

Contents

Xamarin.Android

开始操作

设置和安装

Windows 安装

Android SDK 安装

Android 仿真器设置

硬件加速(Hyper-V 和 HAXM)

Device Manager

设备属性

疑难解答

Android 设备设置

Microsoft Mobile OpenJDK 预览版

了解 Android

第 1 部分:快速入门

第 2 部分:深度分析

了解 Android 多屏幕

第 1 部分:快速入门

第 2 部分:深度分析

面向 Java 开发人员的 Xamarin

应用程序基础知识

可访问性

Android API 级别

Android 资源

Android 资源基础知识

默认资源

备用资源

为不同屏幕创建资源

应用程序本地化和字符串资源

使用 Android 资产

字体

活动生命周期

演练 - 保存活动状态

Android 服务

创建服务

绑定服务

意向服务

启动的服务

前景服务

进程外服务

服务通知

广播接收器

本地化

权限

图形和动画

CPU 体系结构

处理旋转

音频

通知

本地通知

本地通知演练

触控

Android 中的触控

演练 - 在 Android 中使用触控

多点触控跟踪

HttpClient 堆栈和 SSL/TLS

编写响应性应用

用户界面

Android 设计器

使用 Android Designer

设计器基础知识

资源限定符和可视化效果选项

备选布局视图

材料设计功能

材料主题

用户配置文件

初始屏幕

布局

LinearLayout

RelativeLayout

TableLayout

RecyclerView

部件和功能

RecyclerView 示例

扩展示例

ListView

ListView 部件和功能

使用数据填充 ListView

自定义 ListView 的外观

使用 CursorAdapters

使用 ContentProvider

ListView 和活动生命周期

GridView

GridLayout

选项卡式布局

使用 ActionBar 的导航选项卡

控件

ActionBar

自动完成

按钮

单选按钮

切换按钮

CheckBox

自定义按钮

[Calendar](#)

[CardView](#)

[EditText](#)

[库](#)

[导航栏](#)

[选取器](#)

[日期选取器](#)

[时间选取器](#)

[弹出菜单](#)

[RatingBar](#)

[Spinner](#)

[开关](#)

[TextureView](#)

[Toolbar](#)

[替换操作栏](#)

[添加第二个工具栏](#)

[工具栏兼容性](#)

[ViewPager](#)

[带视图的 ViewPager](#)

[带片段的 ViewPager](#)

[WebView](#)

[平台功能](#)

[Android 无线发送](#)

[Android 清单](#)

[使用 Xamarin.Android 的文件访问](#)

[外部存储](#)

[指纹身份验证](#)

[入门](#)

[扫描指纹](#)

[创建 CryptoObject](#)

[响应身份验证回调](#)

[指南和摘要](#)

[注册指纹](#)

[Android 作业计划程序](#)

[Firebase 作业调度程序](#)

[片段](#)

[实现片段](#)

[片段演练 - 第 1 部分](#)

[片段演练 - 第 2 部分](#)

[创建片段](#)

[管理片段](#)

[专用片段类](#)

[提供向后兼容性](#)

[应用链接](#)

[Android 9 Pie](#)

[Android 8 Oreo](#)

[Android 7 Nougat](#)

[Android 6 Marshmallow](#)

[Android 5 Lollipop](#)

[Android 4.4 KitKat](#)

[Android 4.1 Jelly Bean](#)

[Android 4.0 Ice Cream Sandwich](#)

[内容提供商](#)

[工作原理](#)

[使用联系人 ContentProvider](#)

[创建自定义 ContentProvider](#)

[地图和位置](#)

[位置](#)

[映射](#)

[地图应用程序](#)

[地图 API](#)

[获取 Google Maps API 密钥](#)

[使用 Android.Speech](#)

[Java 集成](#)

Android 可调用包装器

使用 JNI

将 Java 移植到 C#

绑定 Java 库

绑定 .JAR

绑定 .AAR

绑定 Eclipse 库项目

自定义绑定

Java 绑定元数据

使用 Javadoc 命名参数

绑定疑难解答

使用本机库

RenderScript

Xamarin.Essentials

入门

加速计

应用信息

气压计

电池

剪贴板

指南针

连接性

数据传输

设备显示信息

设备信息

电子邮件

文件系统帮助程序

手电筒

地理编码

地理位置

陀螺仪

启动器

[磁力计](#)

[主线程](#)

[映射](#)

[打开浏览器](#)

[方向传感器](#)

[电话拨号程序](#)

[电量](#)

[首选项](#)

[屏幕锁定](#)

[安全存储](#)

[SMS](#)

[文本到语音转换](#)

[版本跟踪](#)

[振动](#)

[疑难解答](#)

[数据和云服务](#)

[Azure Active Directory](#)

[入门](#)

[步骤 1。寄存器](#)

[步骤 2。配置](#)

[访问图形 API](#)

[Azure 移动应用](#)

[数据访问](#)

[介绍](#)

[配置](#)

[使用 SQLite.NET ORM](#)

[使用 ADO.NET](#)

[在应用中使用数据](#)

[Google 消息传送](#)

[Firebase 云消息传送](#)

[FCM 通知演练](#)

[Google Cloud Messaging](#)

GCM 通知演练

Web 服务

演练:使用 WCF

部署和测试

应用包大小

构建应用

生成过程

构建特定于 ABI 的 APK

命令行仿真器

调试

在仿真器上调试

在设备上调试

Android 调试日志

可调试属性

环境

GDB

自定义链接器设置

多核设备

性能

分析

准备发布

ProGuard

对 APK 进行签名

对 APK 进行手动签名

查找 Keystore 签名

发布应用

发布到 Google Play

Google 授权服务

APK 扩展文件

手动上传 APK

发布到 Amazon

独立发布

[安装为系统应用](#)

[高级概念和内部机制](#)

[体系结构](#)

[可用程序集](#)

[API 设计](#)

[垃圾回收](#)

[限制](#)

[疑难解答](#)

[疑难解答指南](#)

[常见问题](#)

[应安装哪些 Android SDK 包？](#)

[可以在哪里设置 Android SDK 位置？](#)

[如何更新 Java Development Kit \(JDK\) 版本？](#)

[可否使用 Java Development Kit \(JDK\) 版本 9？](#)

[如何手动安装 Xamarin.Android.Support 包所需的 Android 支持库？](#)

[在 Windows 上调试 Android 需要哪些 USB 驱动程序？](#)

[是否可以从 Windows VM 连接到在 Mac 上运行的 Android 仿真器？](#)

[如何自动化 Android NUnit 测试项目？](#)

[如何在 Android.xml 文件中启用 Intellisense？](#)

[为何我的 Android 发布版本无法连接到 Internet？](#)

[更智能的 Xamarin Android 支持 v4 / v13 NuGet 包](#)

[如何解决 PathTooLongException？](#)

[哪个版本的 Xamarin.Android 添加了 Lollipop 支持？](#)

[Android.Support.v7.AppCompat - 未找到与给定名称匹配的资源:属性“android:actionModeShareDrawable”](#)

[调整 Android Designer 的 Java 内存参数](#)

[Android.Resource.designer.cs 文件不更新](#)

[解决库安装错误](#)

[对 Android SDK 工具的更改](#)

[Xamarin.Android 错误参考](#)

[Wear](#)

[开始操作](#)

[Android 穿戴设备简介](#)

设置和安装

你好, 穿戴设备

用户界面

控件

GridPagerAdapter

平台功能

创建表盘

屏幕大小

部署和测试

在仿真器上调试

在穿戴设备上调试

打包

发行说明

示例

Xamarin.Android 入门

2018/11/2 • [Edit Online](#)

设置和安装

在 Visual Studio 中设置并运行 Xamarin.Android。本部分介绍下载、安装、仿真器配置以及设备预配等内容。

Hello, Android

在这个两部分的指南中，你会使用 Visual Studio 生成第一个 Xamarin.Android 应用程序，并了解使用 Xamarin 进行 Android 应用程序开发的基础知识。同时，本指南会介绍生成和部署 Xamarin.Android 应用程序所需的工具、概念和步骤。

Hello, Android 多屏幕

在这个由两部分组成的指南中，我们会扩展在“了解 Android”指南中创建的应用程序，以便实现第二个屏幕。在此过程中，本指南将介绍基本 Android 应用程序构建基块，并随着更好地了解 Android 应用程序结构和功能来使用户深入了解 Android 体系结构。

面向 Java 开发人员的 Xamarin

本文介绍面向 Java 开发人员的 C# 编程，主要侧重于 Java 开发人员在学习 Xamarin.Android 应用开发时会遇到的 C# 语言功能。

Xamarin University 视频

通过 [Xamarin University](#) 使用 Xamarin for Visual Studio 生成第一个 Android 应用

设置和安装

2018/10/26 • [Edit Online](#)

这部分的主题介绍了如何在 Windows 和 macOS 上安装和配置 Xamarin.Android 以用于 Visual Studio, 如何使用 Android SDK 管理器下载和安装生成和测试应用所需的 Android SDK 工具和组件, 如何配置 Android 仿真器以进行调试, 以及如何将物理 Android 设备连接到开发计算机以调试和最终对应用进行测试。

Windows 安装

本指南提供在 Windows 上安装 Xamarin.Android 所需的安装步骤和配置详细信息。本文结束时, 你需要将一个有效的 Xamarin.Android 安装集成到 Visual Studio 中, 并且准备好开始生成你们的第一个 Xamarin.Android 应用程序。

Mac 安装

本文提供在 Mac 上安装 Xamarin.Android 所需的安装步骤和配置详细信息。本文结束时, 你需要将一个有效的 Xamarin.Android 安装集成到 Visual Studio for Mac 中, 并且准备好开始生成你们的第一个 Xamarin.Android 应用程序。

Android SDK 安装

Visual Studio 包含一个取代 Google 的独立 Android SDK 管理器的 Android SDK 管理器。本文说明如何使用 SDK 管理器下载 Android SDK 工具、平台以及开发 Xamarin.Android 应用所需的其他组件。

Android 仿真器设置

这些文章介绍如何设置 Android Emulator 以测试和调试 Xamarin.Android 应用程序。

Android 设备设置

本文介绍了如何设置物理 Android 设备并将其连接到开发计算机, 这样可以将该设备用于运行和调试 Xamarin.Android 应用程序。

Microsoft Mobile OpenJDK 预览版

本指南介绍了如何切换到 Microsoft 分发的 OpenJDK 预览版。此 OpenJDK 分发适用于移动开发。

Windows 安装

2018/11/2 • [Edit Online](#)

本指南介绍了在 Windows 上安装 Xamarin.Android for Visual Studio 的步骤，并介绍了如何配置 Xamarin.Android 来生成你的第一个 Xamarin.Android 应用程序。

概述

现在，所有版本的 Visual Studio 中都免费附带 Xamarin，并且不需要单独的许可证，可使用 Visual Studio 安装程序下载和安装 Xamarin.Android 工具。（不再需要早期版本的 Xamarin.Android 所需的手动安装和许可步骤。）本指南将介绍以下内容：

- 如何为 Java 开发工具包、Android SDK 和 Android NDK 配置自定义位置。
- 如何启动 Android SDK 管理器，下载并安装其他 Android SDK 组件。
- 如何准备 Android 设备或仿真器进行调试和测试。
- 如何创建第一个 Xamarin.Android 应用项目。

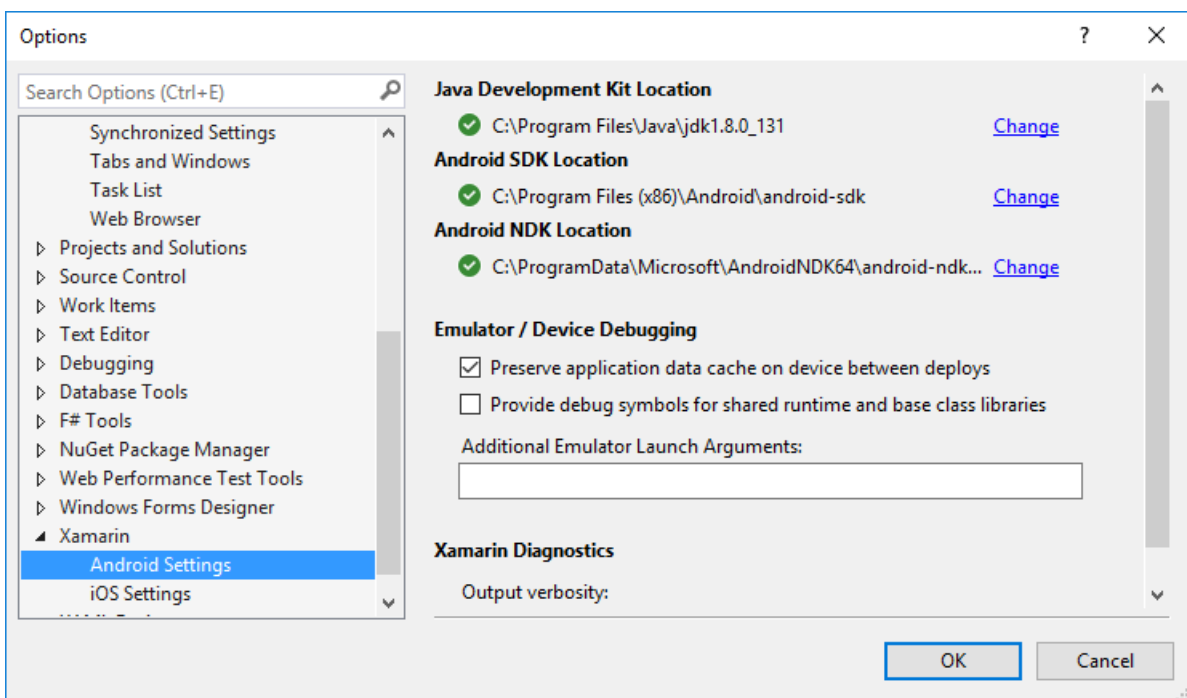
本指南结束时，你需要将一个有效的 Xamarin.Android 安装集成到 Visual Studio 中，并且准备好开始生成你们的第一个 Xamarin.Android 应用程序。

安装

有关安装与 Windows 上的 Visual Studio 配合使用的 Xamarin 的详细信息，请参阅 [Windows 安装指南](#)。

配置

Xamarin.Android 使用 Java 开发工具包 (JDK) 和 Android SDK 生成应用。在安装过程中，Visual Studio 安装程序会将这些工具放置在其默认位置，并使用适当的路径配置来配置开发环境。可单击“工具”>“选项”>“Xamarin”>“Android 设置”查看和更改这些位置：



对于大多数用户，默认位置会起作用，无需进行进一步更改。但是，你可能希望将 Visual Studio 配置为这些工具的自定义位置(例如，如果你已在其他位置安装了 Java JDK、Android SDK 或 NDK)。单击要更改的路径旁边的“更改”，然后导航到新位置。

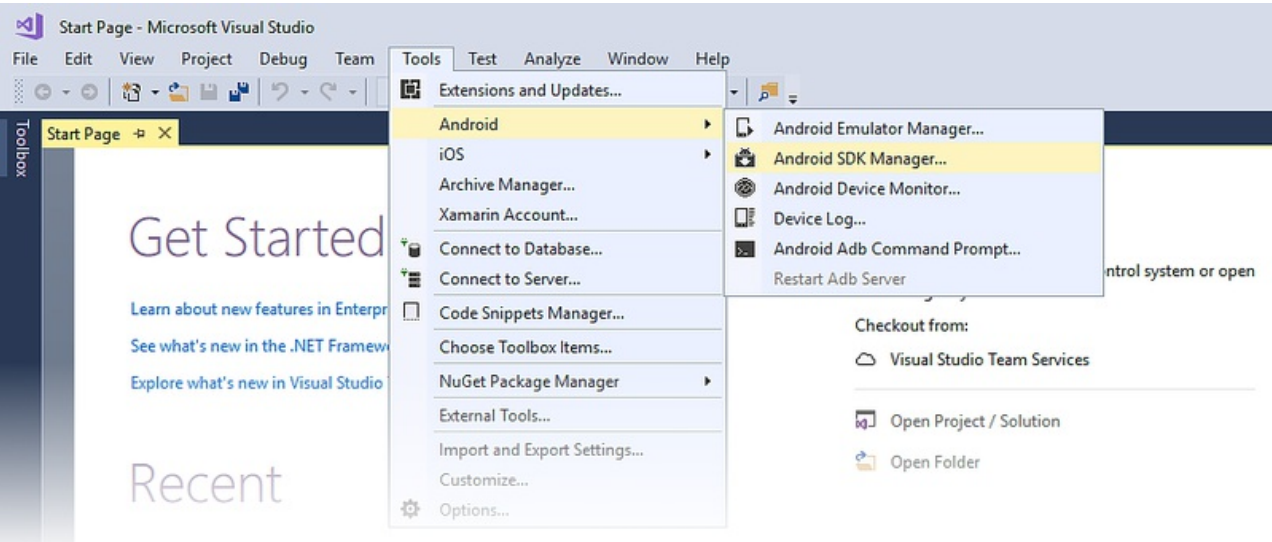
Xamarin.Android 使用JDK 8，这是在为 API 级别 24 或更高级别进行开发时所必需的(JDK 8 还支持低于 24 的 API 级别)。如果专门为 API 级别 23 或更低级别进行开发，可以继续使用 JDK 7。

IMPORTANT

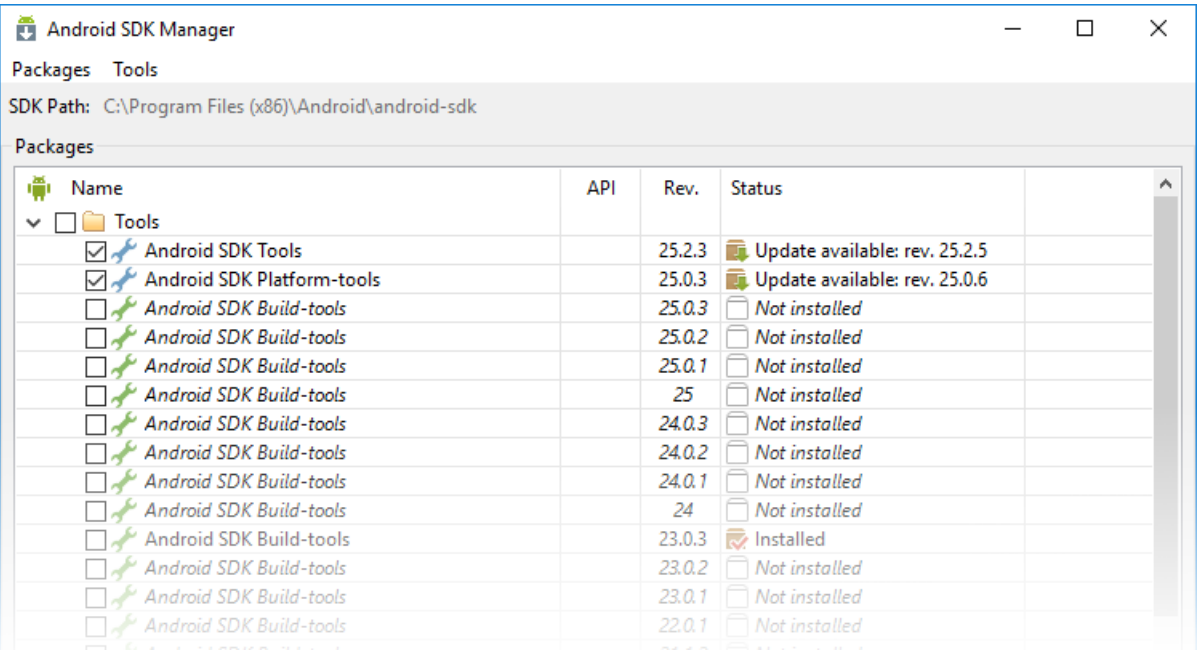
Xamarin.Android 不支持 JDK 9。

Android SDK 管理器

Android 使用多个 Android API 级别设置来确定应用在各种版本的 Android 中的兼容性(有关 Android API 级别的详细信息，请参阅[了解 Android API 级别](#))。根据要面向的 Android API 级别，可能需要下载和安装其他 Android SDK 组件。此外，可能需要安装 Android SDK 中提供的可选工具和仿真器映像。为此，请使用 Android SDK 管理器。可单击“工具”>“Android”>“Android SDK 管理器”，启动“Android SDK管理器”：



默认情况下，Visual Studio 会安装 Google Android SDK 管理器：



可使用 Google Android SDK 管理器安装最高版本为 25.2.3 的 Android SDK 工具包。但是，如果需要使用更高版本的 Android SDK 工具包，则必须安装适用于 Visual Studio 的 Xamarin Android SDK 管理器插件(可从 Visual Studio Marketplace 获取)。这是必需的，因为 Google 的独立 SDK 管理器已在 Android SDK 工具包 25.2.3 版本中

弃用。

有关使用 Xamarin.Android SDK 管理器的详细信息，请参阅 [Android SDK 安装](#)。

Android 仿真器

[Android Emulator](#) 工具可有效地开发和测试 Xamarin.Android 应用。例如，平板电脑等物理设备在部署时可能不可用，或开发人员可能想在提交代码前在计算机上运行某些集成测试。

在计算机上模拟 Android 设备包括以下部分：

- **Google Android Emulator** – 它是基于 [QEMU](#) 的仿真器，用于创建在开发人员的工作站上运行的虚拟化设备。
- **仿真器映像** – 仿真器映像是旨在进行虚拟化的硬件和操作系统的模板或规范。例如，一个仿真器映像可以确定运行安装 Google Play Services 的 Android 7.0 的 Nexus 5X 的硬件要求。另一个仿真器映像可以指定运行 Android 6.0 的 10 英寸平板电脑。
- **Android 虚拟设备 (AVD)** – Android 虚拟设备是从仿真器映像创建的 Android 仿真设备。运行和测试 Android 应用时，Xamarin.Android 将启动 Android Emulator，启动特定 AVD，安装 APK，然后运行应用。

在基于 x86 的计算机上进行开发时，可以通过使用针对 x86 体系结构进行优化的特殊仿真器映像以及以下两项虚拟化技术之一显著提高性能：

1. Microsoft Hyper-V – 可用于运行 Windows 10 的 2018 年 4 月更新或更高版本的计算机。
2. Intel 硬件加速执行管理器 (HAXM) – 可用于运行 OS X、macOS 或较旧 Windows 版本的 x86 计算机。

有关 Android Emulator、Hyper-V 和 HAXM 的详细信息，请参阅[通过硬件加速提高模拟器性能指南](#)。

NOTE

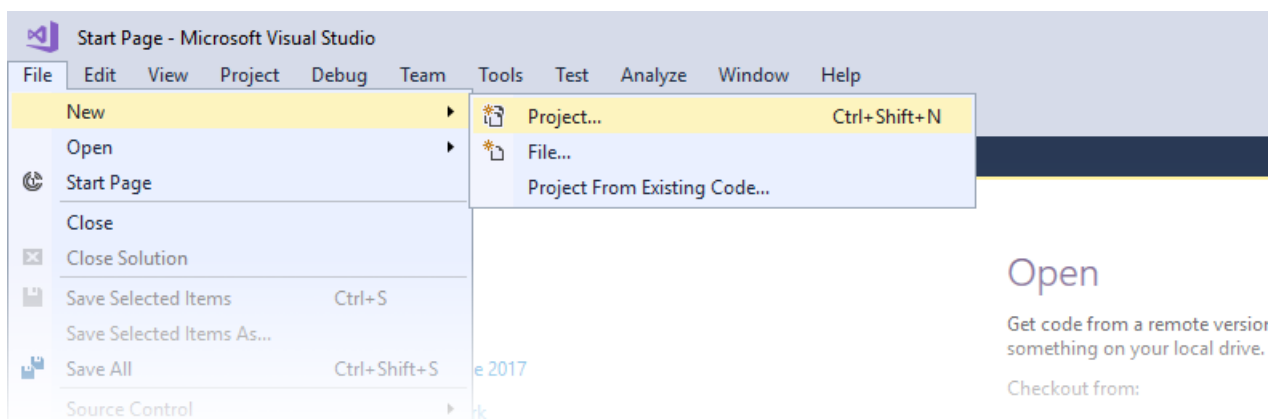
在早于 Windows 10 的 2018 年 4 月更新的 Windows 版本中，HAXM 与 Hyper-V 不兼容。在此情况下，需要[禁用 Hyper-V](#) 或使用不具有 x86 优化的较慢的仿真器映像。

Android 设备

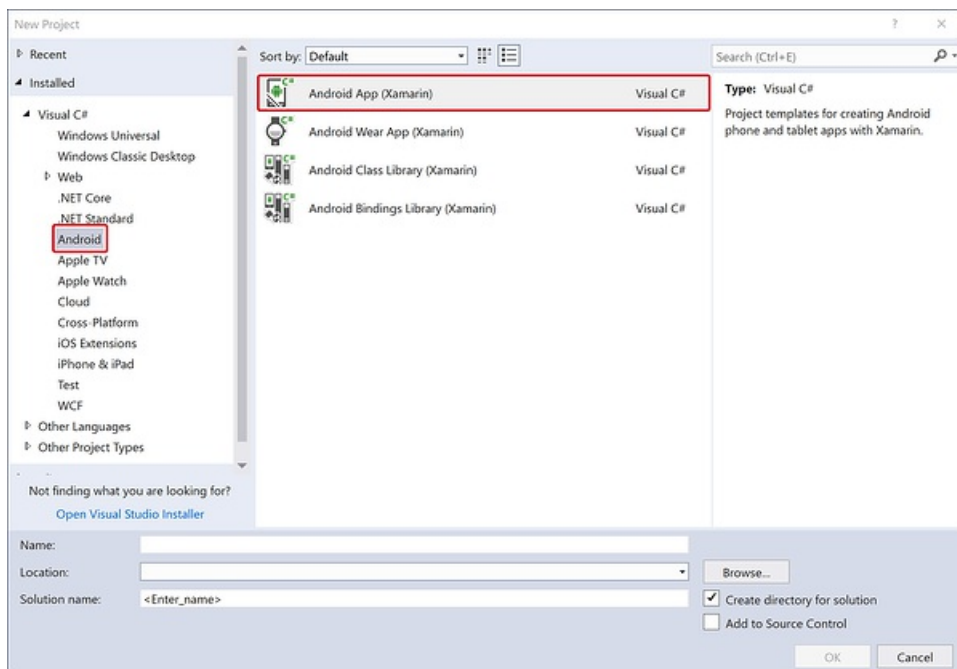
如果有用于测试的 Android 物理设备，现在可设置设备用于开发。通过查看[设置设备进行开发](#)配置 Android 设备进行开发，然后将其连接到计算机以运行和调试 Xamarin.Android 应用程序。

创建应用程序

安装 Xamarin.Android 后，可启动 Visual Studio 创建一个新项目。单击“文件”>“新建”>“项目”，开始创建应用：



在“新建项目”对话框中的“模板”下，选择“Android”，然后单击右窗格中的“Android 应用”。输入应用名称（在下面的屏幕截图中，应用称为 MyApp），然后单击“确定”：



就这么简单！现在即可使用 Xamarin.Android 创建 Android 应用程序！

总结

本文介绍了如何在 Windows 上设置和安装 Xamarin.Android 平台、如何(可选)使用自定义 Java JDK 和 Android SDK 安装位置配置 Visual Studio、如何启动 SDK Manager 安装其他 Android SDK 组件、如何设置 Android 设备或仿真器，以及如何开始构建你的第一个应用程序。

下一步是查看[了解 Android 教程](#)，了解如何创建可用的 Xamarin.Android 应用。

相关链接

- [下载 Visual Studio](#)
- [安装 Visual Studio Tools for Xamarin](#)
- [系统要求](#)
- [Android SDK 安装](#)
- [Android 仿真器设置](#)
- [设置设备进行开发](#)
- [在 Android Emulator 上运行应用](#)

设置用于 Xamarin.Android 的 Android SDK

2018/11/2 • [Edit Online](#)

Visual Studio 包含 Android SDK 管理器, 用于下载 Android SDK 工具、平台以及开发 Xamarin.Android 应用所需的其他组件。

概述

本指南介绍如何在 Visual Studio 和 Visual Studio for Mac 中使用 Xamarin Android SDK 管理器。

NOTE

本指南仅适用于 Visual Studio 2017 和 Visual Studio for Mac。

Xamarin Android SDK 管理器(作为 .NET 移动开发的一部分安装)可帮助你下载开发 Xamarin.Android 应用所需的最新 Android 组件。它取代了已被弃用的 Google 的独立 SDK 管理器。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

要求

若要使用 Xamarin Android SDK 管理器, 需要以下工具:

- Visual Studio 2017 (Community、Professional 或 Enterprise 版本)。需要 Visual Studio 2017 版本 15.7 或更高版本。
- Visual Studio Tools for Xamarin 版本 4.10.0 或更高版本(作为使用 .NET 的移动开发工作负载的一部分安装)。

Xamarin Android SDK 管理器与 Visual Studio 2015 不兼容。Visual Studio 2015 的用户应使用 Android SDK 中由 Google 提供的 SDK 管理工具。

Xamarin Android SDK 管理器还需要 Java 开发工具包(此工具包自动安装在 Xamarin.Android 中)。有多种 JDK 可选方案供选择:

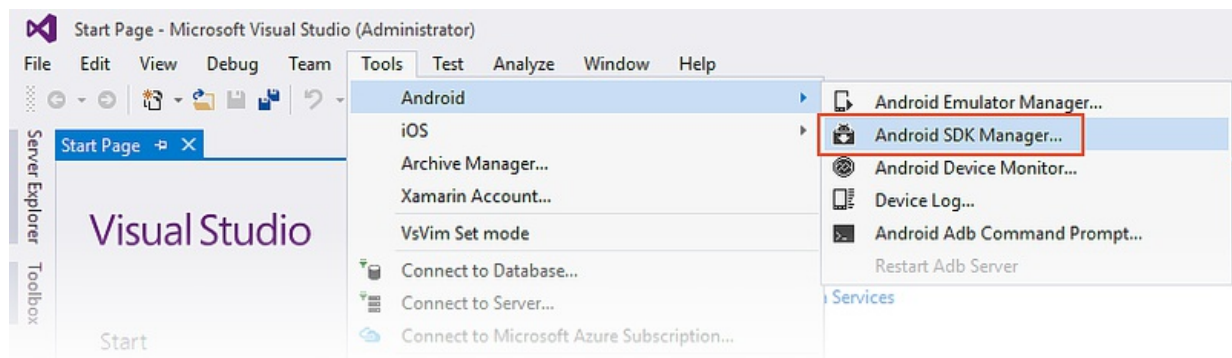
- Xamarin.Android 默认使用 [JDK 8](#), 这是在为 API 级别 24 或更高级别进行开发时所必需的(JDK 8 还支持低于 24 的 API 级别)。
- 如果专门为 API 级别 23 或更低级别进行开发, 可以继续使用 [JDK 7](#)。
- 如果使用 Visual Studio 15.8 Preview 5 或更高版本, 可尝试使用 [Microsoft Mobile OpenJDK 分发](#)(目前处于预览阶段)而不使用 JDK 8。

IMPORTANT

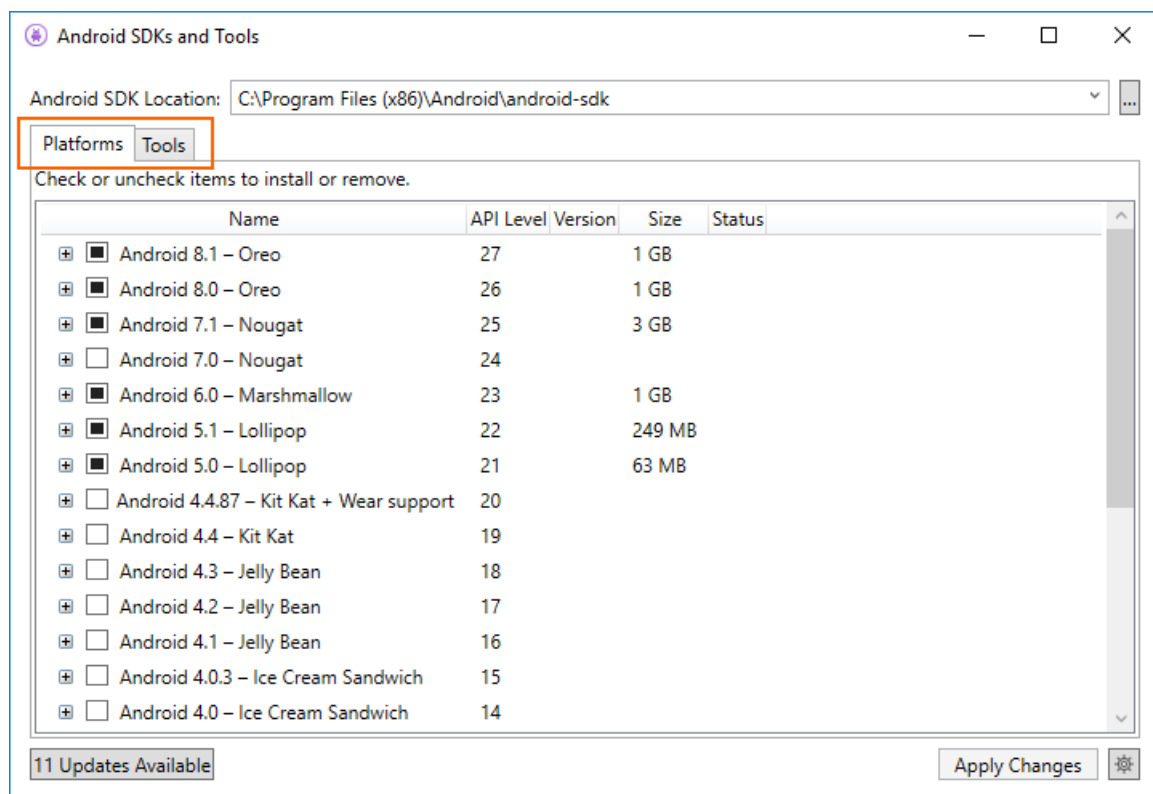
Xamarin.Android 不支持 JDK 9。

SDK 管理器

若要在 Visual Studio 中启动 SDK 管理器, 请单击“工具”>“Android”>“Android SDK 管理器”:



Android SDK 管理器会在“Android SDK 和工具”屏幕中打开。此屏幕上有两个选项卡 –“平台”和“工具”：



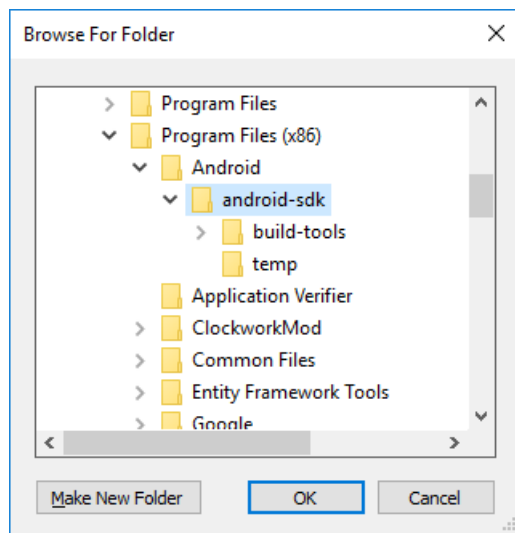
下面的部分将对“Android SDK 和工具”屏幕进行更详细的介绍。

Android SDK 位置

Android SDK 位置是在“Android SDK 和工具”屏幕的顶部进行配置的，如以上屏幕截图中所示。必须正确配置此位置，这样“平台”和“工具”选项卡才能正常工作。出于下面一个或多个原因，可能需要设置 Android SDK 的位置：

1. Android SDK 管理器无法定位 Android SDK。
2. 你已在备用(非默认)位置安装了 Android SDK。

若要设置 Android SDK 的位置，请单击“Android SDK 位置”最右侧的省略号 (...) 按钮。这将打开“浏览文件夹”对话框，用于导航到 Android SDK 的位置。在下面的屏幕截图中，已选中“Program Files (x86)\Android”下面的 Android SDK：

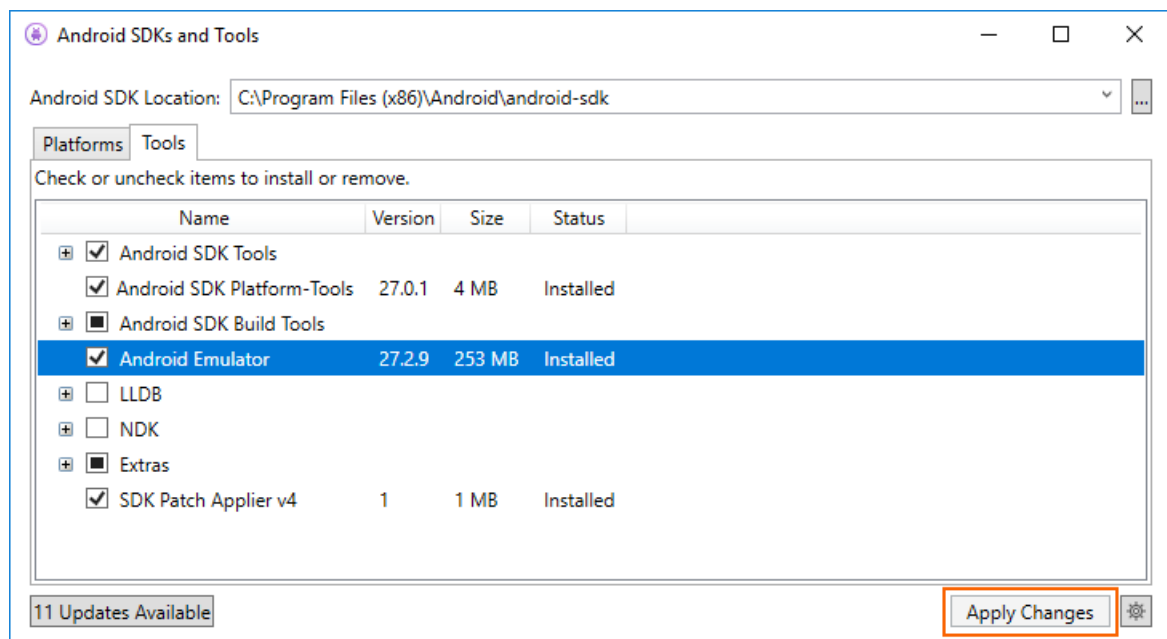


单击“确定”后，SDK 管理器将管理安装在所选位置的 Android SDK。

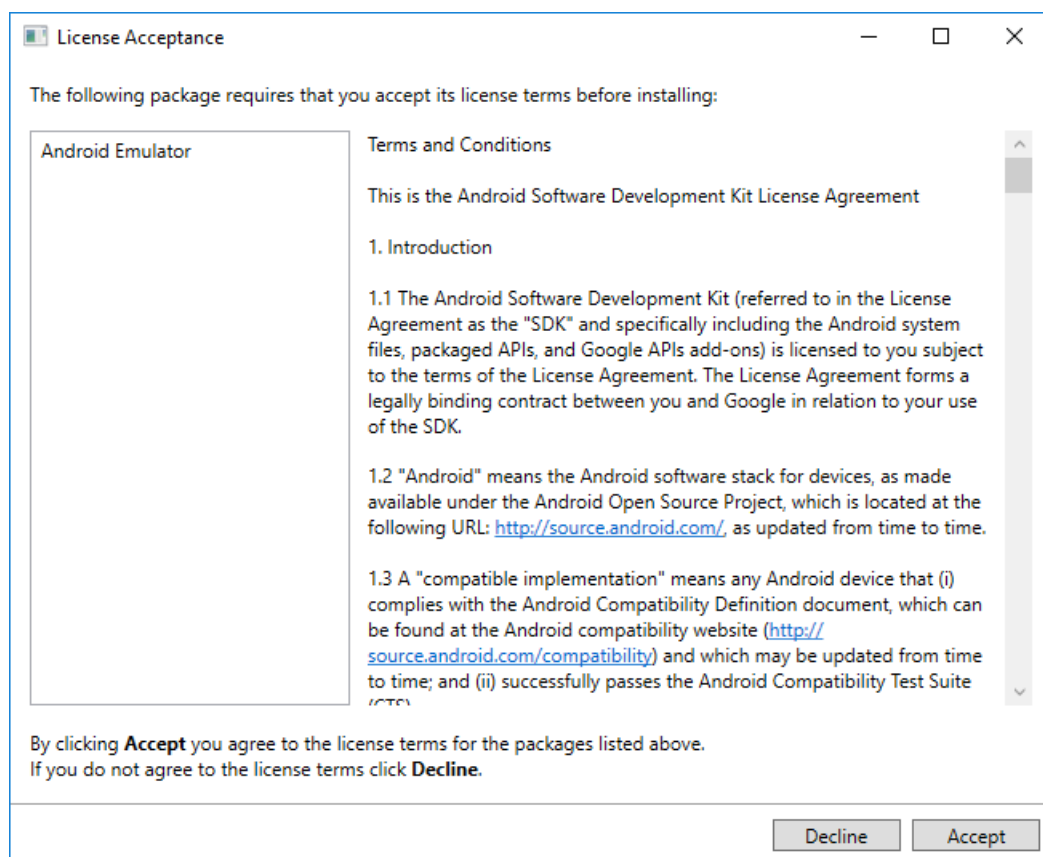
“工具”选项卡

“工具”选项卡显示“工具”和“附加程序”的列表。使用此选项卡以安装 Android SDK 工具、平台工具和生成工具。此外，还可以安装 Android Emulator、低级别调试器 (LLDB)、NDK、HAXM 加速和 Google Play 库。

例如，若要下载 Google Android Emulator 包，请单击“Android Emulator”旁的复选标记，然后单击“应用更改”按钮：



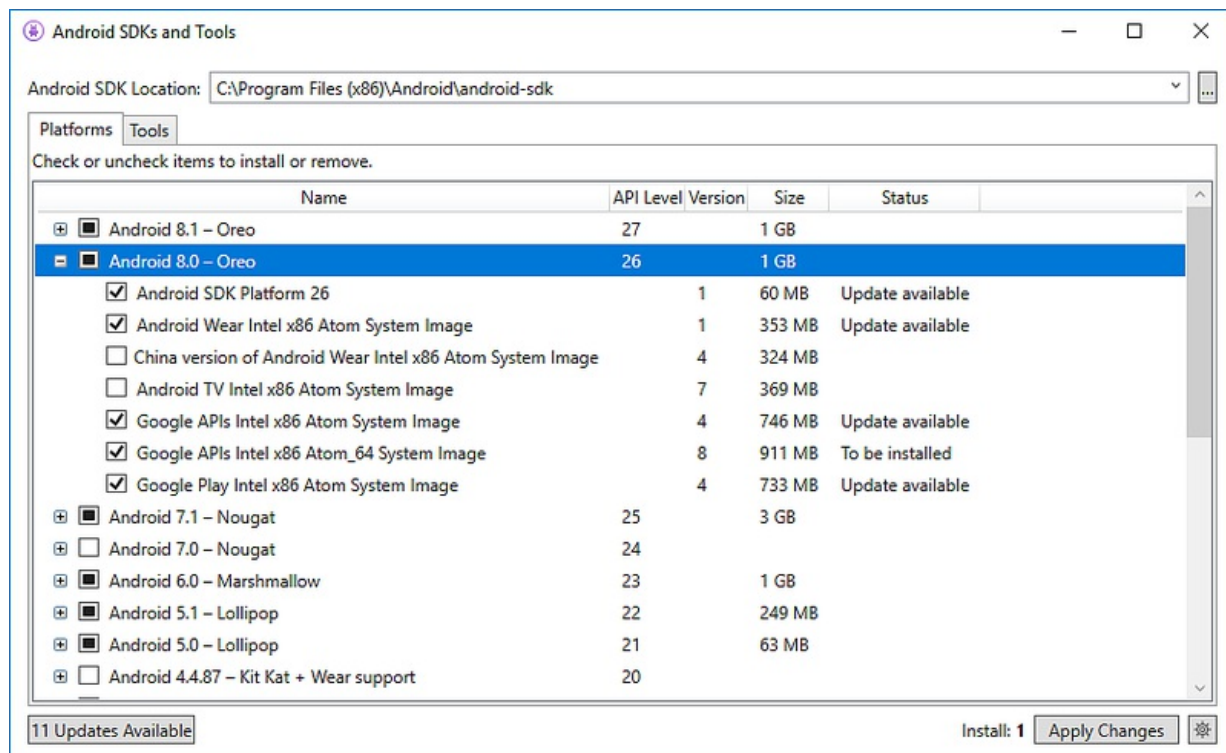
可能显示包含“以下包需要你在安装前接受其许可条款:”消息的对话框



如果接受这些条款和条件，请单击“接受”。在窗口底部，有一个进度栏会指示下载和安装进度。安装完成后，“工具”选项卡将显示已安装所选的工具和附加程序。

“平台”选项卡

“平台”选项卡显示平台 SDK 版本以及适用于每个平台的其他资源(例如系统映像)的列表：

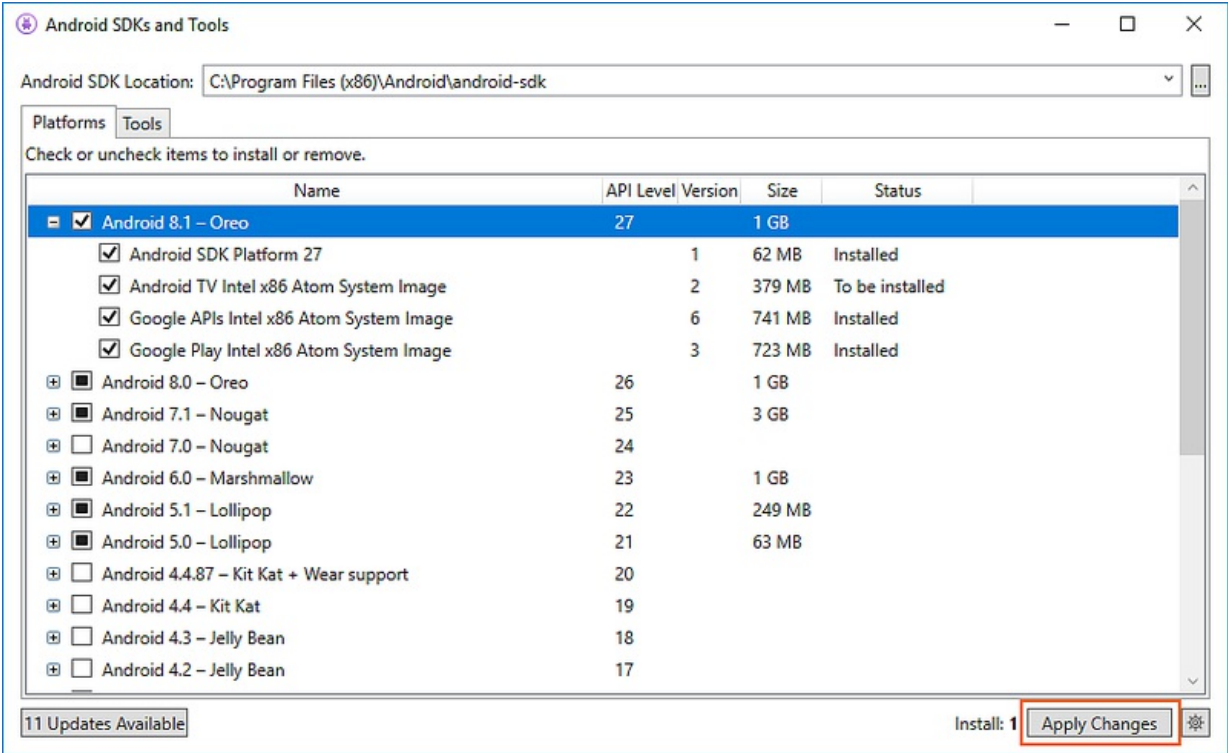


此屏幕列出了 Android 版本(例如“Android 8.0”)、代码名称(“Oreo”)、API 级别(例如“26”)以及平台对应组件的大小(例如“1 GB”)。使用“平台”选项卡安装要面向的 Android API 级别的组件。(有关 Android 版本和 Android API 级别的详细信息，请参阅[了解 Android API 级别](#))。

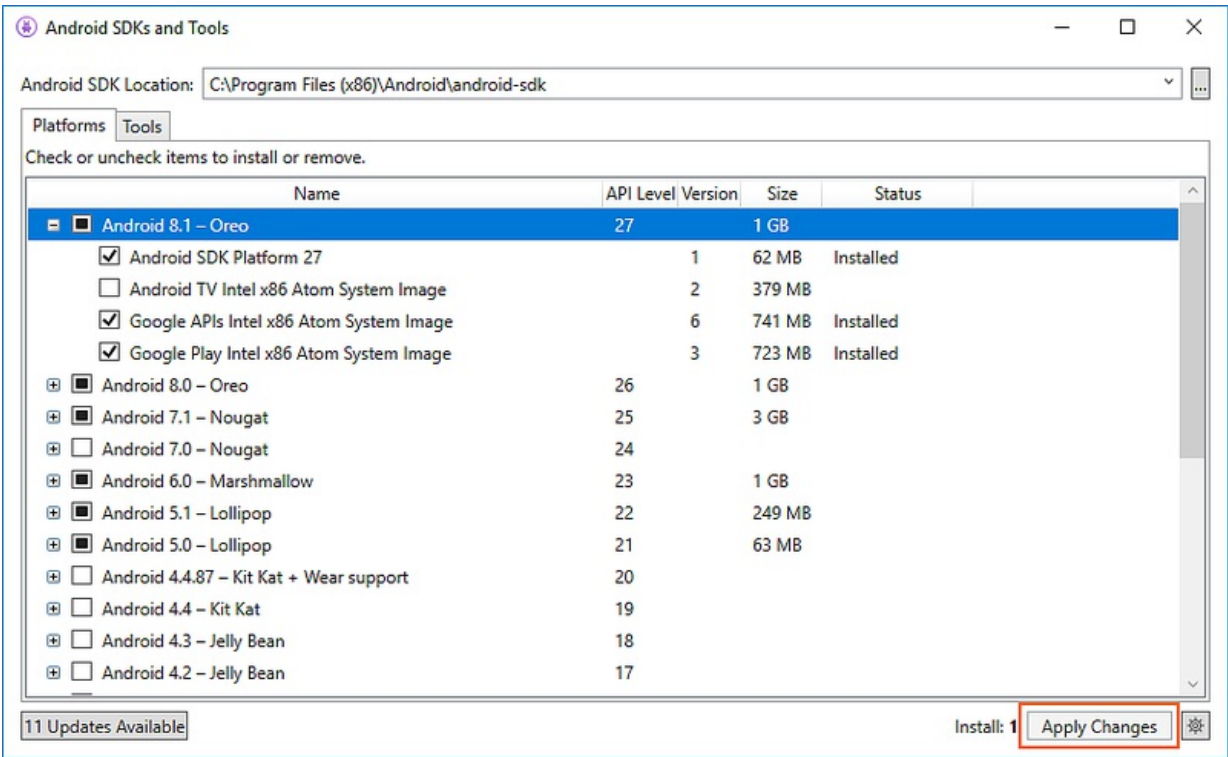
安装某个平台的所有组件后，该平台名称旁边将显示一个复选标记。如果某个平台的组件并非已全部安装，该

平台的此框则会被填充。可以通过单击某个平台左侧的“+”框展开此平台以查看其组件（以及已安装的组件）。单击“-”取消展开某个平台的组件列表。

若要将另一个平台添加到 SDK，请单击此平台旁边的框 - 直到显示复选标记 - 以安装其所有组件，然后单击“应用更改”：



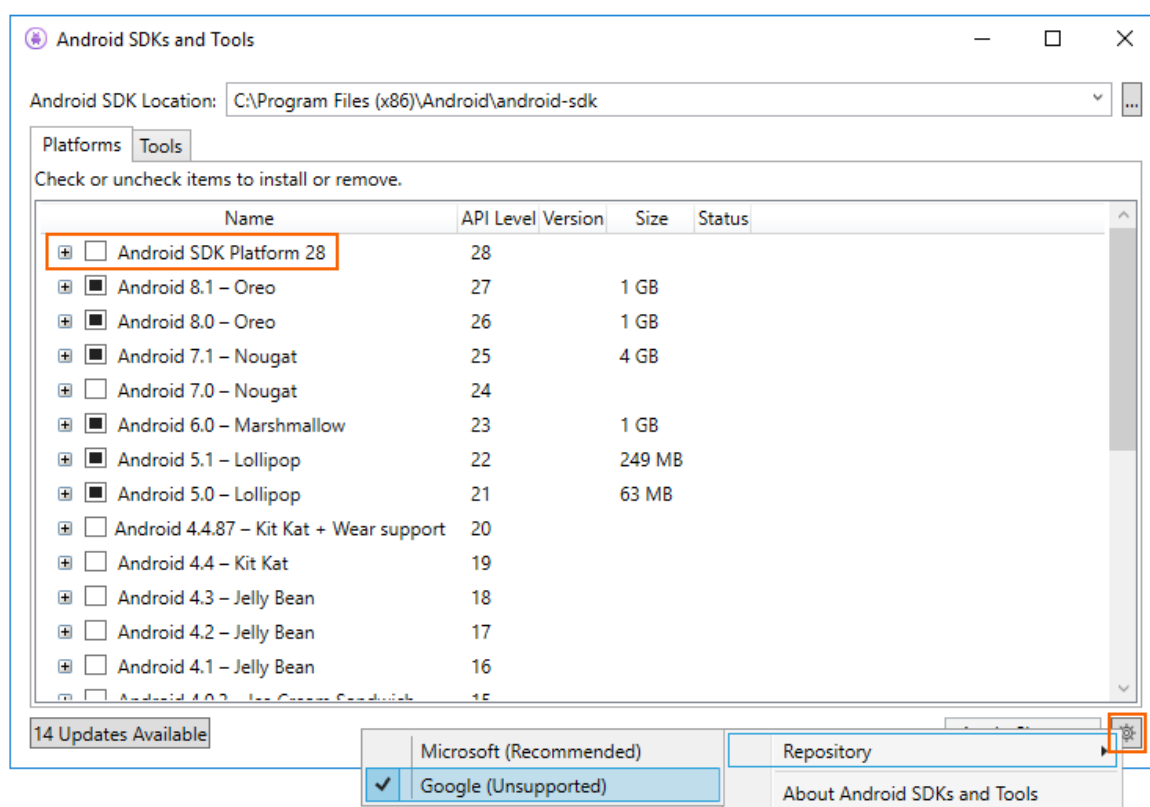
若要仅安装特定组件，请单击一次此平台旁边的框。然后可以选择所需的任何单个组件：



请注意，要安装的组件数量显示在“应用更改”按钮旁边。单击“应用更改”按钮后，将看到上示“许可证接受”屏幕。如果接受这些条款和条件，请单击“接受”。如果有多个组件要安装，则可能会多次看到此对话框。在窗口底部，有一个进度栏会指示下载和安装进度。下载和安装过程完成后（可能需要好几分钟的时间，具体取决于需要下载的组件数量），已添加的组件会被复选标记标记出来并被列为“已安装”。

存储库选择

默认情况下，Android SDK 管理器从由 Microsoft 托管的存储库下载平台组件和工具。如果需要访问实验性 Alpha/Beta 平台和 Microsoft 存储库中尚不可用的工具，可以将 SDK 管理器切换为使用 Google 的存储库。若要进行这种切换，请单击右下角的齿轮图标，然后选择“存储库”>“Google (不受支持)”：



选择 Google 存储库后，以前不可用的“平台”选项卡中可能显示其他包。（在上面的屏幕截图中，切换到 Google 存储库后添加了“Android SDK 平台 28”。）请注意，使用 Google 存储库不受支持，因此不建议将其用于日常开发。

若要切换回平台和工具支持的存储库，请单击“Microsoft (推荐)”。这会将包和工具的列表还原到默认选择。

总结

本指南说明了如何在 Visual Studio 和 Visual Studio for Mac 中安装和使用 Xamarin Android SDK 管理器工具。

相关链接

- [了解 Android API 级别](#)
- [对 Android SDK 工具的更改](#)

Android 仿真器设置

2018/11/2 • [Edit Online](#)

本指南介绍如何准备好 Android Emulator 以测试应用。

概述

可使用各种配置运行 Android Emulator 来模拟不同的设备。每个配置称为虚拟设备。在仿真器上部署和测试应用时，选择模拟物理 Android 设备（如 Nexus 或 Pixel 手机）的预配置或自定义虚拟设备。

下面列出的部分介绍了如何加速 Android Emulator 以最大限度提高性能、如何使用 Android Device Manager 创建和自定义虚拟设备，以及如何自定义虚拟设备的配置文件属性。此外，疑难解答部分说明了常见模拟器问题和解决方法。

部分

通过硬件加速提高仿真器性能

如何使用 Hyper-V 或 HAXM 虚拟化技术为计算机准备最大的 Android Emulator 性能。由于在没有硬件加速的情况下 Android Emulator 的运行可能会极度缓慢，因此，建议在使用模拟器之前在计算机上启用硬件加速。

使用 Android Device Manager 管理虚拟设备

如何使用 Android Device Manager 创建和自定义虚拟设备。

编辑 Android 虚拟设备属性

如何使用 Android Device Manager 编辑虚拟设备的配置文件属性。

Android Emulator 疑难解答

本文介绍运行 Android Emulator 时最常见的警告消息和问题，以及解决方法和相关技巧。

配置 Android Emulator 后，请参阅[使用 Google Android Emulator 进行调试](#)，了解如何启动模拟器以及如何使用它测试并调试应用的信息。

NOTE

自 Android SDK 工具版本 26.0.1 和更高版本开始，Google 已删除了对现有 AVD/SDK 管理器的支持，以支持其新的 CLI（命令行接口）工具。由于此弃用更改，因此现在将 Xamarin SDK/Device Manager 用于 Android 工具 26.0.1 和更高版本，而不使用 Google SDK/Device Manager。有关 Xamarin SDK 管理器的详细信息，请参阅[设置 Xamarin.Android 的 Android SDK](#)。

通过硬件加速提高仿真器性能 (Hyper-V & HAXM)

2018/11/2 • [Edit Online](#)

本文介绍了如何使用计算机的硬件加速功能最大限度提高 Android Emulator 的性能。

Visual Studio 便于开发人员在无法使用 Android 设备的情况下通过使用 Android Emulator 来测试和调试 Xamarin.Android 应用程序。但是，如果硬件加速在运行 Android 仿真器的计算机上不可用，那么它的运行速度太慢。通过将特殊的 x86 虚拟设备映像与计算机的虚拟化功能结合使用，可以极大地提高 Android Emulator 的性能。

在 Windows 上加速 Android Emulator

以下虚拟化技术可用于加速 Android Emulator：

1. **Microsoft 的 Hyper-V 和虚拟机监控程序平台。** [Hyper-V](#) 是 Windows 的虚拟化功能，使虚拟的计算机系统可以在物理主计算机上运行。
2. **Intel 硬件加速执行管理器 (HAXM)。** [HAXM](#) 是运行 Intel CPU 的计算机所用的虚拟化引擎。

为获得最佳性能，建议使用 Hyper-V 加速 Android Emulator。若你的计算机没有 Hyper-V，则可使用 HAXM。如果满足以下条件，Android Emulator 将自动使用硬件加速：

- 硬件加速在开发计算机上可用并已启用。
- 仿真器正在运行为基于 x86 的虚拟设备创建的系统映像。

IMPORTANT

不可在另一 VM (例如由 VirtualBox、VMWare 或 Docker 托管的 VM) 内运行经过 VM 加速的仿真器。必须[直接在系统硬件上运行 Android Emulator](#)。

有关使用 Android Emulator 进行启动和调试的信息，请参阅 [Android Emulator 调试](#)。

使用 HYPER-V 加速

推荐使用 Hyper-V 加速 Android Emulator。在启用 Hyper-V 之前，请阅读以下部分以验证你的计算机是否支持 Hyper-V。

验证对 Hyper-V 的支持

Hyper-V 在 Windows 虚拟机监控程序平台上运行。若要将 Android Emulator 与 Hyper-V 配合使用，计算机必须满足以下条件才能支持 Windows 虚拟机监控程序平台：

- 计算机硬件必须满足以下要求：
 - 支持二级地址转换 (SLAT) 的 64 位 Intel 或 AMD Ryzen CPU。
 - CPU 支持 VM 监视器模式扩展 (Intel CPU 的 VT-c 技术)。
 - 内存至少为 4 GB。
- 在计算机的 BIOS 中，必须启用以下项：
 - 虚拟化技术 (标签可能因主板制造商而不同)。
 - 硬件强制执行数据执行保护。
- 计算机必须更新到 Windows 10 2018 年 4 月更新 (版本 1803) 或更高版本。可使用以下步骤验证

Windows 版本是否为最新版本：

1. 在 Windows 搜索框中，输入“关于”。
2. 在搜索结果中选择“关于你的电脑”。
3. 在“关于”对话框中向下滚动到“Windows 规范”部分。
4. 验证“版本”最低为 1803 版：

| | |
|--------------|-----------------------|
| Edition | Windows 10 Enterprise |
| Version | 1803 |
| Installed on | 4/30/2018 |
| OS build | 17134.1 |

要验证计算机硬件和软件是否与 Hyper-V 兼容，请打开命令提示符并键入以下命令：

```
systeminfo
```

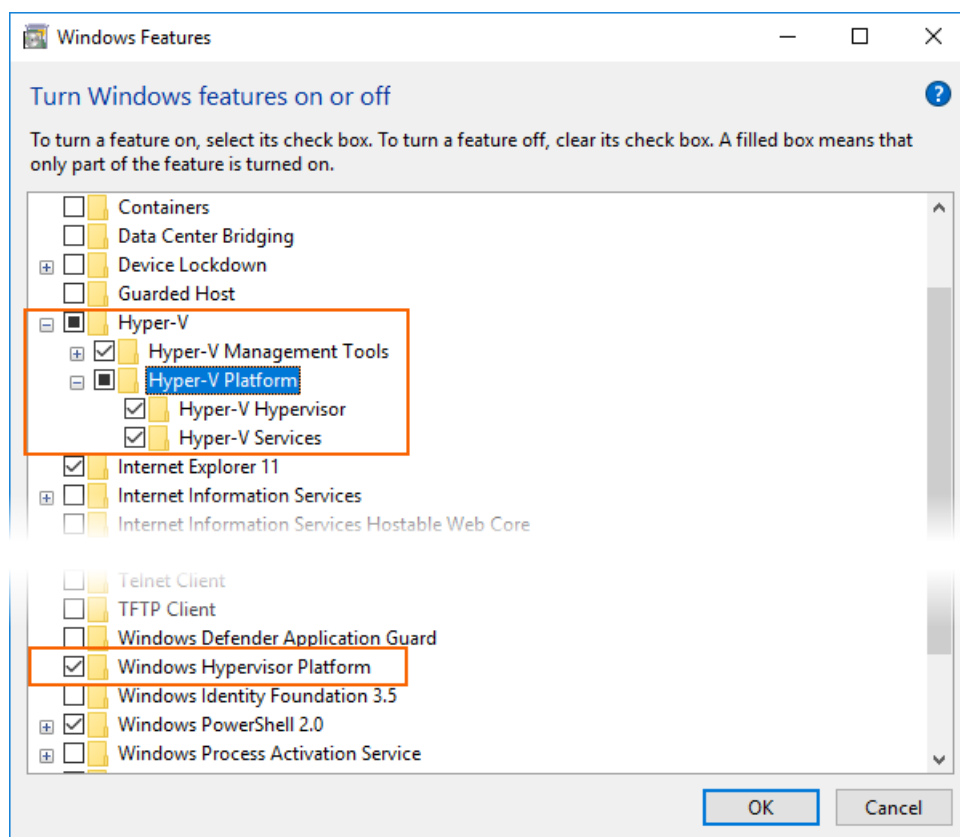
如果列出的所有 Hyper-V 要求的值均为“是”，则计算机可以支持 Hyper-V。例如：

```
Hyper-V Requirements:      VM Monitor Mode Extensions: Yes
                           Virtualization Enabled In Firmware: Yes
                           Second Level Address Translation: Yes
                           Data Execution Prevention Available: Yes
```

启用 HYPER-V 加速

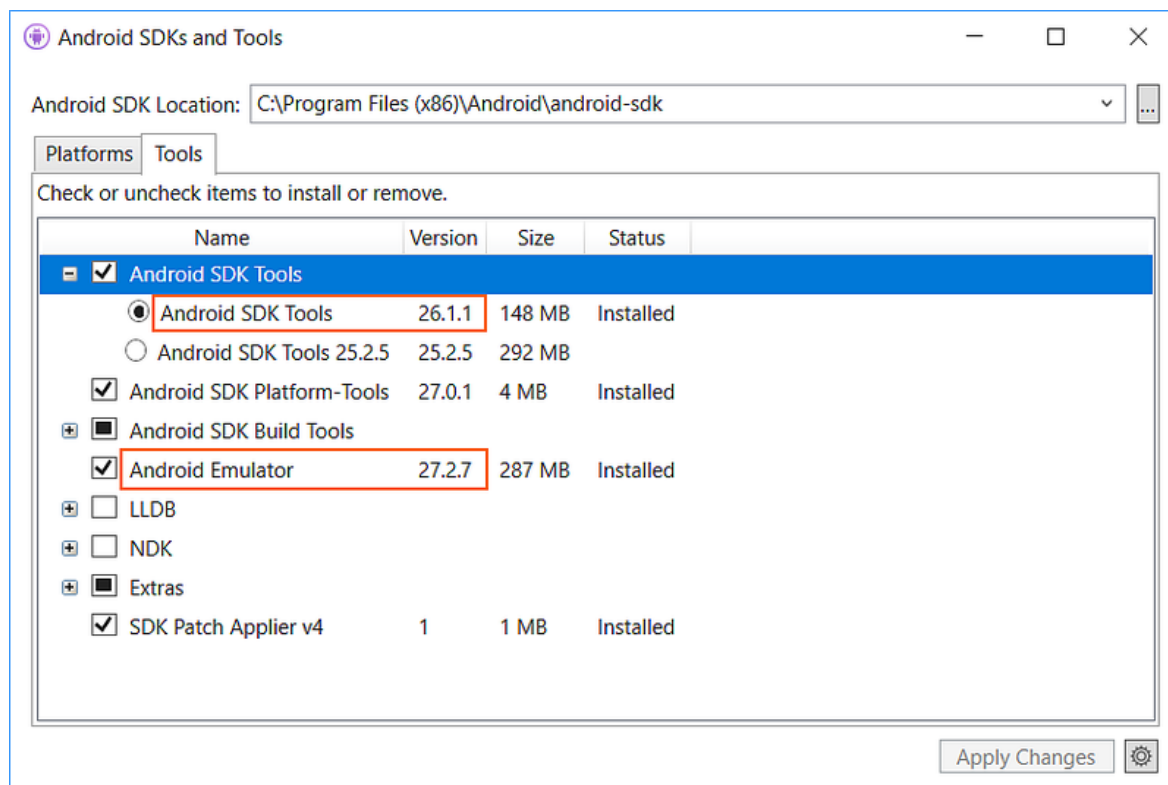
若计算机符合上述条件，请执行以下步骤使用 Hyper-V 加速 Android Emulator：

1. 在 Windows 搜索框中输入“Windows 功能”，然后在搜索结果中选择“打开或关闭 Windows 功能”。
在“Windows 功能”对话框中，启用“Hyper-V”和“Windows 虚拟机监控程序平台”：



进行这些更改后，重新启动计算机。

2. 安装 [Visual Studio 15.8 或更高版本](#) (此版本 Visual Studio 通过 Hyper-V 提供用于运行 Android Emulator 的 IDE 支持)。
3. 安装 **Android Emulator 包 27.2.7 或更高版本**。要安装此包，请在 Visual Studio 中导航到“工具”>“Android”>“Android SDK 管理器”。选择“工具”选项卡，确保 Android Emulator 版本至少为 27.2.7。另请确保 Android SDK Tools 版本为 26.1.1 或更高版本：



创建虚拟设备时(参阅[使用 Android Device Manager 管理虚拟设备](#))，请确保选择基于 x86 的系统映像。如果使用基于 ARM 的系统映像，虚拟设备将无法加速并且运行缓慢。

使用 HAXM 加速

若计算机不支持 Hyper-V，请使用 HAXM 加速 Android Emulator。若要使用 HAXM，必须禁用 Device Guard。

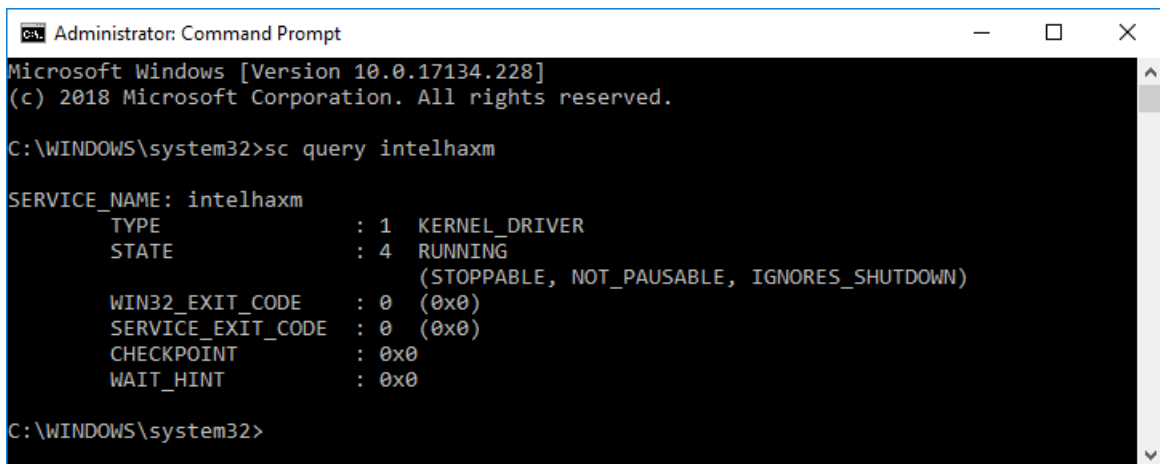
验证 HAXM 支持

要确定硬件是否支持 HAXM，请按照[我的处理器是否支持 Intel 虚拟化技术？](#)中的步骤操作。若硬件支持 HAXM，可以使用以下步骤检查是否已安装 HAXM：

1. 打开命令提示符窗口，然后输入以下命令：

```
sc query intelhaxm
```

2. 检查输出，查看 HAXM 进程是否正在运行。如果它正在运行，你会看到将 `intelhaxm` 状态列为 `RUNNING` 的输出。例如：



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>sc query intelhaxm

SERVICE_NAME: intelhaxm
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE      : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT          : 0x0
        WAIT_HINT           : 0x0

C:\WINDOWS\system32>
```

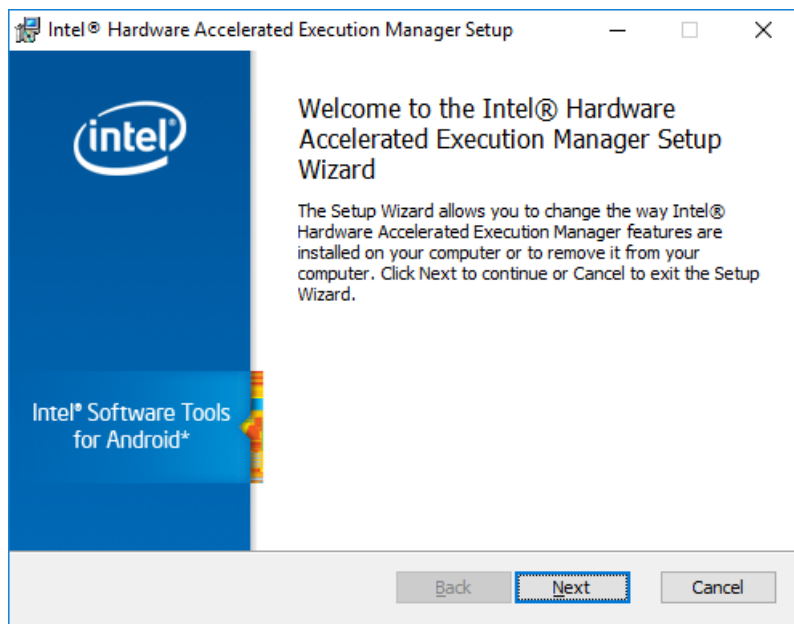
如果未将 `STATE` 设置为 `RUNNING`，则未安装 HAXM。

如果计算机可以支持 HAXM 但未安装 HAXM，请按照下一部分中的步骤安装 HAXM。

安装 HAXM

可以从 [Intel 硬件加速执行管理器](#) 页获取适用于 Windows 的 HAXM 安装包。若要下载并安装 HAXM，请按照下列步骤操作：

1. 从 Intel 网站下载适用于 Windows 的最新 [HAXM 虚拟化引擎](#) 安装程序。直接从 Intel 网站下载 HAXM 安装程序的优点是可以确保使用最新版本。
2. 运行 `intelhaxm android.exe`，启动 HAXM 安装程序。接受安装程序对话框中的默认值：



创建虚拟设备时(参阅[使用 Android Device Manager 管理虚拟设备](#))，请确保选择基于 x86 的系统映像。如果使用基于 ARM 的系统映像，虚拟设备将无法加速并且运行缓慢。

疑难解答

有关解决硬件加速问题的帮助，请参阅 Android Emulator [疑难解答](#) 指南。

在 macOS 上加速 Android Emulator

以下虚拟化技术可用于加速 Android Emulator：

1. **Apple 的虚拟机监控程序框架。**[虚拟机监控程序](#)是 macOS 10.10 及更高版本的一项功能，可在 Mac 上运行虚拟机。
2. **Intel 硬件加速执行管理器 (HAXM)。**[HAXM](#) 是运行 Intel CPU 的计算机所用的虚拟化引擎。

为获得最佳性能，建议使用虚拟机监控程序框架加速 Android Emulator。如果 Mac 上没有虚拟机监控程序框架，则可以使用 HAXM。如果满足以下条件，Android Emulator 将自动使用硬件加速：

- 硬件加速在开发计算机上可用并已启用。
- 仿真器正在运行为基于 x86 的虚拟设备创建的系统映像。

IMPORTANT

不可在另一 VM(例如由 VirtualBox、VMWare 或 Docker 托管的 VM)内运行经过 VM 加速的仿真器。必须[直接在系统硬件上](#)运行 Android Emulator。

有关使用 Android Emulator 进行启动和调试的信息，请参阅 [Android Emulator 调试](#)。

使用虚拟机监控程序框架加速

要将 Android Emulator 与虚拟机监控程序框架配合使用，Mac 必须符合以下条件：

- Mac 必须运行 macOS 10.10 或更高版本。
- Mac 的 CPU 必须能够支持虚拟机监控程序框架。

若 Mac 符合这些条件，Android Emulator 将自动使用虚拟机监控程序框架进行加速(即使已安装 HAXM)。若不

确定 Mac 是否支持虚拟机监控程序框架，请参阅[疑难解答指南](#)，了解验证 Mac 是否支持虚拟机监控程序的方法。

如果 Mac 不支持虚拟机监控程序框架，可以使用 HAXM 加速 Android Emulator (如下所述)。

使用 HAXM 加速

如果你的 Mac 不支持虚拟机监控程序框架 (或者 macOS 版本低于 10.10)，则可使用“Intel 的硬件加速执行管理器”(HAXM) 来加速 Android Emulator。

在首次将 HAXM 与 Android Emulator 配合使用之前，最好先验证 HAXM 是否已安装并可供 Android Emulator 使用。

验证 HAXM 支持

可使用以下步骤检查是否已安装 HAXM：

1. 打开终端，然后输入以下命令：

```
~/Library/Developer/Xamarin/android-sdk-macosx/tools/emulator -accel-check
```

此命令假定将 Android SDK 安装在默认位置“/Library/Developer/Xamarin/android-sdk-macosx”；否则，请在 Mac 上修改 Android SDK 位置的路径。

2. 若已安装 HAXM，则上面的命令将返回类似于以下结果的消息：

```
HAXM version 7.2.0 (3) is installed and usable.
```

若 HAXM 未安装，则返回类似于以下输出的消息：

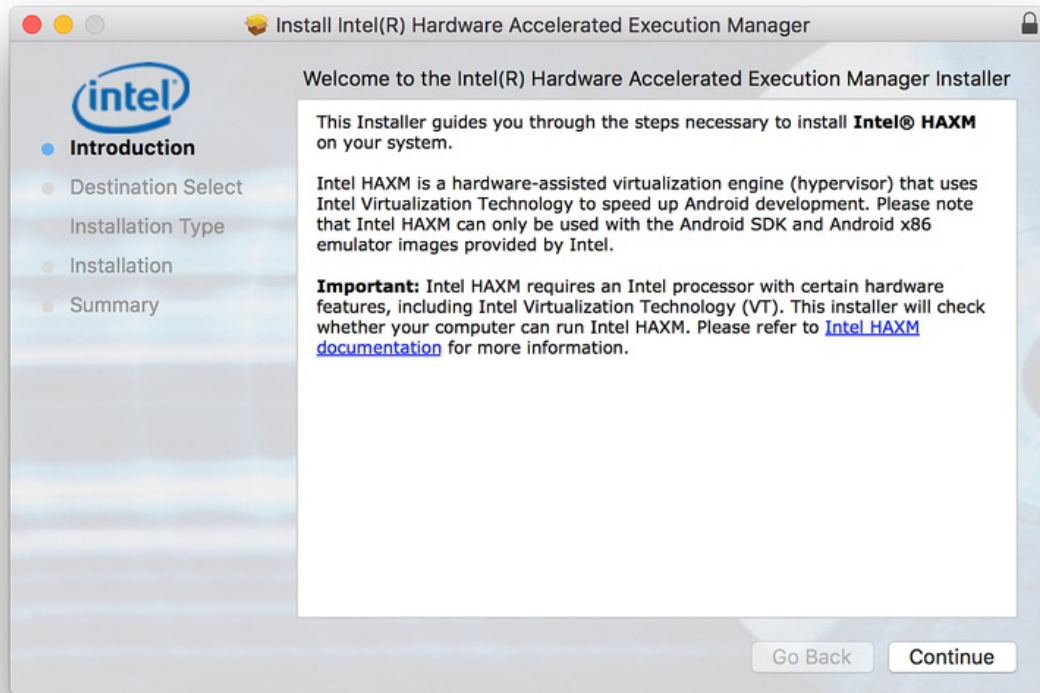
```
HAXM is not installed on this machine (/dev/HAX is missing).
```

如果未安装 HAXM，请按照下一部分的步骤安装 HAXM。

安装 HAXM

可以从[Intel 硬件加速执行管理器](#)页获取适用于 macOS 的 HAXM 安装包。若要下载并安装 HAXM，请按照下列步骤操作：

1. 从 Intel 网站下载适用于 macOS 的最新 [HAXM 虚拟化引擎](#) 安装程序。
2. 运行 HAXM 安装程序。接受安装程序对话框中的默认值：



疑难解答

有关解决硬件加速问题的帮助，请参阅 Android Emulator [疑难解答](#) 指南。

相关链接

- [在 Android Emulator 上运行应用](#)

使用 Android Device Manager 管理虚拟设备

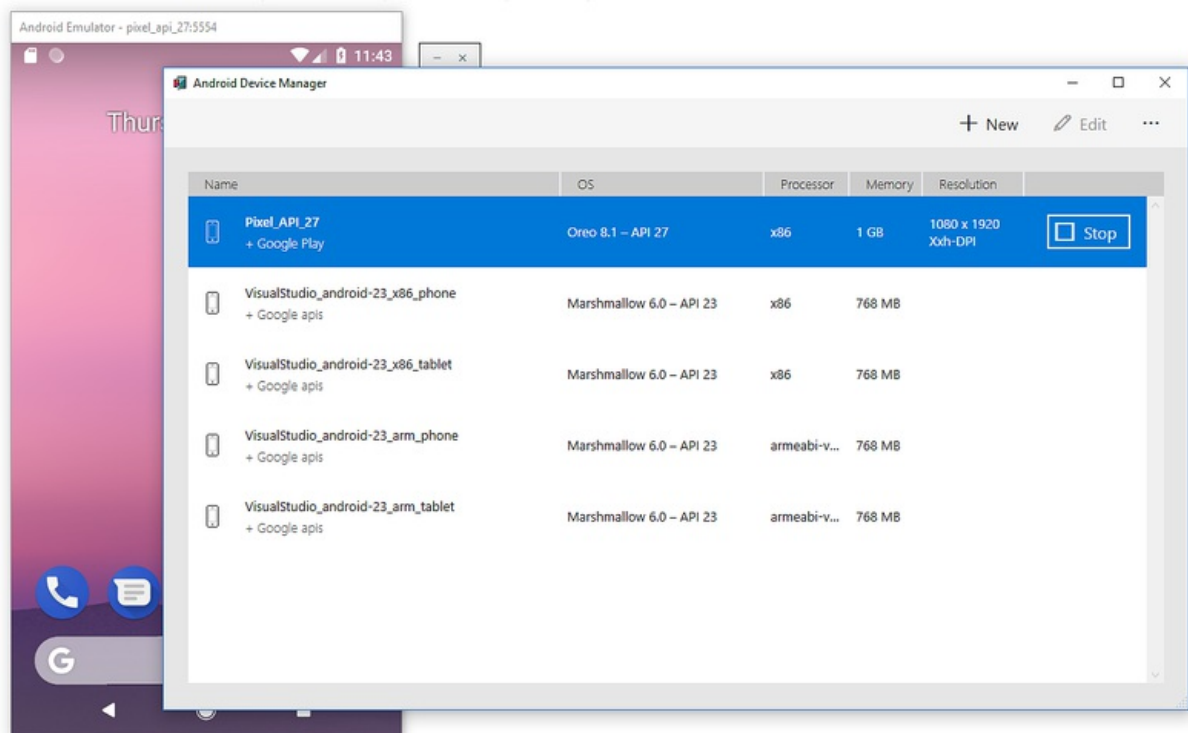
2018/11/2 • [Edit Online](#)

本文介绍如何使用 Android Device Manager 创建和配置对物理 Android 设备进行仿真的 Android 虚拟设备 (AVD)。你可以使用这些虚拟设备运行和测试应用，而不需要依赖物理设备。

验证硬件加速已启用后(如[通过硬件加速提高仿真器性能](#)中所述)，下一步是使用 Android Device Manager(也称为 Xamarin Android Device Manager)创建用于测试和调试应用的虚拟设备。

Windows 上的 Android Device Manager

本文介绍了如何使用 Android Device Manager 创建、复制、自定义和启动 Android 虚拟设备。



使用 Android Device Manager 创建和配置在 [Android Emulator](#) 中运行的 Android 虚拟设备 (AVD)。每台 AVD 是模拟物理 Android 设备的仿真器配置。这样可以在模拟不同物理 Android 设备的多种配置中运行和测试应用。

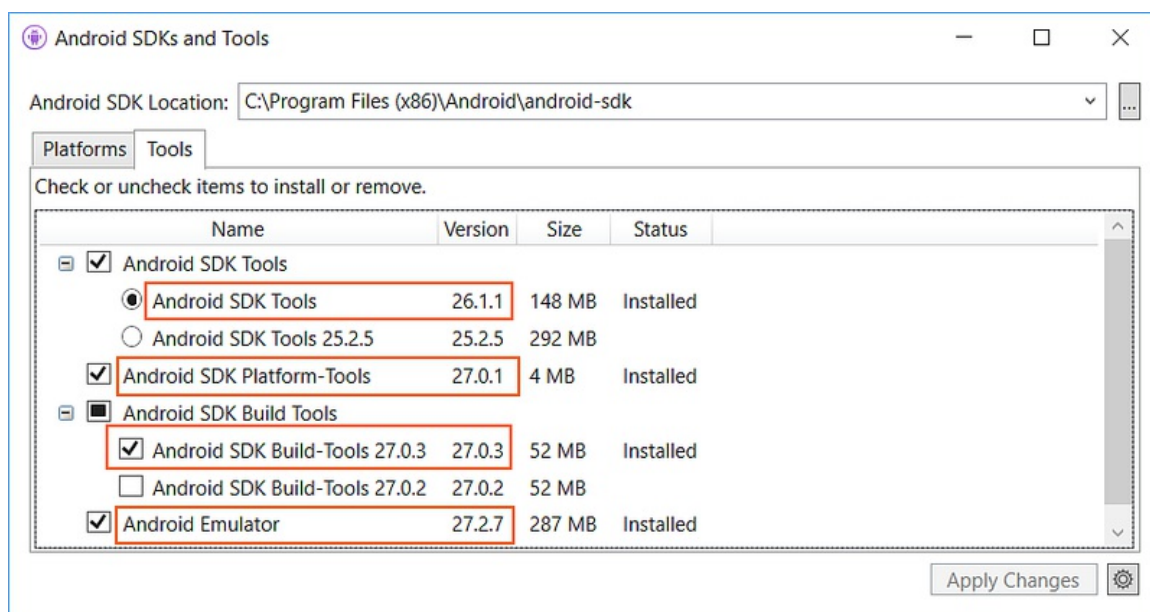
要求

若要使用 Android Device Manager，需要具备以下各项：

- 需要 Visual Studio 2017 版本 15.8 或更高版本。支持 Visual Studio Community、Professional 和 Enterprise 版本。
- Visual Studio Tools for Xamarin 版本 4.9 或更高版本。
- 必须安装 Android SDK(请参阅[设置用于 Xamarin.Android 的 Android SDK](#))。请务必在默认位置安装 Android SDK(如果尚未安装)：C:\Program Files (x86)\Android\android-sdk。
- 必须(通过 [Android SDK 管理器](#))安装以下包：
 - Android SDK Tools 版本 26.1.1 或更高版本

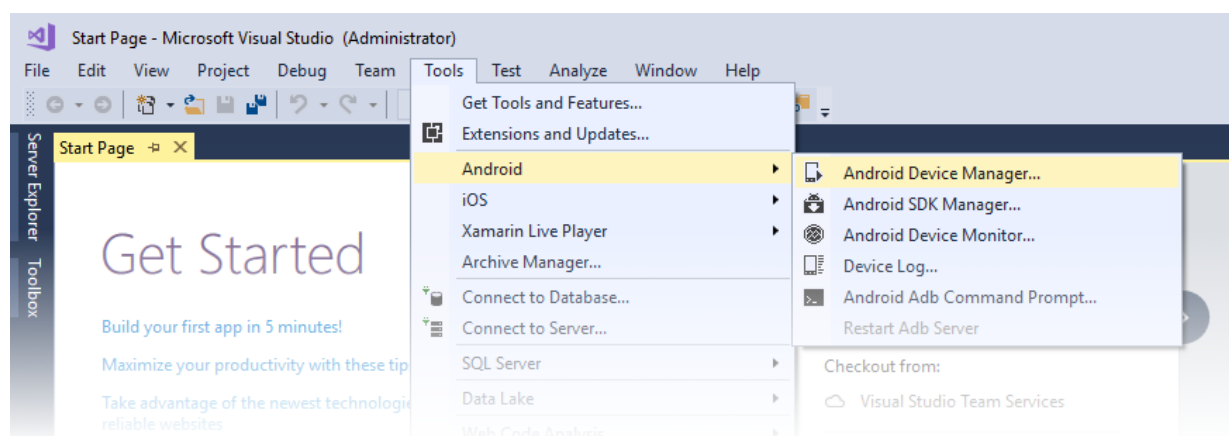
- Android SDK 平台工具 27.0.1 或更高版本
- Android SDK 生成工具 27.0.3 或更高版本
- Android Emulator 27.2.7 或更高版本。

这些包应显示为“已安装”状态，如下面的屏幕截图所示：

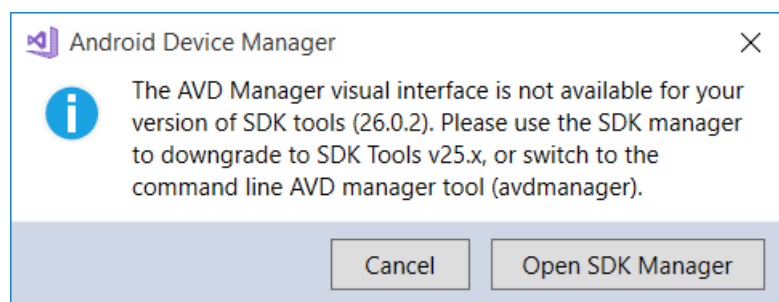


启动设备管理器

通过单击“工具”>“Android”>“Android Device Manager”，从“工具”菜单启动 Android Device Manager：

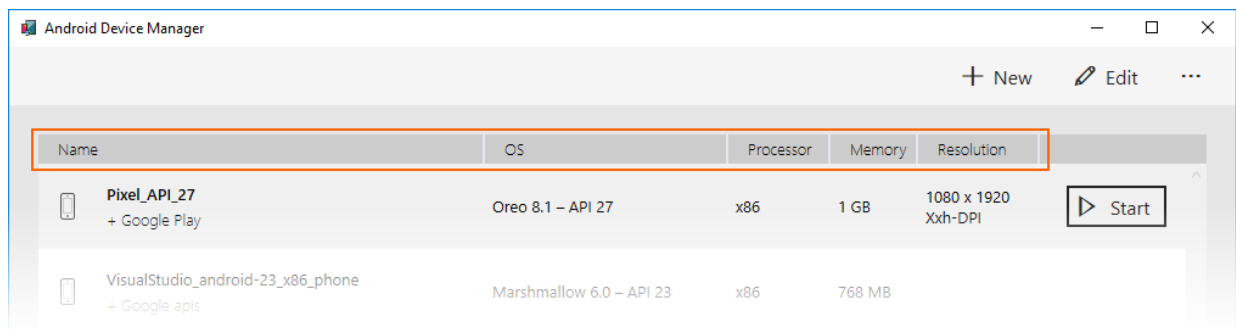


如果在启动时看到以下错误对话框，请参阅[故障排除](#)部分以查找解决方法：

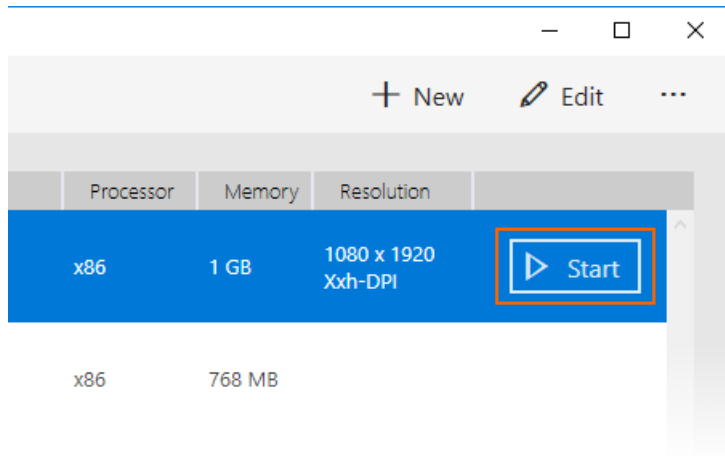


主屏幕

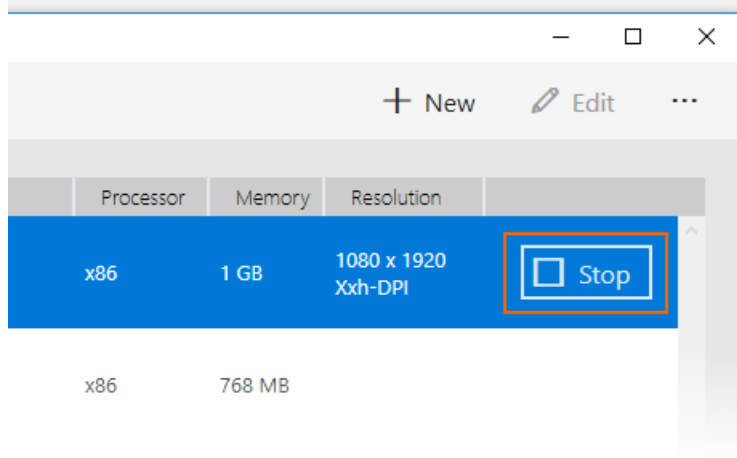
首次启动 Android 设备管理器时，它会展现一个显示所有当前已配置的虚拟设备的屏幕。对于每台虚拟设备，将显示“名称”、“OS”(Android 版)、“处理器”、“内存”大小以及屏幕“分辨率”：



选择列表中的设备时，“启动”按钮出现在右侧。可以单击“启动”按钮以通过此虚拟设备启动仿真器：

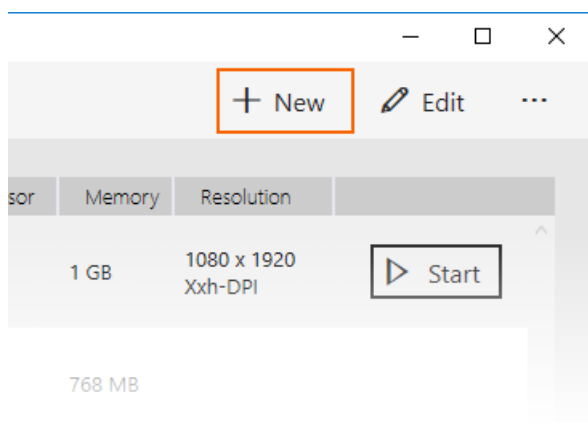


通过所选虚拟设备启动仿真器后，“启动”按钮将更改为可用于终止运行仿真器的“停止”按钮：

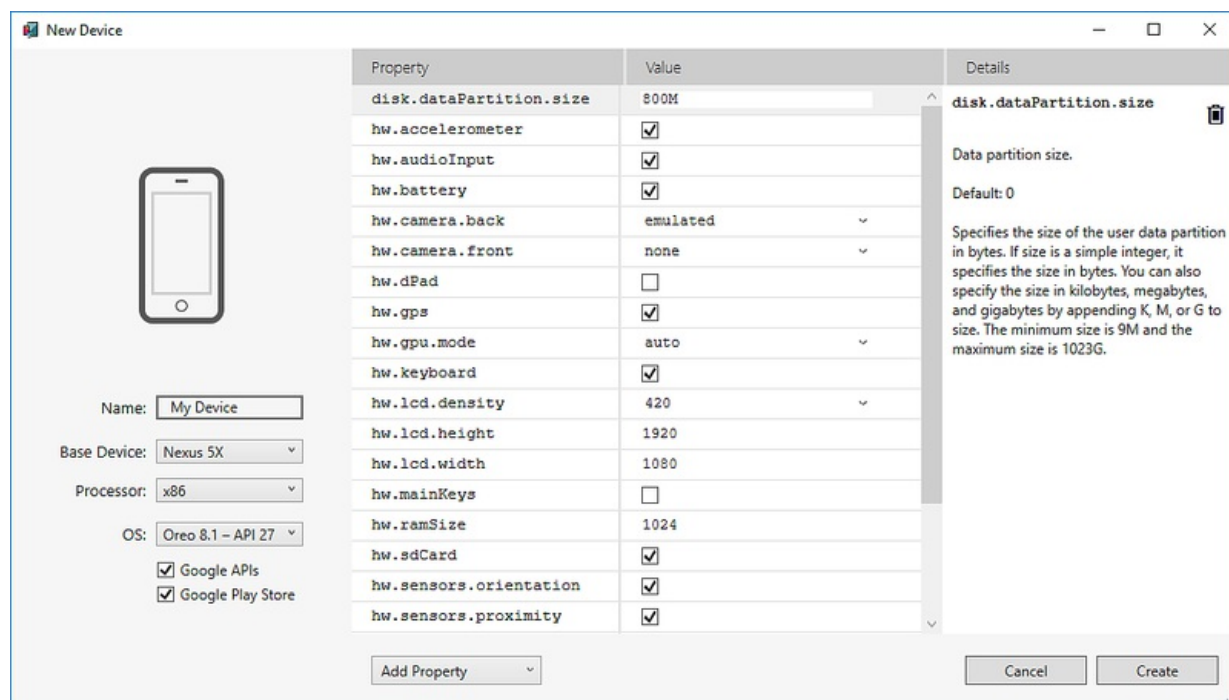


新设备

若要创建新设备，请单击“新建”按钮（位于屏幕的右上方区域）：

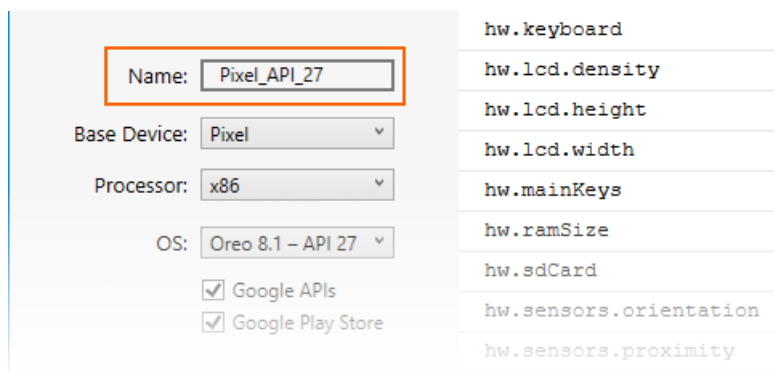


单击“新建”以启动“新设备”屏幕：

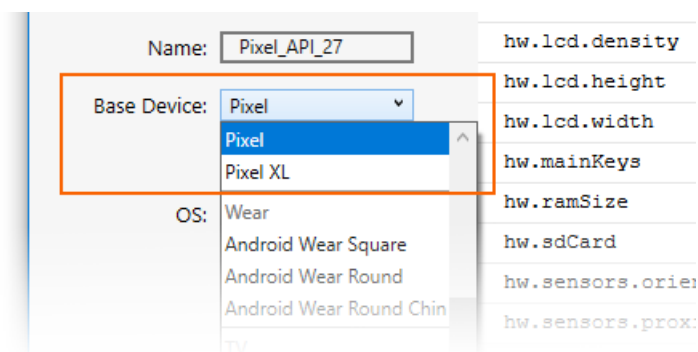


若要在“新设备”中配置新设备，请使用以下步骤：

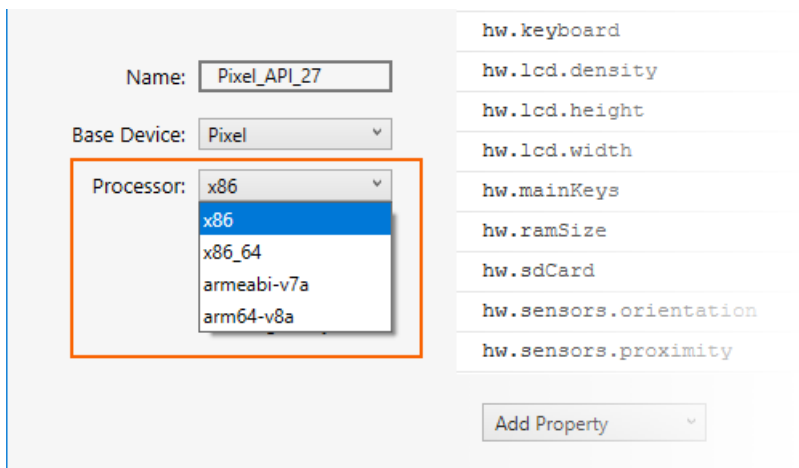
1. 为设备提供新名称。在下面的示例中，新设备名为“Pixel_API_27”：



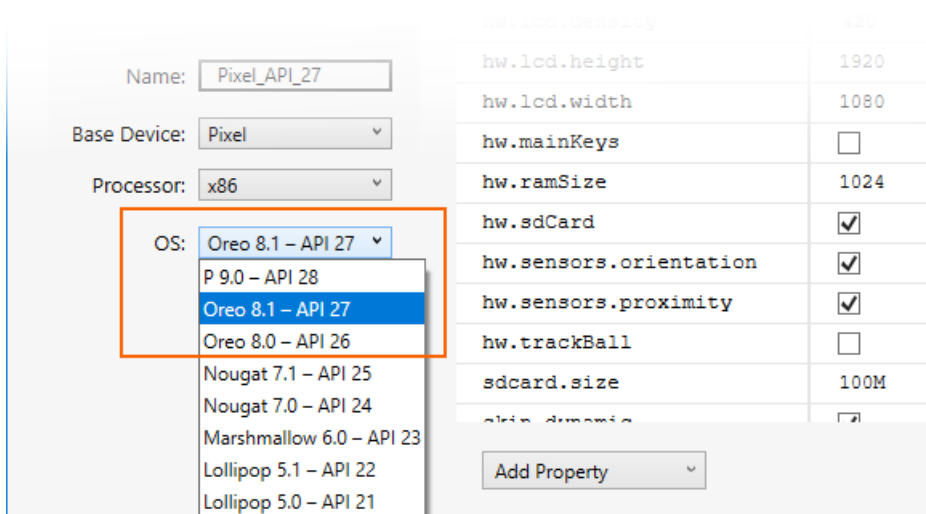
2. 通过单击“基本设备”下拉菜单选择要仿真的物理设备：



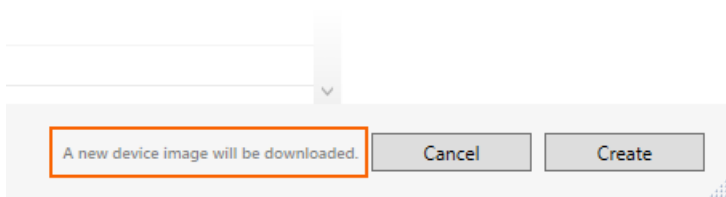
3. 单击“处理器”下拉菜单以选择适用于此虚拟设备的处理器类型。选择“x86”可提供最佳性能，因为它使仿真器能够充分利用[硬件加速](#)。x86_64 选项也将使用硬件加速，但运行速度略慢于 x86 (x86_64 通常用于测试 64 位应用)：



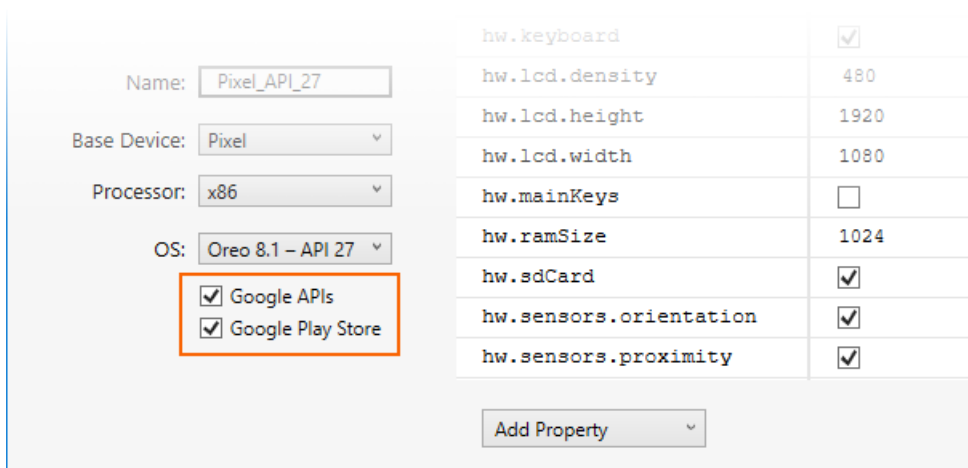
4. 单击“OS”下拉菜单可选择 Android 版本(API 级别)。例如, 选择“Oreo 8.1 - API 27”以创建 API 级别为 27 的虚拟设备:



如果选择尚未安装的 Android API 级别, Device Manager 将在屏幕底部显示“将下载新设备”消息 – 它将在创建新的虚拟设备时, 下载并安装必要的文件:

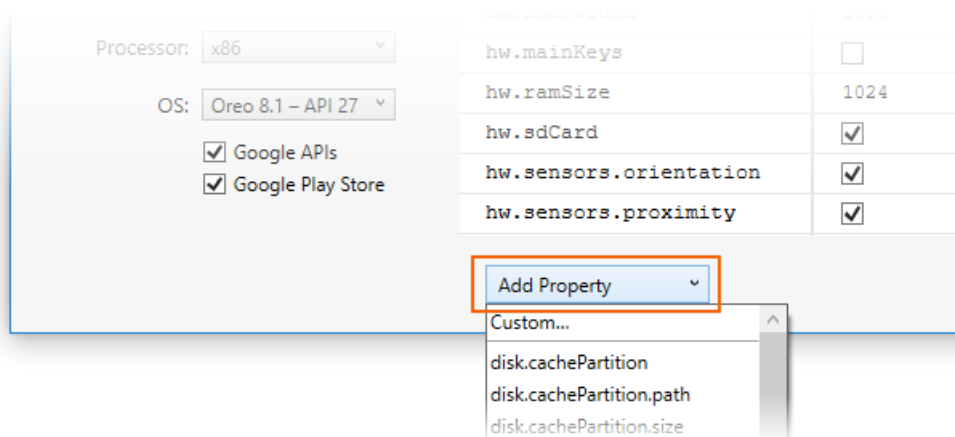


5. 如果希望虚拟设备中包含 Google Play Services API, 请启用“Google API”选项。若要包含 Google Play 商店应用, 请启用“Google Play 商店”选项:



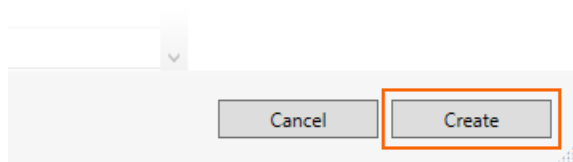
请注意, Google Play 商店图片仅适用于某些基本设备类型, 例如 Pixel、Pixel 2、Nexus 5 和 Nexus 5X。

6. 编辑需要修改的任何属性。若要对属性进行更改, 请参阅[编辑 Android 虚拟设备属性](#)。
7. 添加需要显式设置的任何其他属性。尽管“新设备”屏幕只列出了最常修改的属性, 但你可以单击“添加属性”下拉菜单(位于底部)以添加其他的属性:

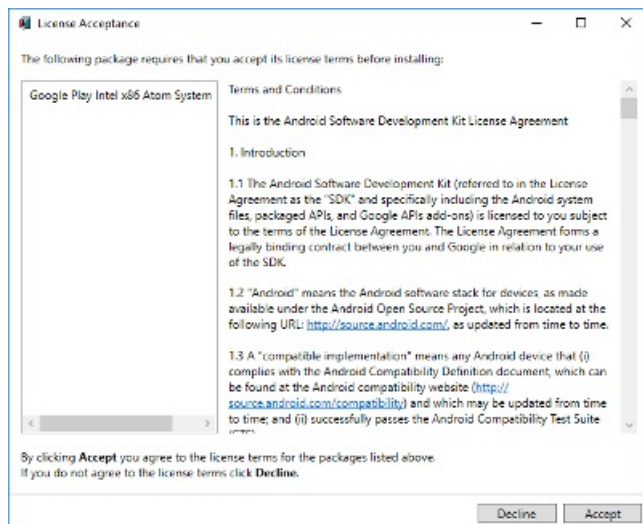


还可以在属性列表顶部选择“自定义...”, 定义自定义属性。

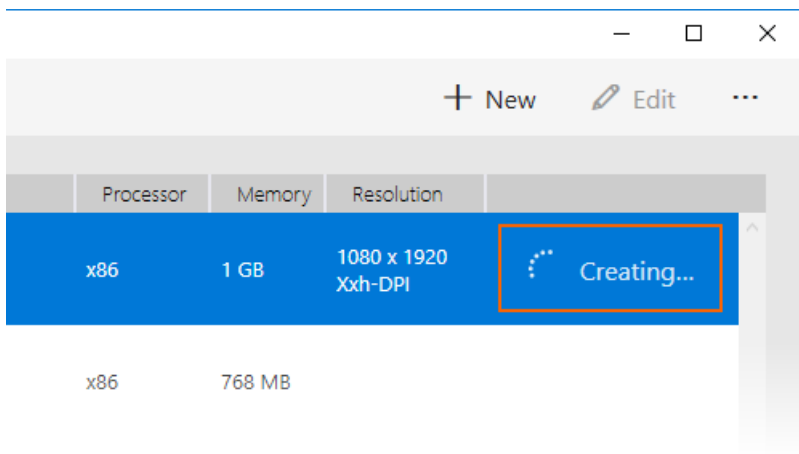
8. 单击“创建”按钮(位于右下角)以创建新设备:



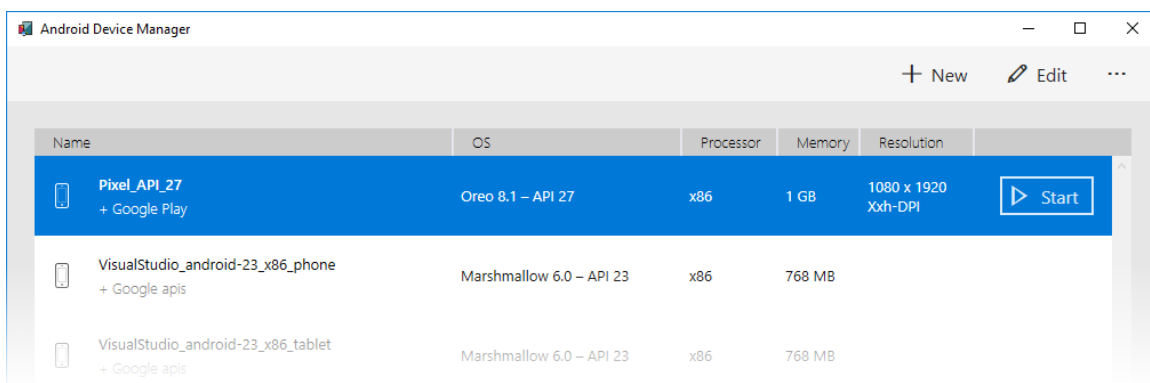
9. 可能会出现“许可证接受”屏幕。如果同意许可条款, 请单击“接受”:



10. 在设备创建期间, Android Device Manager 将新设备添加到已安装虚拟设备列表中, 同时显示“正在创建”进度指示器:

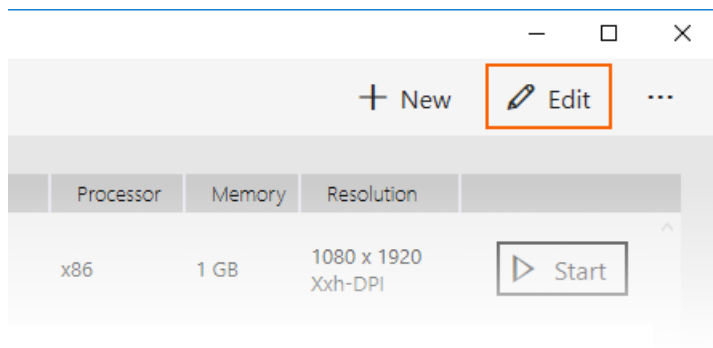


11. 创建过程完成后，新设备会显示在已安装虚拟设备的列表中，并且会显示可以启动的“启动”按钮：

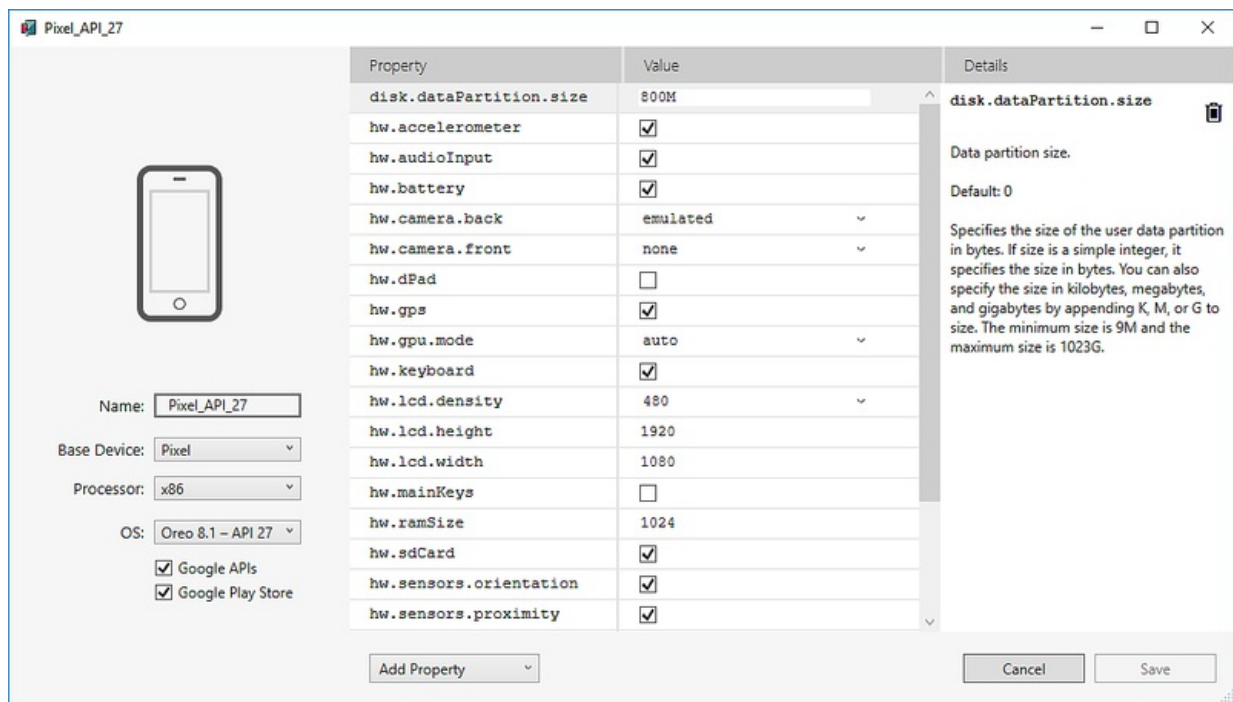


编辑设备

若要编辑现有的虚拟设备，请选择设备并单击“编辑”按钮(位于屏幕的右上方)：

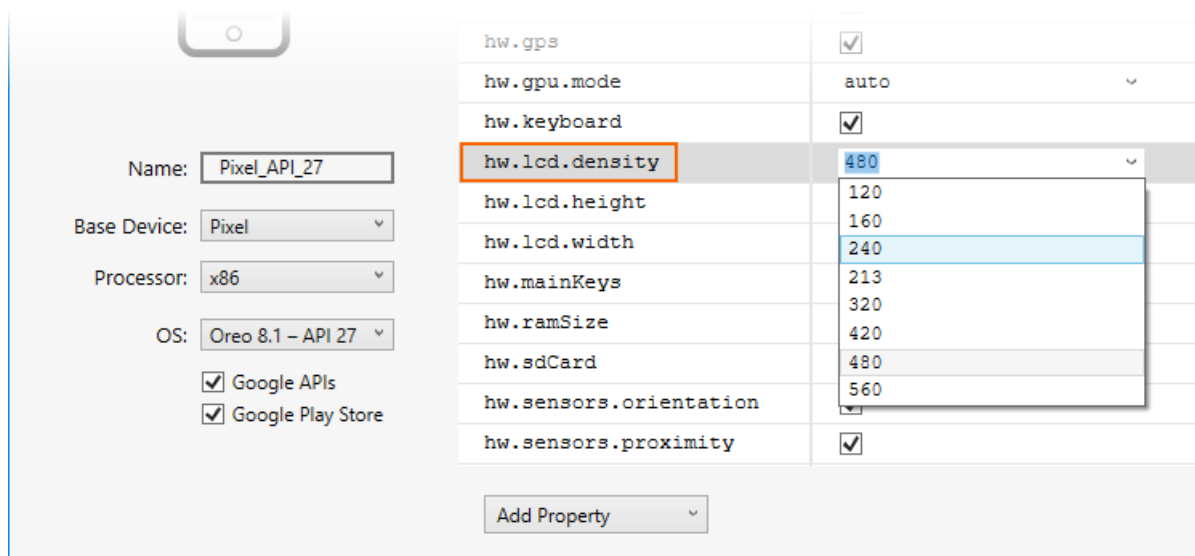


单击“编辑”为所选的虚拟设备启动“设备编辑器”：



“设备编辑器”屏幕在“属性”列下列出了虚拟设备的属性，在“值”列中为每个属性列出相应的值。当选择某个属性时，有关该属性的详细描述会显示在右侧。

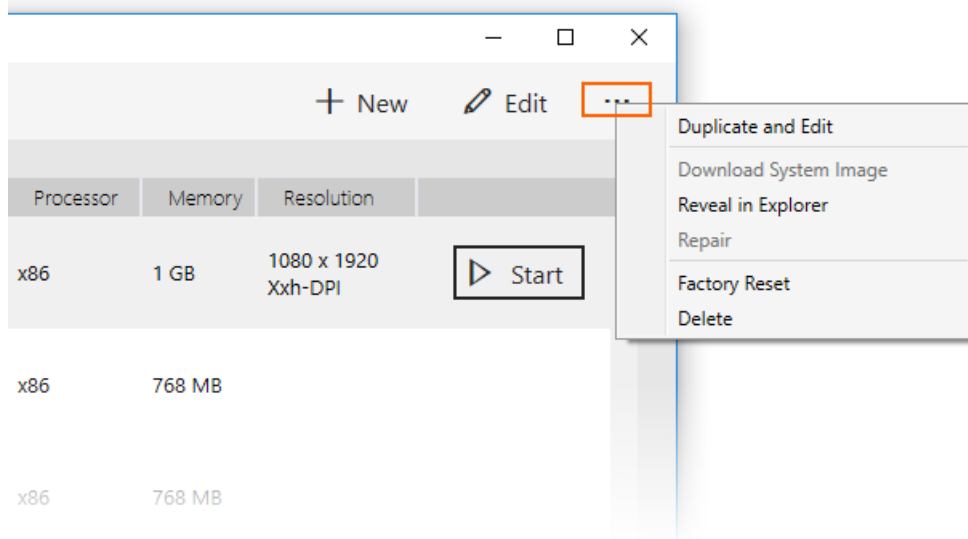
若要更改属性，请在“值”列中编辑其值。例如，在下面的屏幕截图中，`hw.lcd.density` 属性正从“480”更改为“240”：



在进行必要的配置更改后，单击“保存”按钮。有关更改虚拟设备属性的详细信息，请参阅[编辑 Android 虚拟设备属性](#)。

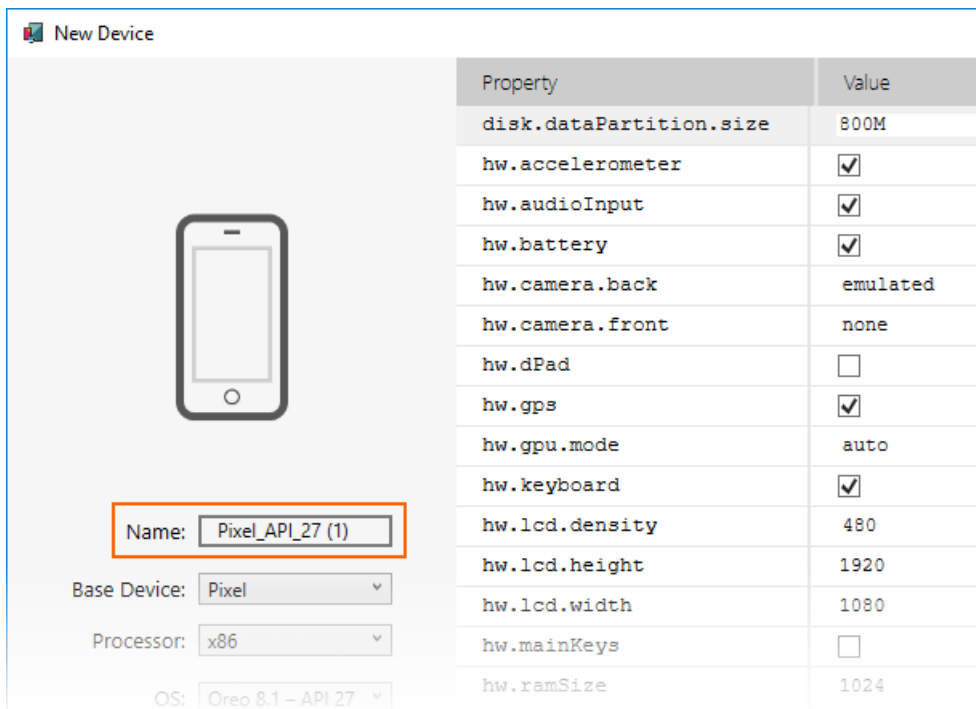
附加选项

右上方的“其他选项”(…) 下拉菜单中提供适用于设备的其他选项：

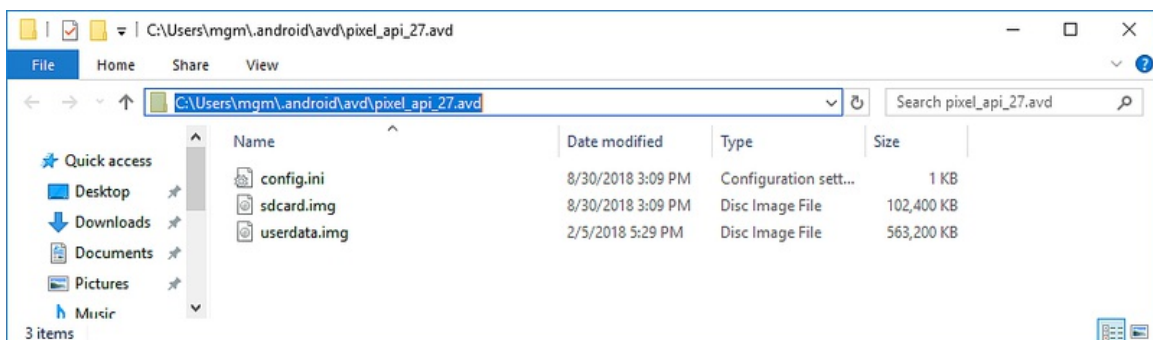


附加选项菜单中包含以下项：

- **复制和编辑** – 复制当前所选的设备，并会在“新建设备”屏幕中使用不同的唯一名称打开。例如，选择“Pixel_API_27”并单击“复制和编辑”可将计数器追加到名称中：

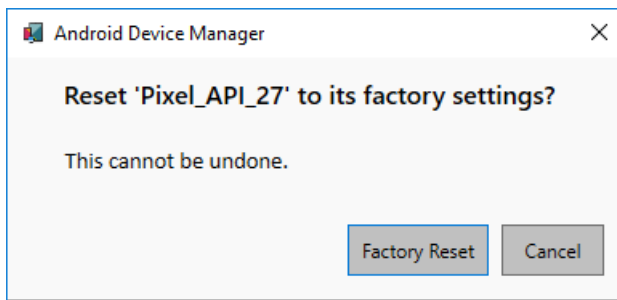


- **在资源管理器中展现** – 在包含用于虚拟设备的文件的文件夹中打开“Windows 资源管理器”窗口。例如，选择“Pixel_API_27”并单击“在资源管理器中展现”以打开一个窗口，如下所示：

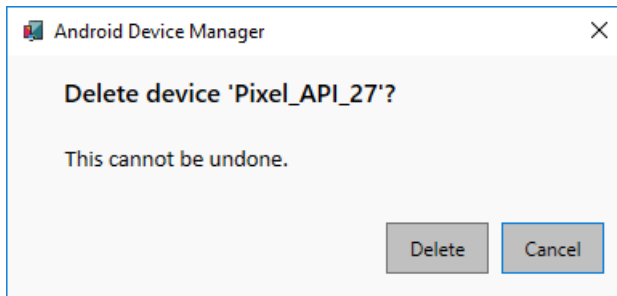


- **恢复出厂设置** – 将所选设备重置为其默认设置，擦除用户在此设备运行时对其内部状态进行的任何更改（这也会擦除当前**快速启动**快照（如果有））。此更改不会影响在创建和编辑期间对虚拟设备做出的修

改。将出现提醒此重置无法被撤消的一个对话框。单击“恢复出厂设置”，确认重置：

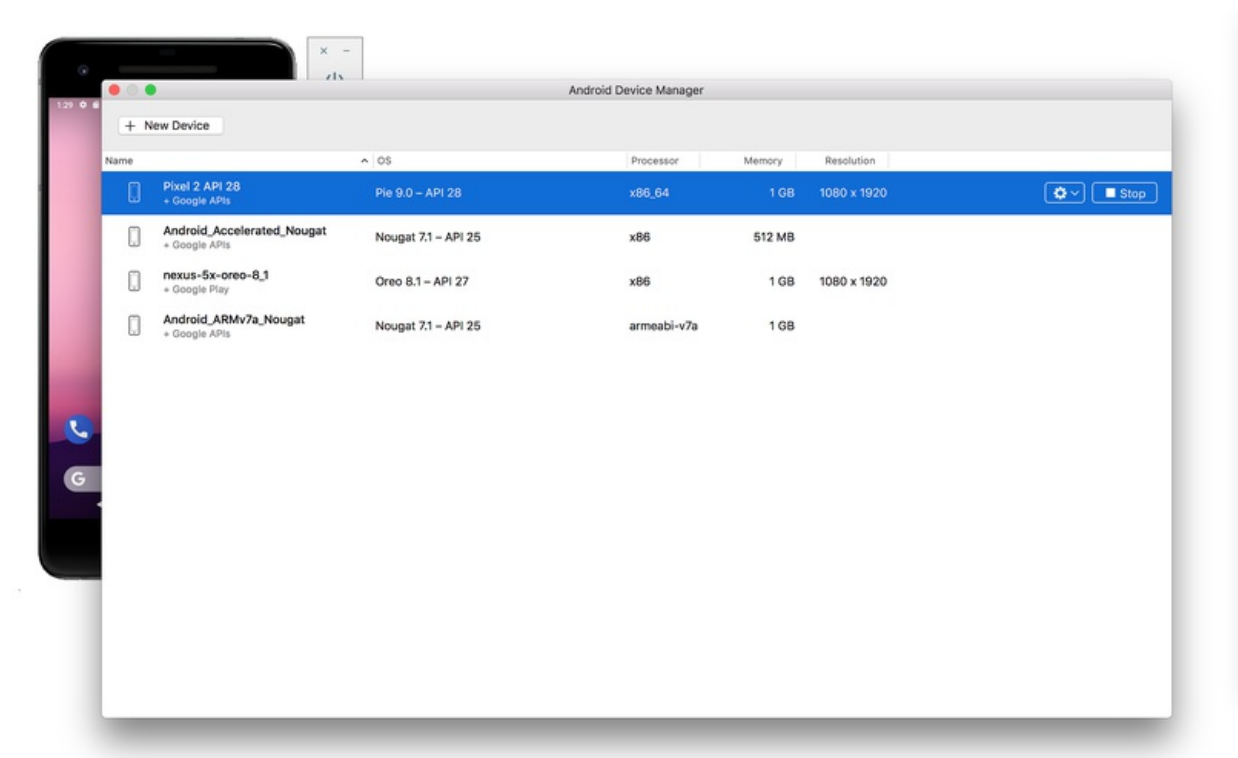


- **删除** – 永久删除所选的虚拟设备。将出现提醒删除设备无法被撤消的一个对话框。如果确定要删除设备，请单击“删除”。



macOS 上的 Android Device Manager

本文介绍了如何使用 Android Device Manager 创建、复制、自定义和启动 Android 虚拟设备。



NOTE

本指南仅适用于 Visual Studio for Mac。Xamarin Studio 与 Android Device Manager 不兼容。

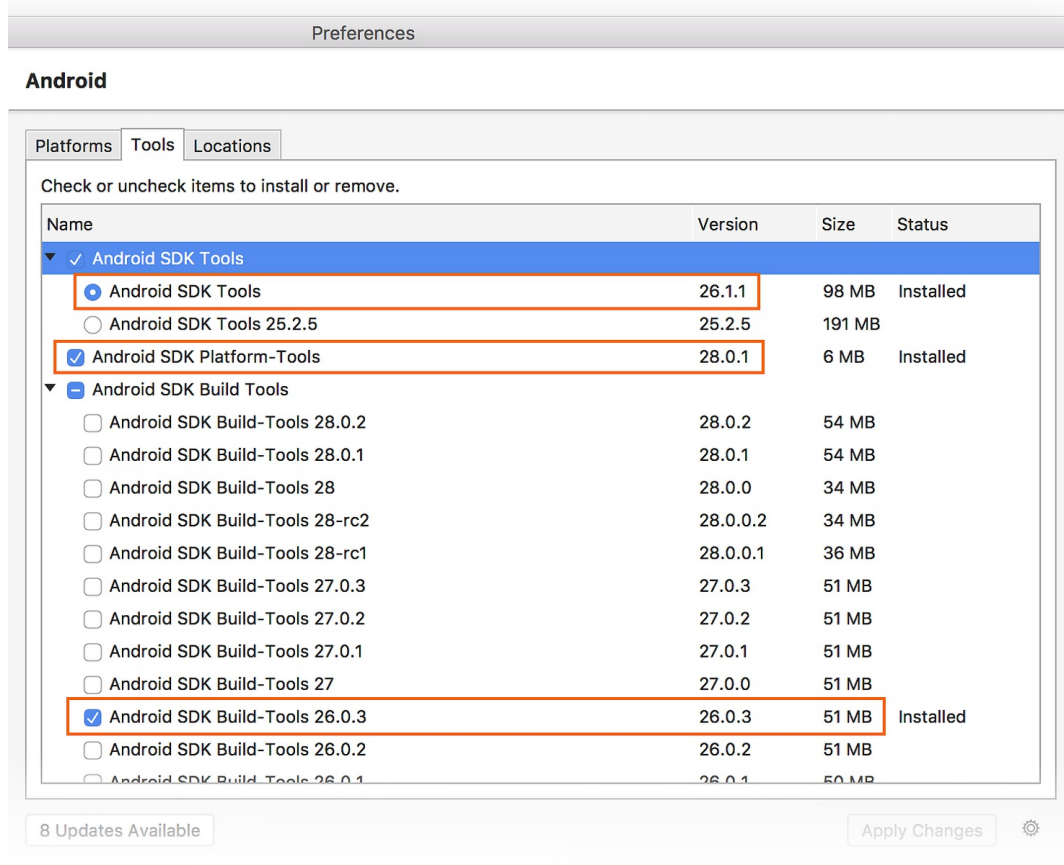
使用 Android Device Manager 创建和配置在 [Android Emulator](#) 中运行的 Android 虚拟设备 (AVD)。每台 AVD 是模拟物理 Android 设备的仿真器配置。这样可以在模拟不同物理 Android 设备的多种配置中运行和测试应用。

要求

若使用 Android Device Manager，需要具备以下各项：

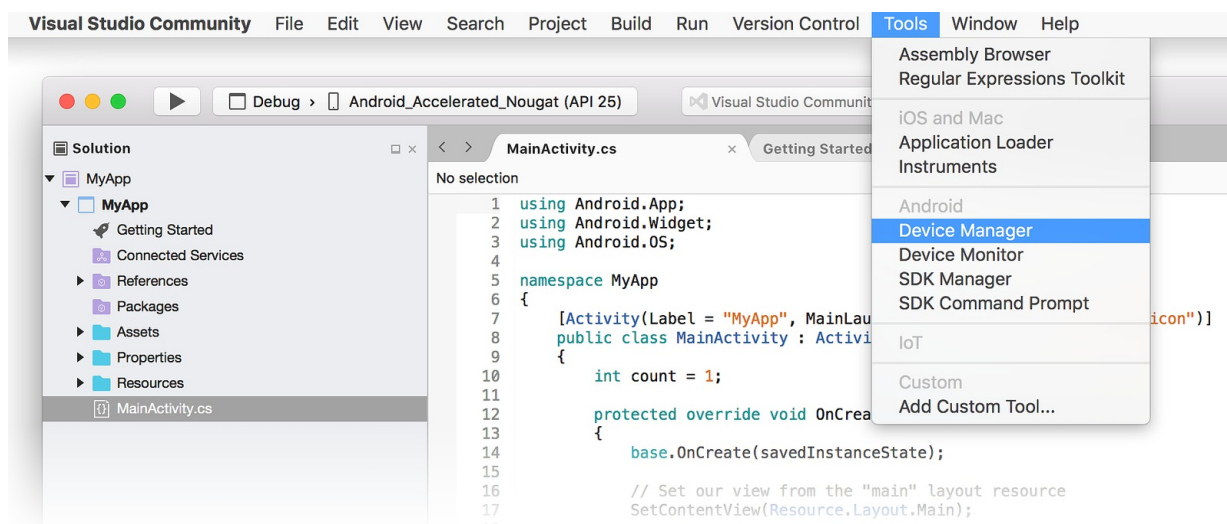
- Visual Studio for Mac 7.6 或更高版本。
- 必须安装 Android SDK (请参阅[设置用于 Xamarin.Android 的 Android SDK](#))。
- 必须(通过 [Android SDK 管理器](#))安装以下包：
 - SDK 工具 26.1.1 版或更高版本
 - Android SDK 平台工具 28.0.1 或更高版本
 - Android SDK 生成工具 26.0.3 或更高版本

这些包应显示为“已安装”状态，如下面的屏幕截图所示：

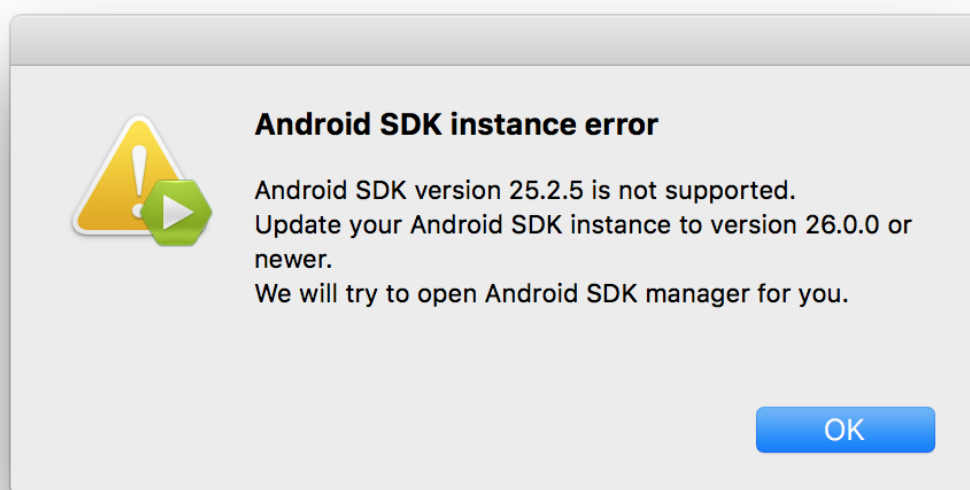


启动设备管理器

通过单击“工具”>“Device Manager”启动 Android Device Manager：

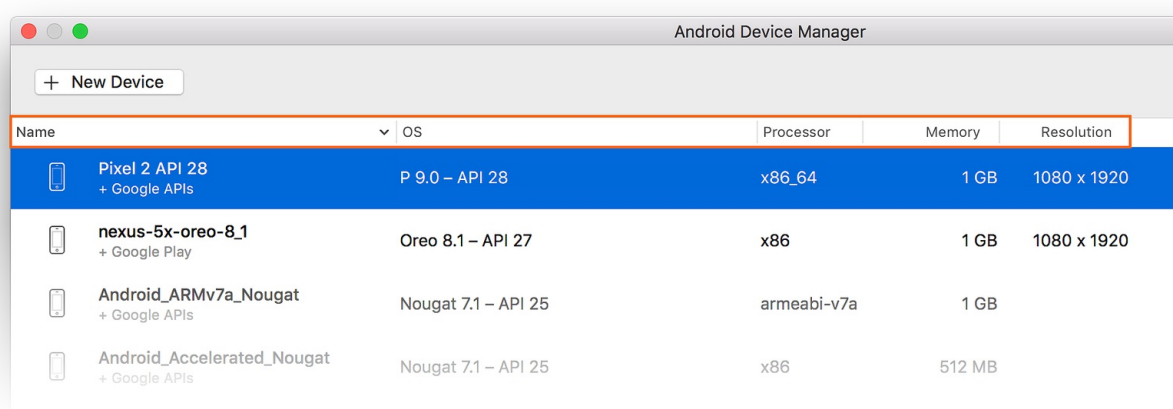


如果在启动时看到以下错误对话框，请参阅[故障排除](#)部分以查找解决方法：

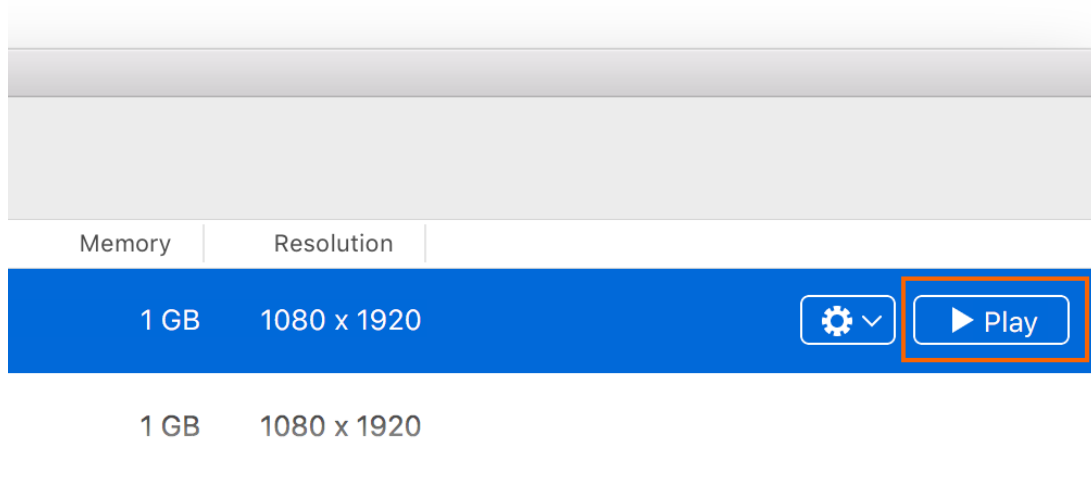


主屏幕

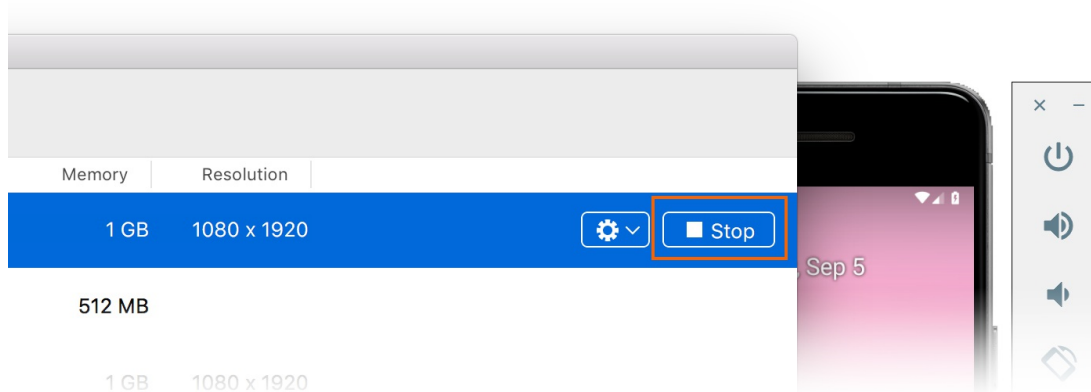
首次启动 Android 设备管理器时，它会展现一个显示所有当前已配置的虚拟设备的屏幕。对于每台虚拟设备，将显示“名称”、“OS”(Android 版)、“处理器”、“内存”大小以及屏幕“分辨率”：



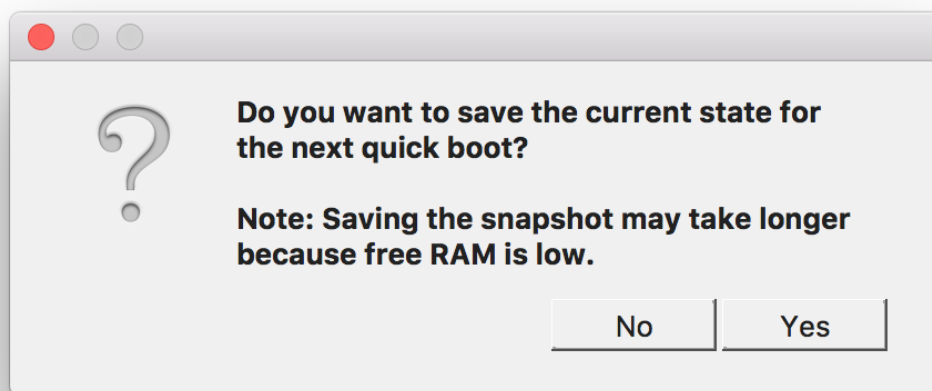
选择列表中的设备时，“播放”按钮出现在右侧。可以单击“播放”按钮以通过此虚拟设备启动仿真器：



通过所选虚拟设备启动仿真器后，“播放”按钮将更改为可用于终止运行仿真器的“停止”按钮：



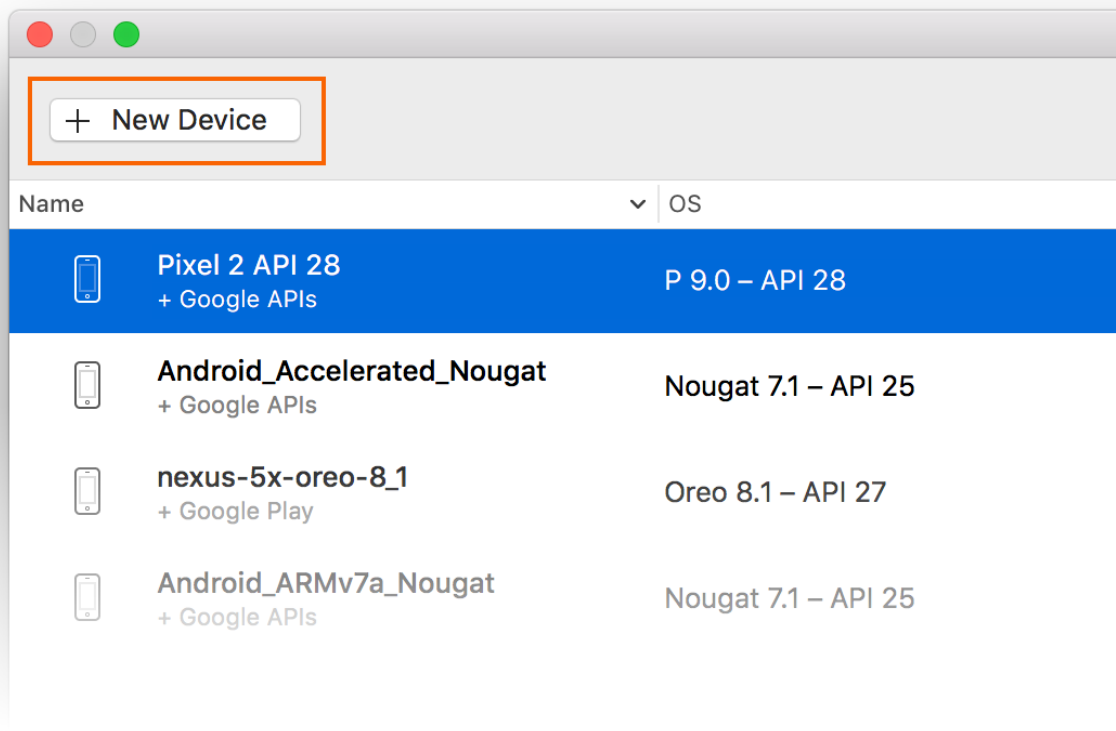
停止仿真器时，可能会收到一个提示，询问是否要保存当前状态以便下一次快速启动：



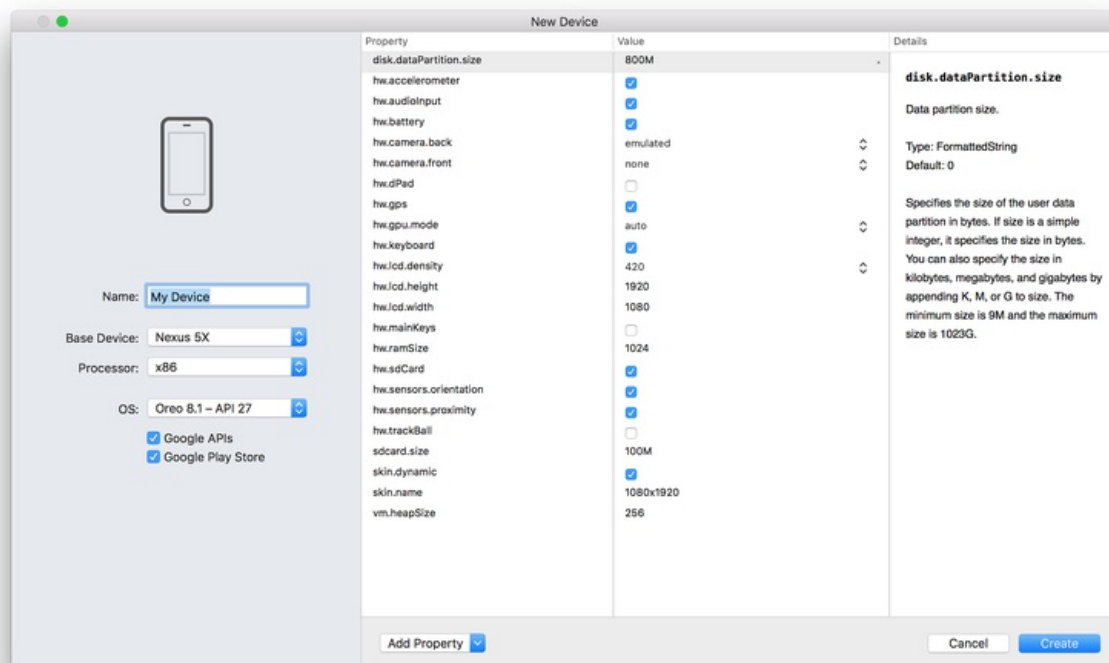
保存当前状态会使再次启动虚拟设备时仿真器的启动速度更快。有关“快速启动”的详细信息，请参阅[快速启动](#)。

新设备

若要创建新设备，请单击“新建设备”按钮（位于屏幕的左上方区域）：

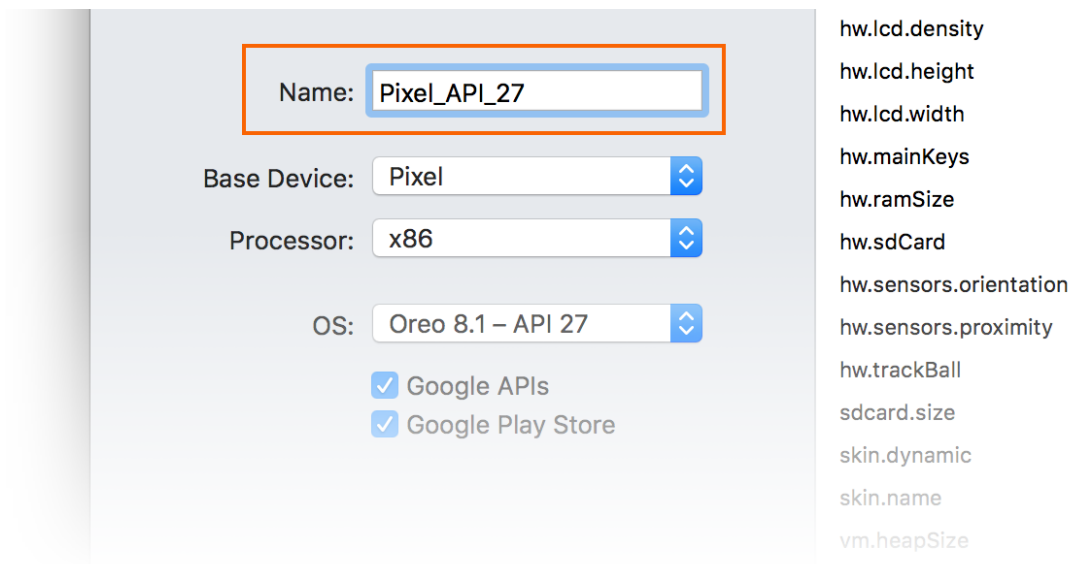


单击“新建设备”以启动“新建设备”屏幕：



使用以下步骤在“新建设备”屏幕中配置新设备：

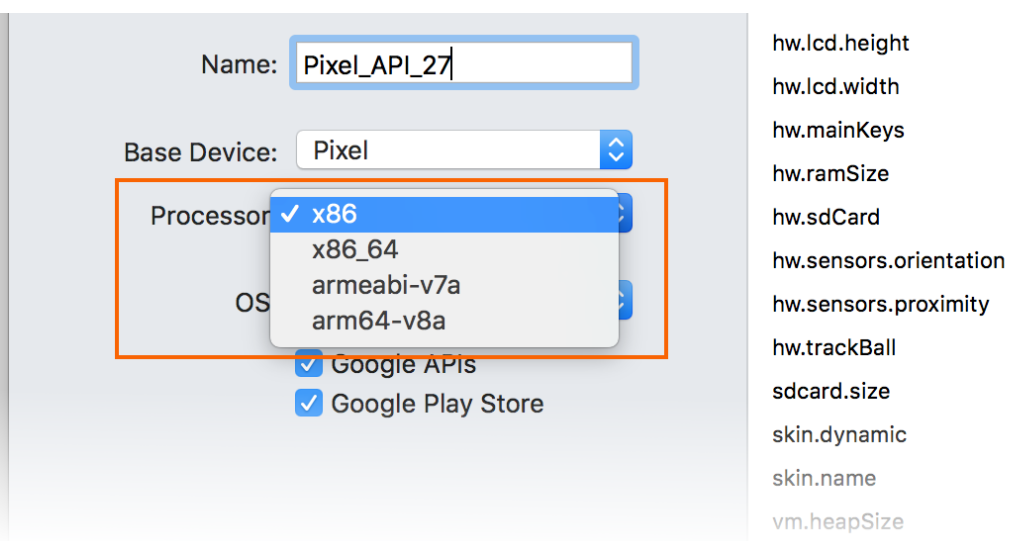
1. 为设备提供新名称。在下面的示例中，新设备名为“Pixel_API_27”：



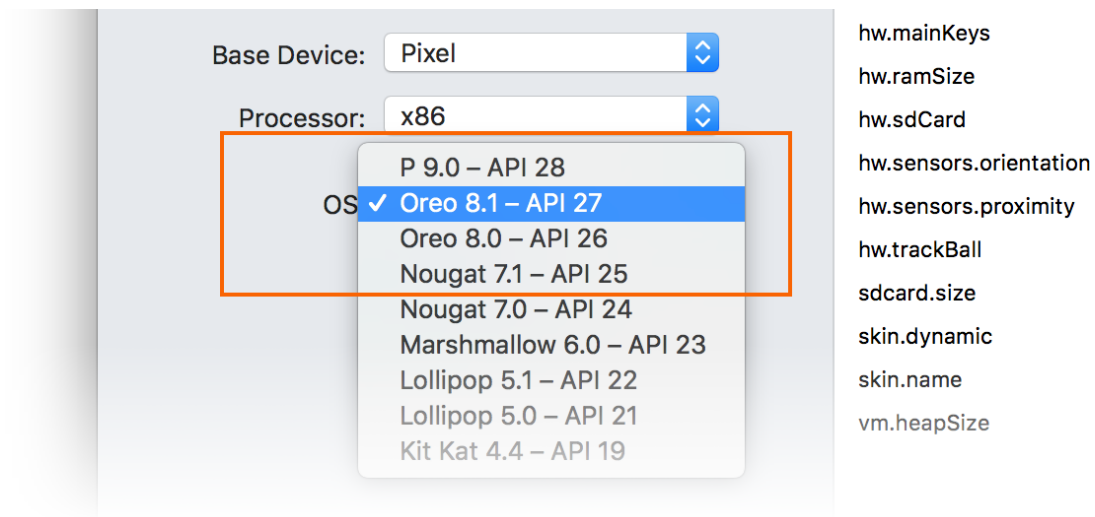
2. 通过单击“基本设备”下拉菜单选择要仿真的物理设备：



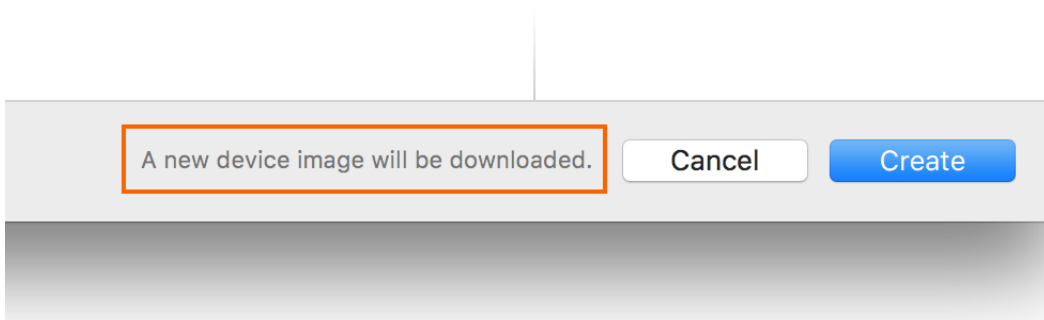
3. 单击“处理器”下拉菜单以选择适用于此虚拟设备的处理器类型。选择“x86”可提供最佳性能，因为它使仿真器能够充分利用[硬件加速](#)。x86_64 选项也将使用硬件加速，但运行速度略慢于 x86 (x86_64 通常用于测试 64 位应用)：



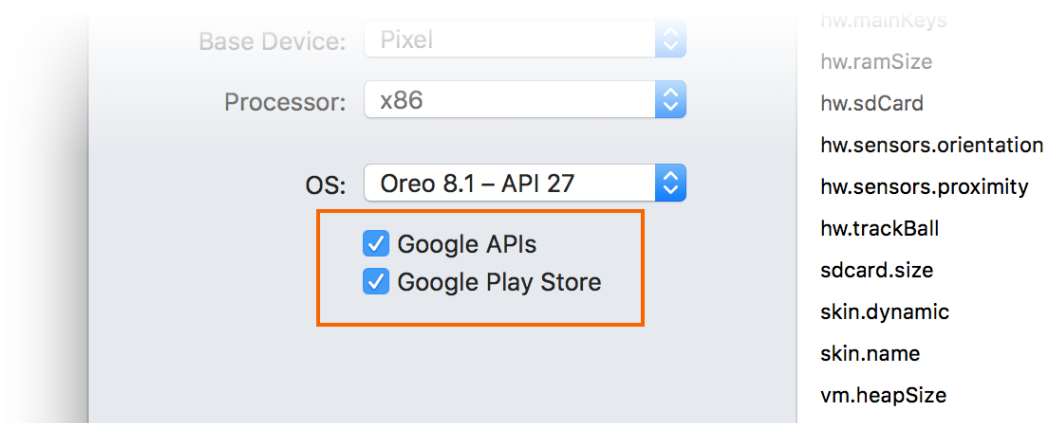
4. 单击“OS”下拉菜单可选择 Android 版本(API 级别)。例如，选择“Oreo 8.1 - API 27”以创建 API 级别为 27 的虚拟设备：



如果选择尚未安装的 Android API 级别，Device Manager 将在屏幕底部显示“将下载新设备”消息 – 它将在创建新的虚拟设备时，下载并安装必要的文件：

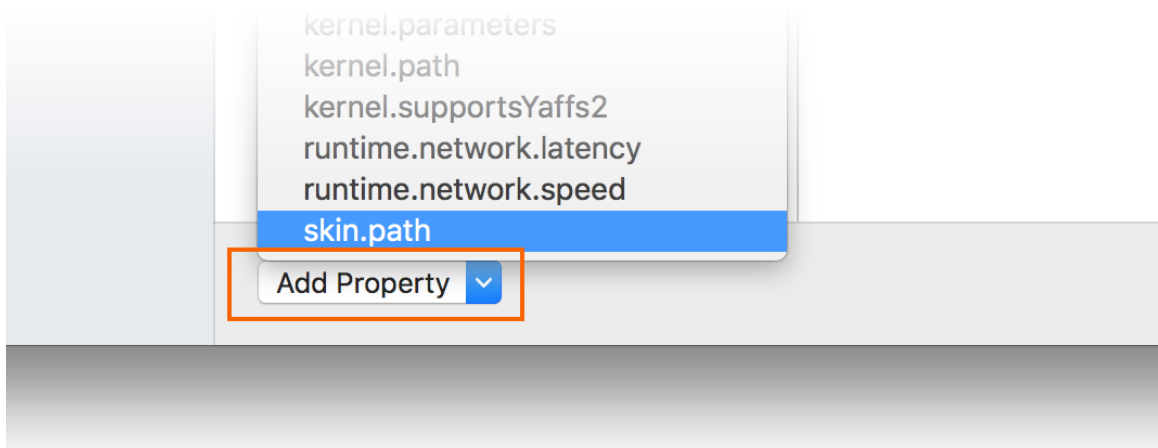


5. 如果希望虚拟设备中包含 Google Play Services API，请启用“Google API”选项。若要包含 Google Play 商店应用，请启用“Google Play 商店”选项：



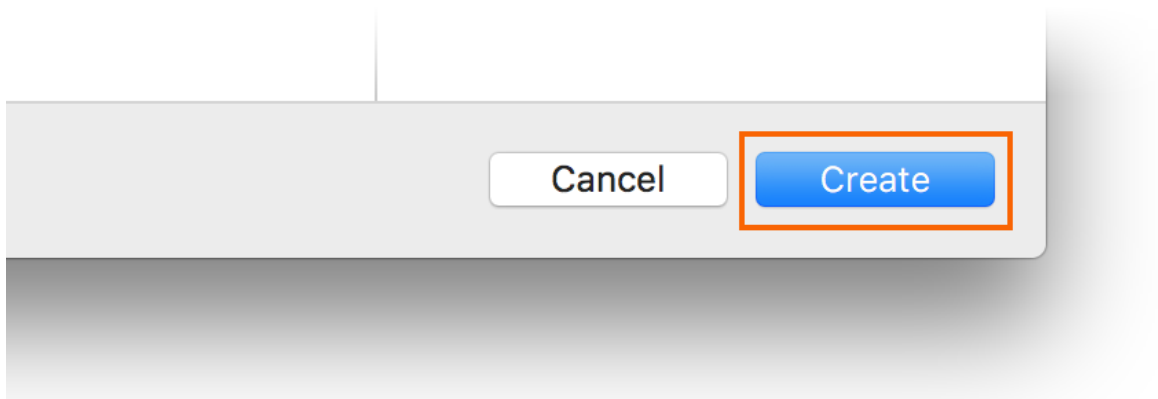
请注意，Google Play 商店图片仅适用于某些基本设备类型，例如 Pixel、Pixel 2、Nexus 5 和 Nexus 5X。

6. 编辑需要修改的任何属性。若要对属性进行更改，请参阅[编辑 Android 虚拟设备属性](#)。
7. 添加需要显式设置的任何其他属性。尽管“新建设备”屏幕只列出了最常修改的属性，但你可以单击“添加属性”下拉菜单（位于底部）以添加其他的属性：

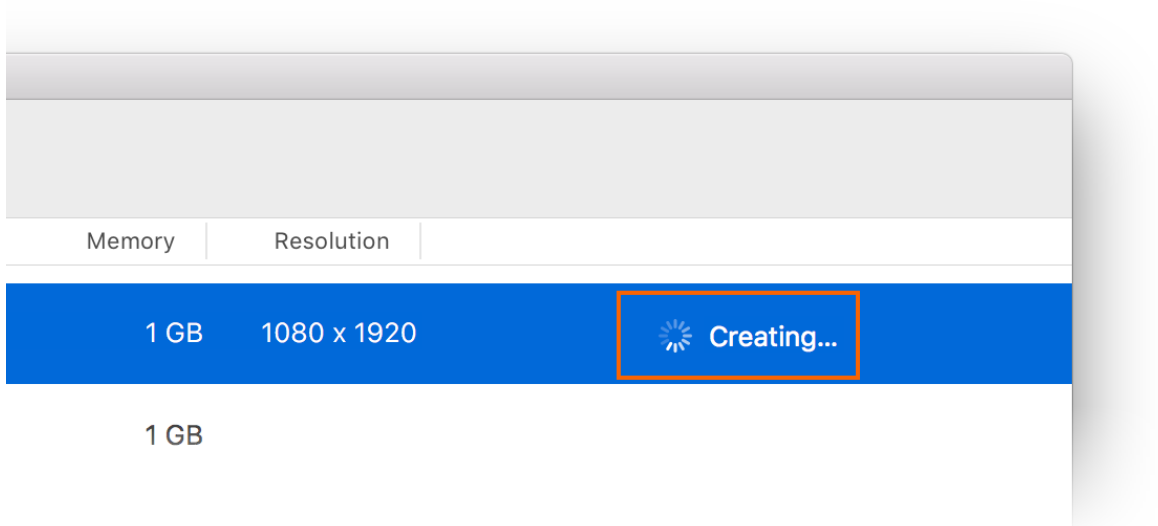


还可以在该属性列表顶部单击“自定义...”，定义自定义属性。

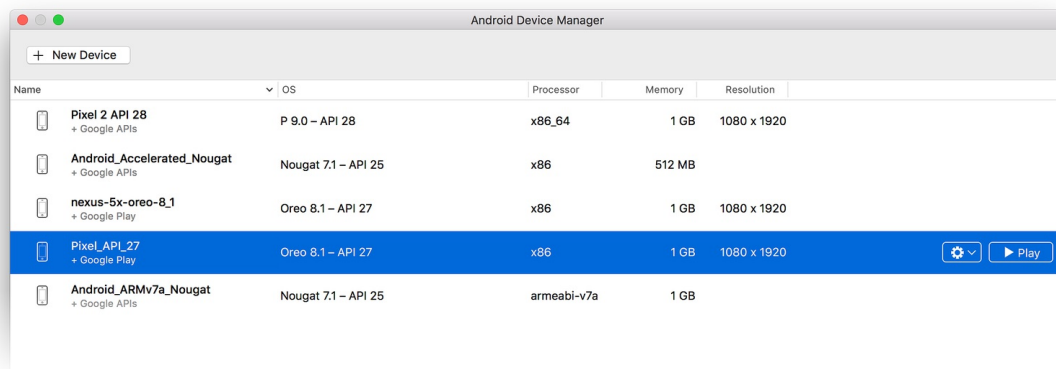
8. 单击“创建”按钮(位于右下角)以创建新设备:



9. 在设备创建期间, Android Device Manager 将新设备添加到已安装虚拟设备列表中, 同时显示“正在创建”进度指示器:

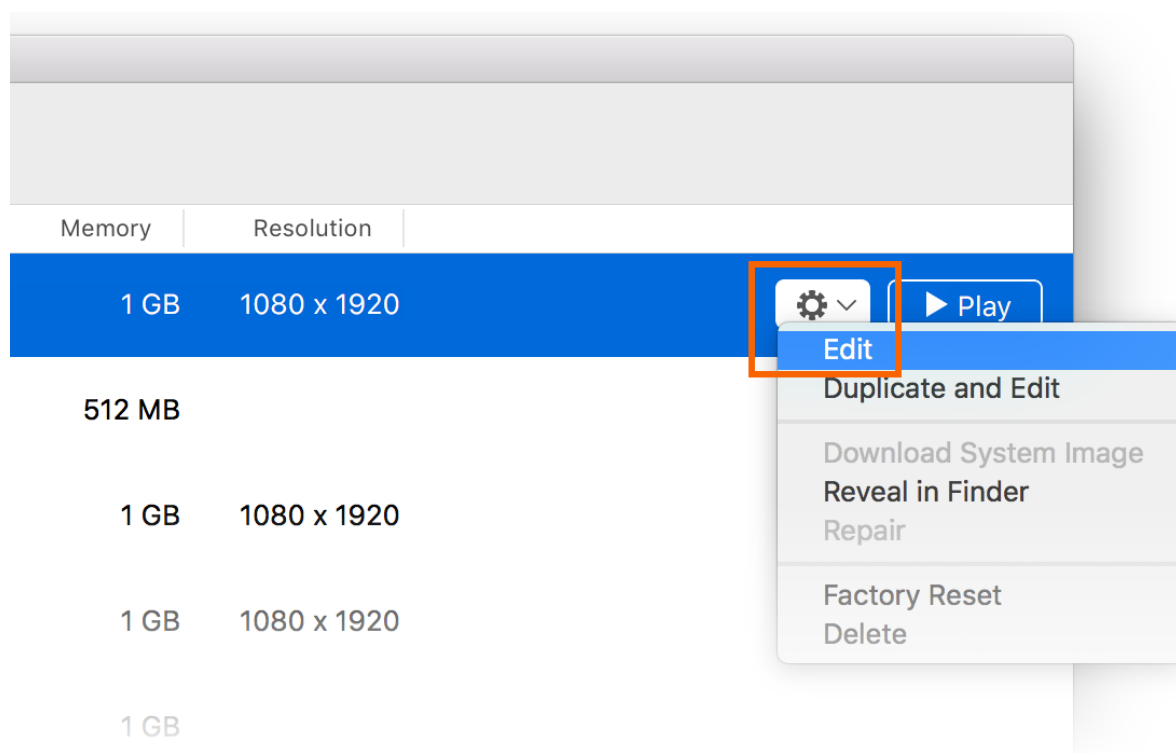


10. 创建过程完成后, 新设备会显示在已安装虚拟设备的列表中, 并且会显示可以启动的“启动”按钮:

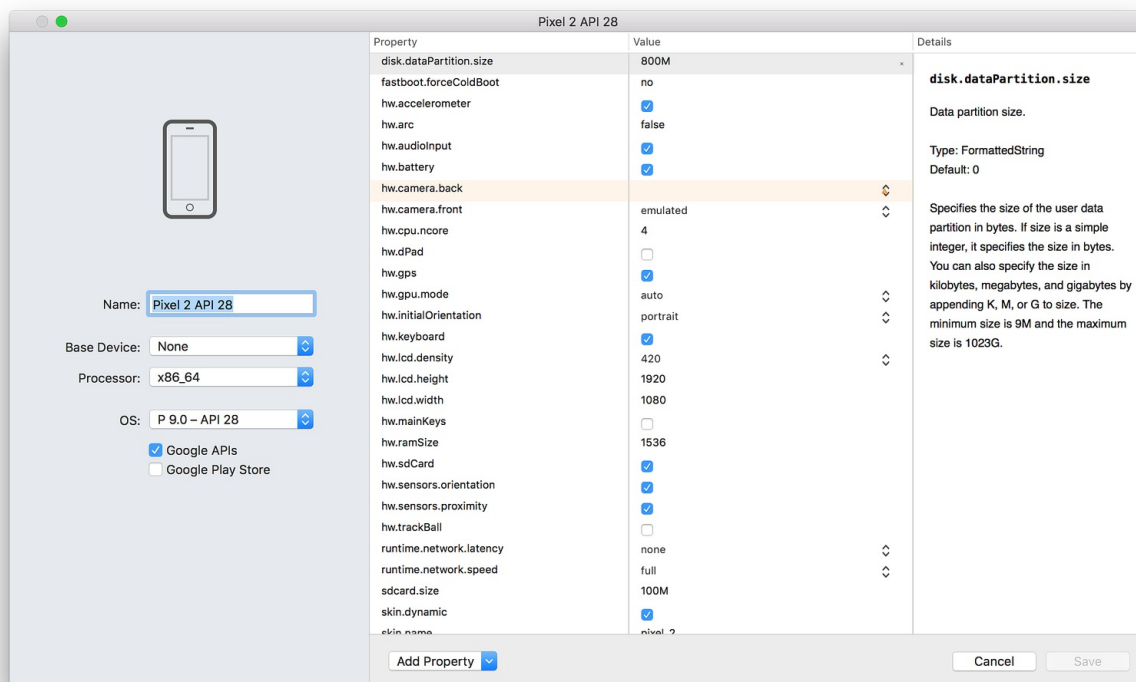


编辑设备

若要编辑现有的虚拟设备，请选择“其他选项”下拉菜单(齿轮图标)，然后选择“编辑”：

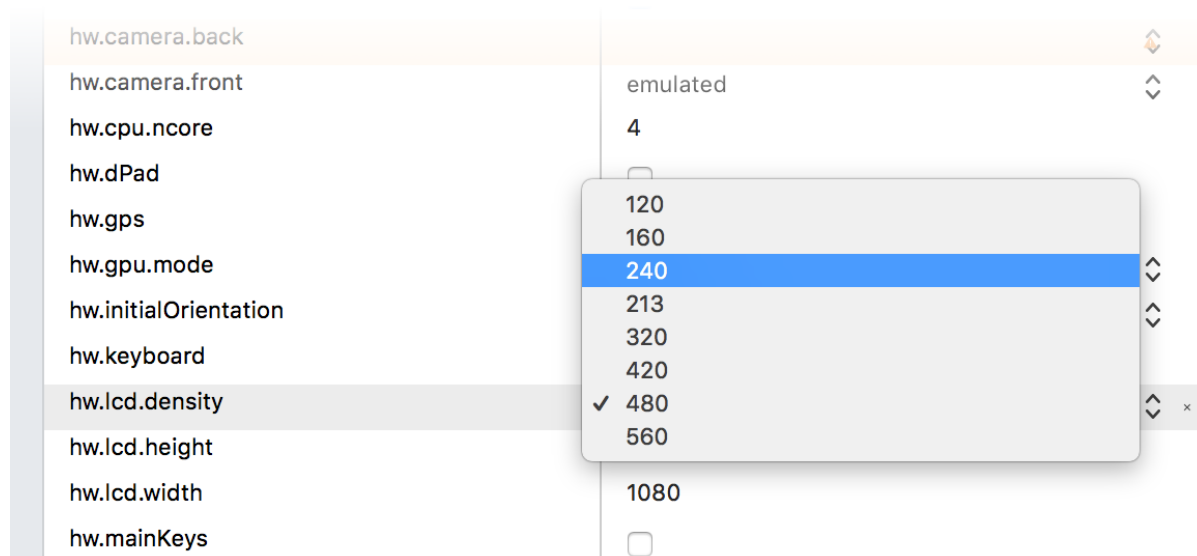


单击“编辑”为所选的虚拟设备启动“设备编辑器”：



“设备编辑器”屏幕在“属性”列下列出了虚拟设备的属性，在“值”列中为每个属性列出相应的值。当选择某个属性时，有关该属性的详细描述会显示在右侧。

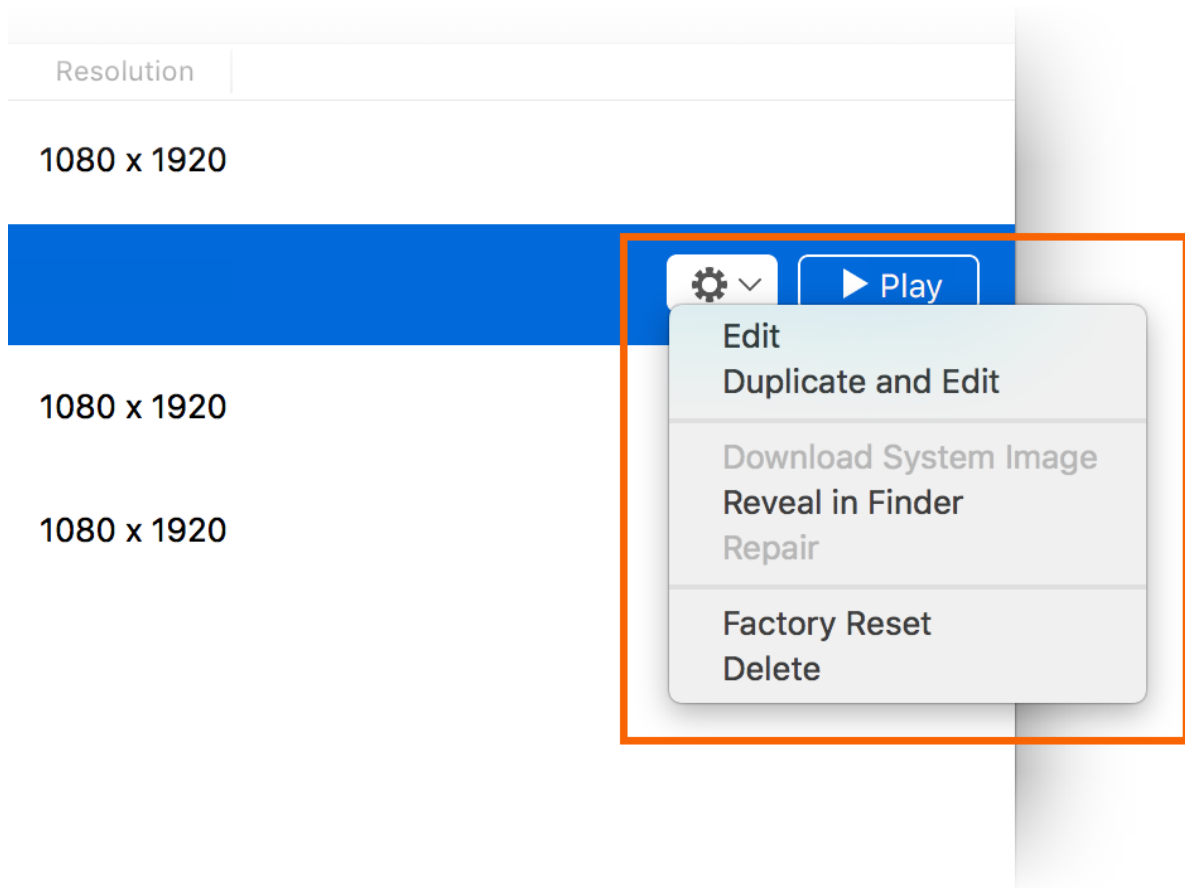
若要更改属性，请在“值”列中编辑其值。例如，在下面的屏幕截图中，`hw.lcd.density` 属性正从“480”更改为“240”：



在进行必要的配置更改后，单击“保存”按钮。有关更改虚拟设备属性的详细信息，请参阅[编辑 Android 虚拟设备属性](#)。

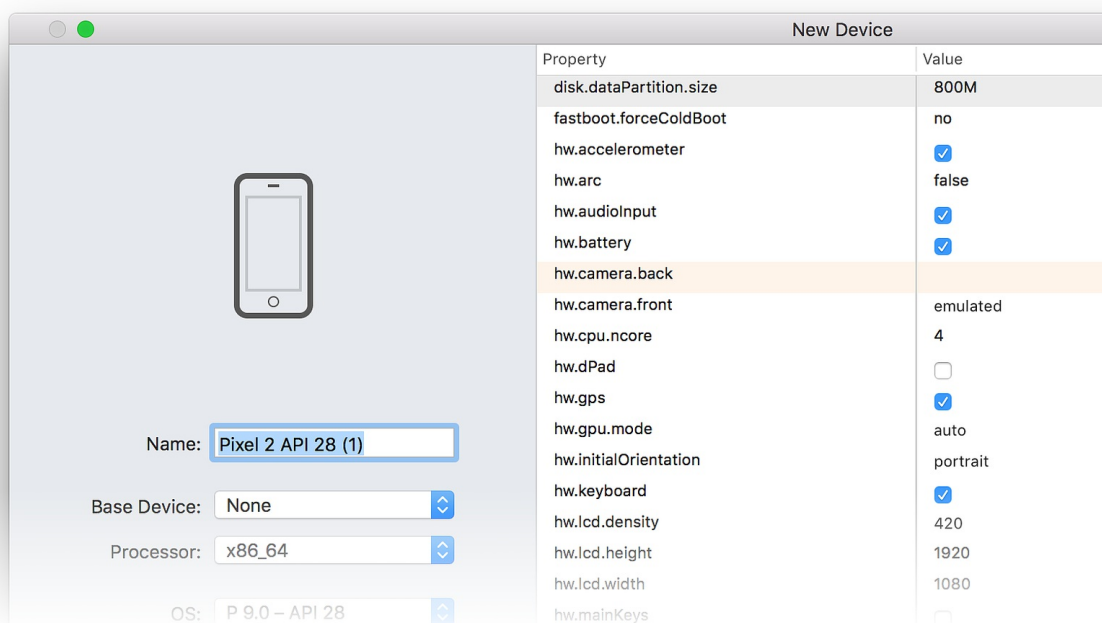
附加选项

“播放”按钮左侧的下拉菜单中提供适用于设备的附加选项：

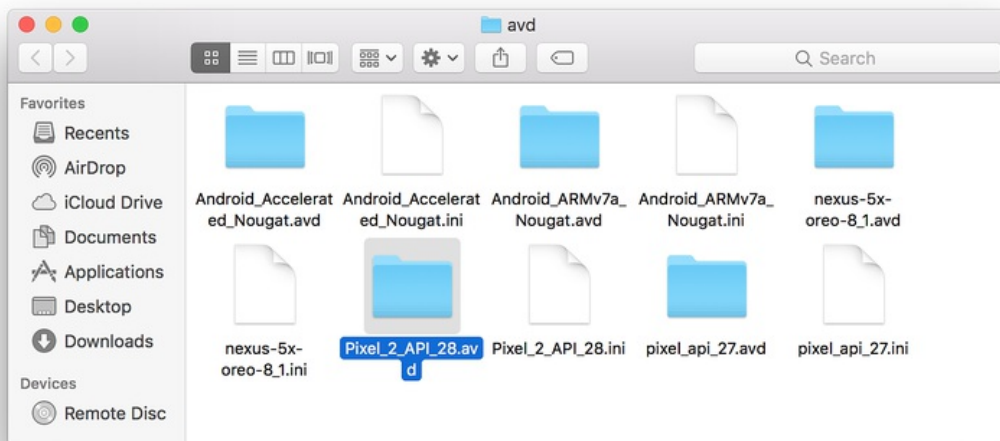


附加选项菜单中包含以下项：

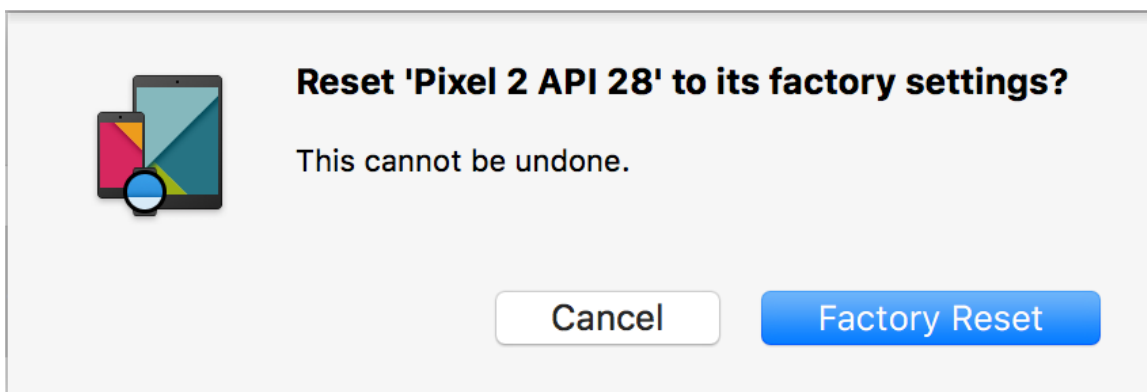
- **编辑** – 在设备编辑器中打开当前所选设备，如之前所述。
- **复制和编辑** – 复制当前所选的设备，并会在“新建设备”屏幕中使用不同的唯一名称打开。例如，选择“Pixel 2 API 28”并单击“复制和编辑”可将计数器追加到名称中：



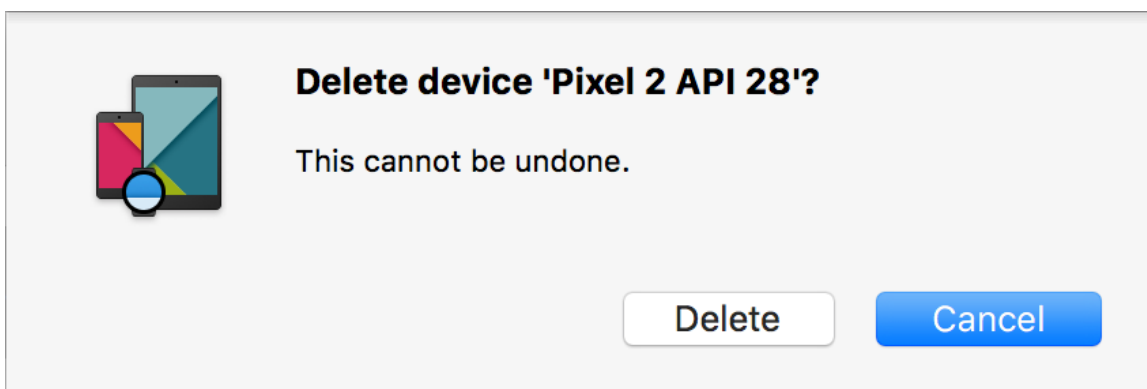
- **在查找器中展现** – 在包含用于虚拟设备的文件的文件夹中打开 macOS Finder 窗口。例如，选择“Pixel 2 API 28”并单击“在查找器中展现”以打开一个窗口，如下所示：



- **恢复出厂设置** – 将所选设备重置为其默认设置，擦除用户在此设备运行时对其内部状态进行的任何更改（这也会擦除当前**快速启动**快照（如果有））。此更改不会影响在创建和编辑期间对虚拟设备做出的修改。将出现提醒此重置无法被撤消的一个对话框。单击“恢复出厂设置”以确认重置。



- **删除** – 永久删除所选的虚拟设备。将出现提醒删除设备无法被撤消的一个对话框。如果确定要删除设备，请单击“删除”。



疑难解答

以下各节介绍如何诊断和避开使用 Android Device Manager 配置虚拟设备时可能发生的问题。

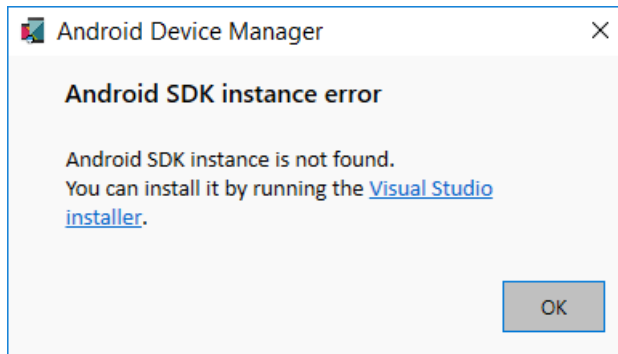
- [Visual Studio](#)
- [Visual Studio for Mac](#)

Android SDK 位于非标准位置

通常情况下，Android SDK 安装在以下位置：

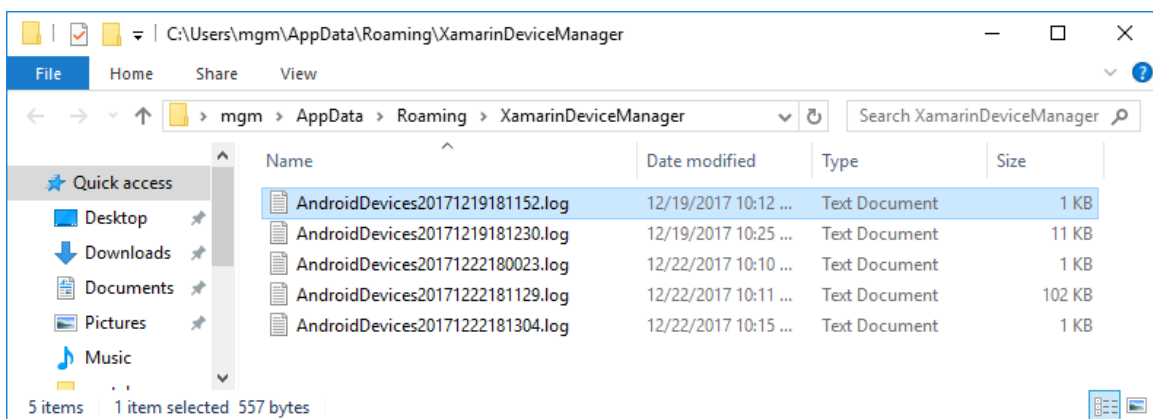
C:\Program Files (x86)\Android\android-sdk

如果 SDK 未安装在此位置, 在启动 Android Device Manager 时可能会遇到此错误:

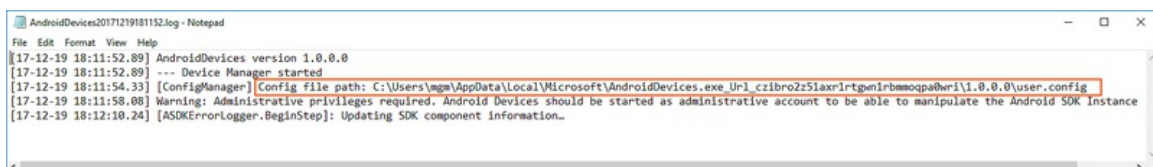


若要解决此问题, 请使用以下步骤:

1. 从 Windows 桌面依次转到 C:\Users\username\AppData\Roaming\XamarinDeviceManager:



2. 双击以打开某个日志文件, 并找到“配置文件路径”。例如:



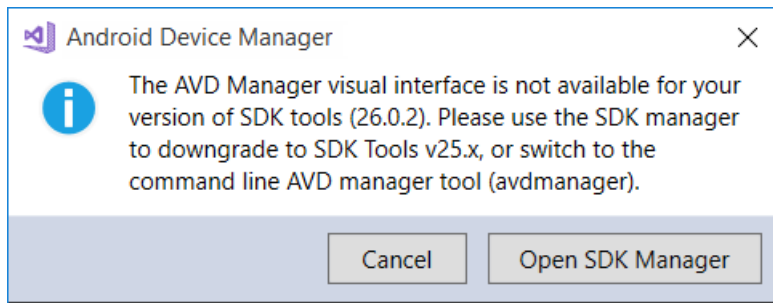
3. 导航到此位置, 然后双击“user.config”以将其打开。
4. 在“user.config”中, 找到 `<UserSettings>` 元素, 并向它添加一个“AndroidSdkPath”特性。将此特性设置为 Android SDK 在开发计算机上的安装位置路径, 然后保存文件。例如, 如果 Android SDK 安装在 C:\Programs\Android\SDK, `<UserSettings>` 将如下所示:

```
<UserSettings SdkLibLastWriteTimeUtcTicks="636409365200000000" AndroidSdkPath="C:\Programs\Android\SDK" />
```

对 user.config 进行更改后, 应该能够启动 Android Device Manager。

错误版本的 Android SDK Tools

如果未安装 Android SDK 工具 26.1.1 或更高版本, 在启动时可能会看到以下错误对话框:



如果看到此错误对话框，请单击“打开 SDK 管理器”以打开 Android SDK 管理器。在 Android SDK 管理器中，单击“工具”选项卡并安装以下包：

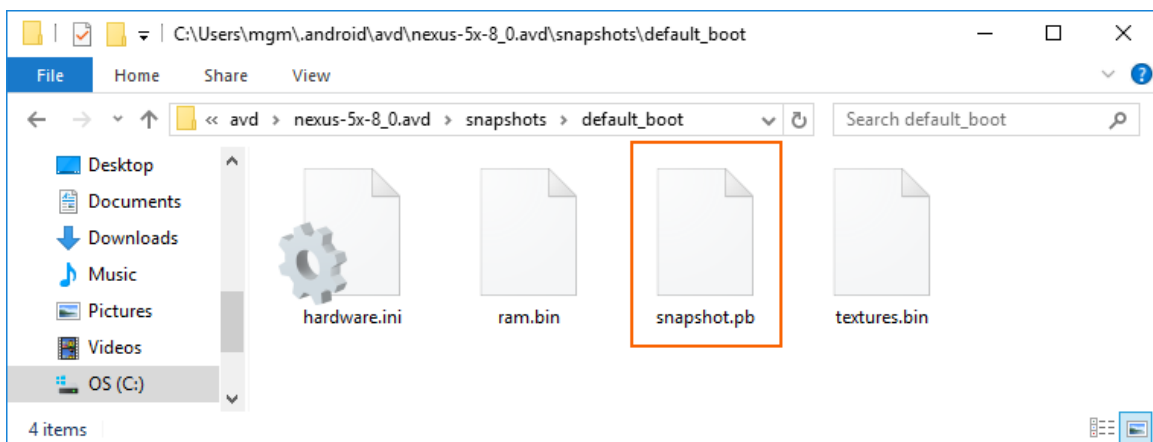
- Android SDK Tools 26.1.1 或更高版本
- Android SDK 平台工具 27.0.1 或更高版本
- Android SDK 生成工具 27.0.3 或更高版本

快照禁用 Android Oreo 上的 WiFi

如果 AVD 配置为使用模拟 Wi-Fi 访问的 Android Oreo，那么在快照之后重启 AVD 可能会导致 Wi-Fi 访问被禁用。

若要解决此问题：

1. 在 Android Device Manager 中选择 AVD。
2. 在其他选项菜单中，单击“在资源管理器中展现”。
3. 导航到“快照”>“default_boot”。
4. 删除 snapshot.pb 文件：



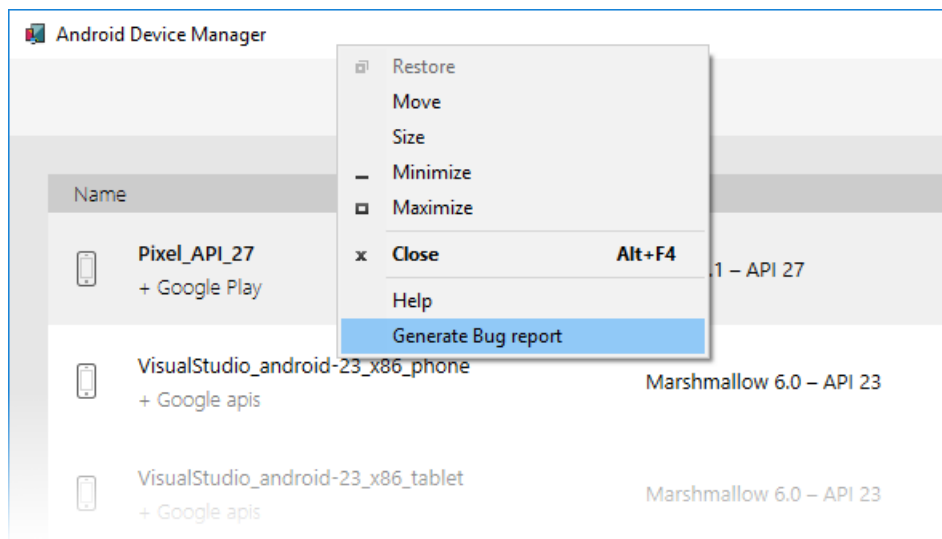
5. 重启 AVD。

进行这些更改后，AVD 将在允许 Wi-Fi 重新工作的状态下重新启动。

生成 Bug 报表

- [Visual Studio](#)
- [Visual Studio for Mac](#)

如果发现 Android Device Manager 出现问题，但无法使用上述疑难解答提示解决，请右键单击标题栏并选择“生成 Bug 报告”，提交该 bug 报告：



总结

本指南介绍 Visual Studio Tools for Xamarin 和 Visual Studio for Mac 中提供的 Android Device Manager。其中介绍了启动和停止 Android 仿真器、选择要运行的 Android 虚拟设备 (AVD)、创建新的虚拟设备以及如何编辑虚拟设备等基本功能。还介绍了如何编辑配置文件硬件属性以便进一步进行自定义并提供了常见问题的故障排除技巧。

相关链接

- [对 Android SDK 工具的更改](#)
- [在 Android Emulator 上调试](#)
- [SDK Tools 发行说明 \(Google\)](#)
- [avdmanager](#)
- [sdkmanager](#)

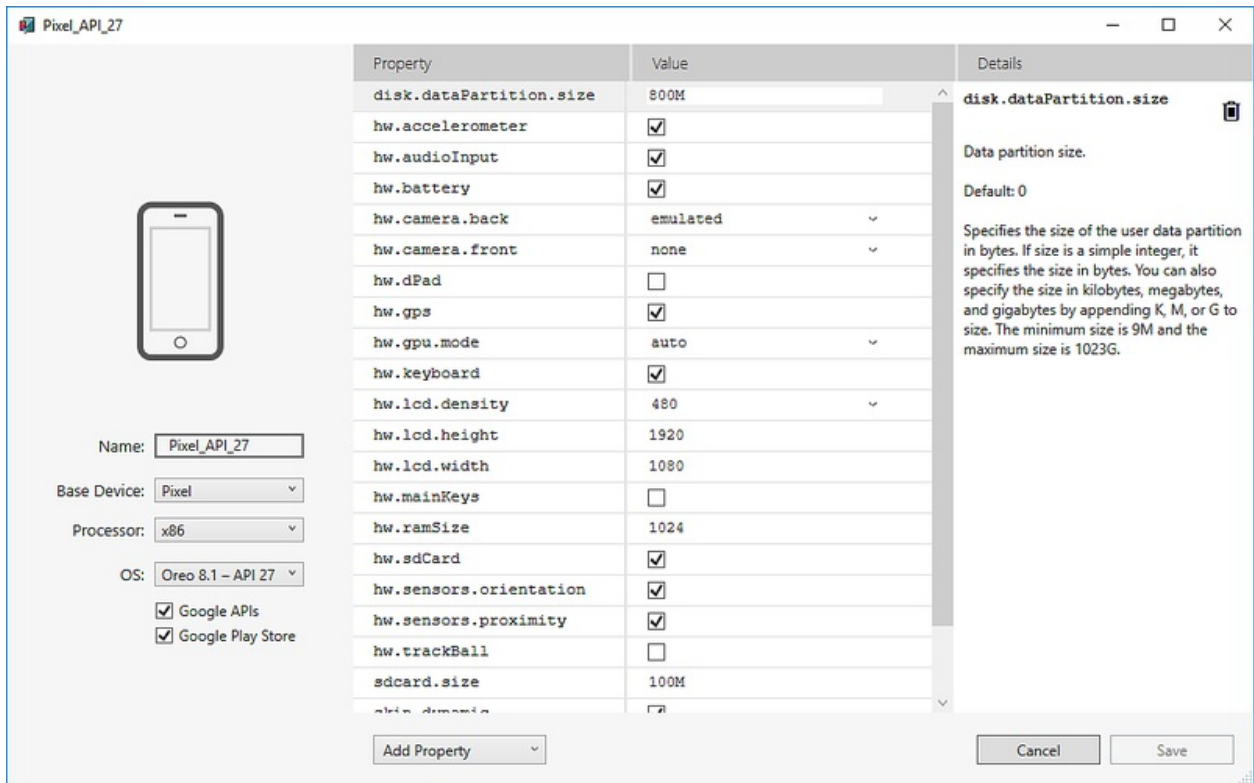
编辑 Android 虚拟设备属性

2018/11/2 • [Edit Online](#)

本文介绍了如何使用 Android Device Manager 编辑 Android 虚拟设备的配置文件属性。

Windows 上的 Android Device Manager

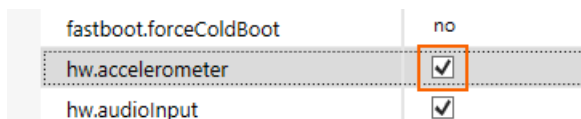
Android Device Manager 支持编辑单个 Android 虚拟设备配置文件属性。“新建设备”和“设备编辑”屏幕在第一列中列出了虚拟设备的属性，第二列中为每个属性相应的值(如此示例中所示)：



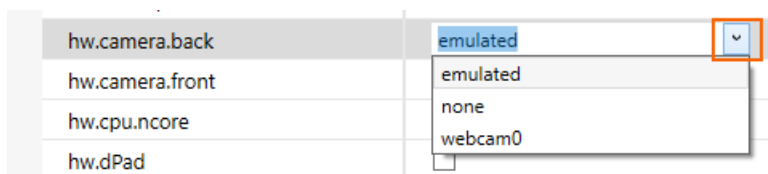
当选择某个属性时，有关该属性的详细描述会显示在右侧。可以修改“硬件配置文件属性”和“AVD 属性”。硬件配置文件属性(如 `hw.ramSize` 和 `hw.accelerometer`)描述仿真设备的物理特征。这些特征包括屏幕大小、可用的 RAM 量以及是否具有加速计。AVD 属性指定 AVD 在运行时的操作。例如，可以配置 AVD 属性以指定 AVD 如何使用开发计算机的图形卡进行呈现。

可以使用以下准则来更改属性：

- 若要更改布尔属性，请单击布尔属性右侧的复选标记：



- 若要更改枚举 (enumerated) 属性，请单击属性右侧的下拉箭头并选择一个新值。

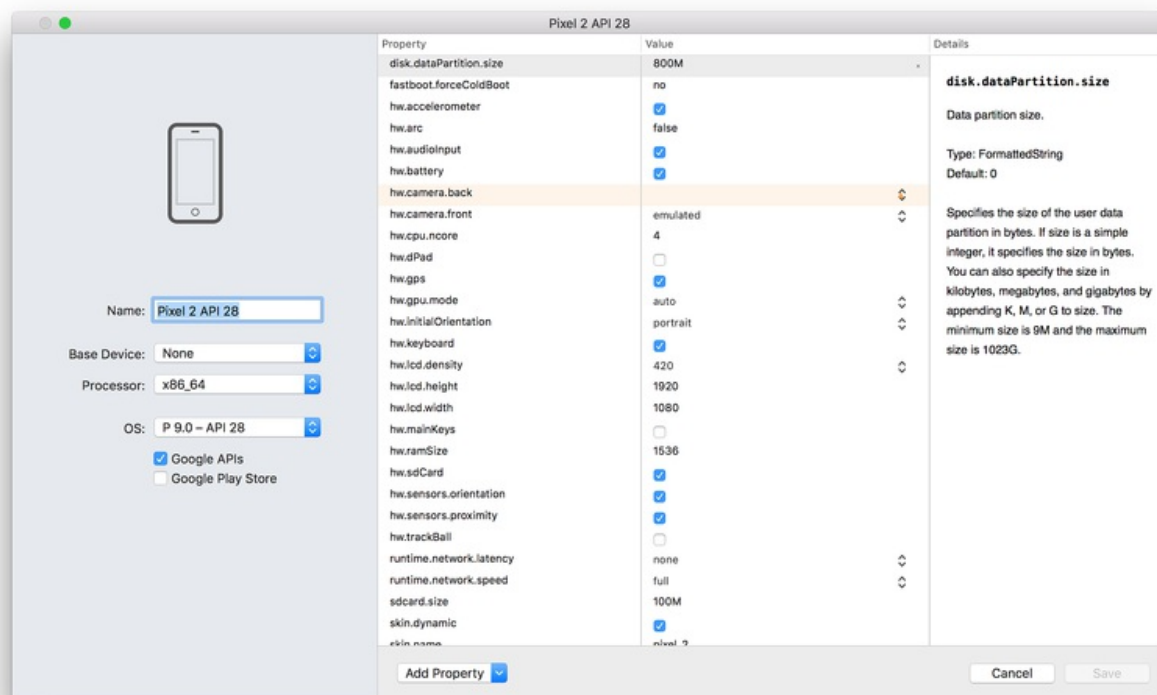


- 若要更改字符串或整数属性，请双击值列中当前的字符串或整数设置并输入一个新值。

| | |
|---------------|--------------------------|
| hw.lcd.height | 1920 |
| hw.lcd.width | 1080 |
| hw.mainKeys | <input type="checkbox"/> |

macOS 上的 Android Device Manager

Android Device Manager 支持编辑单个 Android 虚拟设备配置文件属性。“新建设备”和“设备编辑”屏幕在第一列中列出了虚拟设备的属性，第二列中为每个属性相应的值（如此示例中所示）：



当选择某个属性时，有关该属性的详细描述会显示在右侧。可以修改“硬件配置文件属性”和“AVD 属性”。硬件配置文件属性（如 `hw.ramSize` 和 `hw.accelerometer`）描述仿真设备的物理特征。这些特征包括屏幕大小、可用的 RAM 量以及是否具有加速计。AVD 属性指定 AVD 在运行时的操作。例如，可以配置 AVD 属性以指定 AVD 如何使用开发计算机的图形卡进行呈现。

可以使用以下准则来更改属性：

- 若要更改布尔属性，请单击布尔属性右侧的复选标记：

| | |
|------------------|-------------------------------------|
| abi.type | x86 |
| hw.accelerometer | <input checked="" type="checkbox"/> |
| hw.audioInput | <input checked="" type="checkbox"/> |

- 若要更改枚举 (enumerated) 属性，请单击属性右侧的下拉菜单并选择一个新值。

| | |
|-----------------|--|
| hw.camera.back | emulated |
| hw.camera.front | <input checked="" type="checkbox"/> none |
| hw.dPad | webcam0 |

- 若要更改字符串或整数属性，请双击值列中当前的字符串或整数设置并输入一个新值。

hw.lcd.density

hw.lcd.height

hw.lcd.width

420

1920

1080

下表详细说明了“新建设备”和“设备编辑器”屏幕中列出的属性：

| 属性 | 描述 | 选项 |
|--|---|----------------------------------|
| <code>abi.type</code> | ABI 类型 – 指定仿真设备的 ABI(应用程序二进制接口)类型。x86 选项适用于通常被称为“x86”或“IA-32”的指令集。x86_64 选项适用于 64 位 x86 指令集。armeabi-v7a 选项适用于具有 v7-a ARM 扩展的 ARM 指令集。arm64-v8a 选项适用于支持 AArch64 的 ARM 指令集。 | x86、x86_64、armeabi-v7a、arm64-v8a |
| <code>disk.cachePartition</code> | 缓存分区 – 确定仿真设备是否将在设备上使用 /cache 分区。/cache 分区(最初为空)是 Android 存储经常访问的数据和应用组件的位置。如果设置为“否”，仿真器将不使用 /cache 分区，且其他的 <code>disk.cache</code> 设置将被忽略。 | yes、no |
| <code>disk.cachePartition.path</code> | 缓存分区路径 – 在开发计算机上指定一个缓存分区映像文件。仿真器将此文件用于 /cache 分区。输入绝对路径或相对于仿真器数据目录的路径。如果未设置，仿真器将在开发计算机上创建一个名为 cache.img 的空临时文件。如果文件不存在，它则会被创建为一个空文件。如果 <code>disk.cachePartition</code> 设置为“否”，则忽略此选项。 | |
| <code>disk.cachePartition.size</code> | 缓存分区大小 – 缓存分区文件的大小(以字节为单位)。通常不需要设置此选项，除非应用将下载非常大的文件，这些文件大于 66 MB 的默认缓存大小。如果 <code>disk.cachePartition</code> 设置为“否”，则忽略此选项。如果此值为整数，它则以字节为单位指定大小。还可以通过将 K、M 或 G 追加到值中来以 KB、MB 或 GB 为单位指定大小。最小大小为 9 M；最大为 1023 G。 | |
| <code>disk.dataPartition.initPath</code> | 数据分区的初始路径 – 指定数据分区的初始内容。擦除用户数据后，仿真器将指定文件的内容复制到用户数据(默认情况下为 userdata-qemu.img)，而不是使用 userdata.img 作为初始版本。 | |

| 属性 | 描述 | 选项 |
|--|---|----|
| <code>disk.dataPartition.path</code> | 数据分区的路径 – 指定用户数据分区文件。若要配置永久性用户数据文件, 请在开发计算机上输入文件名和路径。如果文件不存在, 仿真器将从默认文件 <code>userdata.img</code> 创建一个映像, 将它存储在由 <code>disk.dataPartition.path</code> 指定的文件名中, 并在仿真器关闭时将用户数据永久保存到其中。如果不指定路径, 默认文件名为 <code>userdata-qemu.img</code> 。使用特殊值 会使仿真器创建并使用一个临时文件。如果 <code>disk.dataPartition.initPath</code> 已设置, 会再启动时将其内容复制到 <code>disk.dataPartition.path</code> 文件。注意, 此选项不能留空。 | |
| <code>disk.dataPartition.size</code> | 数据分区大小 – 指定用户数据分区的大小(以字节为单位)。如果此值为整数, 它则以字节为单位指定大小。还可以通过将 K、M 或 G 追加到值中来以 KB、MB 或 GB 为单位指定大小。最小大小为 9 M; 最大为 1023 G。 | |
| <code>disk.ramdisk.path</code> | Ramdisk 路径 – 到启动分区 (ramdisk) 映像的路径。ramdisk 映像是装载系统映像之前由内核加载的系统映像的子集。ramdisk 映像通常包含启动时的二进制文件和初始化脚本。如果未指定此选项, 仿真器系统目录中的默认值为 <code>ramdisk.img</code> 。 | |
| <code>disk.snapStorage.path</code> | 快照存储路径 – 到存储所有快照的快照存储文件的路径。在执行期间进行的所有快照将被保存到此文件。只有被保存到此文件的快照在仿真器运行期间才能被还原。如果未指定此选项, 仿真器数据目录中的默认值为 <code>snapshots.img</code> 。 | |
| <code>disk.systemPartition.initPath</code> | 系统分区初始路径 – 到系统映像文件的只读副本的路径; 具体来说, 是指包含系统二进制文件以及与 API 级别和任何变体相对应的数据的分区。如果未指定此选项, 仿真器系统目录中的默认值为 <code>system.img</code> 。 | |
| <code>disk.systemPartition.path</code> | 系统分区路径 – 到读/写系统分区映像的路径。如果未设置此路径, 则将创建一个临时文件并从 <code>disk.systemPartition.initPath</code> 指定的文件内容初始化此文件。 | |

| 属性 | 描述 | 选项 |
|--|--|-----------------------|
| <code>disk.systemPartition.size</code> | 系统分区大小 – 系统分区的理想大小 (以字节为单位)。如果实际的系统分区映像大于此设置, 则会忽略此大小; 否则, 它指定系统分区文件可以增长到的最大大小。如果此值为整数, 它则以字节为单位指定大小。还可以通过将 K、M 或 G 追加到值中来以 KB、MB 或 GB 为单位指定大小。最小大小为 9 M; 最大为 1023 G。 | |
| <code>hw.accelerometer</code> | 加速计 – 确定仿真设备是否包含一个加速度传感器。加速计帮助设备确定方向 (用于自动旋转)。加速计报告设备沿着三个传感器轴的加速情况。 | yes、no |
| <code>hw.audioInput</code> | 音频录制支持 – 确定仿真设备是否可以录制音频。 | yes、no |
| <code>hw.audioOutput</code> | 音频播放支持 – 确定仿真设备是否可以播放音频。 | yes、no |
| <code>hw.battery</code> | 电池支持 – 确定仿真设备是否可以通过电池运行。 | yes、no |
| <code>hw.camera</code> | 照相机支持 – 确定仿真设备是否具有照相机。 | yes、no |
| <code>hw.camera.back</code> | 后置照相机 – 配置后置照相机 (镜头背朝用户的面部)。如果在开发计算机上使用网络摄像头以模拟仿真设备上的后置照相机, 必须将此值设置为 <code>webcamn</code> , 其中 <i>n</i> 选择的是网络摄像头 (如果只有一个网络摄像头, 则选择 <code>webcam0</code>)。如果设置为“emulated”, 仿真器将在软件中模拟照相机。若要禁用后置照相机, 请将此值设置为“none”。如果启用后置照相机, 请确保也要启用 <code>hw.camera</code> 。 | emulated、none、webcam0 |
| <code>hw.camera.front</code> | 前置照相机 – 配置前置照相机 (镜头面向用户)。如果在开发计算机上使用网络摄像头以模拟仿真设备上的前置照相机, 必须将此值设置为 <code>webcamn</code> , 其中 <i>n</i> 选择的是网络摄像头 (如果只有一个网络摄像头, 则选择 <code>webcam0</code>)。如果设置为“emulated”, 仿真器将在软件中模拟照相机。若要禁用前置照相机, 请将此值设置为“none”。如果启用前置照相机, 请确保也要启用 <code>hw.camera</code> 。 | emulated、none、webcam0 |
| <code>hw.camera.maxHorizontalPixels</code> | 最大的照相机水平像素 – 配置仿真设备照相机的最大水平分辨率 (以像素为单位)。 | |
| <code>hw.camera.maxVerticalPixels</code> | 最大的照相机垂直像素 – 配置仿真设备照相机的最大垂直分辨率 (以像素为单位)。 | |

| 属性 | 描述 | 选项 |
|-----------------------------|---|--------------------------------------|
| <code>hw.cpu.arch</code> | CPU 体系结构 – 要由虚拟设备仿真的 CPU 体系结构。如果使用 Intel HAXM 以进行硬件加速, 请为 32 位 CPU 选择 x86。为 64 位 HAXM 加速的设备选择 x86_64。(请务必在 SDK 管理器中安装相应的 Intel x86 系统映像:例如, Intel x86 Atom 或 Intel x86 Atom_64。)若要模拟 ARM CPU, 请为 32 位 ARM CPU 选择 arm 或者为 64 位选择 arm64。请记住, 基于 ARM 的虚拟设备运行速度要比基于 x86 的那些虚拟设备慢得多, 因为硬件加速不适用于 ARM。 | x86、x86_64、arm、arm64 |
| <code>hw.cpu.model</code> | CPU 型号 – 通常不会设置此值(如果未显式设置此值, 它将被设置为派生自 <code>hw.cpu.arch</code> 的一个值)。但是, 可以将它设置为特定于仿真器的字符串以作实验性使用。 | |
| <code>hw.dPad</code> | DPad 密钥 – 确定仿真设备是否支持方向键 (DPad) 密钥。一个 DPad 通常具有四个密钥以指示方向控件。 | yes、no |
| <code>hw.gps</code> | GPS 支持 – 确定仿真设备是否具有 GPS(全球定位系统)接收器。 | yes、no |
| <code>hw.gpu.enabled</code> | GPU 仿真 – 确定仿真设备是否支持 GPU 仿真。启用后, GPU 仿真会使用 Open GL for Embedded Systems 以在屏幕上呈现 2D 和 3D 图形, 并且关联的 GPU 仿真模式设置会确定 GPU 仿真的实现方式。 | yes、no |
| <code>hw.gpu.mode</code> | GPU 仿真模式 – 确定仿真器实现 GPU 仿真的方式。如果选择“auto”, 仿真器将根据开发计算机设置选择硬件加速和软件加速。如果选择“host”, 仿真器将使用开发计算机的图形处理器执行 GPU 仿真以进行更快的呈现。如果 GPU 与仿真器不兼容并且系统为 Windows, 则可以尝试选择“angle”, 而不是“host”。“angle”模式使用 DirectX 以提供与“host”模式类似的性能。如果选择“mesa”, 模拟器将使用 Mesa 3D 软件库来呈现图形。如果通过开发计算机的图形处理器进行呈现存在问题, 请选择“mesa”。可以使用“swiftshader”模式在软件中呈现图形, 不过性能与使用计算机的 CPU 相比稍有降低。“off”选项(禁用图形硬件仿真)是已弃用的一个选项, 使用此选项后可能无法正确呈现某些项, 因此不推荐此选项。 | auto、host、mesa、angle、swiftshader、off |
| <code>hw.gsmModem</code> | GSM 调制解调器支持 – 确定仿真设备是否包含支持 GSM(全球移动通信系统)电话无线电系统的调制解调器。 | yes、no |

| 属性 | 描述 | 选项 |
|------------------------------------|--|---------------------|
| <code>hw.initialOrientation</code> | 初始屏幕方向 – 配置仿真设备上屏幕的初始方向(纵向或横向模式)。在纵向模式下, 屏幕的高度大于宽度。在横向模式下, 屏幕的宽度大于高度。如果设备配置文件中都支持纵向和横向模式, 则可以在运行仿真设备时更改方向。 | portrait、landscape |
| <code>hw.keyboard</code> | 键盘支持 – 确定仿真设备是否支持全键盘。 | yes、no |
| <code>hw.keyboard.charmap</code> | 键盘字符映射名称 – 此设备的硬件字符映射的名称。注意:此应始终为默认值 <code>qwerty2</code> , 除非相应地修改了系统映像。此名称会在启动时发送到内核。使用不正确的名称将导致虚拟设备不可用。 | |
| <code>hw.keyboard.lid</code> | 键盘盖支持 – 在启用键盘支持的情况下, 此设置确定是否可以关闭/隐藏或打开/显示全键盘。如果“hw.keyboard”设置为“false”, 则将忽略此设置。注意:如果仿真设备面向 API 级别 12 或更高版本, 默认值则为“false”。 | yes、no |
| <code>hw.lcd.backlight</code> | LCD 背光 – 确定 LCD 背光是否由仿真设备模拟。 | yes、no |
| <code>hw.lcd.density</code> | LCD 密度 – 仿真 LCD 显示的密度, 以与密度无关的像素或 dp(dp 是一个虚拟像素单位)为单位测量。当设置为 160 dp 时, 每个 dp 将对应一个物理像素。在运行时, Android 使用此值选择和缩放适当的资源/资产以进行正确的显示呈现。 | 120、160、240、213、320 |
| <code>hw.lcd.depth</code> | LCD 颜色深度 – 保留位图以驱动 LCD 显示的仿真帧缓冲区的颜色位深度。此值可以为 16 位(65,536 种可能的颜色)或 32 位(16,777,216 种颜色和透明度)。尽管 32 位设置使仿真器运行较为缓慢, 但颜色准确度更高。 | 16, 32 |
| <code>hw.lcd.height</code> | LCD 像素高度 – 构成仿真 LCD 显示的垂直维度的像素数量。 | |
| <code>hw.lcd.width</code> | LCD 像素宽度 – 构成仿真 LCD 显示的水平维度的像素数量。 | |
| <code>hw.mainKeys</code> | 硬件返回/主页键 – 确定仿真设备是否支持硬件“返回”和“主页”导航按钮。如果仅在软件中实现按钮, 可以将此值设置为“yes”。如果将 <code>hw.mainKeys</code> 设置为“yes”, 仿真器将不会在屏幕上显示导航按钮, 但你可以使用仿真器侧面板来“按”这些按钮。 | yes、no |

| 属性 | 描述 | 选项 |
|--|---|----------------------------|
| <code>hw.ramSize</code> | 设备 RAM 大小 – 仿真器上的物理 RAM 量(以 MB 为单位)。默认值将根据屏幕大小或外观版本来计算。尽管增加大小可以提供更快的仿真器操作, 但这将耗费开发计算机中更多的资源。 | |
| <code>hw.screen</code> | 触控屏类型 – 定义仿真器上的屏幕类型。多点触控屏可以在触控界面上跟踪两根或更多的手指。触控屏仅可以检测单个手指触控事件。无触控屏不会检测触控事件。 | touch、multi-touch、no-touch |
| <code>hw.sdCard</code> | SDCard 支持 – 确定仿真设备是否支持插入和移除虚拟 SD(数字安全)卡。仿真器使用存储在开发计算机上可装载的磁盘映像来模拟真实 SD 卡设备的分区(请参阅 <code>hw.sdCard.path</code>)。 | yes、no |
| <code>sdcard.size</code> | SDCard 大小 – 指定虚拟 SD 卡文件的大小(以字节为单位), 此文件位于由设备上可用的 <code>hw.sdCard.path</code> 指定的位置。如果此值为整数, 它则以字节为单位指定大小。还可以通过将 K、M 或 G 追加到值中来以 KB、MB 或 GB 为单位指定大小。最小大小为 9 M; 最大为 1023 G。 | |
| <code>hw.sdCard.path</code> | SDCard 映像路径 – 指定开发计算机上 SD 卡分区映像文件的文件名和路径。例如, 可以在 Windows 上将此路径设置为 <code>C:\sd\sdcard.img</code> 。 | |
| <code>hw.sensors.magnetic_field</code> | 磁场传感器 – 确定仿真设备是否支持磁场传感器。磁场传感器(也被称为磁力计)报告沿三个传感器轴测量的环境磁场。为需要访问指南针读数的应用启用此设置。例如, 导航应用可能需要此传感器以检测用户面朝的方向。 | yes、no |
| <code>hw.sensors.orientation</code> | 方向传感器 – 确定仿真设备是否提供方向传感器值。方向传感器测量某个设备围绕三个物理坐标轴(x、y、z)旋转的度数。注意, 自 Android 2.2(API 级别 8)起, 方向传感器已被弃用。 | yes、no |
| <code>hw.sensors.proximity</code> | 邻近感应传感器 – 确定仿真设备是否支持邻近感应传感器。此传感器测量某个物体相对于设备的视图屏幕的邻近度。此传感器通常用于确定话筒是否正在向上靠近一个人的耳朵。 | yes、no |
| <code>hw.sensors.temperature</code> | 温度传感器 – 确定仿真设备是否支持温度传感器。此传感器以摄氏度(°C)为单位测量设备的温度。 | yes、no |

| 属性 | 描述 | 选项 |
|-------------------------------------|---|-------------------|
| <code>hw.touchScreen</code> | 触控屏支持 – 确定仿真设备是否支持触控屏。触控屏用于在屏幕上直接操作对象。 | yes、no |
| <code>hw.trackBall</code> | 轨迹球支持 – 确定仿真设备是否支持轨迹球。 | yes、no |
| <code>hw.useext4</code> | EXT4 文件系统支持 – 确定仿真设备是否将 Linux EXT4 文件系统用于分区。因为文件系统类型是自动检测的, 因此, 此选项已被弃用并忽略。 | 否 |
| <code>kernel.newDeviceNaming</code> | 内核新设备命名 – 用于指定内核是否需要新的设备命名方案。这通常与 Linux 3.10 内核和更高版本配合使用。如果设置为“autodetect”, 仿真器将自动检测内核是否需要新的设备命名方案。 | autodetect、yes、no |
| <code>kernel.parameters</code> | 内核参数 – 指定 Linux 内核启动参数的字符串。默认情况下, 此设置将留空。 | |
| <code>kernel.path</code> | 内核路径 – 指定 Linux 内核的路径。如果未指定此路径, 仿真器会在仿真器系统目录中查找 kernel-ranchu。 | |
| <code>kernel.supportsYaffs2</code> | YAFFS2 分区支持 – 确定内核是否支持 YAFFS2 (Yet Another Flash File System 2) 分区。通常情况下, 这仅适用于 Linux 3.10 之前的内核。如果设置为“autodetect”, 仿真器将自动检测内核是否可以装载 YAFFS2 文件系统。 | autodetect、yes、no |
| <code>skin.name</code> | 外观名称 – Android 仿真器外观的名称。外观是定义仿真显示的视觉对象和控件元素的文件集合, 描述了 AVD 的窗口在开发计算机上的外观。外观描述屏幕大小、按钮和整体设计, 但不会影响应用的操作。 | |
| <code>skin.path</code> | 外观路径 – 包含在 skin.name 中指定的仿真器外观文件的目录的路径。此目录包含 hardware.ini 布局文件以及外观显示元素的映像文件。 | |
| <code>skin.dynamic</code> | 外观动态 – 外观是否为动态的。如果仿真器是基于指定的宽度和高度构造给定大小的外观, 仿真器外观则为动态外观。 | 否 |

有关这些属性的详细信息, 请参阅[硬件配置文件属性](#)。

Android Emulator 疑难解答

2018/11/2 • [Edit Online](#)

本文介绍配置和运行 Android Emulator 时最常见的警告消息和问题。此外，其中还介绍了这些错误的解决方案以及各种疑难解答提示，以帮助诊断仿真器出现的问题。

在 Windows 上部署时出现的问题

部署应用时，仿真器可能会显示一些错误消息。此处介绍最常见的错误和解决方案。

部署错误

若出现有关无法在仿真器上安装 APK 或无法运行 Android Debug Bridge (adb) 的错误消息，请验证 Android SDK 能否连接到仿真器。要验证仿真器的连接情况，请使用以下步骤：

1. 通过“Android Device Manager”启动仿真器(选择虚拟设备并单击“启动”)。
2. 打开命令提示符，转到 adb 的安装文件夹。如果 Android SDK 安装在其默认位置，则 adb 位于 C:\Program Files (x86)\Android\android-sdk\platform-tools\adb.exe；如果不是，请在计算机上修改 Android SDK 所在的路径。
3. 键入以下命令：

```
adb devices
```

4. 如果可以通过 Android SDK 访问仿真器，那么仿真器应该就显示在附加设备列表中。例如：

```
List of devices attached
emulator-5554    device
```

5. 如果仿真器不在此列表中，请启动“Android SDK 管理器”，应用所有更新，再尝试重启仿真器。

MMIO 访问错误

若出现“发生 MMIO 访问错误”消息，请重启仿真器。

缺少 Google Play Services

如果在仿真器中运行的虚拟设备未安装 Google Play Services 或 Google Play 商店，则在未安装这些软件包时创建虚拟设备通常会出现此情况。创建虚拟设备时(请参阅[使用 Android Device Manager 管理虚拟设备](#))，请务必选择以下一个或两个选项：

- **Google API** – 在虚拟设备中包含 Google Play Services。
- **Google Play 商店** – 在虚拟设备中包含 Google Play 商店。

例如，此虚拟设备将包含 Google Play Services 和 Google Play 商店：

Name:

Base Device:

Processor:

OS:

☒ Google APIs

☒ Google Play Store

NOTE

Google Play 商店图片仅适用于某些基本设备类型，例如 Pixel、Pixel 2、Nexus 5 和 Nexus 5X。

性能问题

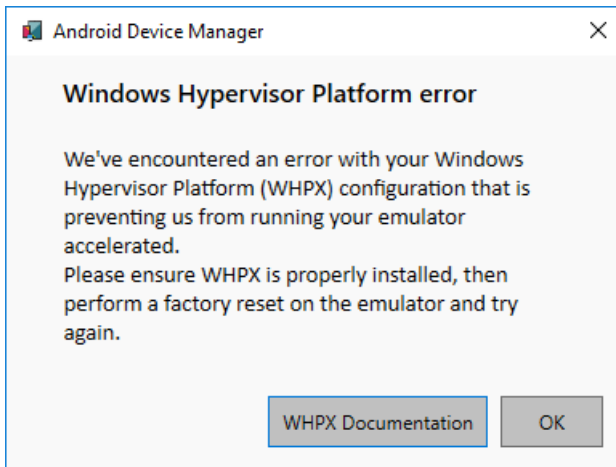
性能问题通常由以下某个问题引起：

- 仿真器在没有硬件加速的情况下运行。
- 在仿真器中运行的虚拟设备未使用基于 x86 的系统映像。

以下部分更详细地介绍了这些方案。

未启用硬件加速

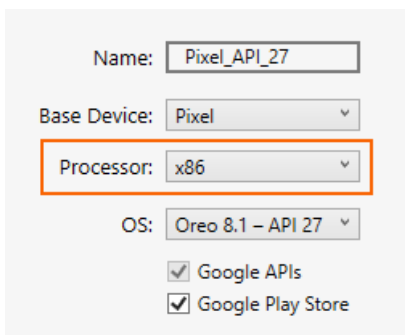
如果未启用硬件加速，则从设备管理器启动虚拟设备时将生成一个对话框，其中显示一条错误消息，指出未正确配置 Windows 虚拟机监控程序平台 (WHPX)：



如果显示此错误消息，请参阅下面的[硬件加速问题](#)，了解可用于验证和启用硬件加速的步骤。

加速已启用但仿真器运行速度过慢

导致此问题的常见原因是虚拟设备 (AVD) 中未使用基于 x86 的映像。创建虚拟设备时 (请参阅[使用 Android Device Manager 管理虚拟设备](#))，请确保选择基于 x86 的系统映像：



硬件加速问题

无论使用 Hyper-V 还是 HAXM 进行硬件加速，都可能会遇到配置问题或与计算机上的其他软件发生冲突。可通过打开命令提示符并输入以下命令来验证是否已启用硬件加速（以及仿真器正在使用哪种加速方法）：

```
"C:\Program Files (x86)\Android\android-sdk\emulator\emulator-check.exe" accel
```

此命令假定将 Android SDK 安装在默认位置 C:\Program Files (x86)\Android\android-sdk；如果不是，请在计算机上修改上述 Android SDK 所在的路径。

硬件加速不可用

如果 Hyper-V 可用，将从 emulator-check.exe accel 命令返回类似以下示例的消息：

```
HAXM is not installed, but Windows Hypervisor Platform is available.
```

如果 HAXM 可用，将返回类似以下示例的消息：

```
HAXM version 6.2.1 (4) is installed and usable.
```

如果硬件加速不可用，将显示如下示例的消息（如果无法找到 Hyper-V，仿真器将查找 HAXM）：

```
HAXM is not installed on this machine
```

如果硬件加速不可用，请参阅[使用 Hyper-V 加速](#)了解如何在计算机上启用硬件加速。

BIOS 设置不正确

如果未正确配置 BIOS 以支持硬件加速，则在运行 emulator-check.exe accel 命令时将显示类似于以下示例的消息：

```
VT feature disabled in BIOS/UEFI
```

要解决此问题，请重启计算机的 BIOS 并启用以下选项：

- 虚拟化技术（标签可能因 motherboard 制造商而不同）。
- 硬件强制执行数据执行保护。

如果启用了硬件加速并且 BIOS 配置正确，则仿真器可通过硬件加速成功运行。然而，一些特定于 Hyper-V 和 HAXM 的问题仍可能引起某些问题，下面将对此进行解释。

HYPER-V 问题

在某些情况下，在“打开或关闭 Windows 功能”对话框中启用 Hyper-V 和 Windows 虚拟机监控程序平台后可能无法正确启用 Hyper-V。要验证是否已启用 Hyper-V，请使用以下步骤：

1. 在 Windows 搜索框中, 输入“powershell”。
2. 右键单击搜索结果中的 Windows PowerShell, 然后选择“以管理员身份运行”。
3. 在 PowerShell 控制台中, 输入以下命令:

```
Get-WindowsOptionalFeature -FeatureName Microsoft-Hyper-V-All -Online
```

如果未启用 Hyper-V, 将显示类似于以下示例的消息, 指示 Hyper-V 的状态为“已禁用”:

```
FeatureName      : Microsoft-Hyper-V-All
DisplayName      : Hyper-V
Description      : Provides services and management tools for creating and running virtual machines
                  and their resources.
RestartRequired  : Possible
State            : Disabled
CustomProperties :
```

4. 在 PowerShell 控制台中, 输入以下命令:

```
Get-WindowsOptionalFeature -FeatureName HypervisorPlatform -Online
```

如果未启用虚拟机监控程序, 将显示类似于以下示例的消息, 指示虚拟机监控程序平台的状态为“已禁用”:

```
FeatureName      : HypervisorPlatform
DisplayName      : Windows Hypervisor Platform
Description      : Enables virtualization software to run on the Windows hypervisor
RestartRequired  : Possible
State            : Disabled
CustomProperties :
```

如果未启用 Hyper-V 和/或虚拟机监控程序平台, 请使用以下 PowerShell 命令进行启用:

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
Enable-WindowsOptionalFeature -Online -FeatureName HypervisorPlatform -All
```

完成这些命令后, 请进行重启。

有关启用 Hyper-V 的详细信息(包括使用部署映像服务和管理工具启用 Hyper-V 的技术), 请参阅[安装 Hyper-V](#)。

HAXM 问题

导致 HAXM 问题的原因通常包括: 与其他虚拟化技术冲突、设置不正确或 HAXM 驱动器不是最新版本。

HAXM 进程未运行

如果已安装 HAXM, 则可通过打开命令提示符并输入以下命令来验证 HAXM 进程是否正在运行:

```
sc query intelhaxm
```

如果 HAXM 进程正在运行, 应看到类似于下面结果的输出:

```
SERVICE_NAME: intelhaxm
    TYPE           : 1  KERNEL_DRIVER
    STATE          : 4  RUNNING
                   (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
    WIN32_EXIT_CODE : 0  (0x0)
    SERVICE_EXIT_CODE : 0  (0x0)
    CHECKPOINT      : 0x0
    WAIT_HINT       : 0x0
```

如果 `STATE` 未设置为 `RUNNING`，请参阅[如何使用 Intel 硬件加速执行管理器](#)解决该问题。

HAXM 虚拟化冲突

HAXM 可能与其他使用虚拟化的技术(如 Hyper-V、Windows Device Guard 和某防病毒软件)冲突：

- **Hyper-V** – 如果使用的是 Windows 10 2018 年 4 月更新(内部版本 1803)之前的 Windows 版本，并启用了 Hyper-V，请按照[禁用 Hyper-V](#)中的步骤操作以便启用 HAXM。
- **Device Guard** – Device Guard 和 Credential Guard 可阻止在 Windows 计算机上禁用 Hyper-V。若要禁用 Device Guard 和 Credential Guard，请参阅[禁用 Device Guard](#)。
- **防病毒软件** – 如果运行的防病毒软件(如 Avast)使用硬件协助虚拟化，请禁用或卸载此软件，再重启并重试运行 Android Emulator。

BIOS 设置不正确

若要在 Windows PC 上使用 HAXM，只有在 BIOS 中启用虚拟化技术 (Intel VT-x)，HAXM 才能正常运行。如果已禁用 VT-x，则尝试启用 Android Emulator 时会出现如下错误消息：

此计算机符合 **HAXM** 要求，但未启用 **Intel 虚拟化技术 (VT-x)**。

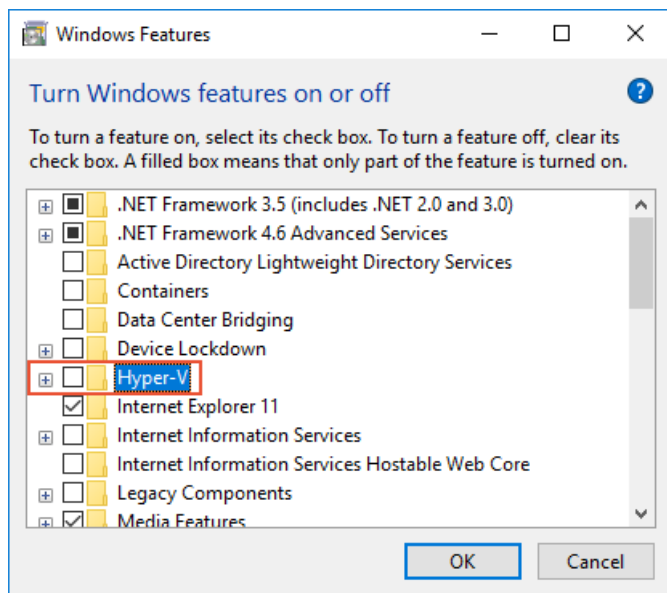
若要更正此错误，请将计算机引导到 BIOS，同时启用 VT-x 和 SLAT(第二级地址转换)，再重启计算机，以返回到 Windows。

禁用 Hyper-V

如果使用的是 Windows 10 2018 年 4 月更新(内部版本 1803)之前的 Windows 版本，并启用了 Hyper-V，则必须禁用 Hyper-V 并重启计算机才能安装和使用 HAXM。如果使用的是 Windows 10 2018 年 4 月更新(内部版本 1803)或更高版本，则 Android Emulator 27.2.7 或更高版本可以使用 Hyper-V(而不是 HAXM)进行硬件加速，因此不需要禁用 Hyper-V。

可以按照下列步骤操作，在“控制面板”中禁用 Hyper-V：

1. 在 Windows 搜索框中输入“Windows 功能”，然后在搜索结果中选择“打开或关闭 Windows 功能”。
2. 取消选中 Hyper V：



3. 重新启动计算机。

或者，可使用以下 PowerShell 命令禁用 Hyper-V 虚拟机监控程序：

```
Disable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-Hypervisor
```

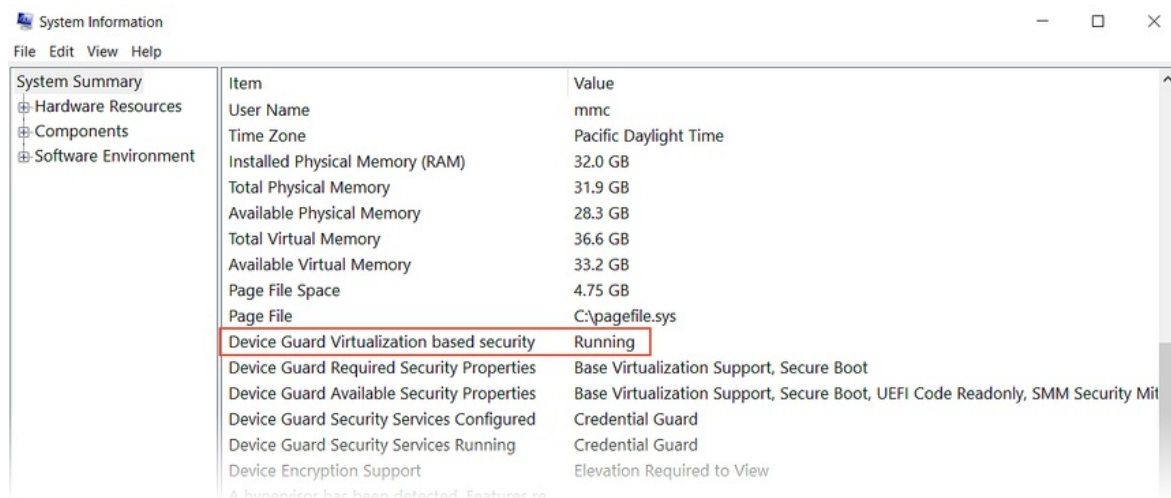
Intel HAXM 和 Microsoft Hyper-V 不能同时处于活动状态。遗憾的是，没有办法在不重启计算机的情况下在 Hyper-V 和 HAXM 之间切换。

在某些情况下，如果启用了 Device Guard 和 Credential Guard，按照前述步骤操作将无法成功禁用 Hyper-V。如果无法禁用 Hyper-V（或似乎已禁用，但仍无法安装 HAXM），请按照下一部分中的步骤操作，禁用 Device Guard 和 Credential Guard。

禁用 Device Guard

Device Guard 和 Credential Guard 可阻止在 Windows 计算机上禁用 Hyper-V。对于由负责组织配置和控制的域加入计算机而言，这种情况通常都是一个需要解决的问题。在 Windows 10 上，请按照下列步骤操作，检查 Device Guard 是否在运行：

1. 在 Windows 搜索框中输入“系统信息”，然后在搜索结果中选择“系统信息”。
2. 在“系统摘要”中，检查是否有“基于 Device Guard 虚拟化的安全性”；若有，检查是否处于“正在运行”状态：

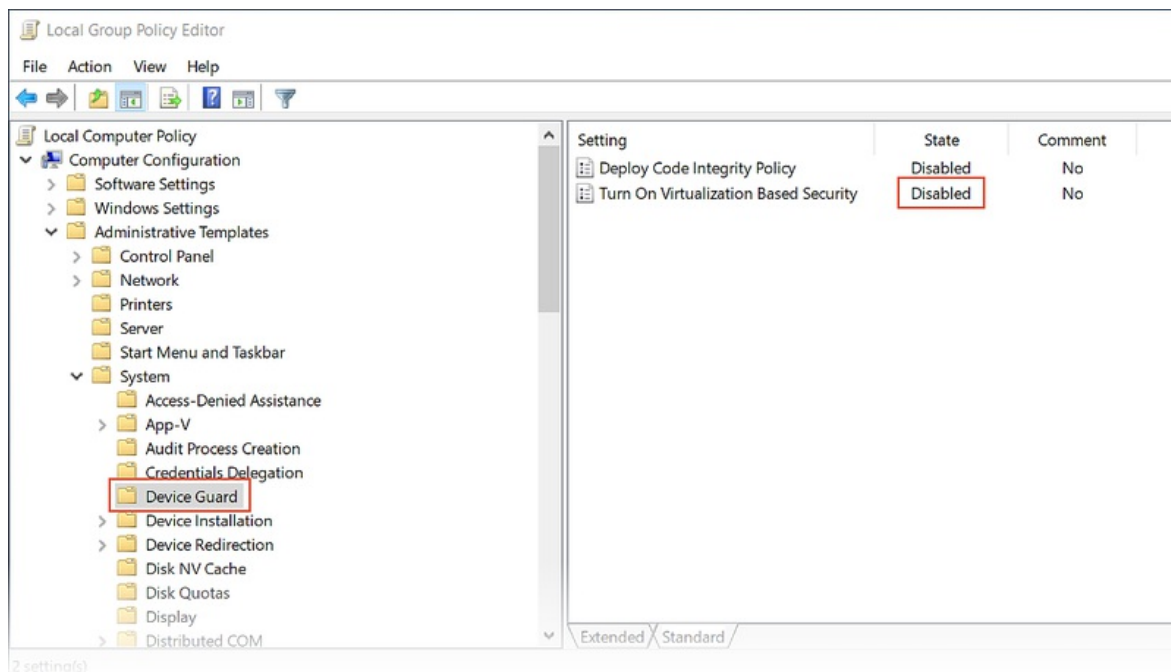


如果已启用 Device Guard，请按照下列步骤操作，禁用 Device Guard：

1. 确保已按照上一部分所述禁用“Hyper-V”（在“打开或关闭 Windows 功能”下）。
2. 在 Windows 搜索框中，输入“gpedit”，再选择“编辑组策略”搜索结果。这些步骤将启动“本地组策略编辑

器”。

- 在“本地组策略编辑器”中，依次转到“计算机配置”>“管理模板”>“系统”>“Device Guard”：



- 将“打开基于虚拟化的安全性”更改为“已禁用”(如上所示)，再退出“本地组策略编辑器”。
- 在 Windows 搜索框中，输入“cmd”。右键单击搜索结果中的“命令提示符”，再选择“以管理员身份运行”。
- 将以下命令复制并粘贴到命令提示符窗口(如果正在使用驱动器 Z:，请改为选择未使用的驱动器号)：

```
mountvol Z: /s
copy %WINDIR%\System32\SecConfig.efi Z:\EFI\Microsoft\Boot\SecConfig.efi /Y
bcdedit /create {0cb3b571-2f2e-4343-a879-d86a476d7215} /d "DebugTool" /application osloader
bcdedit /set {0cb3b571-2f2e-4343-a879-d86a476d7215} path "\EFI\Microsoft\Boot\SecConfig.efi"
bcdedit /set {bootmgr} bootsequence {0cb3b571-2f2e-4343-a879-d86a476d7215}
bcdedit /set {0cb3b571-2f2e-4343-a879-d86a476d7215} loadoptions DISABLE-LSA-ISO,DISABLE-VBS
bcdedit /set {0cb3b571-2f2e-4343-a879-d86a476d7215} device partition=Z:
mountvol Z: /d
```

- 重新启动计算机。在启动屏幕上，应该会出现类似以下消息的提示：

是否要禁用 Credential Guard?

按下指示的键，以根据提示禁用 Credential Guard。

- 重启计算机后，再次检查，以确保 Hyper-V 已禁用(如前述步骤所述)。

如果 Hyper-V 仍未禁用，域加入计算机的策略可能会阻止禁用 Device Guard 或 Credential Guard。在这种情况下，可以向域管理员申请豁免，以便能够选择禁用 Credential Guard。或者，如果必须使用 HAXM，则可以使用未加入域的计算机。

其他故障排除提示

以下建议通常有助于诊断 Android Emulator 问题。

从命令行启动仿真器

如果仿真器尚未运行，则可以从命令行(而不是从 Visual Studio 中)启动它以查看其输出。通常，Android Emulator AVD 图像存储在以下位置(将“用户名”替换为 Windows 用户名)：

C:\Users\username\.android\avd

可通过传入 AVD 的文件夹名称从此位置启动带有 AVD 图像的仿真器。例如，此命令将启动名为“Pixel_API_27”的 AVD：

```
"C:\Program Files (x86)\Android\android-sdk\emulator\emulator.exe" -partition-size 512 -no-boot-anim -verbose -feature WindowsHypervisorPlatform -avd Pixel_API_27 -prop monodroid.avdname=Pixel_API_27
```

此示例假定将 Android SDK 安装在默认位置 C:\Program Files (x86)\Android\android-sdk；如果不是，请在计算机上修改上述 Android SDK 所在的路径。

运行此命令时，它将在仿真器启动时生成许多行输出。具体而言，如果硬件加速已启用并且正常运行（在此示例中，使用 HAXM 进行硬件加速），将出现如下所示的行：

```
emulator: CPU Acceleration: working
emulator: CPU Acceleration status: HAXM version 6.2.1 (4) is installed and usable.
```

查看设备管理器日志

通常可通过查看设备管理器日志来诊断仿真器问题。这些日志将写入以下位置：

C:\Users\username\AppData\Roaming\XamarinDeviceManager

可使用文本编辑器（如记事本）查看每个 DeviceManager.log 文件。以下示例日志项目表示，在计算机上未找到 HAXM：

```
Component Intel x86 Emulator Accelerator (HAXM installer) r6.2.1 [Extra: (Intel Corporation)] not present on the system
```

在 macOS 上部署时出现的问题

部署应用时，仿真器可能会显示一些错误消息。下面介绍最常见的错误和解决方案。

部署错误

若出现有关无法在仿真器上安装 APK 或无法运行 Android Debug Bridge (adb) 的错误消息，请验证 Android SDK 能否连接到仿真器。要验证连接情况，请使用以下步骤：

1. 通过“Android Device Manager”启动仿真器（选择虚拟设备并单击“启动”）。
2. 打开命令提示符，转到 adb 的安装文件夹。如果将 Android SDK 安装在其默认位置，则“adb”位于“/Library/Developer/Xamarin/android-sdk-macosx/platform-tools/adb”；如果不是，请在计算机上修改 Android SDK 所在的路径。
3. 键入以下命令：

```
adb devices
```

4. 如果可以通过 Android SDK 访问仿真器，那么仿真器应该就显示在附加设备列表中。例如：

```
List of devices attached
emulator-5554    device
```

5. 如果仿真器不在此列表中，请启动“Android SDK 管理器”，应用所有更新，再尝试重启仿真器。

MMIO 访问错误

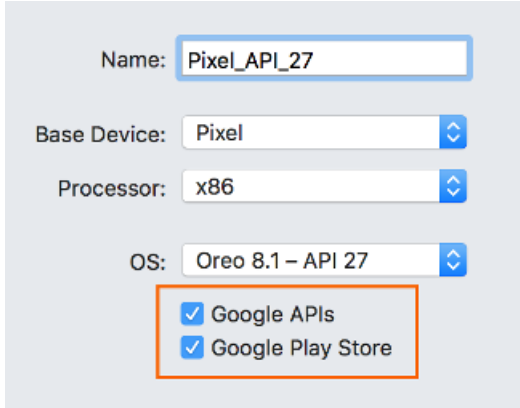
若显示“发生 MMIO 访问错误”，请重启仿真器。

缺少 Google Play Services

如果在仿真器中运行的虚拟设备未安装 Google Play Services 或 Google Play 商店，则在未安装这些软件包时创建虚拟设备通常会出现此情况。创建虚拟设备时(请参阅[使用 Android Device Manager 管理虚拟设备](#))，请务必选择以下一项或两项：

- **Google API** – 在虚拟设备中包含 Google Play Services。
- **Google Play 商店** – 在虚拟设备中包含 Google Play 商店。

例如，此虚拟设备将包含 Google Play Services 和 Google Play 商店：



NOTE

Google Play 商店图片仅适用于某些基本设备类型，例如 Pixel、Pixel 2、Nexus 5 和 Nexus 5X。

性能问题

性能问题通常由以下某个问题引起：

- 仿真器在没有硬件加速的情况下运行。
- 在仿真器中运行的虚拟设备未使用基于 x86 的系统映像。

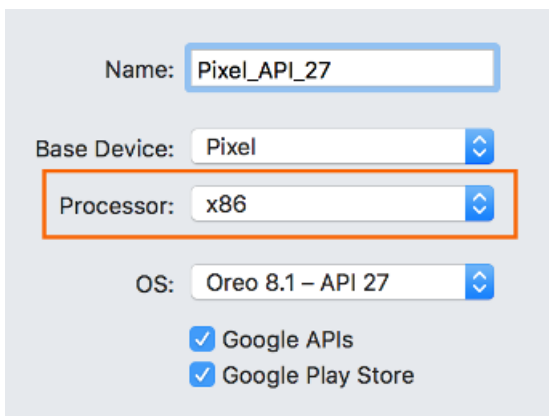
以下部分更详细地介绍了这些方案。

未启用硬件加速

如果未启用硬件加速，则在将应用部署到 Android Emulator 时，可能会弹出一个对话框，其中显示“设备将不加速运行”。如果不确定计算机上是否启用了硬件加速(或想知道哪种技术可以提供加速)，请参阅下面的[硬件加速问题](#)，了解验证和启用硬件加速的步骤。

加速已启用但仿真器运行速度过慢

导致此问题的常见原因是虚拟设备中未使用基于 x86 的映像。创建虚拟设备时(请参阅[使用 Android Device Manager 管理虚拟设备](#))，请确保选择基于 x86 的系统映像：



硬件加速问题

无论使用虚拟机监控程序框架还是 HAXM 进行仿真器的硬件加速，都可能遇到因安装不当或 macOS 版本过期引起的问题。以下部分可帮助你解决此问题。

虚拟机监控程序框架问题

如果在新版 Mac 上使用 macOS 10.10 或更高版本，则 Android Emulator 将自动使用虚拟机监控程序框架进行硬件加速。但是，某些旧版 Mac 或运行早于 10.10 版的 macOS 的 Mac 可能无法提供虚拟机监控程序框架支持。

要确定 Mac 是否支持虚拟机监控程序框架，请打开终端并输入以下命令：

```
sysctl kern.hv_support
```

若 Mac 支持虚拟机监控程序框架，则上述命令将返回以下结果：

```
kern.hv_support: 1
```

若 Mac 上没有虚拟机监控程序框架，则可以按照[使用 HAXM 加速](#)中的步骤来使用 HAXM 进行加速。

HAXM 问题

如果 Android Emulator 无法正常启动，这通常是 HAXM 问题所致。导致 HAXM 问题的原因通常包括：与其他虚拟化技术冲突、设置不正确或 HAXM 驱动器不是最新版本。按照[安装 HAXM](#)中介绍的步骤，尝试重新安装 HAXM 驱动程序。

其他故障排除提示

以下建议通常有助于诊断 Android Emulator 问题。

从命令行启动仿真器

如果仿真器尚未运行，则可以从命令行（而不是从 Visual Studio for Mac 中）启动它以查看其输出。通常，Android Emulator AVD 图像存储在以下位置：

```
./android/avd
```

可通过传入 AVD 的文件夹名称从此位置启动带有 AVD 图像的仿真器。例如，此命令将启动名为“Pixel_2_API_28”的 AVD：

```
~/Library/Developer/Xamarin/android-sdk-macosx/emulator/emulator -partition-size 512 -no-boot-anim -verbose -feature WindowsHypervisorPlatform -avd Pixel_2_API_28 -prop monodroid.avdname=Pixel_2_API_28
```

如果将 Android SDK 安装在其默认位置，则仿真器位于“~/Library/Developer/Xamarin/android-sdk-macosx/emulator”目录；如果不

是，请在 Mac 上修改 Android SDK 所在的路径。

运行此命令时，它将在仿真器启动时生成许多行输出。具体而言，如果硬件加速已启用并且正常运行（在此示例中，使用虚拟机监控程序框架进行硬件加速），将出现如下示例的行：

```
emulator: CPU Acceleration: working  
emulator: CPU Acceleration status: Hypervisor.Framework OS X Version 10.13
```

查看设备管理器日志

通常可通过查看设备管理器日志来诊断仿真器问题。这些日志将写入以下位置：

/Library/Logs/XamarinDeviceManager

可查看每个“Android Devices.log”文件（通过双击文件在控制台应用中将其打开）。以下示例日志项目指示未找到 HAXM：

```
Component Intel x86 Emulator Accelerator (HAXM installer) r6.2.1 [Extra: (Intel Corporation)] not present on  
the system
```


设置设备进行开发

2018/11/2 • [Edit Online](#)

本文介绍如何设置 Android 设备并将其连接到计算机，使设备可用于运行和调试 Xamarin.Android 应用程序。

目前为止，你可能已经看到了新应用程序在 Android 仿真程序上运行，并希望看到它在闪亮的 Android 设备上运行。以下步骤涉及如何将设备连接到计算机以进行调试：

1. 在设备上启用调试 - 默认情况下，无法在 Android 设备上调试应用程序。
2. 安装 USB 驱动程序 - 对于 OS X 计算机，无需此步骤。Windows 计算机可能需要安装 USB 驱动程序。
3. 将设备连接到计算机 - 通过 USB 或 WiFi 将设备连接到计算机，这是最后一步。

上述每个步骤将在以下部分中详细介绍。

在设备上启用调试

可使用 Android 设备测试 Android 应用程序。但是，必须先正确配置设备，才能开始调试。所涉及的步骤略有不同，具体取决于设备上运行的 Android 版本。

Android 4.0 到 Android 4.1

针对 Android 4.0.x 到 Android 4.1.x，请按照以下步骤启用调试：

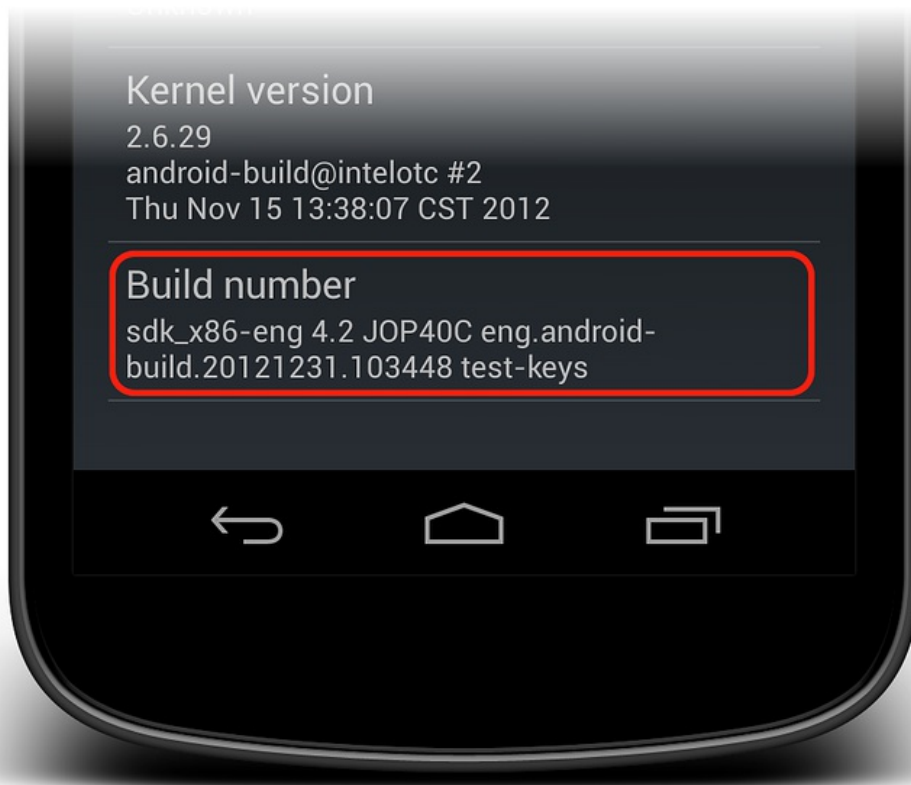
1. 转到“设置”屏幕。
2. 选择“开发人员选项”。
3. 选中“USB 调试”选项。

此屏幕截图显示运行 Android 4.0.3 的设备上的“开发人员选项”屏幕：

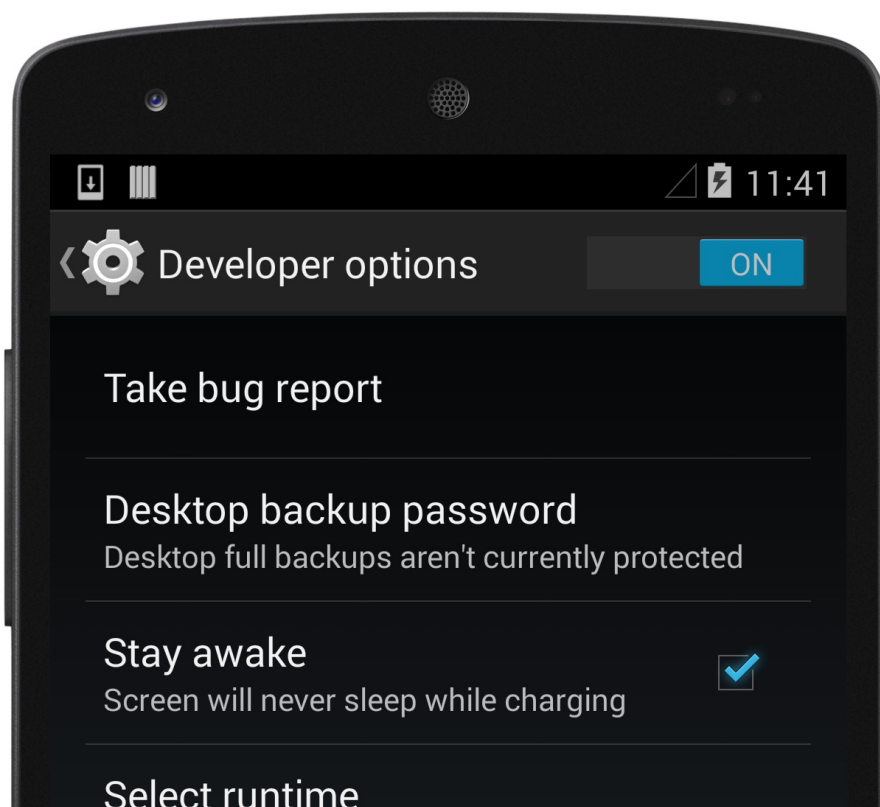


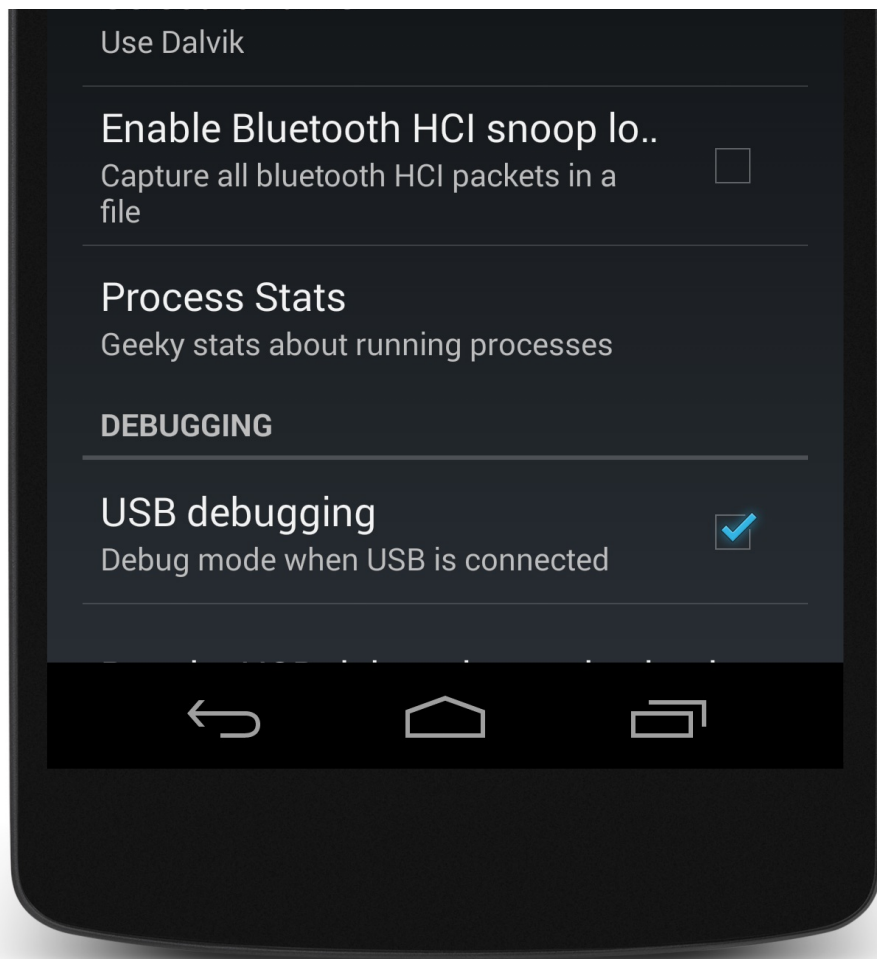
Android 4.2 及更高版本

从 Android 4.2 及更高版本开始，默认情况下，“开发人员选项”是隐藏的。若要启用，请转到“设置”>“关于手机”，然后点击七次“内部版本号”项以显示“开发人员选项”选项卡：



“开发人员选项”选项卡可用后，请在“设置”>“系统”下将其打开，以显示开发人员设置：





从此处可启用开发人员选项，例如 USB 调试和保持唤醒状态模式。

安装 USB 驱动程序

对于 OS X，无需此步骤。只需通过 USB 线将设备连接到 Mac。

可能需要先安装一些额外的驱动程序，Windows 计算机才能识别通过 USB 连接的 Android 设备。

NOTE

这些是设置 Google Nexus 设备的步骤，将作为参考提供。适用于特定设备的步骤可能有所不同，但遵循的模式是类似的。如果遇到问题，请在 Internet 上搜索你的设备。

在 **[Android SDK install path]\tools** 目录中，运行 **android.bat** 应用程序。默认情况下，Xamarin.Android 安装程序会将 Android SDK 放置在 Windows 计算机上的以下位置中：

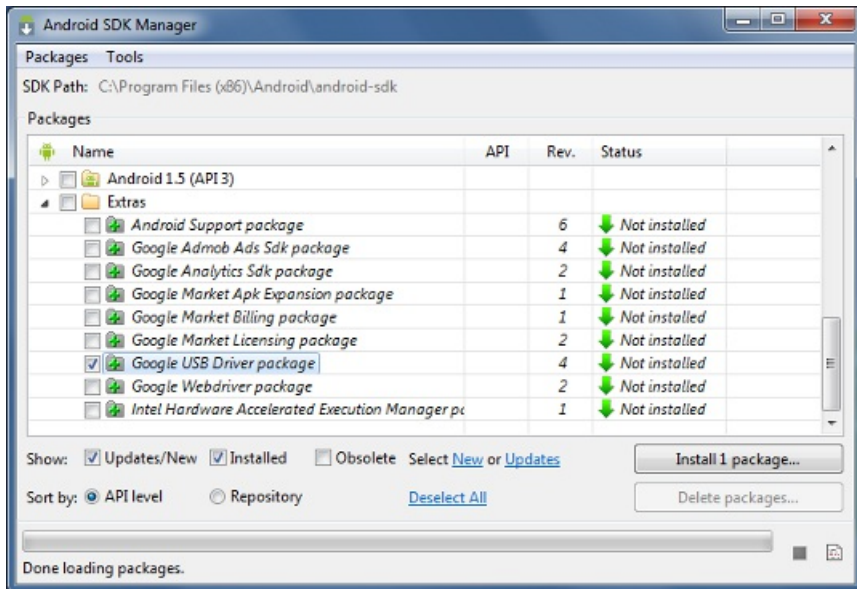
```
C:\Users\[username]\AppData\Local\Android\android-sdk
```

下载 USB 驱动程序

Google Nexus 设备 (Galaxy Nexus 除外) 需要 Google USB 驱动程序。Galaxy Nexus 的驱动程序由 [Samsung 分发](#)。所有其他 Android 设备应使用 [来自其各自制造商的 USB 驱动程序](#)。

通过启动 Android SDK 管理器并展开“附加程序”文件夹，安装 **Google USB 驱动程序包**，如下面的屏幕截图所示

示：



选中“Google USB 驱动程序”框，然后单击“安装”按钮。驱动程序文件将下载到以下位置：

```
[Android SDK install path]\extras\google\usb_driver
```

Xamarin.Android 安装的默认路径为：

```
C:\Users\[username]\AppData\Local\Android\android-sdk\extras\google\usb_driver
```

安装 USB 驱动程序

USB 驱动程序下载完成后，请将其安装。在 Windows 7 上安装驱动程序：

1. 通过 USB 线，将设备连接到计算机。
2. 从桌面或 Windows 资源管理器，右键单击“计算机”，然后选择“管理”。
3. 在左窗格中，选择“设备”。
4. 在右窗格中，找到并展开“其它设备”。
5. 右键单击设备名，并选择“更新驱动程序软件”。这将启动硬件更新向导。
6. 选择“浏览计算机以查找驱动程序软件”，然后单击“下一步”。
7. 单击“浏览”，找到 USB 驱动程序文件夹 (Google USB 驱动程序位于 **[Android SDK install path]\extras\google\usb_driver**)。
8. 单击“下一步”安装驱动程序。

在 Windows 8 中安装未经验证的驱动程序

若要在 Windows 8 中安装未经验证的驱动程序，可能需要执行额外步骤。以下步骤介绍如何安装 Galaxy Nexus 的驱动程序：

1. 访问 **Windows 8 高级启动选项** - 此步骤包括重启计算机以访问高级启动选项。通过使用以下命令，启动命令行提示符和重启计算机：

```
shutdown.exe /r /o
```

2. 连接设备 - 将设备连接到计算机

3. 启动设备管理器 - 运行 `devmgmt.msc`; 应可看到你的设备已列出, 其上有一个黄色的三角形。

4. 安装设备驱动程序 - 安装设备驱动程序, 如上所述。

将设备连接到计算机

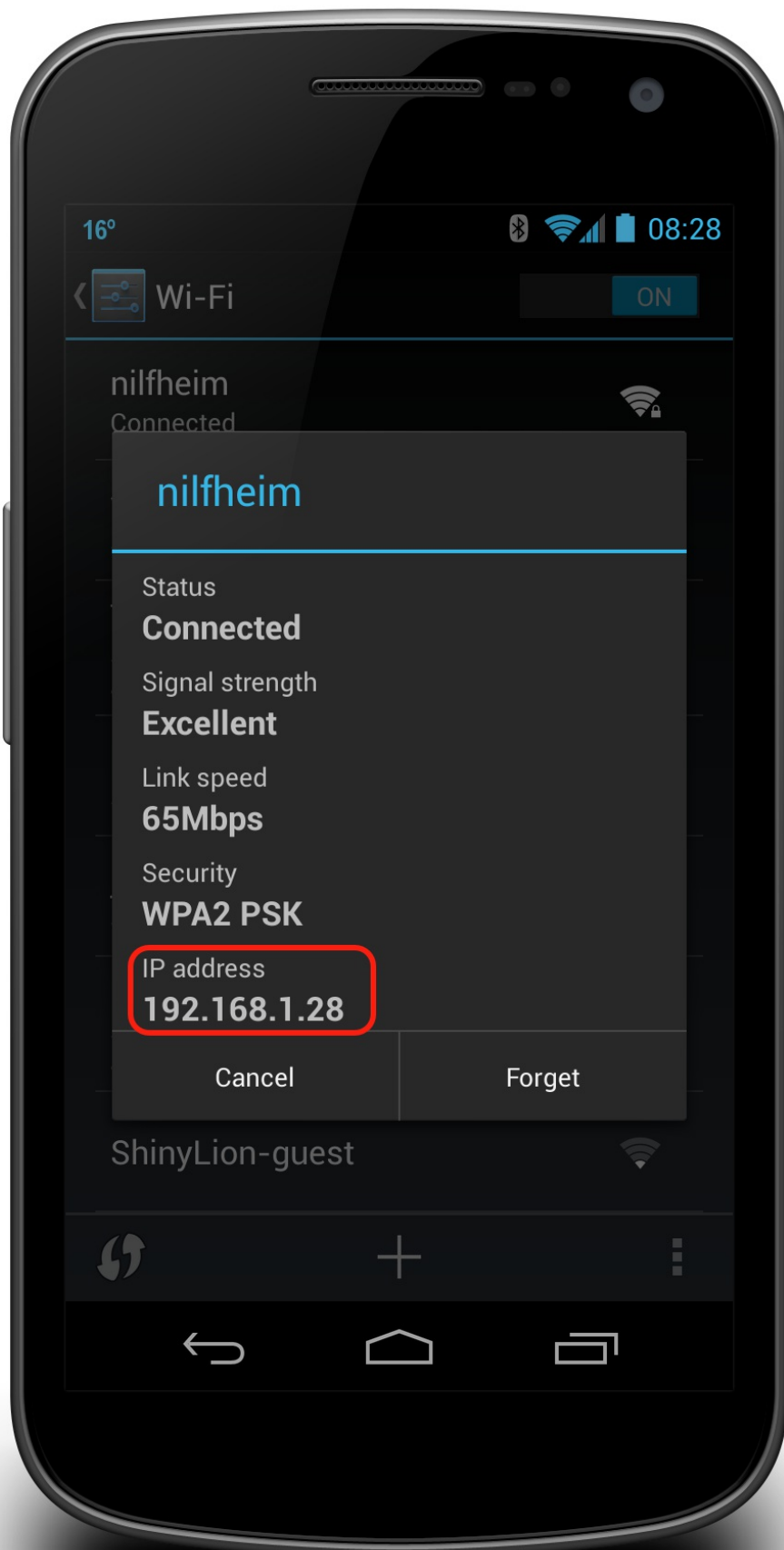
最后一步是将设备连接到计算机。有两种方法可以实现此目的:

- **USB 线** - 这是最简单、最常见的方式。只需将 USB 线插入设备, 然后插入计算机。
- **WiFi** - 可以通过 WiFi 将 Android 设备连接到计算机, 无需使用 USB 线。使用此技术需要一些操作, 但在没有 USB 线或没有用于设备的足够长的 USB 线的情况, 此技术会有帮助。下一部分中将介绍如何通过 WiFi 连接。

通过 WiFi 连接

默认情况下, [Android Debug Bridge \(ADB\)](#) 配置为通过 USB 与 Android 设备进行通信。可将其重新配置为使用 TCP/IP, 而不是使用 USB。为此, 设备和计算机必须处于同一 WiFi 网络上。若要通过 WiFi 设置调试环境, 请从命令行执行以下步骤:

1. 确定 Android 设备的 IP 地址。查找 IP 地址的一种方法是在“设置”>“Wi-Fi”下查看, 然后点击设备所连接的 WiFi 网络。此时会弹出设置屏幕, 显示网络连接的相关信息, 类似下面的屏幕截图中所示:



16°

08:28

Wi-Fi

ON

nilfheim
Connected



nilfheim

Status

Connected

Signal strength

Excellent

Link speed

65Mbps

Security

WPA2 PSK

IP address

192.168.1.28

Cancel

Forget

ShinyLion-guest



某些版本的 Android 上不会列出 IP 地址，但可在“设置”>“关于手机”>“状态”下找到 IP 地址。

2. 通过 USB 将 Android 设备连接到计算机。
3. 接下来，重启 ADB，以便可在端口 5555 上使用 TCP。在命令提示符处，键入以下命令：

```
adb tcpip 5555
```

发出此命令后，计算机不能侦听通过 USB 连接的设备。

4. 断开将设备连接到计算机的 USB 线连接。
5. 配置 ADB，使其可在上面步骤 1 中指定的端口上连接 Android 设备：

```
adb connect 192.168.1.28:5555
```

此命令完成后，Android 设备即可通过 WiFi 连接到计算机。

通过 WiFi 完成调试后，可通过以下命令将 ADB 重置回 USB 模式：

```
adb usb
```

可要求 ADB 列出连接到计算机的设备。无论设备通过何种方式连接，可在命令提示符发出以下命令，查看连接的设备：

```
adb devices
```

总结

本文介绍如何通过设备上启用调试，配置用于开发的 Android 设备。还介绍了如何使用 USB 或 WiFi 将设备连接到计算机。

相关链接

- [Android Debug Bridge](#)
- [使用硬件设备](#)
- [Samsung 驱动程序下载](#)
- [OEM USB 驱动程序](#)
- [Google USB 驱动程序](#)
- [XDA 开发人员:已解决 Windows 8 - ADB/快速启动驱动程序问题](#)

Microsoft 分发的 OpenJDK 预览版

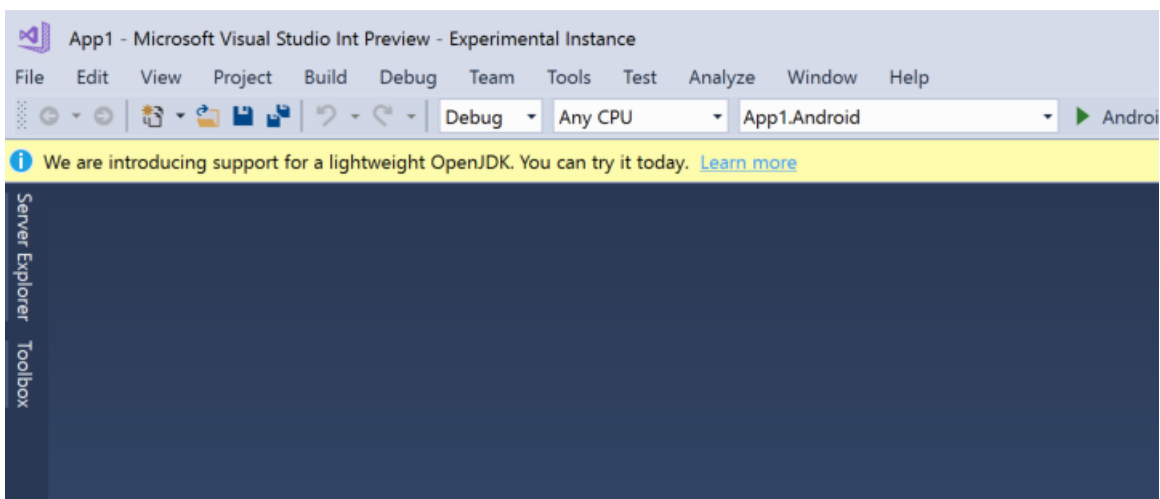
2018/8/13 • [Edit Online](#)

本指南介绍了如何切换到 Microsoft 分发的 OpenJDK 预览版。此分发适用于移动开发。



概述

自 Visual Studio 15.9 和 Visual Studio for Mac 7.7 起, Visual Studio Tools for Xamarin 将从 Oracle JDK 迁移到仅用于 Android 开发的 OpenJDK 轻型版本:



此迁移带来以下优势:

- 始终拥有适用于 Android 开发的 OpenJDK 版本。
- 下载 JDK 9 或 10 不会影响开发体验。
- 显著减少了下载大小和占用空间。
- 第三方服务器和安装程序不再出现问题。

若要更快地迁移到改进后体验, 可以使用 Microsoft 分发的 Mobile OpenJDK 内部版本在 Windows 和 Mac 上进行测试。下面介绍了具体设置过程, 随时可以还原回 Oracle JDK。

下载

首先, 下载适用于系统的内部版本:

- **Mac** – <https://dl.xamarin.com/OpenJDK/mac/microsoft-dist-openjdk-1.8.0.9.zip>
- **Windows x86** – <https://dl.xamarin.com/OpenJDK/win32/microsoft-dist-openjdk-1.8.0.9.zip>
- **Windows x64** – <https://dl.xamarin.com/OpenJDK/win64/microsoft-dist-openjdk-1.8.0.9.zip>

配置

解压缩到正确位置:

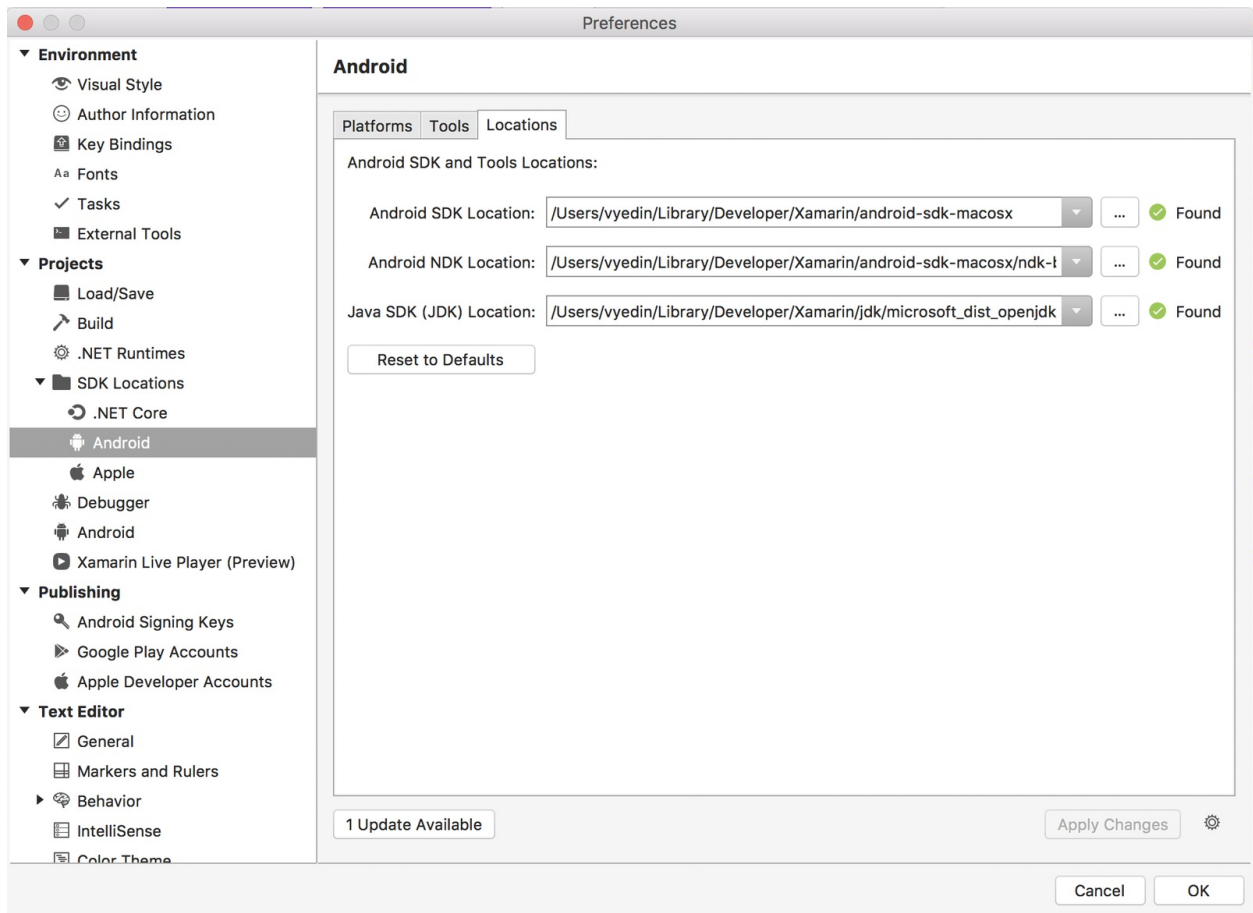
- **Mac** – `$HOME/Library/Developer/Xamarin/jdk/microsoft_dist_openjdk_1.8.0.9`
- **Windows** – `C:\Program Files\Android\jdk\microsoft_dist_openjdk_1.8.0.9`

IMPORTANT

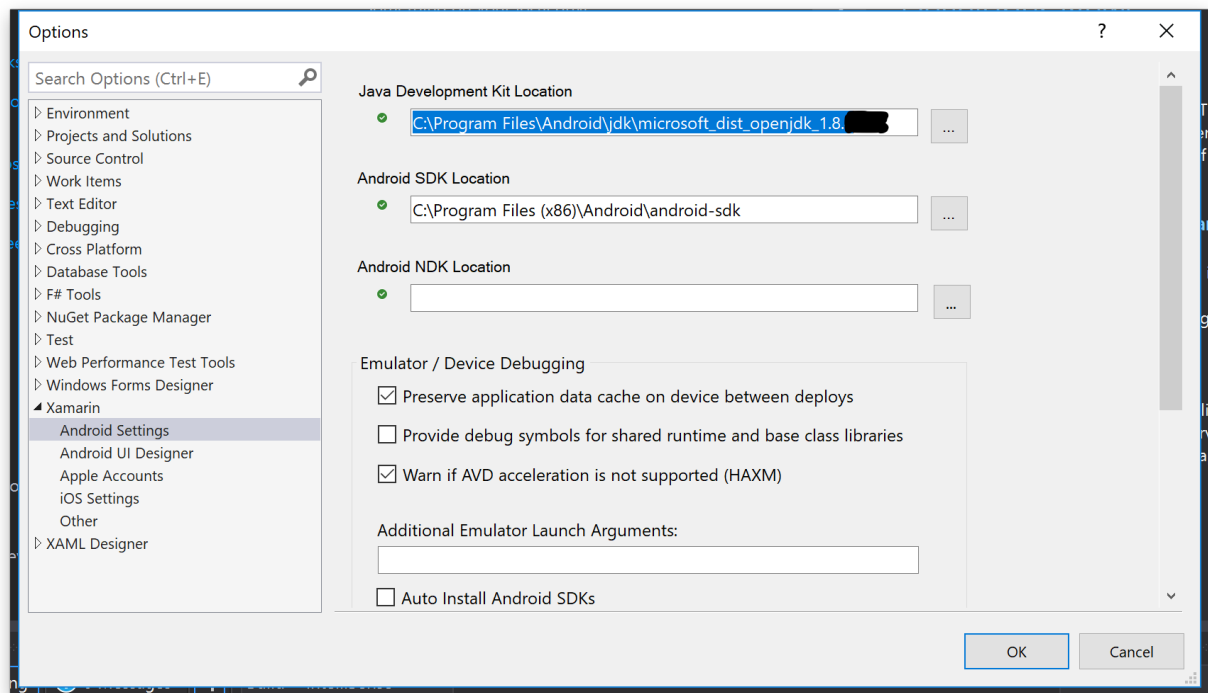
此示例使用内部版本 1.8.0.9;但你可以下载更高版本。

将 IDE 定目标到新 JDK:

- **Mac** – 依次单击“工具 > SDK 管理器 > 位置”，并将“Java SDK (JDK)位置”更改为 OpenJDK 安装的完整路径。在以下示例中，此路径设置为“`$HOME/Library/Developer/Xamarin/jdk/microsoft_dist_openjdk_1.8.0.9`”。



- **Windows** – 依次单击“工具 > 选项 > Xamarin > Android 设置”，并将“Java 开发工具包位置”设置为 OpenJDK 安装的完整路径。在以下示例中，此路径设置为“`C:\Program Files\Android\jdk\microsoft_dist_openjdk_1.8.0.9`”:



还原

若要还原回 Oracle JDK，请将 Java SDK 位置更改为以前使用的 Oracle JDK 路径，并重新生成解决方案。在 Mac 上，可以通过单击“重置为默认值”来还原 Oracle JDK 路径。

若对 Microsoft 分发的 Mobile OpenJDK 有任何疑问，请使用 IDE 中的反馈工具来报告问题，以便我们能够快速跟踪和修复问题。

已知问题和计划修复日期

`JAVA_HOME` 环境变量可能无法正确导出到 SDK 和设备管理器。作为解决方法，可以将此环境变量设置为计算机上的 OpenJDK 位置。15.9 预览版已修复此问题。

总结

本文介绍了如何将 IDE 配置为使用 Microsoft 分发的 Mobile OpenJDK 预览版(计划于 2018 年早些时候发布稳定版本)。

了解 Android

2018/10/26 • [Edit Online](#)

在此由两部分构成的指南中，你将使用 Visual Studio for Mac 或 Visual Studio 生成第一个 Xamarin.Android 应用程序，并进一步了解使用 Xamarin 进行 Android 应用程序开发的基础知识。在此过程中，会介绍生成和部署 Xamarin.Android 应用程序所需的工具、概念和步骤。

第 1 部分:快速入门

在本指南的第一部分，用户将创建一个应用程序，该应用程序可将用户输入的字母数字电话号码转换为数字电话号码，然后呼叫该号码。

第 2 部分:深度分析

在本文档的第二部分，用户将回顾生成的应用程序，并从根本上了解 Android 应用程序的工作原理。

相关链接

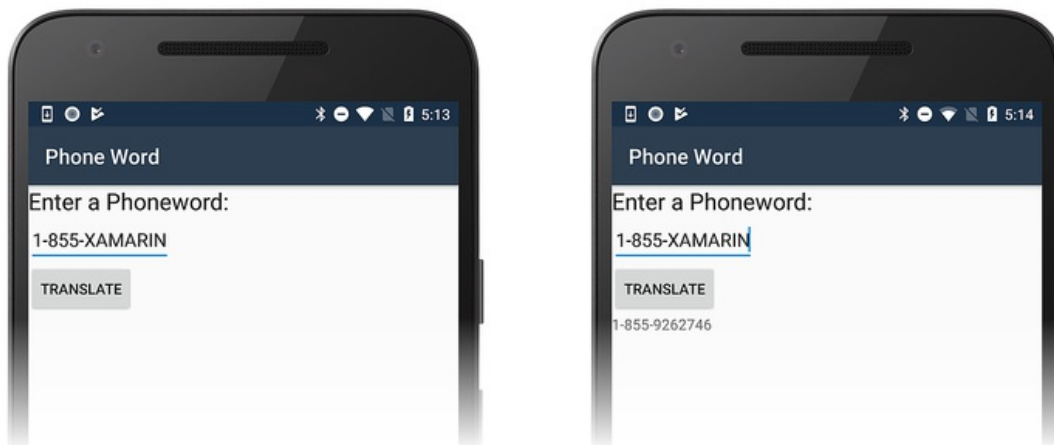
- [Android 入门](#)
- [在 Visual Studio 中进行调试](#)
- [Visual Studio for Mac 方案 - 调试](#)

Hello, Android: 快速入门

2018/11/2 • [Edit Online](#)

在这个两部分的指南中，你将使用 Visual Studio 生成第一个 Xamarin.Android 应用程序，并了解使用 Xamarin 进行 Android 应用程序开发的基础知识。

你将创建一个应用程序，它将字母数字电话号码(由用户输入)转换为数字电话号码，然后向用户显示此数字电话号码。最终应用程序如下所示：



Windows 要求

若要按照本演练进行操作，你需要以下内容：

- Windows 10。
- Visual Studio 2017 Community、Professional 或 Enterprise (版本 15.8 或更高版本)。

macOS 要求

若要按照本演练进行操作，你需要以下内容：

- Visual Studio for Mac 的最新版本。
- 运行 macOS High Sierra (10.13) 或更高版本的 Mac。

本演练假设最新版本的 Xamarin.Android 已在你选择的平台上安装并运行。有关安装 Xamarin.Android 的指南，请参阅 [Xamarin.Android 安装指南](#)。

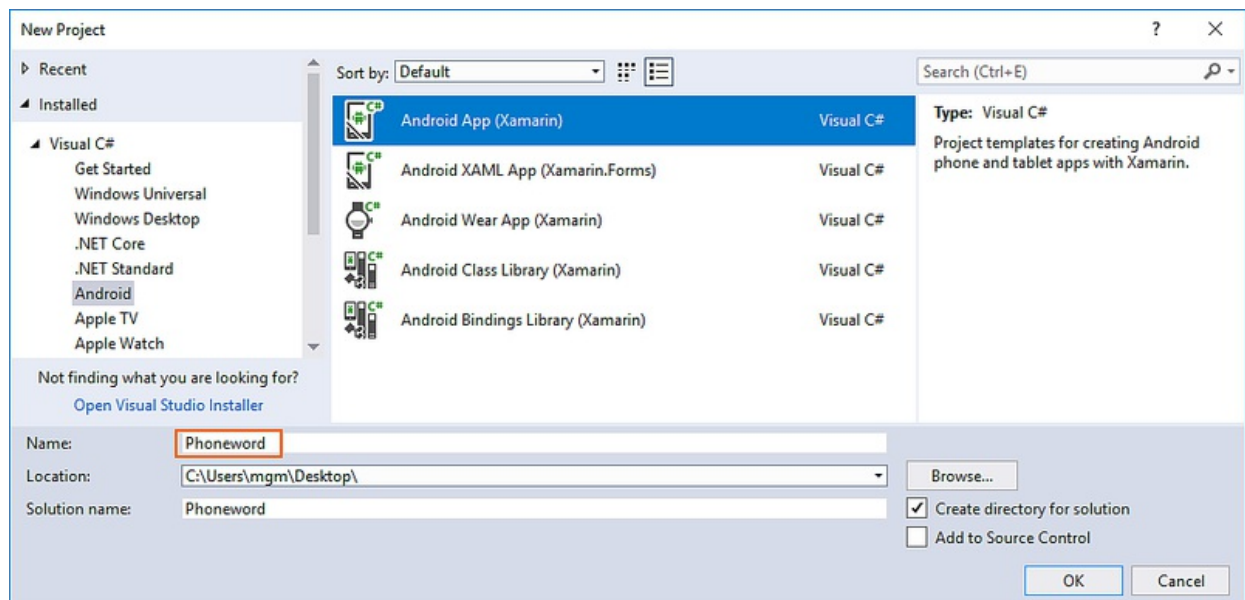
配置仿真器

如果使用的是 Android 仿真器，建议将仿真器配置为使用硬件加速。[通过硬件加速提高仿真器性能](#)中提供了有关配置硬件加速的说明。

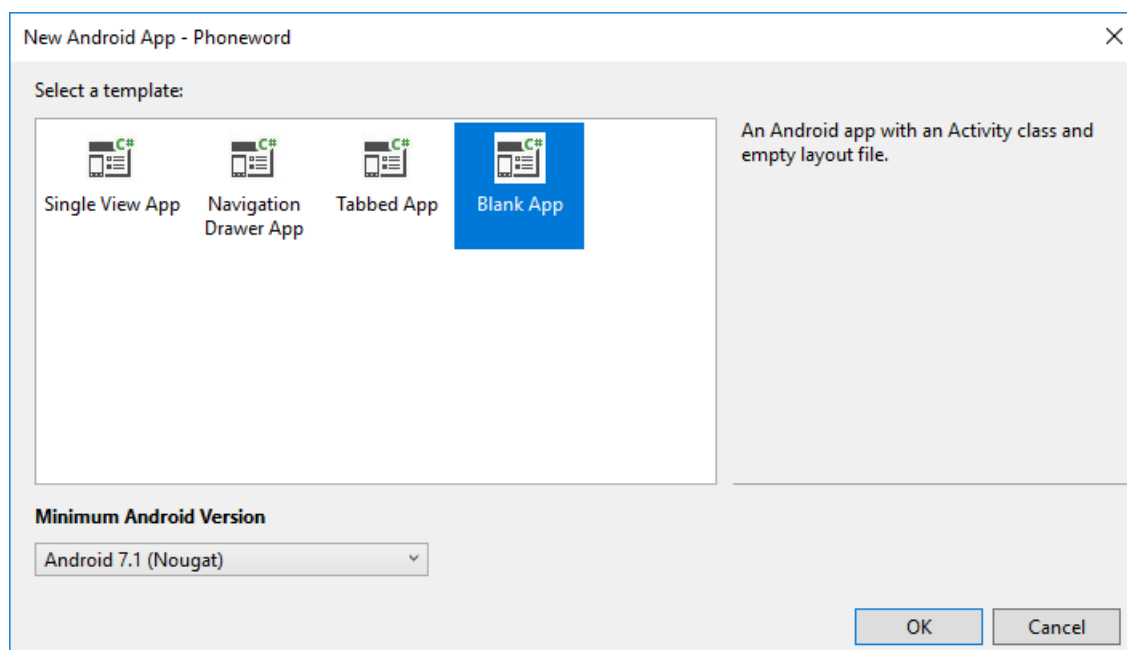
创建项目

启动 Visual Studio。单击“文件”>“新建”>“项目”以创建新项目。

在“新建项目”对话框中，单击“Android 应用”模板。将新项目命名为 `Phoneword`，然后单击“确定”：

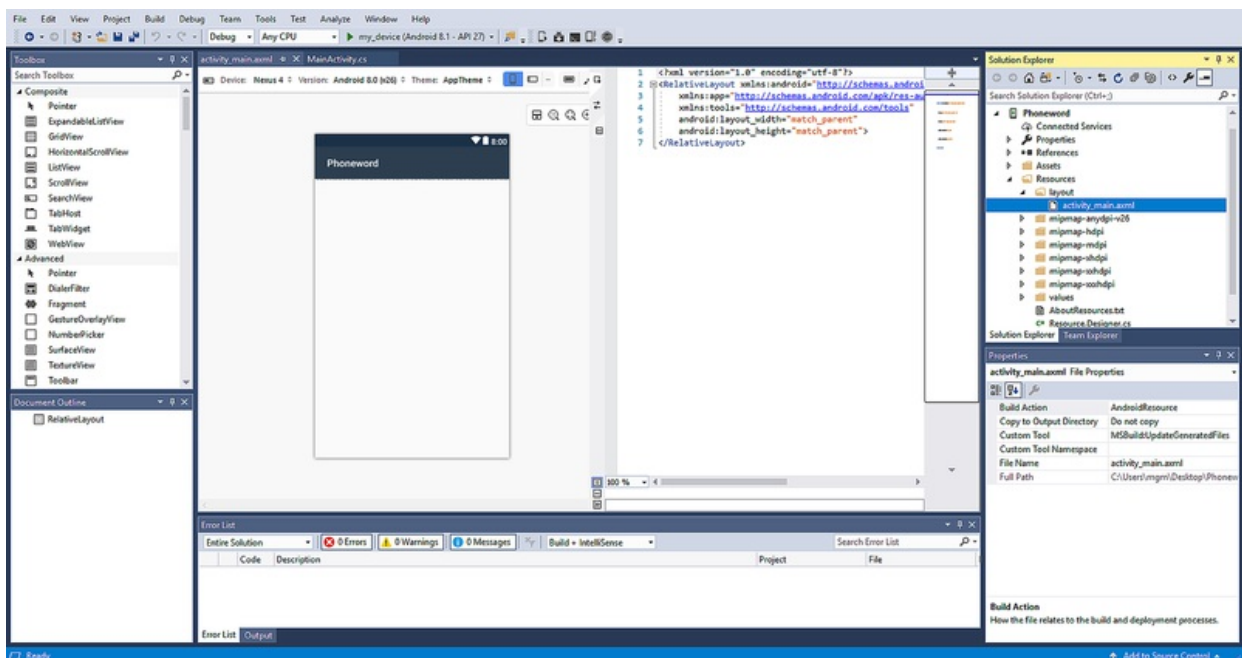


在“新 Android 应用”对话框中，依次单击“空白应用”和“确定”，以新建项目：

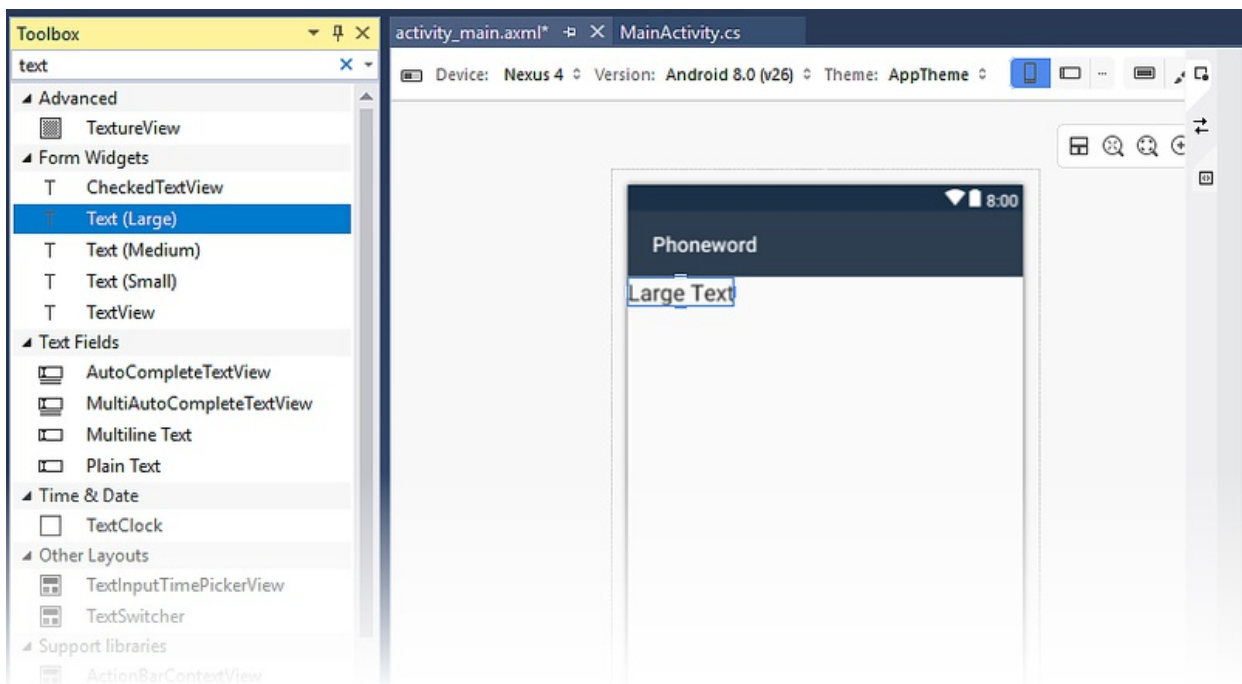


创建布局

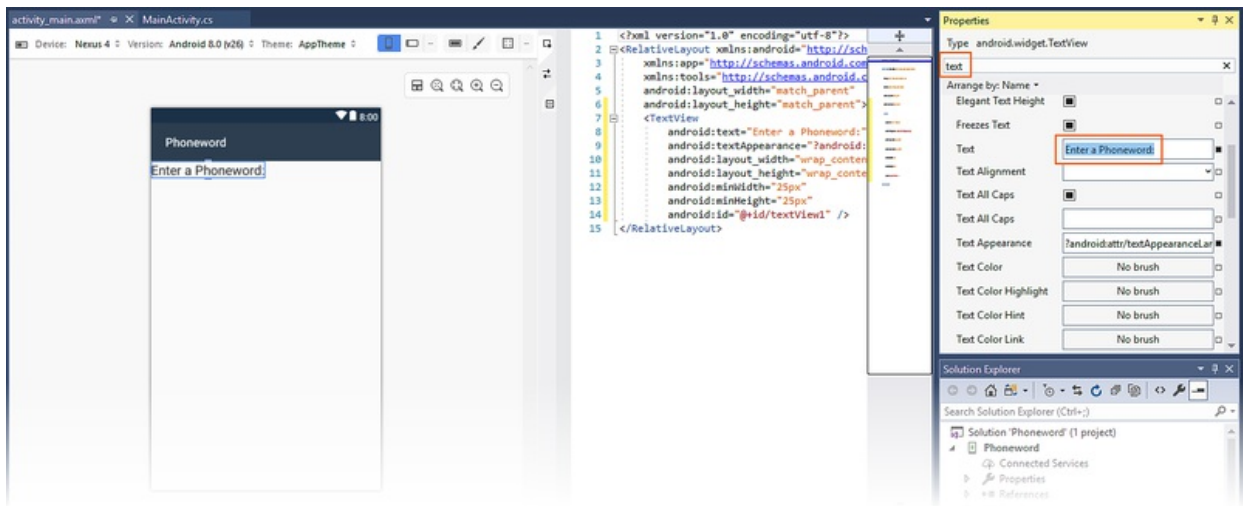
创建新项目之后，在“解决方案资源管理器”中展开 **Resources** 文件夹，然后展开 **layout** 文件夹。双击“activity_main.xml”，以在 Android Designer 中打开它。这是应用屏幕的布局文件：



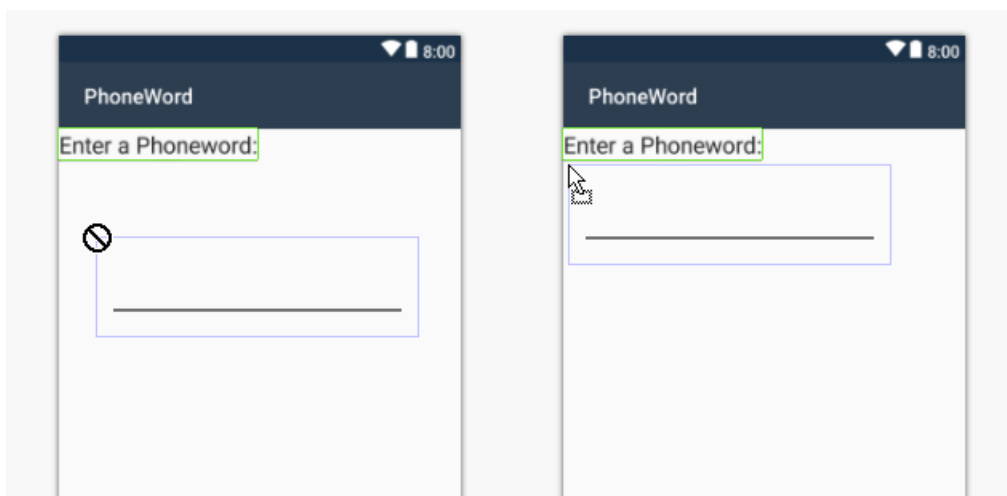
在“工具箱”(左侧区域)的搜索字段中输入 `text`，并将一个“文本(大)”小组件拖动至 Design Surface 上(中央区域)：



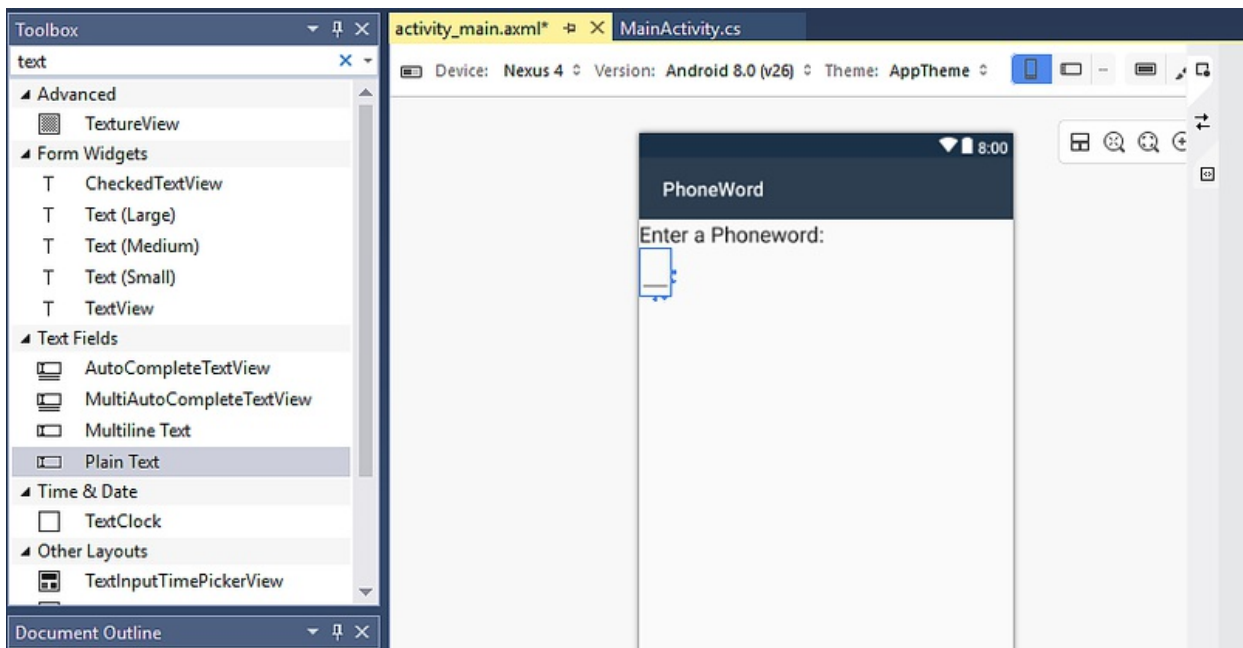
在 Design Surface 上选中“文本(大)”控件的情况下，可使用“属性”窗格将“文本(大)”小组件的 `Text` 属性更改为 `Enter a Phoneword:`：



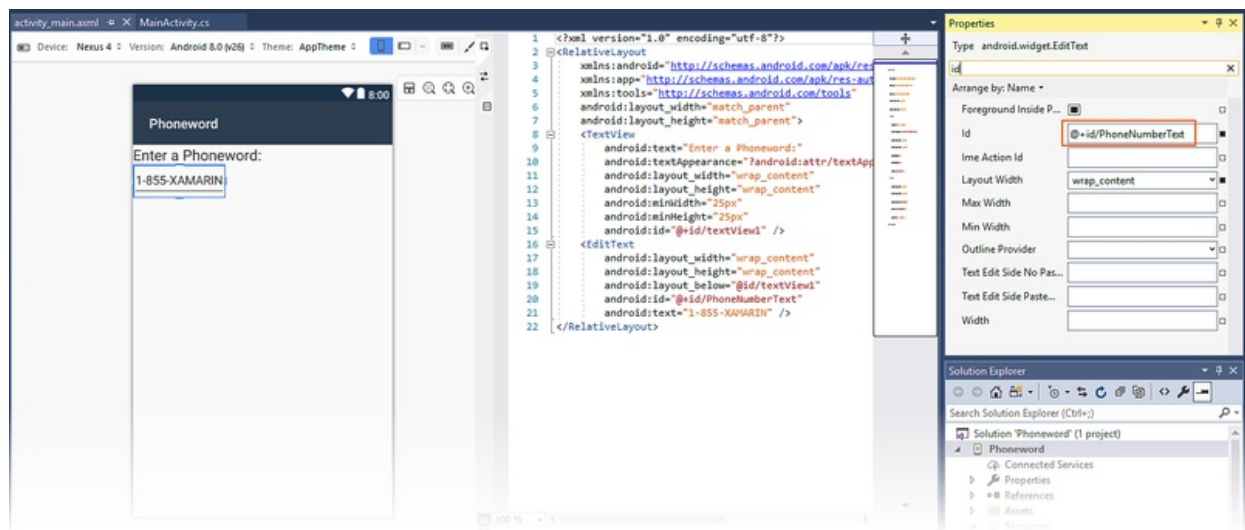
将一个“纯文本”小组件从“工具箱”拖动到设计图面上，并将它放置在“文本(大)”小组件下。直到将鼠标指针移动到布局中可接受小组件的位置，才会放置小组件。在下面的屏幕截图中，直到鼠标指针移动到前一个 `TextView` 的下方(如右图所示)，才能放置小组件(如左图所示)：



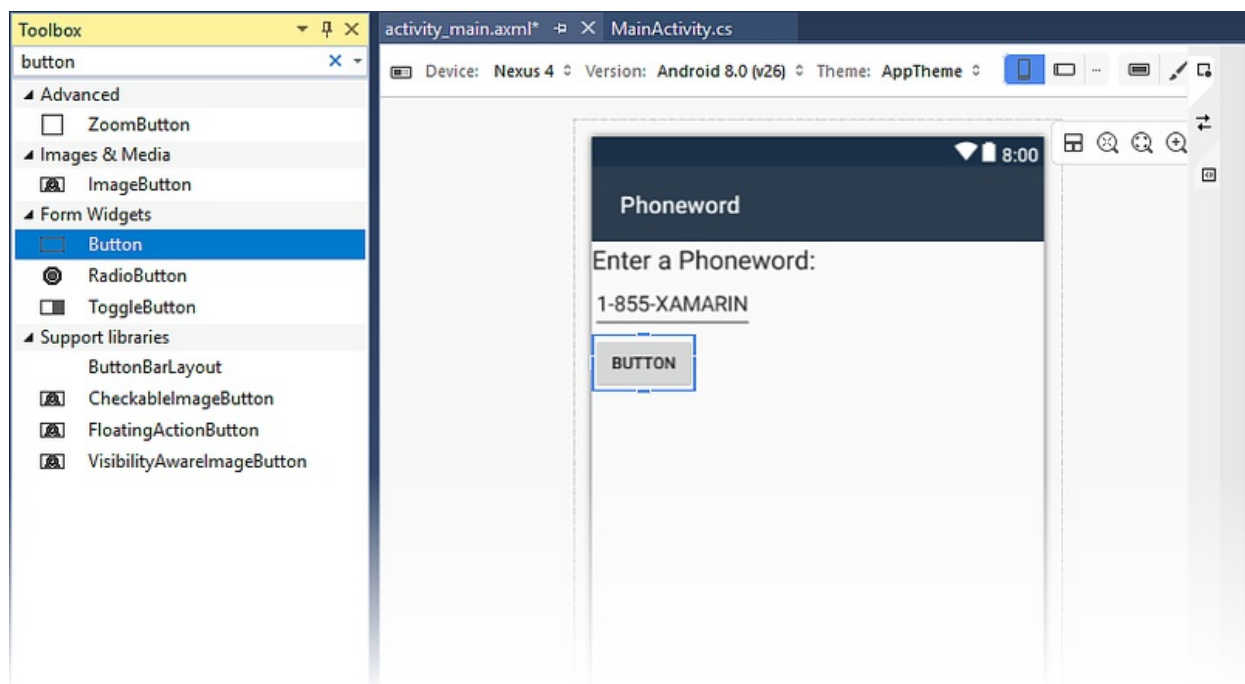
正确放置“纯文本”(`EditText`)小组件后，它将如下屏幕截图所示：



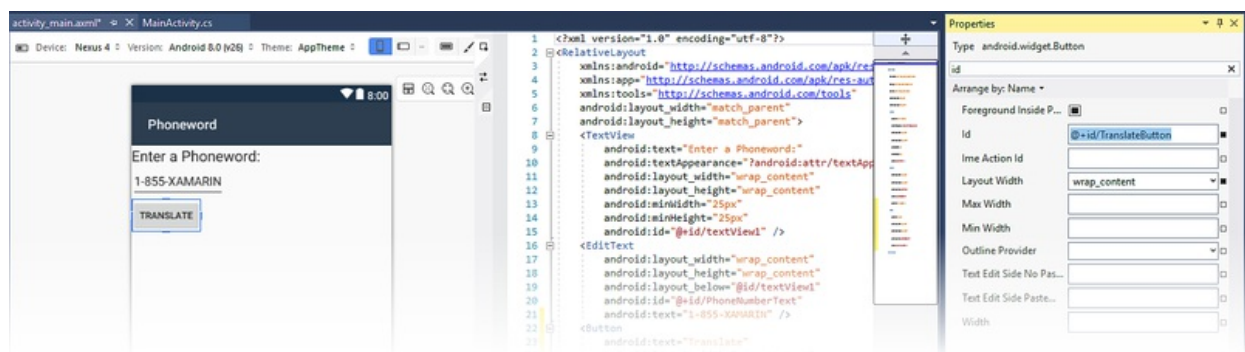
在 Design Surface 上选中“纯文本”小组件的情况下，可使用“属性”窗格将“纯文本”小部件的 `Id` 属性更改为 `@+id/PhoneNumberText`，并将 `Text` 属性更改为 `1-855-XAMARIN`：



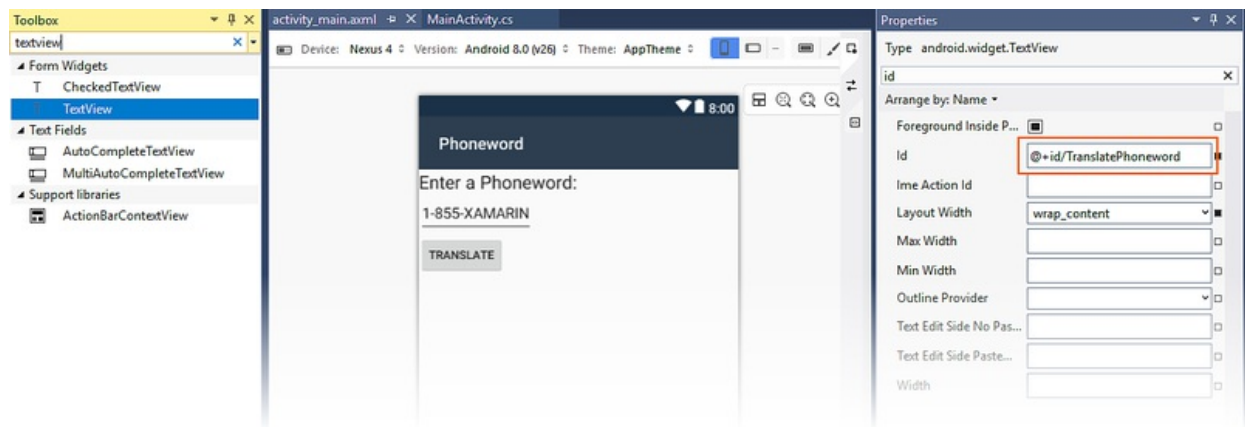
将一个“按钮”从“工具箱”拖动到设计图面上，并将它放置在“纯文本”小组件下方：



在 Design Surface 上选中“按钮”后，使用“属性”窗格将其 Text 属性更改为 Translate，将其 Id 属性更改为 @+id/TranslateButton：



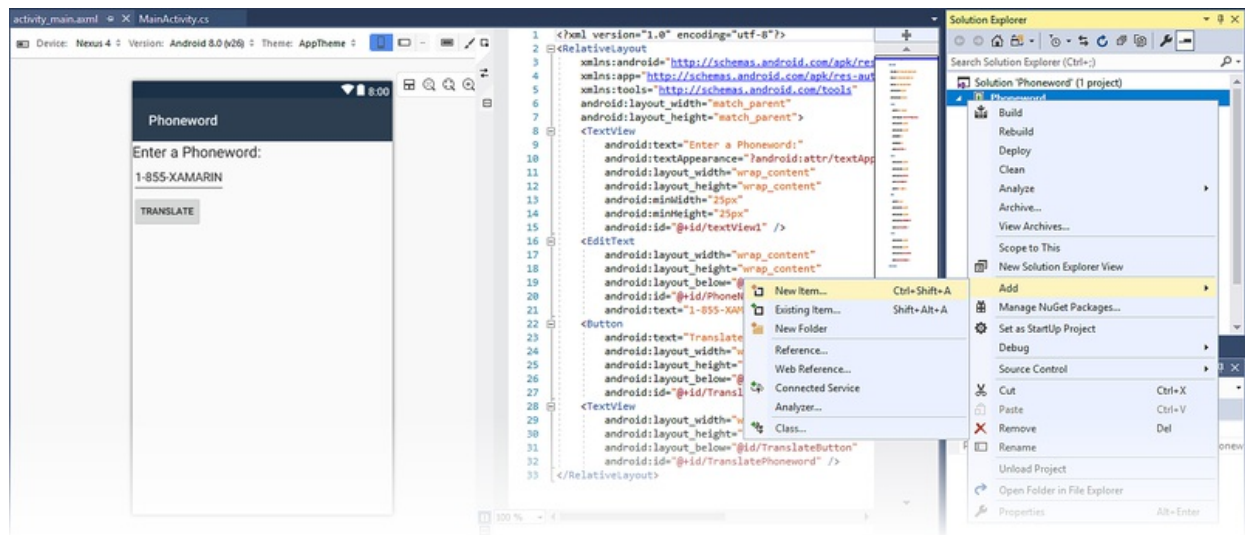
将一个“TextView”从“工具箱”拖动到 Design Surface 上，并将其置于“按钮”小组件下方。将“TextView”的 Text 属性更改为空字符串，并将其 Id 属性设置为 @+id/TranslatedPhoneword：



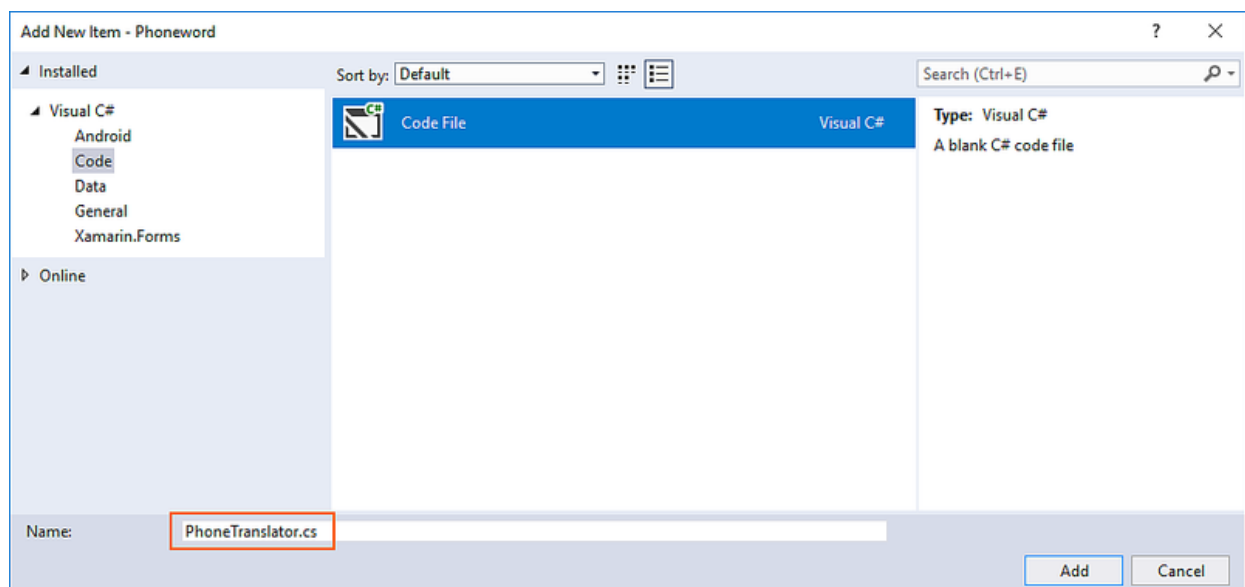
通过按 **CTRL+S** 来保存工作。

编写一些代码

下一步是添加一些代码，以将电话号码从字母数字转换为数字。通过在“解决方案资源管理器”窗格中右键单击“Phoneword”项目，然后选择“添加”>“新建项...”以向项目添加新文件，如下所示：



在“添加新项”对话框中，选择“Visual C#”>“代码”>“代码文件”，然后将新代码文件命名为“PhoneTranslator.cs”：



这将创建新的空 C# 类。在此文件中插入以下代码：

```

using System.Text;
using System;
namespace Core
{
    public static class PhonewordTranslator
    {
        public static string ToNumber(string raw)
        {
            if (string.IsNullOrEmpty(raw))
                return "";
            else
                raw = raw.ToUpperInvariant();

            var newNumber = new StringBuilder();
            foreach (var c in raw)
            {
                if ("0123456789".Contains(c))
                {
                    newNumber.Append(c);
                }
                else
                {
                    var result = TranslateToNumber(c);
                    if (result != null)
                        newNumber.Append(result);
                }
                // otherwise we've skipped a non-numeric char
            }
            return newNumber.ToString();
        }
        static bool Contains (this string keyString, char c)
        {
            return keyString.IndexOf(c) >= 0;
        }
        static int? TranslateToNumber(char c)
        {
            if ("ABC".Contains(c))
                return 2;
            else if ("DEF".Contains(c))
                return 3;
            else if ("GHI".Contains(c))
                return 4;
            else if ("JKL".Contains(c))
                return 5;
            else if ("MNO".Contains(c))
                return 6;
            else if ("PQRS".Contains(c))
                return 7;
            else if ("TUV".Contains(c))
                return 8;
            else if ("WXYZ".Contains(c))
                return 9;
            return null;
        }
    }
}

```

通过单击“文件”>“保存”(或按 **CTRL+S**)来保存对 **PhoneTranslator.cs** 文件进行的更改，然后关闭该文件。

关联用户界面

接下来，通过将支持代码插入到 `MainActivity` 类中来添加代码以关联用户界面。首先关联“Translate”按钮。在 `MainActivity` 类中找到 `OnCreate` 方法。接下来，在 `OnCreate` 内的 `base.OnCreate(savedInstanceState)` 和 `SetContentView(Resource.Layout.activity_main)` 调用下添加该按钮代码。首先，修改模板代码，使 `OnCreate` 方

法与以下内容相似：

```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Runtime;
using Android.Widget;

namespace Phoneword
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);

            // New code will go here
        }
    }
}
```

获取对通过 Android 设计器在布局文件中创建的控件的引用。在 `OnCreate` 方法中将以下代码添加到 `SetContentView` 调用后面：

```
// Get our UI controls from the loaded layout
EditText phoneNumberText = FindViewById<EditText>(Resource.Id.PhoneNumberText);
TextView translatedPhoneWord = FindViewById<TextView>(Resource.Id.TranslatedPhoneword);
Button translateButton = FindViewById<Button>(Resource.Id.TranslateButton);
```

添加对用户按“Translate”按钮进行响应的代码。将以下代码添加到 `OnCreate` 方法(在上一步中添加的行后)：

```
// Add code to translate number
translateButton.Click += (sender, e) =>
{
    // Translate user's alphanumeric phone number to numeric
    string translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (string.IsNullOrEmpty(translatedNumber))
    {
        translatedPhoneWord.Text = string.Empty;
    }
    else
    {
        translatedPhoneWord.Text = translatedNumber;
    }
};
```

通过选择“文件”>“全部保存”(或按 **CTRL-SHIFT-S**)来保存工作，然后通过选择“生成”>“重新生成解决方案”(或按 **CTRL-SHIFT-B**)来生成应用程序。

如果发生错误，则完成前面的步骤并更正任何错误，直到应用程序成功生成。如果收到生成错误(如“资源在当前上下文中不存在”)，请验证 **MainActivity.cs** 中的命名空间名称是否与项目名称 (`Phoneword`) 匹配，然后完全重新生成解决方案。如果仍收到生成错误，请验证已安装最新 Visual Studio 更新。

设置应用名称

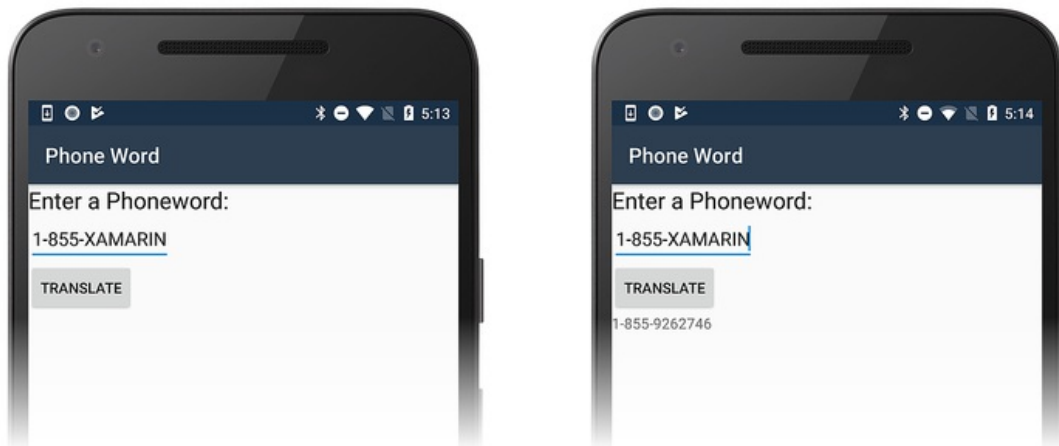
你现在应具有可正常工作的应用程序 – 该设置应用名称了。展开“值”文件夹(在“资源”文件夹中)并打开文

件“strings.xml”。将应用名称字符串更改为 `Phone Word`，如下所示：

```
<resources>
  <string name="app_name">Phone Word</string>
  <string name="action_settings">Settings</string>
</resources>
```

运行应用

通过在 Android 设备或仿真器上运行应用程序来测试该应用程序。单击“转换”按钮，将“1-855-XAMARIN”转换为电话号码：

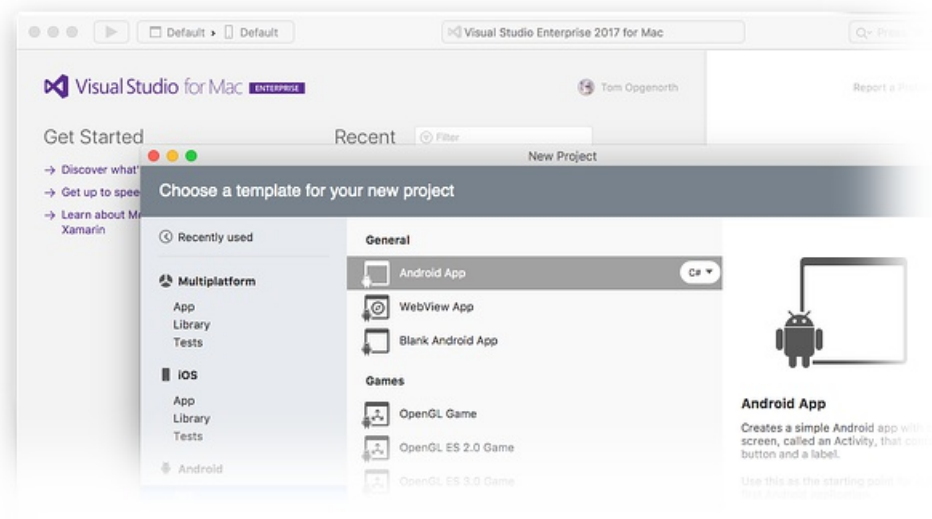


若要在 Android 设备上运行应用，请参阅[如何设置设备以进行开发](#)。

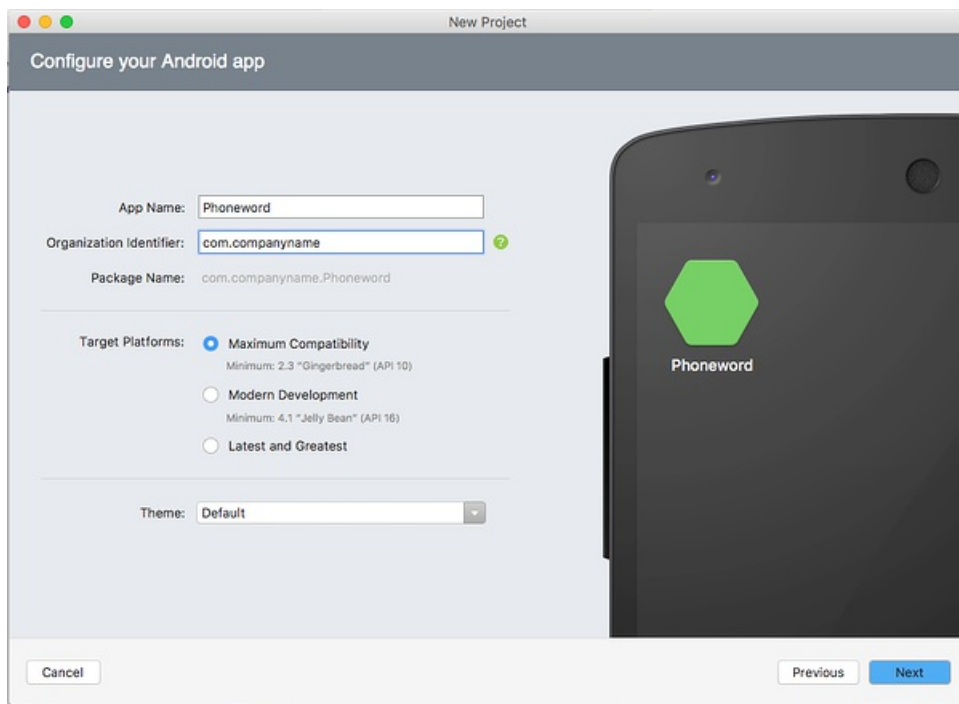
从“应用程序”文件夹或从 Spotlight 启动 Visual Studio for Mac。

单击“新建项目...”以创建新项目。

在“为新项目选择模板”对话框中，单击“Android”>“应用”，然后选择“Android 应用”模板。单击“下一步”。



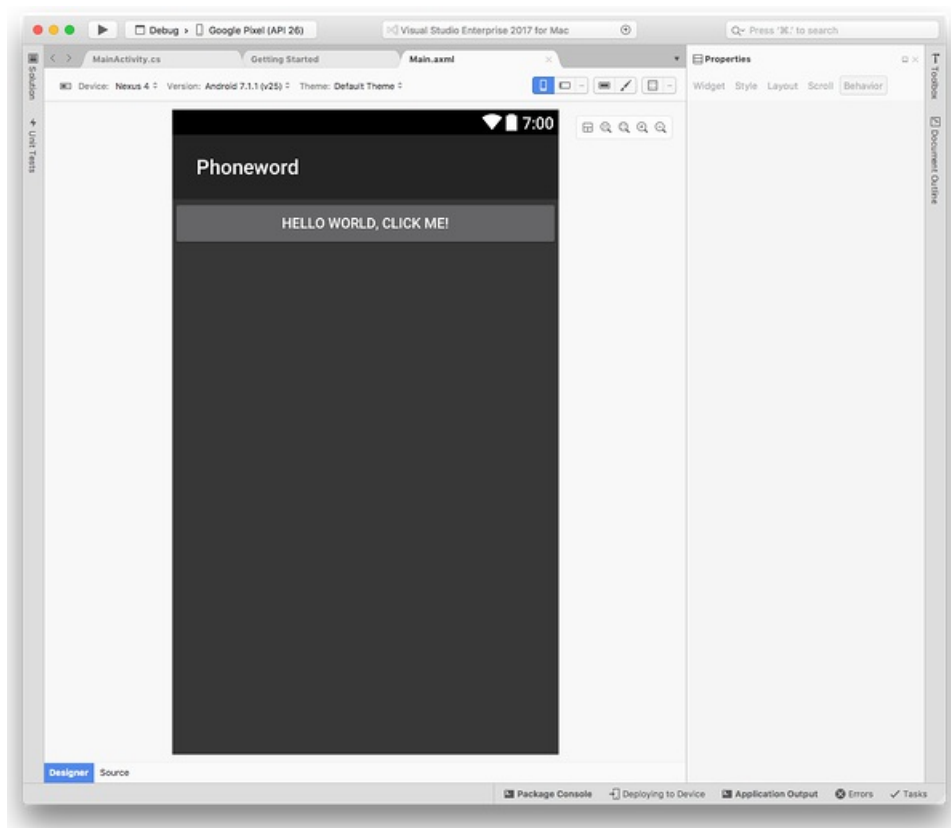
在“配置 Android 应用”对话框中，将新应用命名为 `Phoneword`，然后单击“下一步”。



在“配置新的 Android 应用”对话框中，将“解决方案”和“项目”名称保留设置为 `Phoneword`，然后单击“创建”以创建项目。

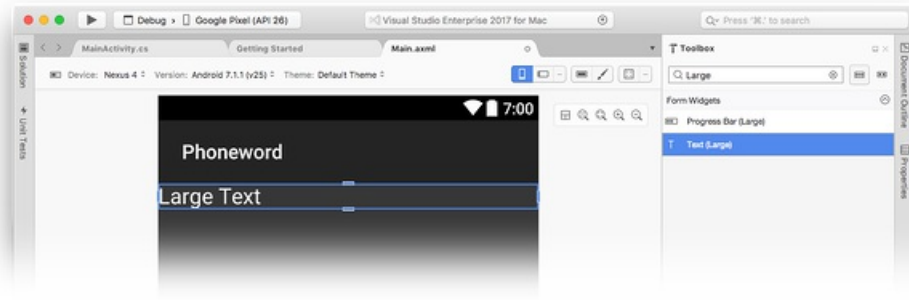
创建布局

创建新项目之后，在“解决方案”板中展开 **Resources** 文件夹，然后展开 **layout** 文件夹。双击 **Main.xml** 以在 Android 设计器中打开它。这是在 Android Designer 中进行查看时屏幕的布局文件：

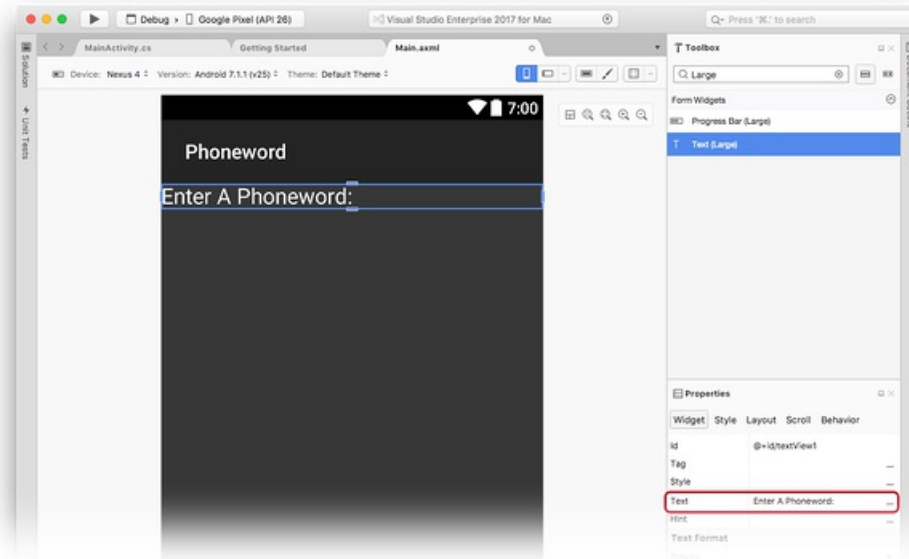


选择设计图面上的“Hello World, Click Me!” 按钮，然后按 **Delete** 键以删除它。

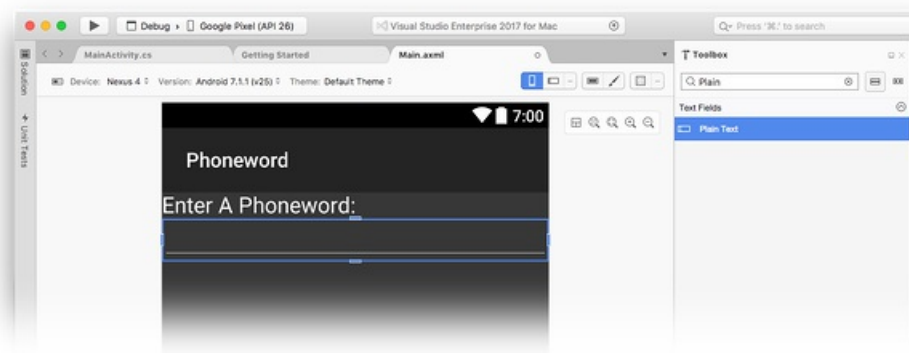
在“工具箱”(右侧区域)中，向搜索字段中输入 `text` 并将一个“文本(大)”小部件拖动到设计图面上(中央区域)：



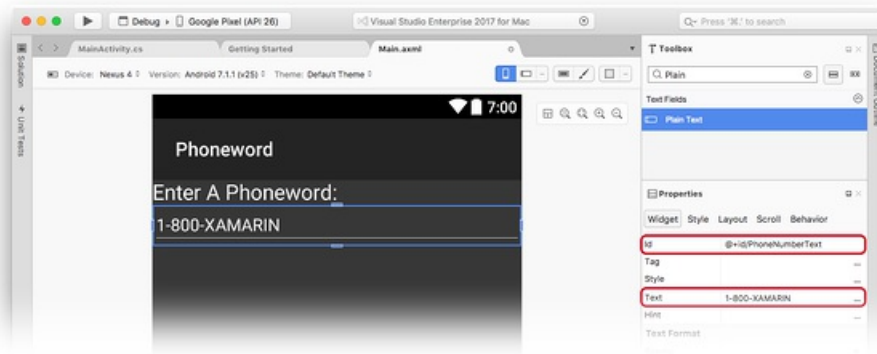
在设计图面上选择了“文本(大)”小部件的情况下，可以使用“属性”板将“文本(大)”小部件的 `Text` 属性更改为 `Enter a Phoneword:`，如下所示：



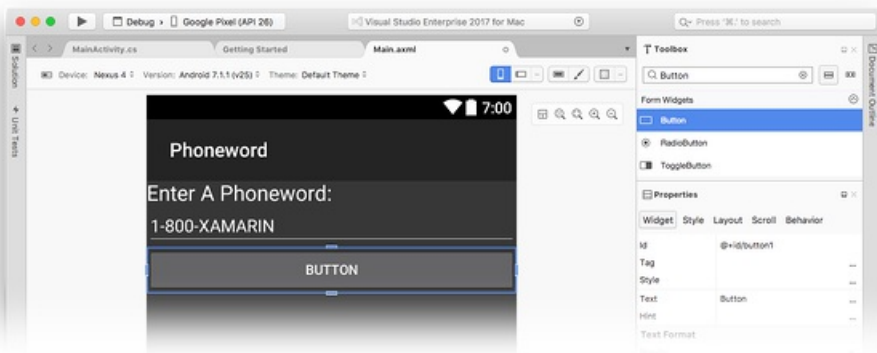
接下来，将一个“纯文本”小组件从“工具箱”拖动到设计图面上，并将它放置在“文本(大)”小组件下。请注意，可以使用搜索字段帮助按名称查找小组件：



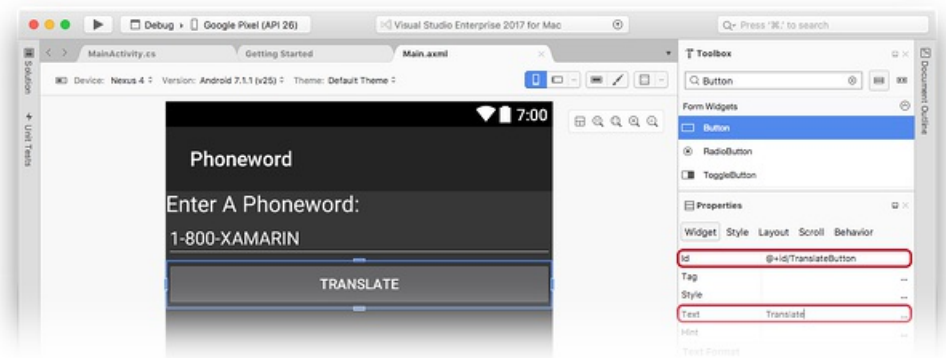
在设计图面上选择了“纯文本”小部件的情况下，可以使用“属性”板将“纯文本”小部件的 `Id` 属性更改为 `@+id/PhoneNumberText`，并将 `Text` 属性更改为 `1-855-XAMARIN`：



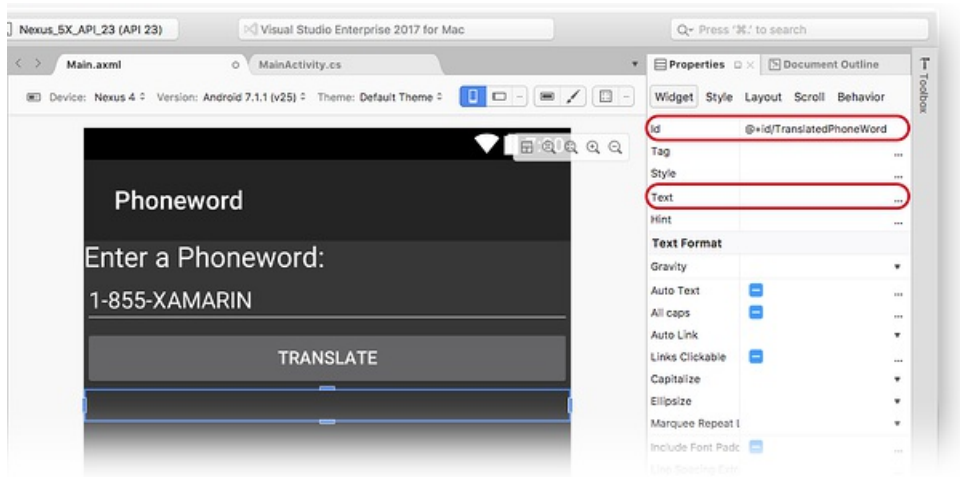
将一个“按钮”从“工具箱”拖动到设计图面上，并将它放置在“纯文本”小组件下方：



在设计图面上选择了“按钮”的情况下，可以使用“属性”板将“按钮”的 `Id` 属性更改为 `@+id/TranslateButton`，并将 `Text` 属性更改为 `Translate`：



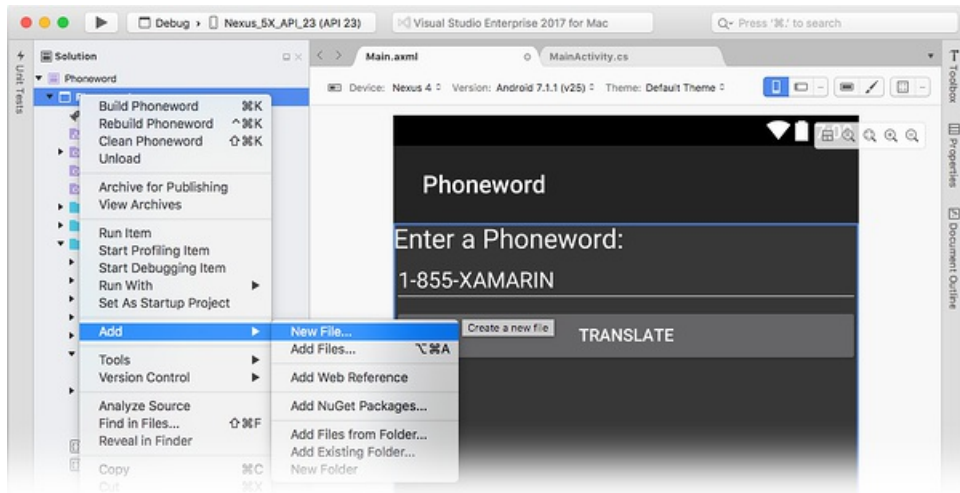
将一个“TextView”从“工具箱”拖动到 Design Surface 上，并将其置于“按钮”小组件下方。选中“TextView”后，将“TextView”的 `id` 属性设置为 `@+id/TranslatedPhoneWord`，并将 `text` 更改为一个空字符串：



通过按 **⌘ + S** 来保存工作。

编写一些代码

现在添加一些代码，以将电话号码从字母数字转换为数字。通过在“解决方案”板中单击“Phoneword”项目旁的齿轮图标，然后选择“添加”>“新建文件...”，向项目添加新文件：



在“新建文件”对话框中，选择“常规”>“空类”，将新文件命名为 PhoneTranslator，然后单击“新建”。这会为我们创建新的空 C# 类。

在新类中删除所有模板代码，并将其替换为以下代码：


```

using System.Text;
using System;
namespace Core
{
    public static class PhonewordTranslator
    {
        public static string ToNumber(string raw)
        {
            if (string.IsNullOrEmpty(raw))
                return "";
            else
                raw = raw.ToUpperInvariant();

            var newNumber = new StringBuilder();
            foreach (var c in raw)
            {
                if ("0123456789".Contains(c))
                {
                    newNumber.Append(c);
                }
                else
                {
                    var result = TranslateToNumber(c);
                    if (result != null)
                        newNumber.Append(result);
                }
                // otherwise we've skipped a non-numeric char
            }
            return newNumber.ToString();
        }
        static bool Contains (this string keyString, char c)
        {
            return keyString.IndexOf(c) >= 0;
        }
        static int? TranslateToNumber(char c)
        {
            if ("ABC".Contains(c))
                return 2;
            else if ("DEF".Contains(c))
                return 3;
            else if ("GHI".Contains(c))
                return 4;
            else if ("JKL".Contains(c))
                return 5;
            else if ("MNO".Contains(c))
                return 6;
            else if ("PQRS".Contains(c))
                return 7;
            else if ("TUV".Contains(c))
                return 8;
            else if ("WXYZ".Contains(c))
                return 9;
            return null;
        }
    }
}

```

通过选择“文件”>“保存”(或按 **⌘ + S**) 来保存对 **PhoneTranslator.cs** 文件进行的更改，然后关闭该文件。通过重新生成解决方案来确保没有编译时错误。

关联用户界面

下一步是通过向 `MainActivity` 类中添加支持代码来添加代码以关联用户界面。在“Solution Pad”中双击“MainActivity.cs”中以打开它。

首先将事件处理程序添加到“转换”按钮。在 `MainActivity` 类中找到 `OnCreate` 方法。在 `OnCreate` 中的 `base.OnCreate(bundle)` 和 `SetContentView (Resource.Layout.Main)` 调用下添加按钮代码。删除全部现有按钮处理代码(即引用 `Resource.Id.myButton` 并为其创建单击处理程序的代码), 让 `OnCreate` 方法如下所示:

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace Phoneword
{
    [Activity (Label = "Phone Word", MainLauncher = true)]
    public class MainActivity : Activity
    {
        protected override void OnCreate (Bundle bundle)
        {
            base.OnCreate (bundle);

            // Set our view from the "main" layout resource
            SetContentView (Resource.Layout.Main);

            // Our code will go here

        }
    }
}
```

接下来, 需要引用使用 Android 设计器在布局文件中创建的控件。将以下代码添加到 `OnCreate` 方法中(在 `SetContentView` 调用后面):

```
// Get our UI controls from the loaded layout
EditText phoneNumberText = FindViewById<EditText>(Resource.Id.PhoneNumberText);
TextView translatedPhoneWord = FindViewById<TextView>(Resource.Id.TranslatedPhoneWord);
Button translateButton = FindViewById<Button>(Resource.Id.TranslateButton);
```

通过向 `OnCreate` 方法添加以下代码(在最后一步中添加的行后面), 来添加对用户按“Translate”按钮进行响应的代码:

```
// Add code to translate number
string translatedNumber = string.Empty;

translateButton.Click += (sender, e) =>
{
    // Translate user's alphanumeric phone number to numeric
    translatedNumber = PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (string.IsNullOrEmpty(translatedNumber))
    {
        translatedPhoneWord.Text = string.Empty;
    }
    else
    {
        translatedPhoneWord.Text = translatedNumber;
    }
};
```

通过选择“生成”>“全部生成”(或按 `⌘ + B`), 来保存工作并生成应用程序。如果应用程序进行了编译, 则会在 Visual Studio for Mac 的顶部收到成功消息:

如果发生错误, 则完成前面的步骤并更正任何错误, 直到应用程序成功生成。如果收到生成错误(如“资源在当前

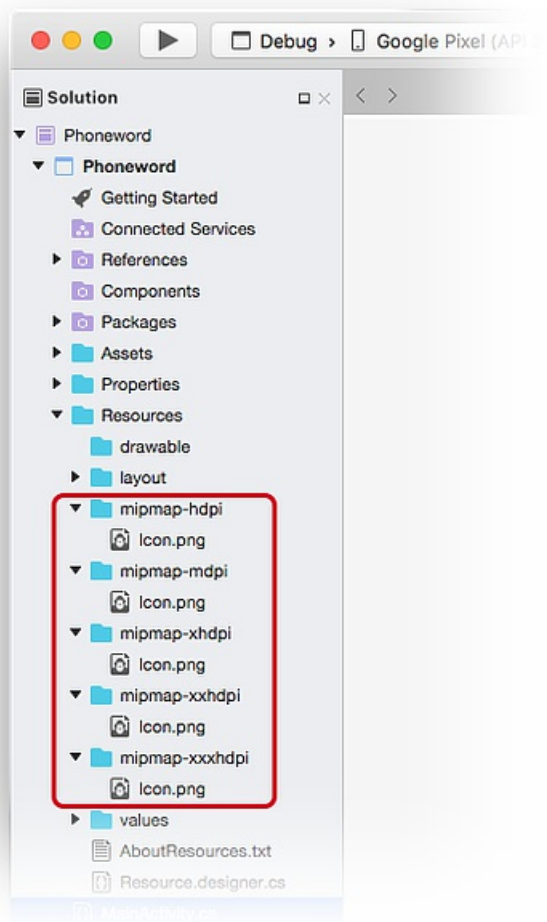
上下文中不存在”)，请验证 **MainActivity.cs** 中的命名空间名称是否与项目名称 (**Phoneword**) 匹配，然后完全重新生成解决方案。如果仍收到生成错误，请验证是否已安装最新的 Xamarin.Android 和 Visual Studio for Mac 更新。

设置标签和应用图标

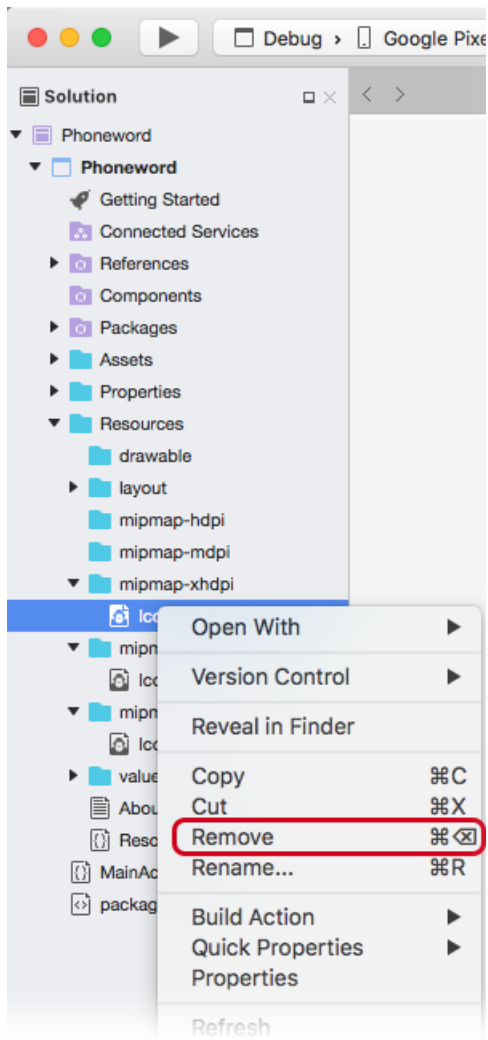
你现在具有可正常工作的应用程序，可以添加完成收尾工作了！首先编辑 **MainActivity** 的 **Label**。**Label** 是 Android 屏幕顶部显示的内容，用于让用户知道他们在应用程序中所处的位置。在 **MainActivity** 类顶部，将 **Label** 更改为 **Phone Word**，如下所示：

```
namespace Phoneword
{
    [Activity (Label = "Phone Word", MainLauncher = true)]
    public class MainActivity : Activity
    {
        ...
    }
}
```

现在可以设置应用程序图标了。默认情况下，Visual Studio for Mac 将为项目提供默认图标。从解决方案中删除这些文件，然后使用不同的图标替换它们。在“Solution Pad”中展开“Resources”文件夹。请注意，有 5 个前缀为“mipmap-”的文件夹，每个文件夹都包含一个“Icon.png”文件：

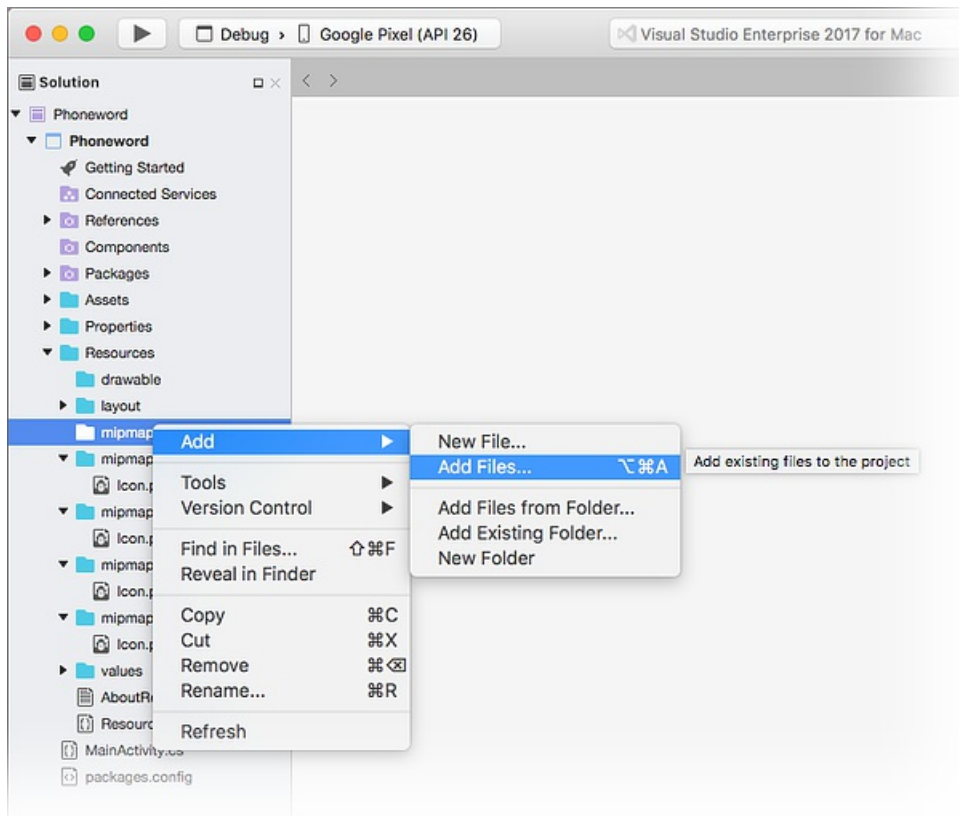


需要从项目中删除每个图标文件。右键单击每个“Icon.png”文件，然后从上下文菜单中选择“删除”：



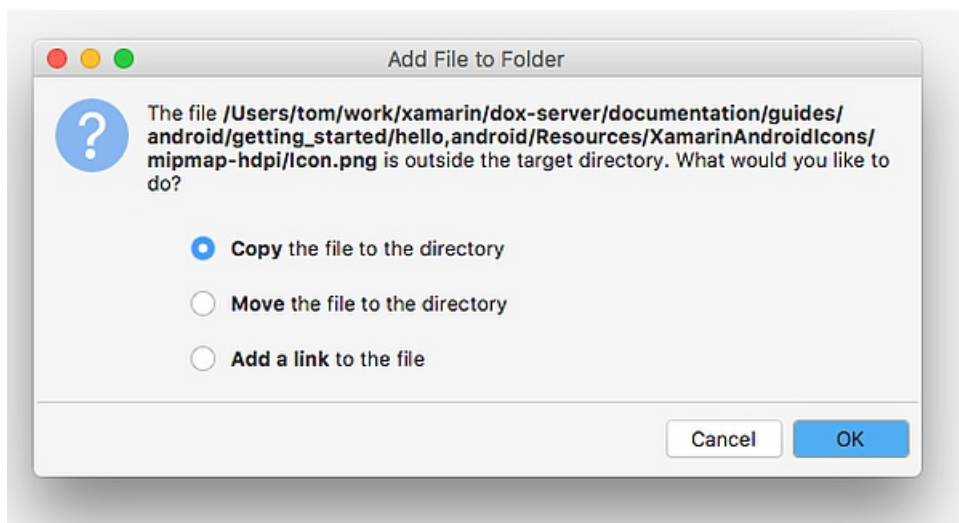
单击对话框中的“删除”按钮。

接着，下载并解压缩 [Xamarin 应用图标集](#)。此 zip 文件包含应用程序的图标。尽管每个图标看上去都相同，但它们具有不同的分辨率，使它们能在具有不同屏幕密度的不同设备上正确呈现。必须将此文件集复制到 Xamarin.Android 项目中。在 Visual Studio for Mac 的“Solution Pad”中，右键单击 mipmap-hdpi 文件夹，然后选择“添加”>“添加文件”：



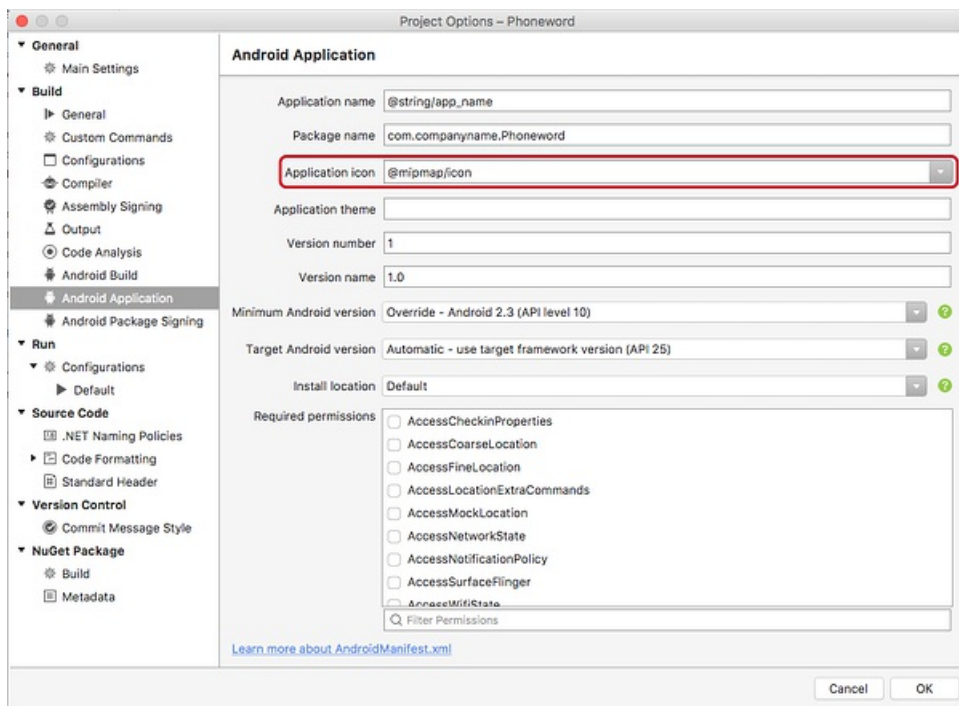
从选择对话框中，导航到已解压缩的 Xamarin AdApp 图标目录并打开 mipmap-hdpi 文件夹。选择“Icon.png”，然后单击“打开”。

在“将文件添加到文件夹”对话框中，选择“将文件复制到目录中”，然后单击“确定”：



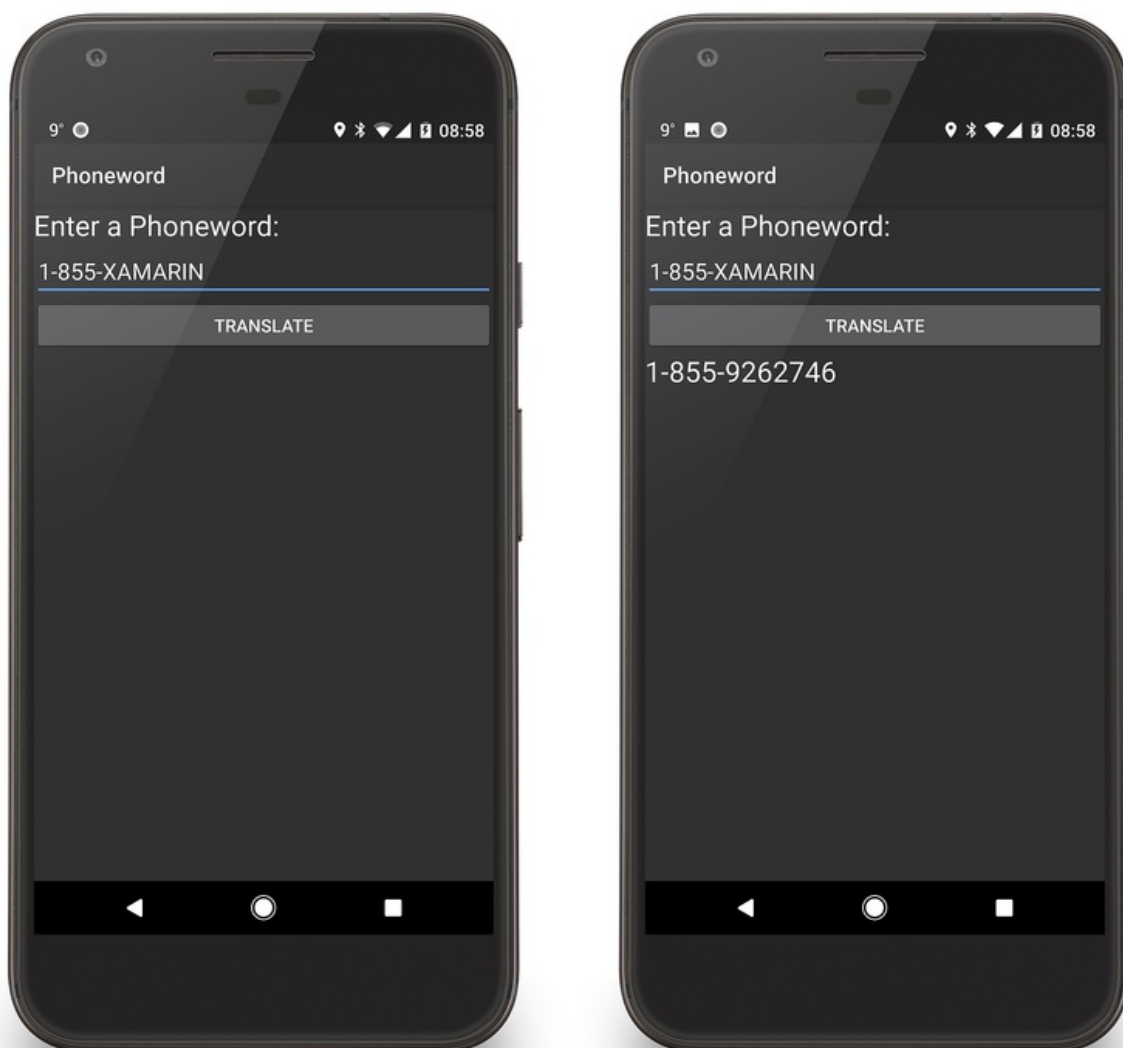
为每个 mipmap- 文件夹重复执行这些步骤，直到 mipmap- Xamarin 应用图标文件夹的内容复制到其在 Phoneword 项目中对应的 mipmap- 文件夹为止。

在所有图标都复制到 Xamarin.Android 项目中后，在“Solution Pad”中右键单击项目，打开“项目选项”对话框。选择“生成”>“Android 应用程序”，然后从“应用程序图标”组合框中选择 `@mipmap/icon`：



运行应用

最后，通过在 Android 设备或仿真器上运行应用程序并转换 Phoneword 以测试此应用程序：



若要在 Android 设备上运行应用，请参阅[如何设置设备以进行开发](#)。

祝贺你完成第一个 Xamarin.Android 应用程序！现在可以仔细分析刚刚学习的工具和技能。接下来是 [Hello, Android 深入了解](#)。

相关链接

- [Xamarin Android 应用图标 \(ZIP\)](#)
- [Phoneword\(示例\)](#)

了解 Android: 深度分析

2018/11/2 • [Edit Online](#)

本指南由两部分构成, 在本指南中, 用户将生成第一个 `Xamarin.Android` 应用程序, 并了解使用 `Xamarin` 进行 `Android` 应用程序开发的基础知识。在此过程中, 会向你介绍生成和部署 `Xamarin.Android` 应用程序所需的工具、概念和步骤。

在[了解 Android 快速入门](#)中, 生成并运行第一个 `Xamarin.Android` 应用程序。现在可以更深入地了解 `Android` 应用程序的工作原理, 以便可以生成更复杂的程序。本指南回顾在《了解 Android》演练中执行的步骤, 使用户了解所执行的操作并开始了解 `Android` 应用程序开发的基础知识。

本指南涉及以下主题:

- **Visual Studio 简介** – Visual Studio 以及创建新 `Xamarin.Android` 应用程序的简介。
- **Xamarin.Android 应用程序剖析** – `Xamarin.Android` 应用程序基本部分的教程。
- **应用基础知识和体系结构基础知识** – 活动、`Android` 清单, 以及常规 `Android` 开发风格简介。
- **用户界面 (UI)** – 使用 `Android Designer` 创建用户界面。
- **活动和活动生命周期** – 活动生命周期和使用代码连接用户界面的简介。
- **测试、部署和完成收尾工作** – 完成应用程序(提供了有关测试、部署、生成图稿等方面的建议)。
- **Visual Studio for Mac 简介** – Visual Studio for Mac 以及创建新 `Xamarin.Android` 应用程序的简介。
- **Xamarin.Android 应用程序剖析** – `Xamarin.Android` 应用程序基本部分的教程。
- **应用基础知识和体系结构基础知识** – 活动、`Android` 清单, 以及常规 `Android` 开发风格简介。
- **用户界面 (UI)** – 使用 `Android Designer` 创建用户界面。
- **活动和活动生命周期** – 活动生命周期和使用代码连接用户界面的简介。
- **测试、部署和完成收尾工作** – 完成应用程序(提供了有关测试、部署、生成图稿等方面的建议)。

本指南可帮助你掌握生成单屏幕 `Android` 应用程序所需的技能和知识。完成本指南之后, 应了解 `Xamarin.Android` 应用程序的不同部分以及它们如何组合在一起。

Visual Studio 简介

Visual Studio 是 Microsoft 提供的强大 IDE。它采用完全集成的可视化设计器、包含重构工具的文本编辑器、程序集浏览器、源代码集成等。在本指南中, 将了解如何使用 Visual Studio 的一些基本功能以及 `Xamarin` 插件。

Visual Studio 将代码组织为解决方案和项目。解决方案是可以容纳一个或多个项目的容器。项目可以是应用程序(如 `iOS` 或 `Android`)、支持库、测试应用程序等。在 **Phoneword** 应用中, 你已使用了“`Android` 应用程序”模板将新的 `Android` 项目添加到了在[了解 Android](#)指南中所创建的 **Phoneword** 解决方案中。

Visual Studio for Mac 简介

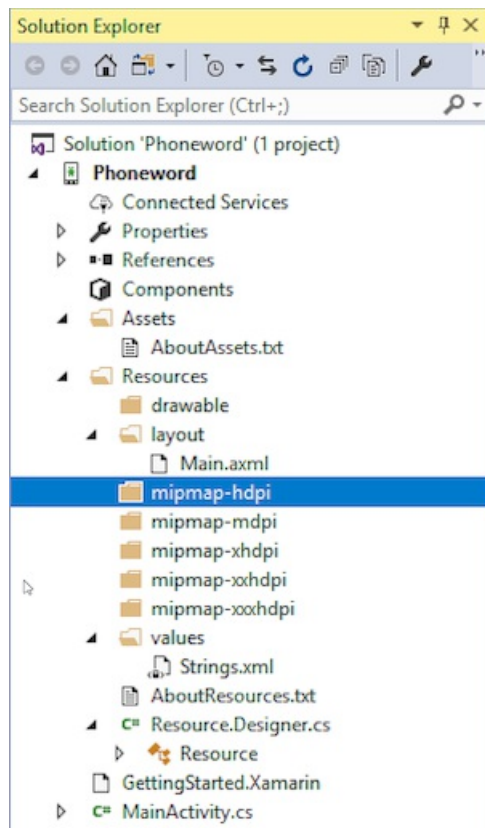
Visual Studio for Mac 是免费的开源 IDE, 类似于 Visual Studio。它采用完全集成的可视化设计器、包含重构工具的文本编辑器、程序集浏览器、源代码集成等。本指南介绍如何使用 Visual Studio for Mac 中的一些基本功能。如果是初次接触 Visual Studio for Mac, 可能需阅读更深入的 [Visual Studio for Mac 简介](#)。

Visual Studio for Mac 遵循将代码组织为解决方案和项目的 Visual Studio 做法。解决方案是可以容纳一个或多个

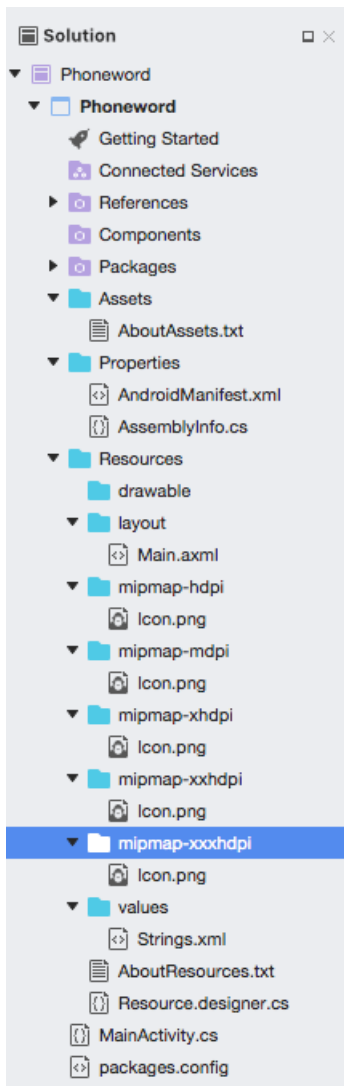
项目的容器。项目可以是应用程序(如 iOS 或 Android)、支持库、测试应用程序等。在 **Phoneword** 应用中, 你已使用了“Android 应用程序”模板将新的 Android 项目添加到了在[了解 Android](#)指南中所创建的 **Phoneword** 解决方案中。

Xamarin.Android 应用程序剖析

以下屏幕截图列出了解决方案的内容。下面是一个解决方案资源管理器, 它包含与解决方案关联的目录结构和所有文件:



以下屏幕截图列出了解决方案的内容。下面是一个 Solution Pad, 它包含与解决方案关联的目录结构和所有文件:



创建了一个名为 **Phoneword** 的解决方案，并在其中放入了 Android 项目 **Phoneword**。

查看项目中的项，查看每个文件夹及其用途：

- **属性** – 包含 [AndroidManifest.xml](#) 文件，该文件描述了对 Xamarin.Android 应用程序的所有要求（包括名称、版本号和权限）。**Properties** 文件夹还包括 .NET 程序集元数据文件 [AssemblyInfo.cs](#)。最好在此文件中填写一些应用程序相关的基本信息。
- **引用** – 包含生成和运行应用程序所需的程序集。如果展开“引用”目录，可查看对 .NET 程序集（如 [System](#)、[System.Core](#) 和 [System.Xml](#)）的引用以及对 Xamarin 的 Mono.Android 程序集的引用。
- **资产** – 包含应用程序需要运行的文件（包括字体、本地数据文件和文本文件）。此处包括的文件可通过生成的 `Assets` 类访问。有关 Android 资产的详细信息，请参阅 Xamarin [使用 Android 资产](#) 指南。
- **资源** – 包含应用程序资源，例如字符串、图像和布局。可以通过生成的 `Resource` 类访问代码中的这些资源。[Android 资源](#) 指南提供有关“资源”目录的更多详细信息。应用程序模板在 **AboutResources.txt** 文件中还包含有“资源”的简明指南。

资源

“资源”目录包含 4 个文件夹（drawable、layout、mipmap 和 values），还有一个名为 Resource.designer.cs 的文件。

下表总结了这些项：

- **drawable** – 可绘制目录包含 [可绘制资源](#)，如图像和位图。
- **mipmap** – mipmap 目录包含适用于不同启动器图标密度的可绘制文件。在默认模板中，drawable 目录包含应用程序图标文件“Icon.png”。

- **layout** – 布局目录包含 *Android 设计器文件* (.xml), 该文件定义每个屏幕或活动的用户界面。该模板创建名为 `activity_main.xml` 的默认布局。
- **layout** – 布局目录包含 *Android 设计器文件* (.xml), 该文件定义每个屏幕或活动的用户界面。该模板创建名为 **Main.xml** 的默认布局。
- **values** – 此目录包含存储简单值(如字符串、整数和颜色)的 XML 文件。该模板创建名为 **Strings.xml** 的文件, 用于存储字符串值。
- **Resource.designer.cs** – 也称为 `Resource` 类, 此文件是一个分部类, 存放分配给每个资源的唯一 ID。它由 Xamarin.Android 工具自动创建, 并在必要时重新生成。不应手动编辑此文件, 因为 Xamarin.Android 将覆盖对其进行的任何手动更改。

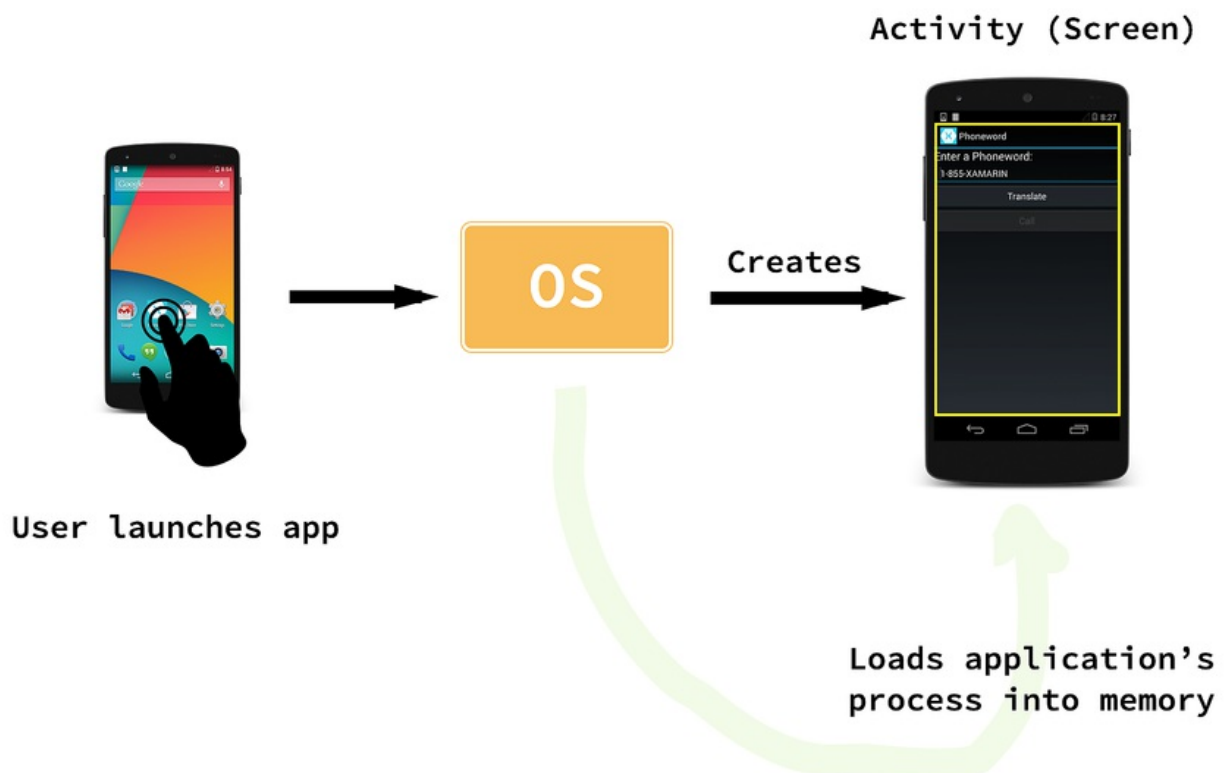
应用基础知识和体系结构基础知识

Android 应用程序不具有单一入口点;也就是说, 应用程序中没有操作系统可用来启动该应用程序的任何代码行。相反, 当 Android 实例化应用程序的一个类时, 会启动该应用程序, 在此期间 Android 将整个应用程序的进程加载到内存中。

设计复杂应用程序或与 Android 操作系统交互时, Android 的这一特有功能极其有用。但是, 这些选项也使 Android 在处理 **Phoneword** 应用程序等基本方案时变得复杂。出于此原因, 分两种情况来探索 Android 体系结构。本指南剖析使用 Android 应用最常见入口点(第一个屏幕)的应用程序。在[了解 Android 多屏幕](#)中, 讨论了以不同方式启动应用程序, 全面探讨了 Android 体系结构的复杂性。

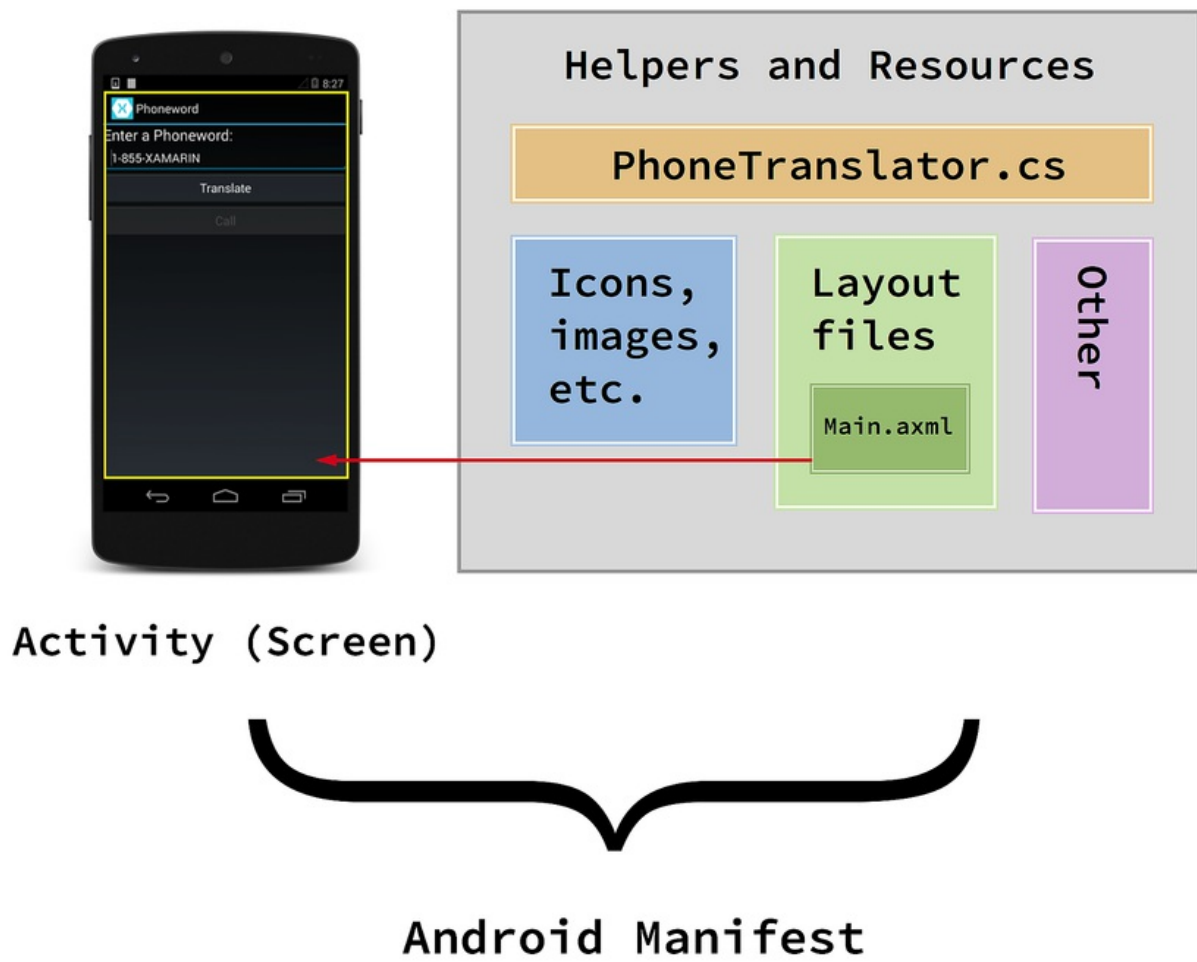
Phoneword 解决方案 - 以活动开始

在仿真器或设备中首次打开 **Phoneword** 应用程序时, 操作系统会创建第一个活动。活动是特殊的 Android 类, 对应于单个应用程序屏幕, 负责绘制和支持用户界面。Android 创建应用程序的第一个活动时, 会加载整个应用程序:



由于 Android 应用程序中没有线性发展(可以通过多个点启动应用程序), Android 采用一种独特方式来跟踪哪些类和文件组成应用程序。在 **Phoneword** 示例中, 将名为“Android 清单”的特殊 XML 文件注册组成应用程序的所有部分。“Android 清单”的作用是跟踪应用程序的内容、属性和权限, 并将这些信息告知 Android 操作系统。可以将 **Phoneword** 应用程序当作单一活动(屏幕)和由 Android 清单文件捆绑在一起的资源文件和帮助程序文件的

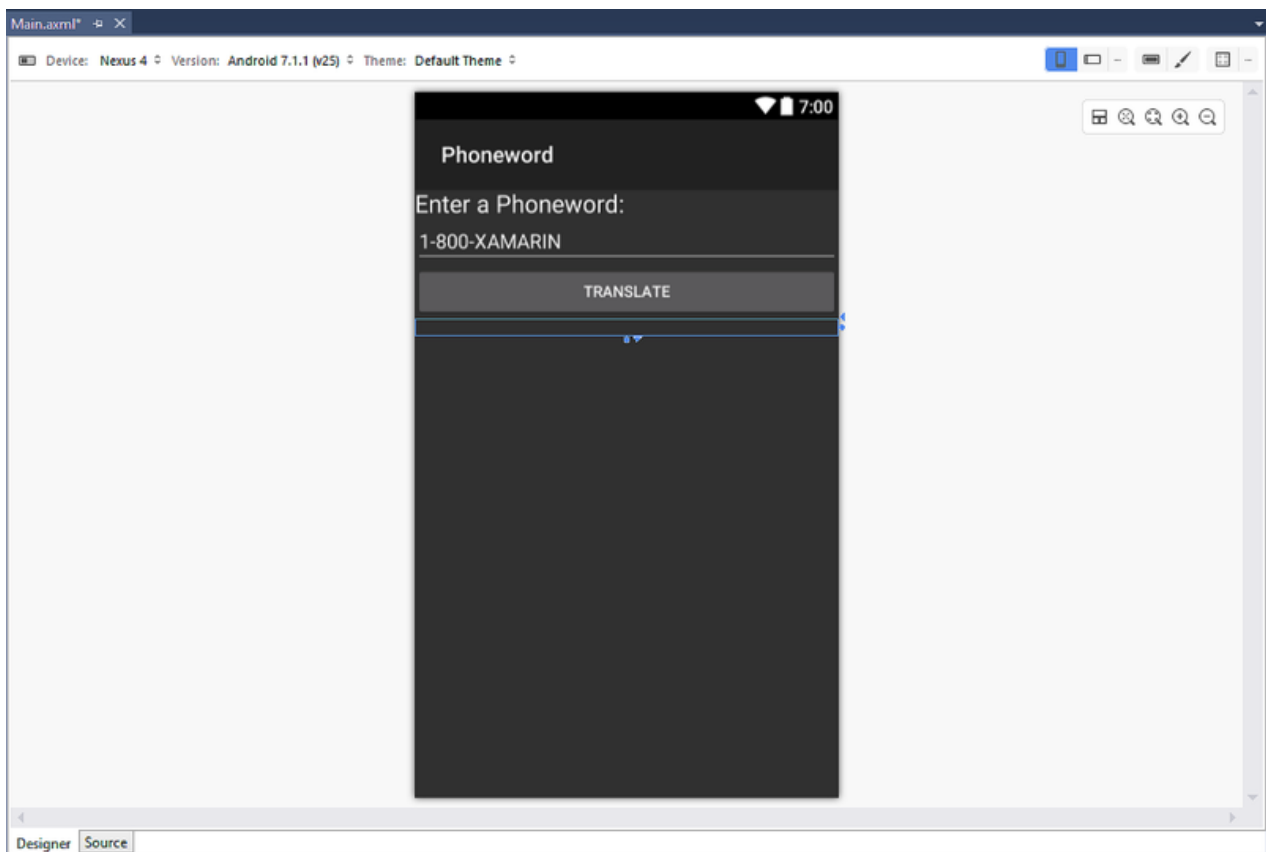
集合，如以下关系图所示：



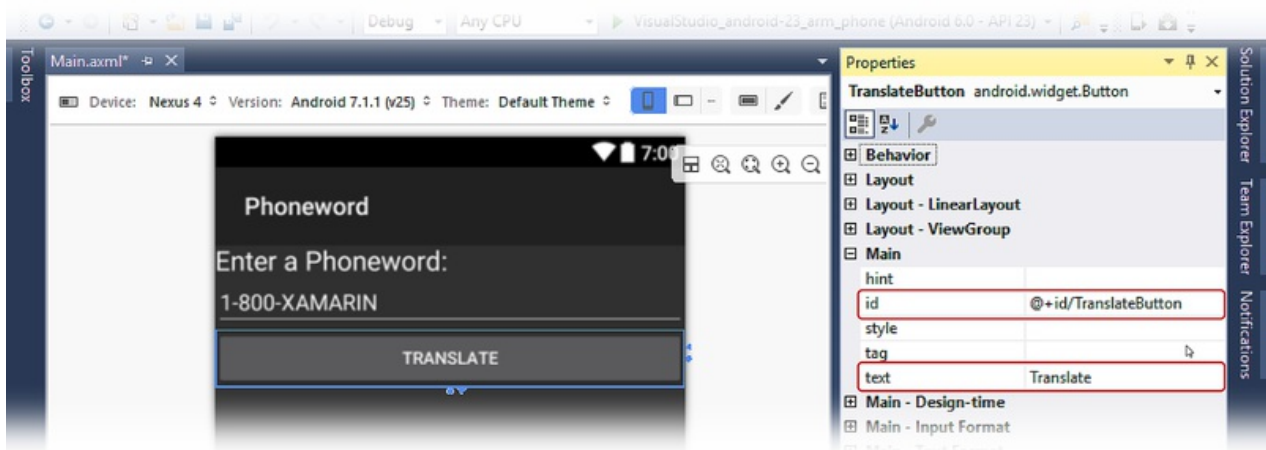
以下几个部分将探索 **Phoneword** 应用程序各部分的关系；使你能更好地理解上面的关系图。此探索先从用户界面开始，会讨论 Android 设计器和布局文件。

用户界面

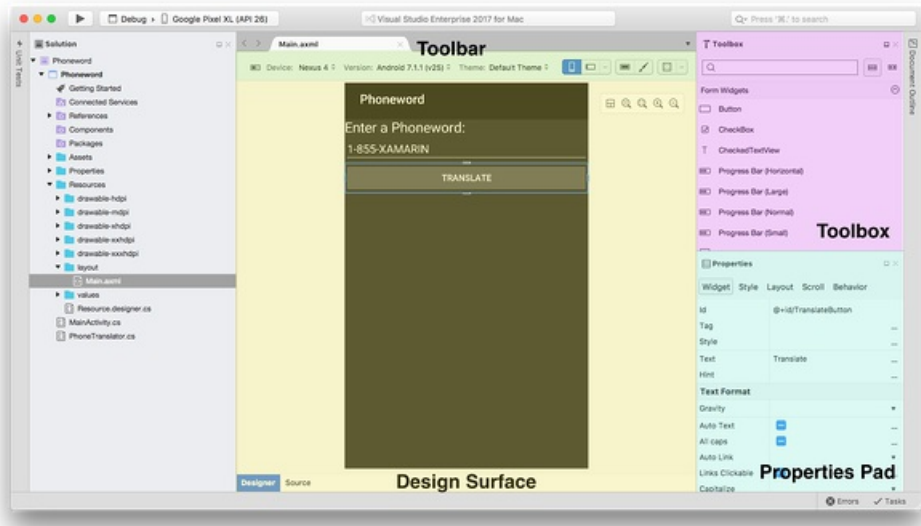
activity_main.xml 是应用程序中第一个屏幕的用户界面布局文件。.xml 指示这是 Android 设计器文件 (AXML 表示 *Android XML*)。名称 Main 对 Android 而言是任意的 – 可将布局文件命名为其他名称。在 IDE 中打开 activity_main.xml 时，会显示名为“Android Designer”的 Android 布局文件的可视编辑器：



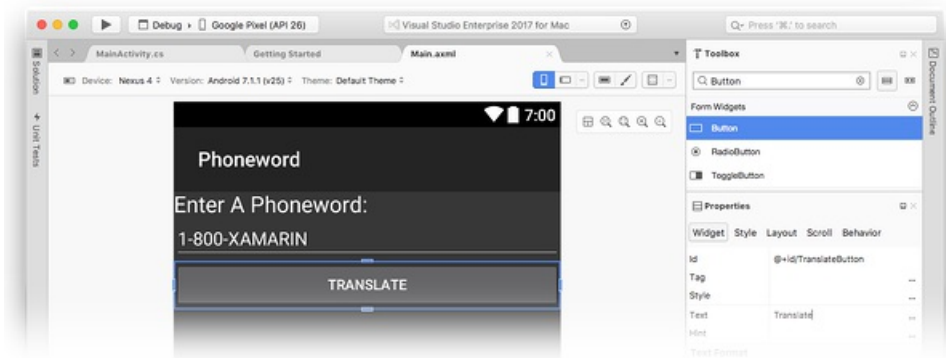
在 **Phoneword** 应用中, **TranslateButton** 的 ID 设置为 `@+id/TranslateButton` :



Main.xml 是应用程序中第一个屏幕的用户界面布局文件。xml 指示这是 Android 设计器文件 (AXML 表示 *Android XML*)。名称 Main 对 Android 而言是任意的 – 可将布局文件命名为其他名称。在 IDE 中打开 **Main.xml** 时, 会显示名为“Android 设计器”的 Android 布局文件的可视编辑器:



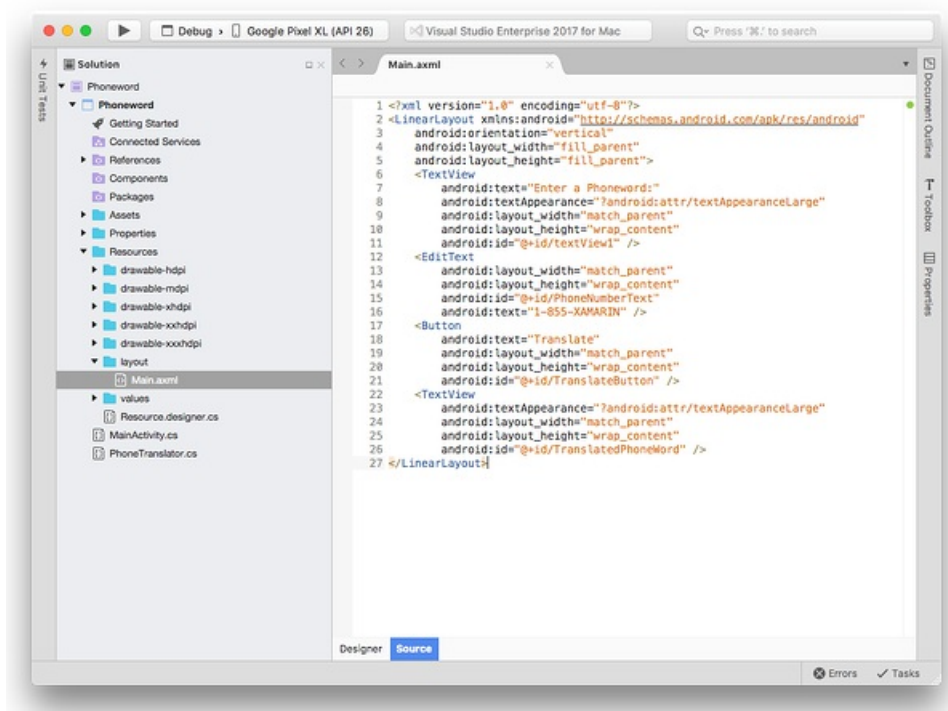
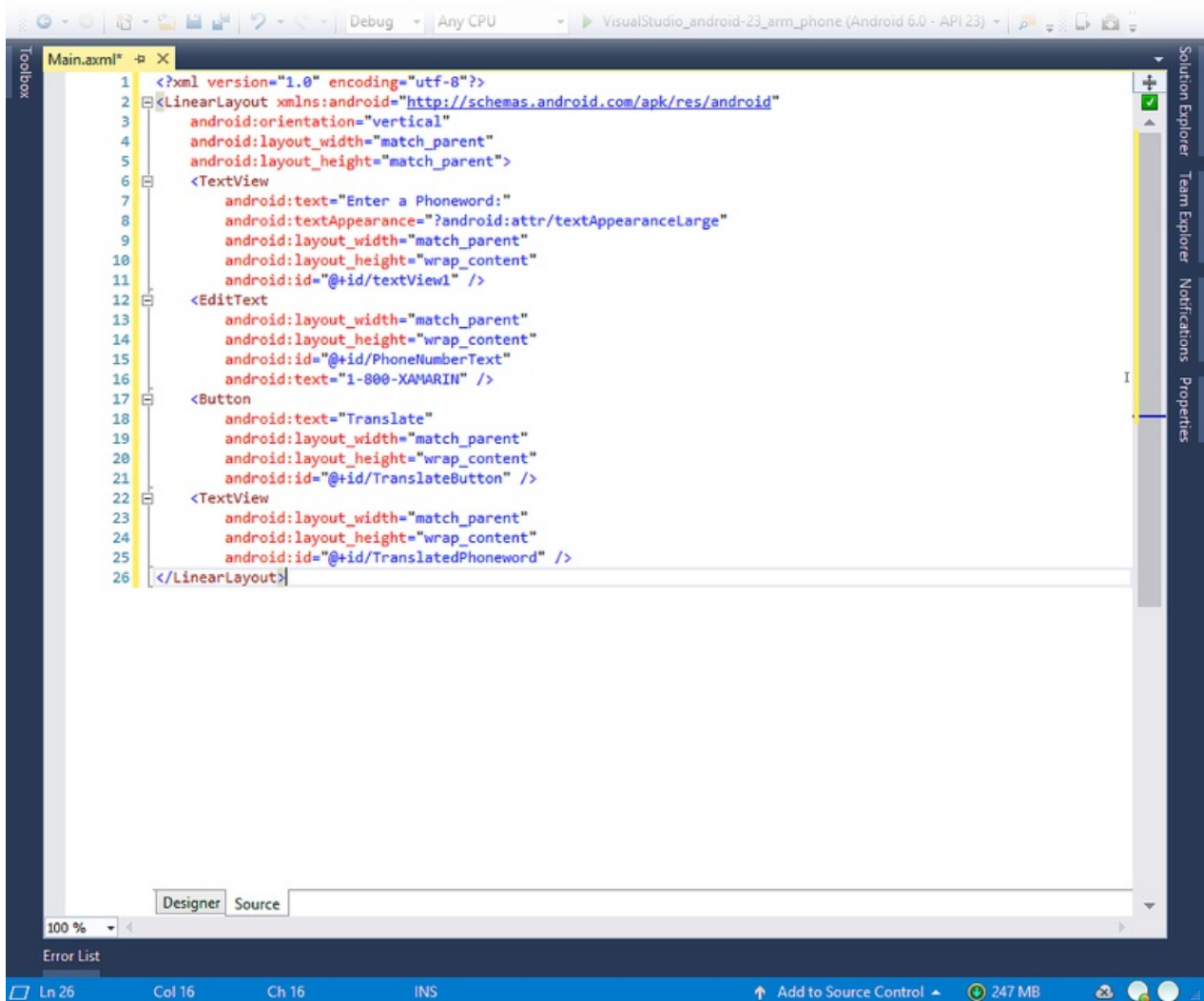
在 **Phoneword** 应用中, **TranslateButton** 的 ID 设置为 `@+id/TranslateButton` :



设置 **TranslateButton** 的 `id` 属性时, Android Designer 会将 **TranslateButton** 控件映射到 `Resource` 类, 并为其分配 `TranslateButton` 的资源 ID。通过将可视控件映射到类, 可以找到并使用 **TranslateButton** 和应用代码中的其他控件。当你剖析为控件提供支持的代码时, 会更详细地了解这一内容。此时, 只需知道控件的代码表示形式是通过 `id` 属性链接到设计器中控件的可视表示形式即可。

源视图

在设计界面上定义的所有内容都会转换成 XML, 以供 Xamarin.Android 使用。Android 设计器提供源视图, 此源视图包含从可视化设计器生成的 XML。可以切换到设计器视图左下角的“源”面板以查看此 XML, 如下屏幕截图所示:



此 XML 源代码应包含文本 (大型)、纯文本和两个按钮元素。有关 Android 设计器的更深入教程, 请参阅 [Xamarin Android 设计器概述](#) 指南。

现在已讨论了用户界面可视部分背后的工具和概念。接下来, 通过探索活动和活动生命周期介绍为用户界面提供支持的代码。

活动和活动生命周期

Activity 类包含为用户界面提供支持的代码。活动负责响应用户交互，并创建动态用户体验。本部分将介绍

Activity 类，讨论活动生命周期，并剖析为 **Phoneword** 应用程序中的用户界面提供支持的代码。

Activity 类

Phoneword 应用程序只有一个屏幕(活动)。为屏幕提供支持的类称为 **MainActivity**，位于 **MainActivity.cs** 文件中。名称 **MainActivity** 在 Android 中没有特别的意义 – 虽然约定是命名应用程序 **MainActivity** 中的第一个活动，但 Android 并不在意将其命名为其他名称。

打开 **MainActivity.cs** 时，可以看到，**MainActivity** 类是 **Activity** 类的子类并且活动标有 **Activity** 属性：

```
[Activity (Label = "Phone Word", MainLauncher = true)]
public class MainActivity : Activity
{
    ...
}
```

Activity 属性向 Android 清单注册活动；这能让 Android 知道此类是该清单所管理的 **Phoneword** 应用程序的一部分。**Label** 属性设置将显示在屏幕顶部的文本。

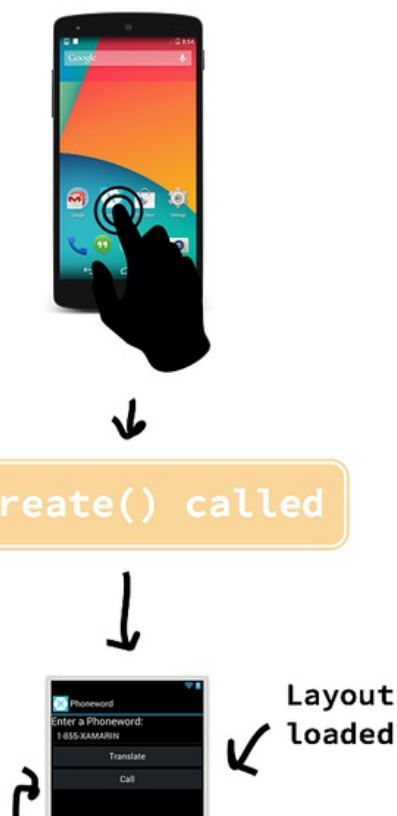
MainLauncher 属性告知 Android 在启动应用程序时显示此活动。如[了解 Android 多屏幕指南](#)中所述，当你向应用程序添加更多活动(屏幕)时，此属性会变得很重要。

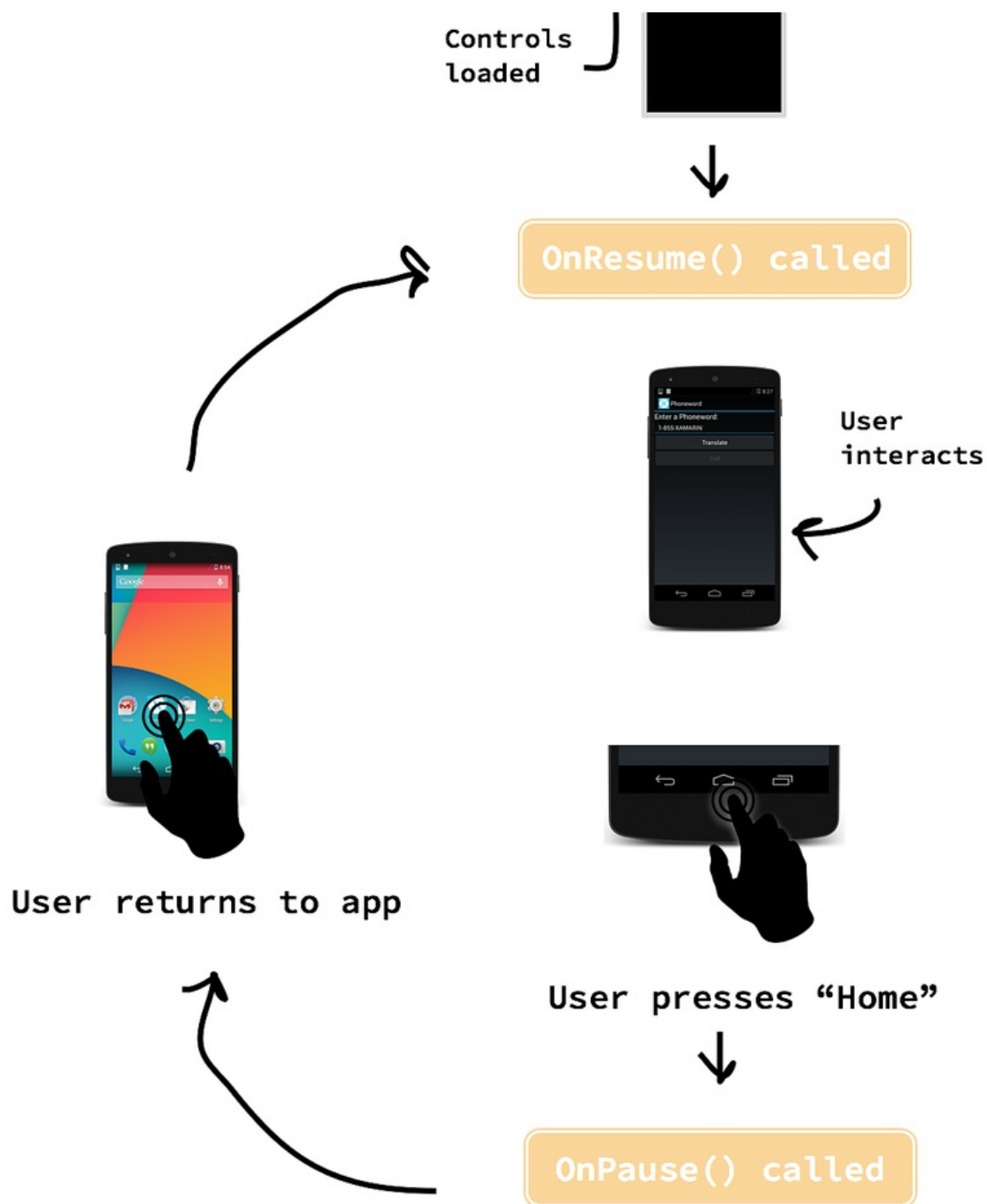
在了解了 **MainActivity** 的基础知识，现在将通过介绍_活动生命周期_来深入研究活动代码。

活动生命周期

在 Android 中，活动会根据与用户的交互经历生命周期的不同阶段。可以对活动进行创建、启动和暂停、恢复和销毁等操作。**Activity** 类包含方法，系统会在屏幕生命周期的特定时间点调用这些方法。下图说明了活动的典型生命周期以及一些相应的生命周期方法：

User launches app





通过重写 `Activity` 生命周期方法，可以控制活动的加载方式和与用户的互动方式，甚至还可以控制活动从设备屏幕消失后会发生的状况。例如，可以重写上图中的生命周期方法，以执行以下重要任务：

- **OnCreate** – 创建视图、初始化变量，并执行在用户能看到活动之前其他必须完成的准备工作。只有将活动加载到内存时，才会调用此方法一次。
- **OnResume** – 执行每当活动返回到设备屏幕时必须发生的任何任务。
- **OnPause** – 执行每当活动离开设备屏幕时必须发生的任何任务。

向 `Activity` 中的生命周期方法添加自定义代码时，重写该生命周期方法的基实现。可以利用现有的生命周期方法（已在其中附加了一些代码）并使用自己的代码来扩展该方法。从方法内调用基实现，确保原始代码在新代码之前运行。下一部分对此提供示例说明。

活动生命周期是 Android 中一个重要且复杂的部分。完成_入门_系列后，如果想要了解有关活动的详细信息，请阅读[活动生命周期](#)指南。本指南下一步的重点是活动生命周期的第一个阶段 - `OnCreate`。

OnCreate

Android 创建活动时会调用 `Activity` 的 `onCreate` 方法(向用户显示屏幕之前)。可以重写 `onCreate` 生命周期方法, 以创建视图并准备向用户显示活动:

```
protected override void onCreate (Bundle bundle)
{
    base.onCreate (bundle);

    // Set our view from the "main" layout resource
    setContentView (Resource.Layout.Main);
    // Additional setup code will go here
}
```

在 **Phoneword** 应用中, 首先在 `onCreate` 中加载在 Android Designer 中创建的用户界面。若要加载 UI, 请调用 `setContentView` 并向其传递布局文件的资源布局名称: `activity_main.xml`。布局位于 `Resource.Layout.activity_main`:

```
setContentView (Resource.Layout.activity_main);
```

当 `MainActivity` 开始运行后, 会基于 `activity_main.xml` 文件的内容创建一个视图。

在 **Phoneword** 应用中, 首先在 `onCreate` 中加载在 Android Designer 中创建的用户界面。若要加载 UI, 请调用 `setContentView` 并向其传递布局文件的资源布局名称: `Main.xml`。布局位于 `Resource.Layout.Main`:

```
setContentView (Resource.Layout.Main);
```

当 `MainActivity` 开始运行后, 会基于 **Main.xml** 文件中内容创建一个视图。请注意, 布局文件名称应与活动名称匹配 – `Main.xml` 是 `Main` 活动的布局。这不是 Android 要求的, 但当你开始向应用程序添加更多屏幕时, 会发现此命名约定便于匹配布局文件和代码文件。

准备好布局文件后, 可以开始查找控件。若要查找控件, 请调用 `findViewById`, 并传入控件的资源 ID:

```
EditText phoneNumberText = findViewById<EditText>(Resource.Id.PhoneNumberText);
Button translateButton = findViewById<Button>(Resource.Id.TranslateButton);
TextView translatedPhoneWord = findViewById<TextView>(Resource.Id.TranslatedPhoneWord);
```

现在布局文件中已具有对控件的引用, 可以开始对其进行编程, 以响应用户交互。

响应用户交互

在 Android 中, `click` 事件侦听用户的触控。在此应用中, `click` 事件将由 lambda 处理, 不过也可改用委托或命名事件处理程序。最终的 **TranslateButton** 代码如下所示:

```
translateButton.Click += (sender, e) =>
{
    // Translate user's alphanumeric phone number to numeric
    translatedNumber = PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (string.IsNullOrEmpty(translatedNumber))
    {
        translatedPhoneWord.Text = string.Empty;
    }
    else
    {
        translatedPhoneWord.Text = translatedNumber;
    }
};
```

测试、部署和完成收尾工作

Visual Studio for Mac 和 Visual Studio 均提供许多用于测试和部署应用程序的选项。此部分介绍调试选项，演示如何在设备上测试应用程序，并介绍用于针对不同的屏幕密度创建自定义应用图标的工具。

调试工具

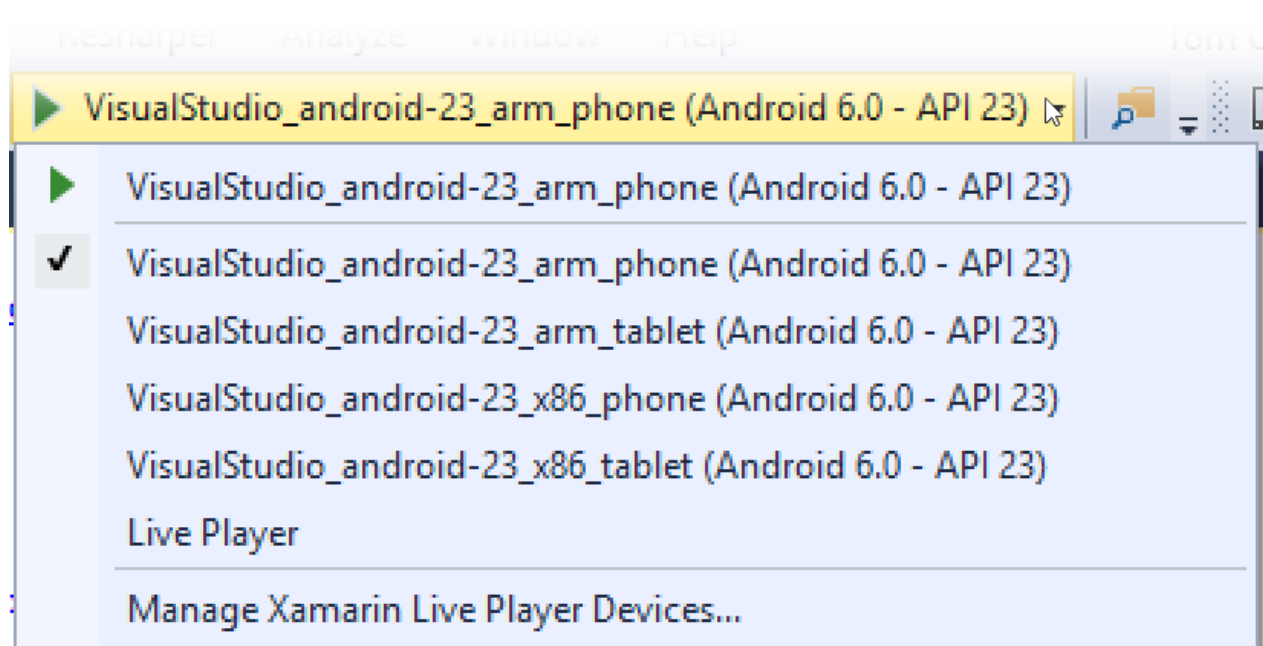
应用程序代码中的问题可能难以进行诊断。为了帮助诊断复杂的代码问题，可以[设置断点](#)、[逐行执行代码](#)或[将信息输出到日志窗口](#)。

部署到设备

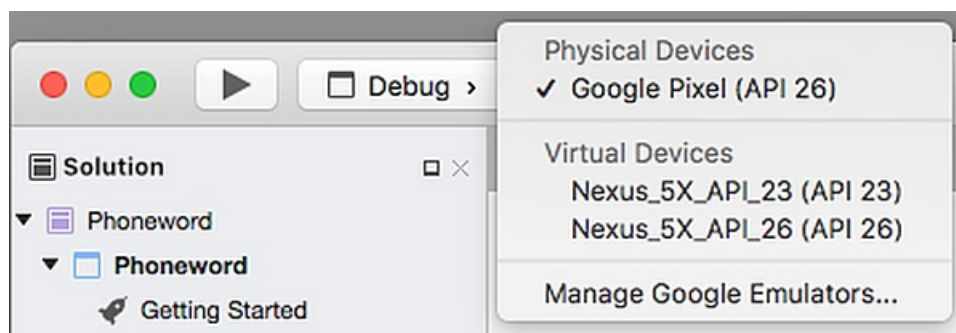
仿真器是用于部署和测试应用程序的良好开端，但用户不会在仿真器中使用最终应用程序。最好尽早并经常在实际设备上测试应用程序。

使用 Android 设备测试应用程序之前，需要对其进行开发配置。[设置设备进行开发](#)指南提供了有关准备设备进行开发的详尽说明。

配置设备后，可通过插入设备，从“选择设备”对话框中将其选中，然后启动应用程序，对设备进行部署：



配置设备后，可通过插入设备，按“开始(播放)”，从“选择设备”对话框中将其选中，然后按“确定”，对设备进行部署：

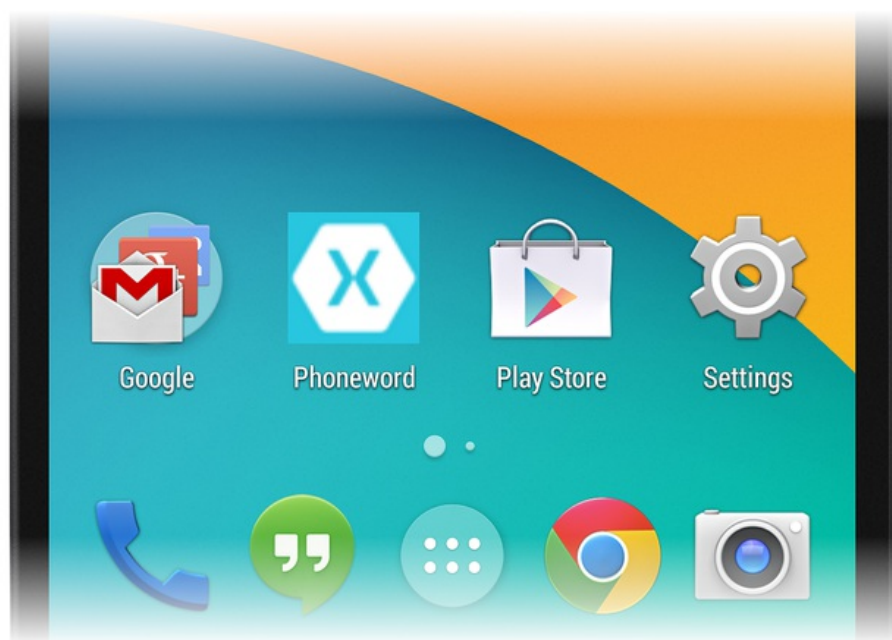


这将启动设备上的应用程序：

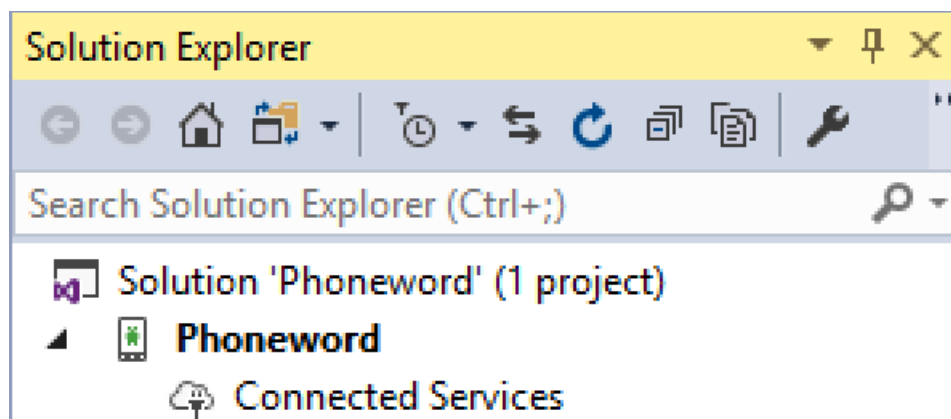























为不同的屏幕密度设置图标

Android 设备具有不同的屏幕大小和分辨率，不是所有图像都能清晰显示在屏幕上。例如，下面是一张高密度 Nexus 5 上低密度图标的屏幕截图。可以看到，与周边的图标相比，它很模糊：



考虑到这一点，最好将不同分辨率的图标添加到 **Resources** 文件夹。Android 提供了不同版本的 mipmap 文件夹来处理不同密度的启动器图标，包括针对中等密度屏幕的 mdpi、针对高密度屏幕的 hdpi，以及针对超高密度屏幕的 xhdpi、xxhdpi 和 xxxhdpi。不同大小的图标存储在相应的 mipmap- 文件夹中：

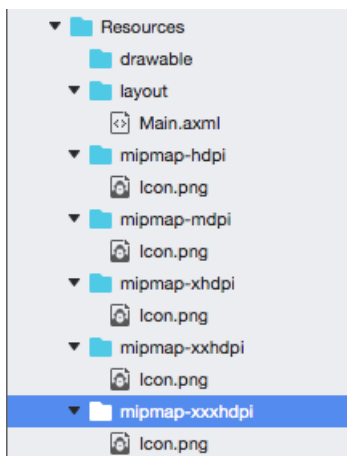


- ▶  Properties
- ▶  References
-  Components
- ▲  Assets
 -  AboutAssets.txt
- ▲  Resources
 -  drawable
 - ▲  layout
 -  Main.xml
 -  mipmap-hdpi
 -  mipmap-mdpi
 -  mipmap-xhdpi
 -  mipmap-xxhdpi
 -  mipmap-xxxhdpi
- ▲  values
 -  Strings.xml
 -  AboutResources.txt
- ▲  Resource.Designer.cs
 - ▶  Resource
-  GettingStarted.Xamarin
- ▶  MainActivity.cs

Solution Explorer

Team Explorer

Notifications



Android 根据合适的密度选择图标：



生成自定义图标

并非每个人都具有可用于创建自定义图标和启动图像(让应用与众不同)的设计器。下面是几种生成自定义应用图像的备选方法：

- [Android Asset Studio](#) – 是一个基于 Web 的浏览器生成器，针对所有类型 Android 图标，带有其他有用社区工具的链接。在 Google Chrome 中性能最佳。
- Visual Studio – 可以用于直接在 IDE 中为应用创建简单图标集。
- [Glyphish](#) – 可免费下载和购买的高质量预生成图标集。
- [Fiverr](#) – 从各种设计器中进行选择以便为你创建图标集(最低 5 美元)。可以漫无目标，不过如果需要动态设计的图标，这是很好的资源。
- [Android Asset Studio](#) – 是一个基于 Web 的浏览器生成器，针对所有类型 Android 图标，带有其他有用社区工具的链接。在 Google Chrome 中性能最佳。
- [Sketch 3](#) – Sketch 是用于设计用户界面、图标等的 Mac 应用。这是用于设计 Xamarin 应用图标和启动图像集的应用。App Store 提供 Sketch 3，价格约 80 美元。还可以免费试用 [Sketch 工具](#)。
- [Pixelmator](#) – 适用于 Mac 的通用图像编辑应用(价格约 30 美元)。
- [Glyphish](#) – 可免费下载和购买的高质量预生成图标集。
- [Fiverr](#) – 从各种设计器中进行选择以便为你创建图标集(最低 5 美元)。可以漫无目标，不过如果需要动态设

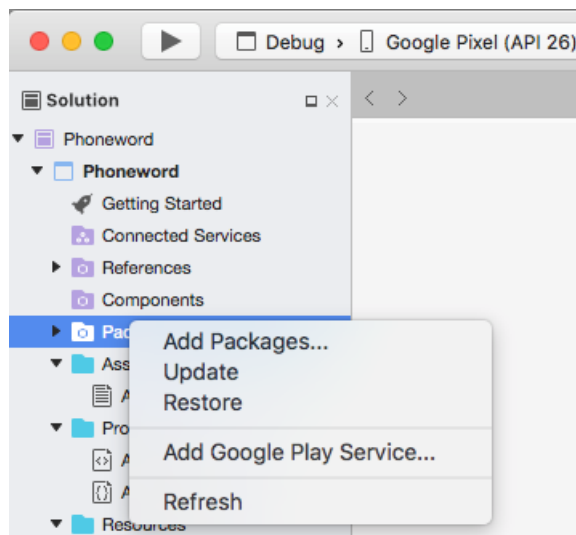
计的图标，这是很好的资源。

有关图标大小和要求的详细信息，请参阅 [Android 资源指南](#)。

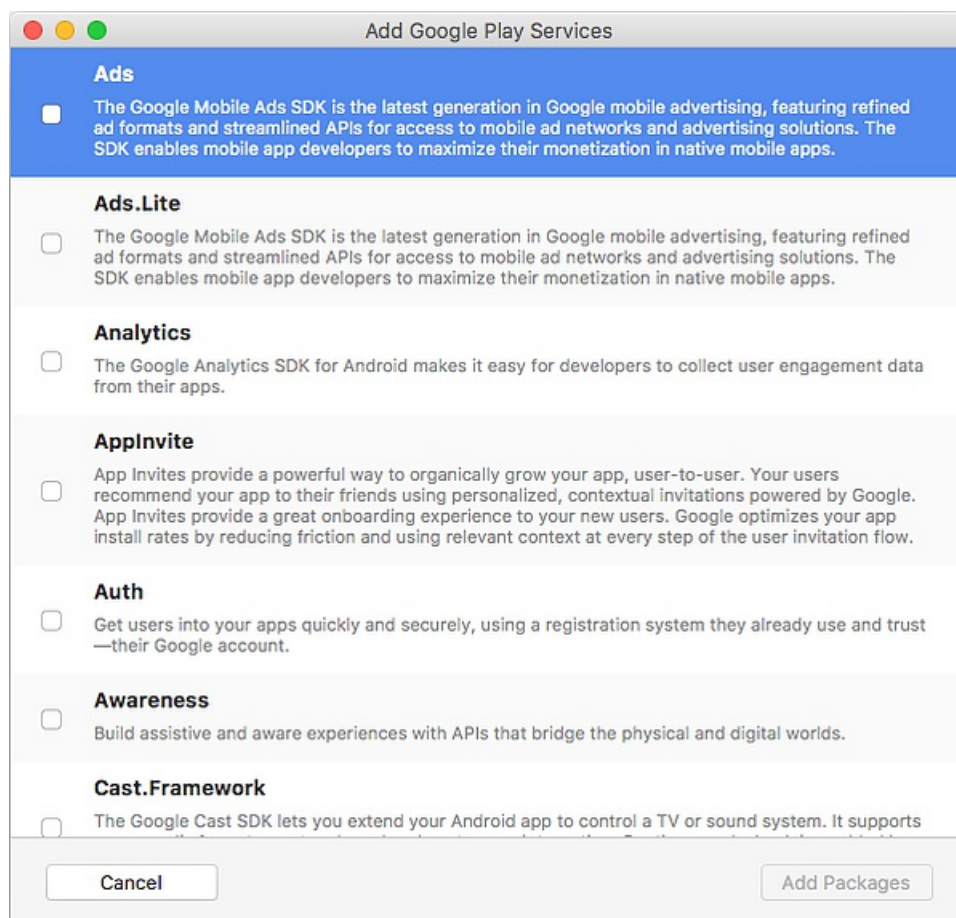
添加 Google Play Services 包

Google Play Services 是一套外接程序库，让 Android 开发人员可以利用 Google 的最新功能，如 Google Maps、Google Cloud Messaging 和 In-App Billing。以前，由 Xamarin 以单一包的形式提供对所有 Google Play Services 库的绑定 – 从 Visual Studio for Mac 开始，提供了一个新的项目对话框，用于选择要包括在应用中的 Google Play Services 包。

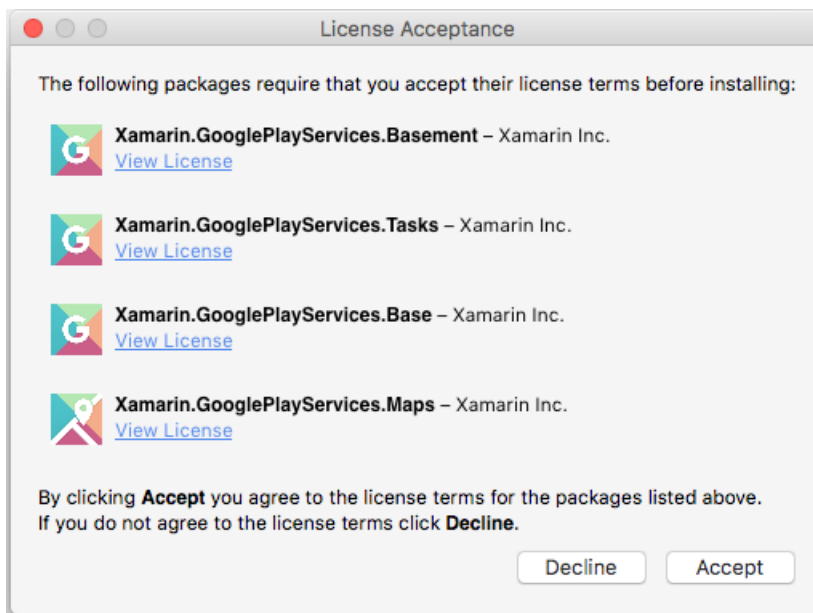
若要添加一个或多个 Google Play Service 库，右键单击项目树中的“包”节点，并单击“添加 Google Play Service...”：



出现“添加 Google Play Services”对话框时，选择想要添加到项目的包 (nuget)：



选择服务并单击“添加包”时，Visual Studio for Mac 会下载并安装所选包及其所需的任何 Google Play Services 依赖包。在某些情况下，可能会看到“接受许可”对话框，要求在安装包之前单击“接受”：



总结

祝贺你！你现在应已充分了解了 Xamarin.Android 应用程序的组件以及创建它们所需的工具。

在入门系列的另一个教程中，你将扩展应用程序以处理多个屏幕，同时探索更高级的 Android 体系结构和概念。

了解 Android 多屏幕

2018/10/26 • [Edit Online](#)

本指南由两部分构成，在本指南中，用户将扩展在《了解 Android》指南中创建的 Phoneword 应用程序，以处理第二个屏幕。在此过程中，本指南将介绍基本 Android 应用程序构建基块，并随着更好地了解 Android 应用程序结构和功能来使用户深入了解 Android 体系结构。

第 1 部分:快速入门

在本指南的第一部分，用户将向 Phoneword 应用程序添加第二个屏幕，用于跟踪从应用呼叫的号码的历史记录。最终的应用将显示第二个屏幕，其中将列出呼叫历史记录。

第 2 部分:深度分析

在本文档的第二部分，用户将回顾已生成的应用，并讨论体系结构、导航以及此过程中遇到的其他 Android 新概念。

相关链接

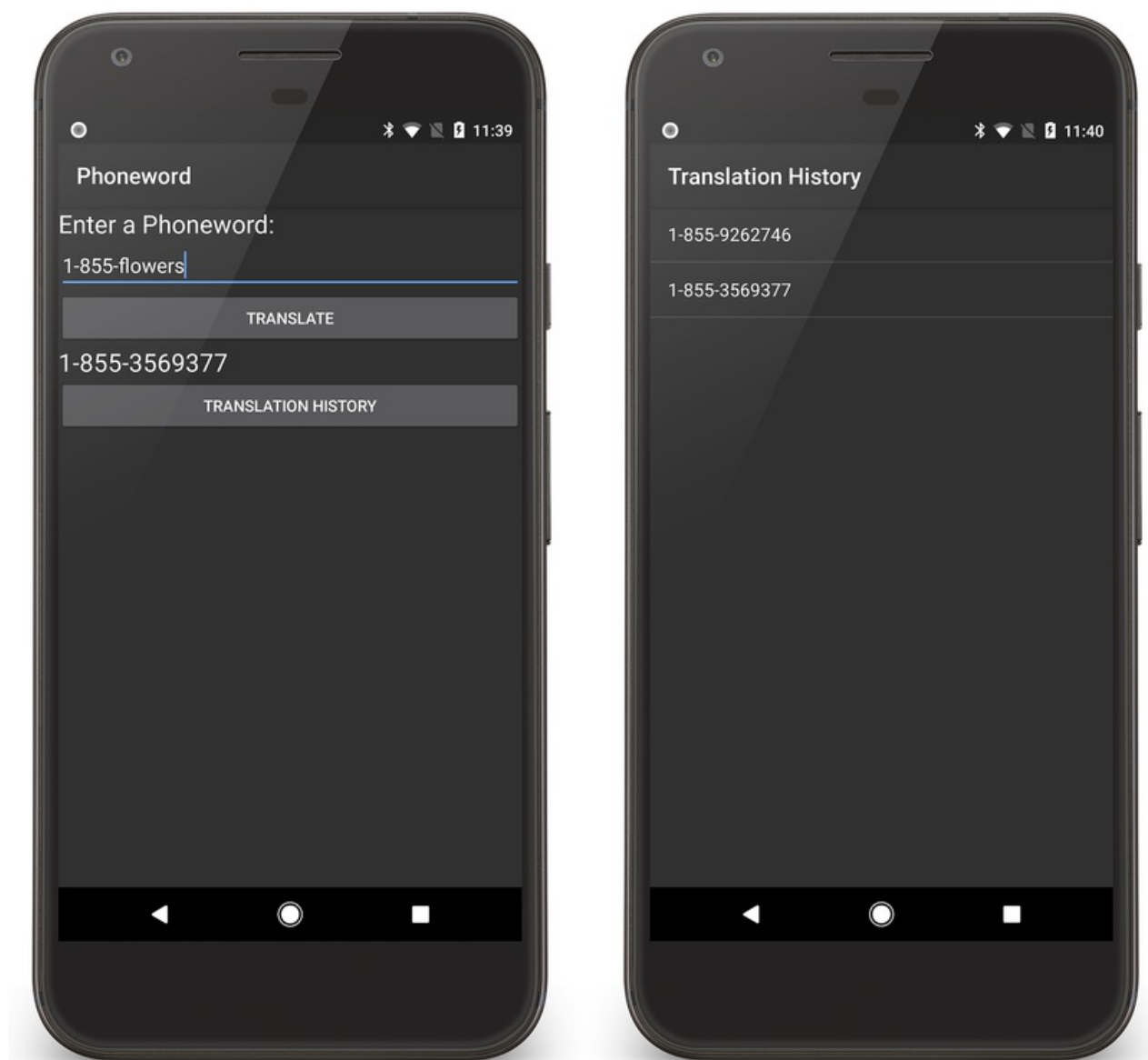
- [Android 入门](#)
- [在 Visual Studio 中进行调试](#)
- [Visual Studio for Mac 方案 - 调试](#)

了解 Android 多屏显示:快速入门

2018/11/2 • [Edit Online](#)

本指南分为两部分, 介绍了扩展 Phoneword 应用程序以处理第二个屏幕的情况。与此同时, 介绍了基础的 Android 应用程序构建基块, 便于更深入探讨 Android 体系结构。

在本指南的演练部分中, 将向 [Phoneword](#) 应用程序添加第二个屏幕, 用于跟踪使用此应用转换的号码的相关历史记录。**最终的应用程序**将具有第二个屏幕, 可显示“已转换”的号码, 如右侧屏幕截图所示:



附随的[深入了解](#)将回顾构建的内容, 并讨论体系结构、导航和此过程中遇到的其他 Android 新概念。

要求

由于本指南紧接[了解 Android](#)中的内容, 因此需要完成[了解 Android 快速入门](#)。如果想要直接跳到以下演练, 可下载完整版的 [Phoneword](#)(参见“了解 Android 快速入门”), 然后使用该版本进行演练。

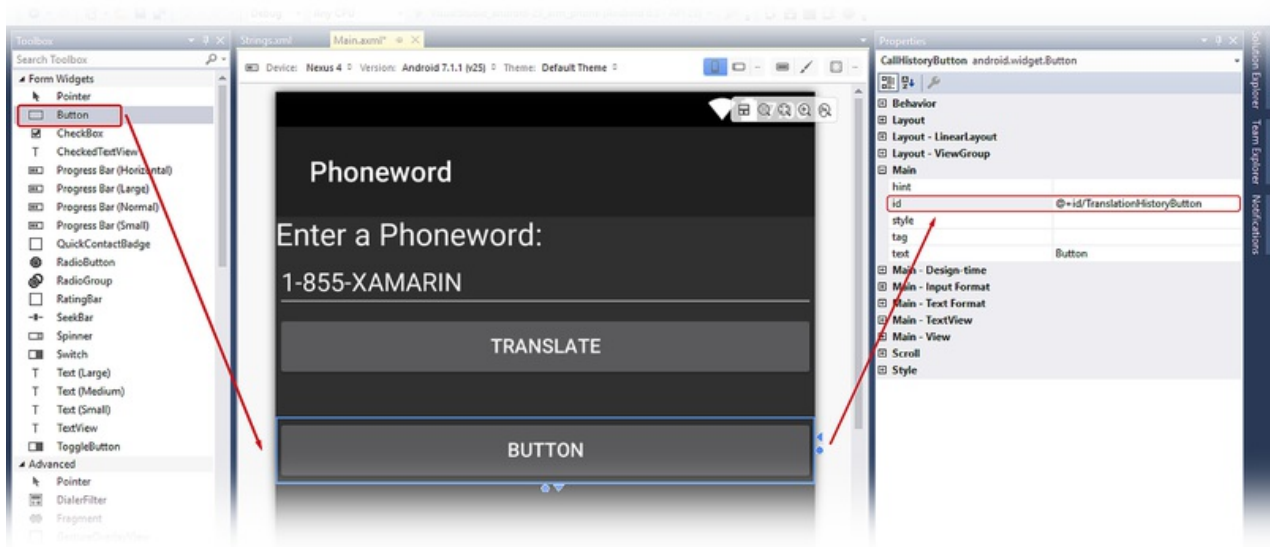
演练

在本演练中, 将向 Phoneword 应用程序添加一个“转换历史记录”屏幕。

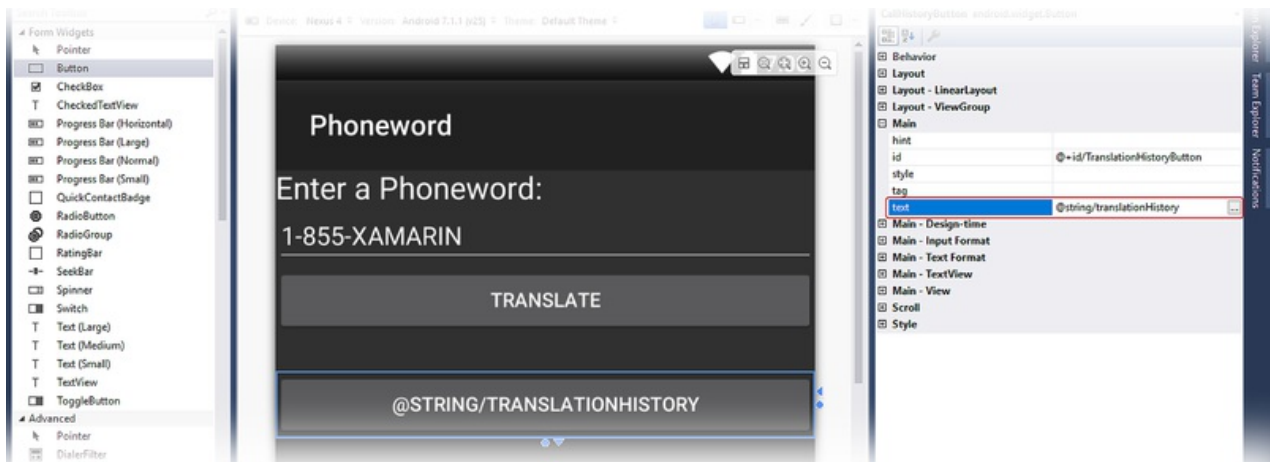
首先在 Visual Studio 中打开 Phoneword 应用程序，然后从“解决方案资源管理器”中编辑 Main.xml 文件。

更新布局

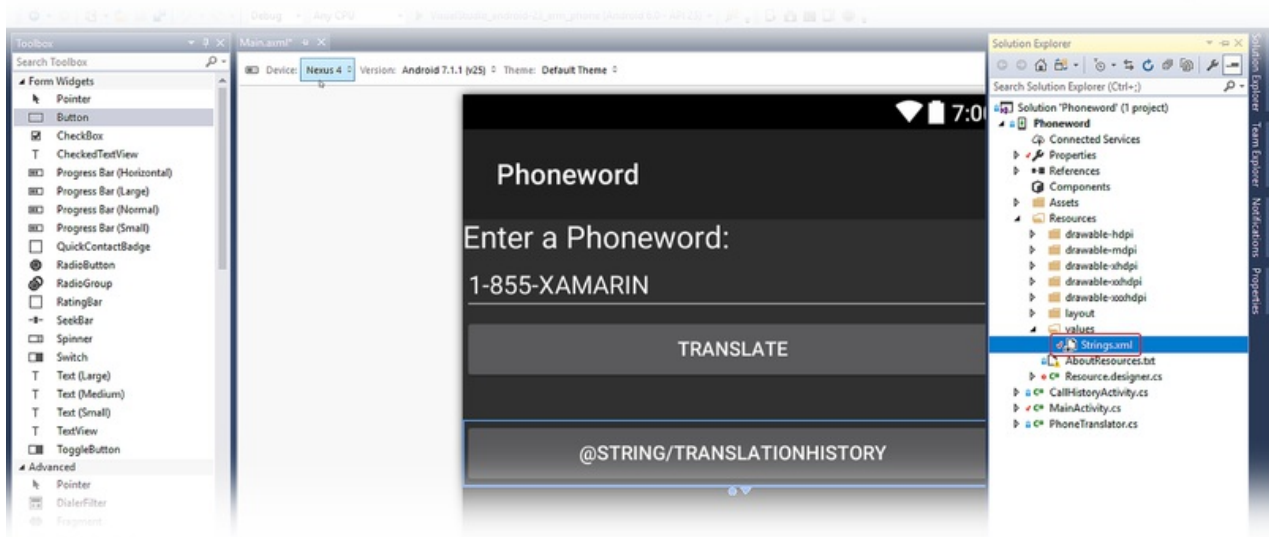
从“工具箱”中将“按钮”拖动到 Design Surface 上，然后将其置于“TranslatedPhoneWord”TextView 下方。在“属性面板”窗格中，将按钮“ID”更改为 `@+id/TranslationHistoryButton`



将按钮的 **Text** 属性设为 `@string/translationHistory`。Android 设计器将按字面意思解读此属性，但用户需要做些更改，使按钮的文本正确显示：



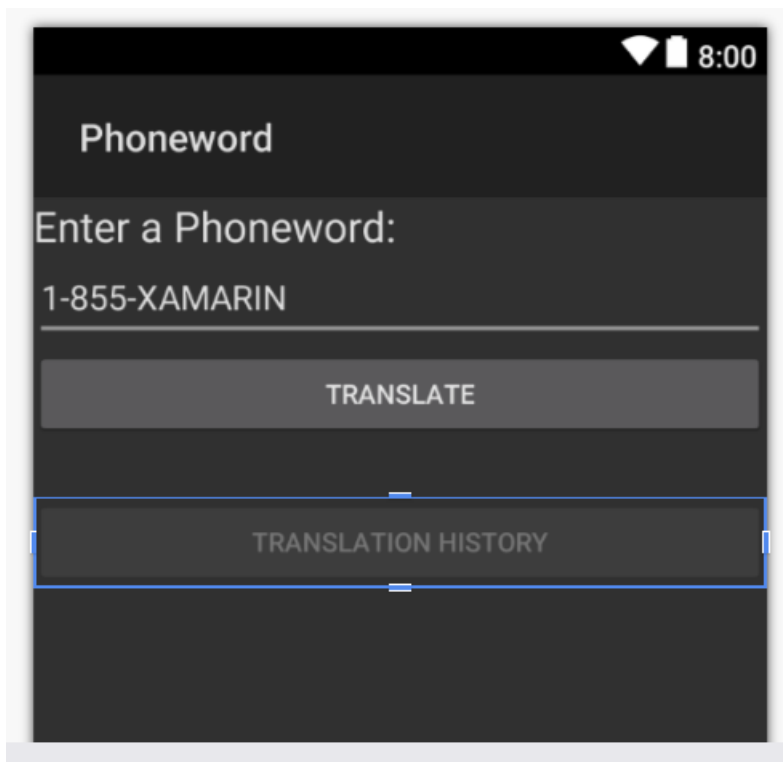
在解决方案资源管理器的“资源”文件夹下展开“值”节点，然后双击字符串资源文件 **Strings.xml**：



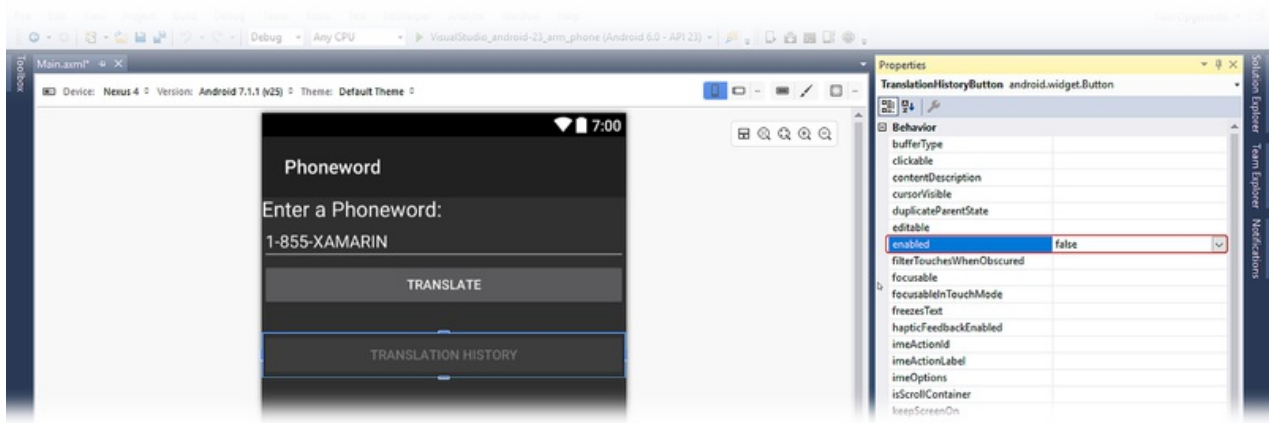
向 **Strings.xml** 文件添加 `translationHistory` 字符串名称和值，然后保存该文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="translationHistory">Translation History</string>
    <string name="ApplicationName">Phoneword</string>
</resources>
```

“转换历史记录”按钮文本应会更新以反映新的字符串值：

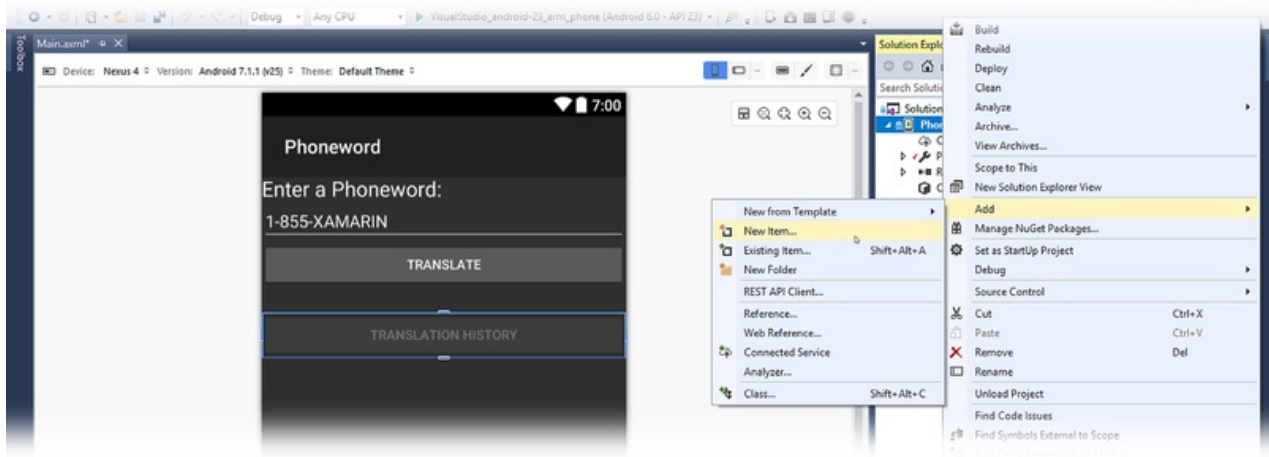


在 Design Surface 上选中“转换历史记录”按钮后，在“属性面板”窗格中查找 `enabled` 设置，然后将其值设为 `false` 以禁用此按钮。这将导致按钮在设计图面上颜色变暗：



创建第二个活动

再创建一个“活动”以支持第二个屏幕。在解决方案资源管理器中，右键单击 **Phoneword** 项目，然后选择“添加”>“新项...”：



在“添加新项”对话框中，选择“Visual C#”>“活动”，然后将活动文件命名为 TranslationHistoryActivity.cs。

将 TranslationHistoryActivity.cs 中的模板代码替换为以下代码：

```
using System;
using System.Collections.Generic;
using Android.App;
using Android.OS;
using Android.Widget;
namespace Phoneword
{
    [Activity(Label = "@string/translationHistory")]
    public class TranslationHistoryActivity : ListActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            // Create your application here
            var phoneNumbers = Intent.Extras.GetStringArrayList("phone_numbers") ?? new string[0];
            this.ListAdapter = new ArrayAdapter<string>(this, Android.Resource.Layout.SimpleListItem1,
            phoneNumbers);
        }
    }
}
```

在此类中，将按编程方式创建和填充 `ListActivity`，因此无需新建此活动的布局文件。有关更详细的信息，请参阅[了解 Android 多屏显示详述](#)。

添加列表

此应用会收集电话号码(用户已在第一个屏幕上转换的)，然后传递给第二个屏幕。电话号码以字符串列表的形式存储。若要支持列表(和稍后使用的“意向”)，请将以下 `using` 指令添加到 `MainActivity.cs` 顶部：

```
using System.Collections.Generic;
using Android.Content;
```

然后请创建可使用电话号码填充的空白列表。`MainActivity` 类将如下所示：

```
[Activity(Label = "Phoneword", MainLauncher = true)]
public class MainActivity : Activity
{
    static readonly List<string> phoneNumbers = new List<string>();
    ...// OnCreate, etc.
}
```

在 `MainActivity` 类中，添加以下代码以注册“转换历史记录”按钮(将此行放在 `translateButton` 声明后)：

```
Button translationHistoryButton = FindViewById<Button> (Resource.Id.TranslationHistoryButton);
```

将以下代码添加到 `OnCreate` 方法的末尾，以关联“转换历史记录”按钮：

```
translationHistoryButton.Click += (sender, e) =>
{
    var intent = new Intent(this, typeof(TranslationHistoryActivity));
    intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
    StartActivity(intent);
};
```

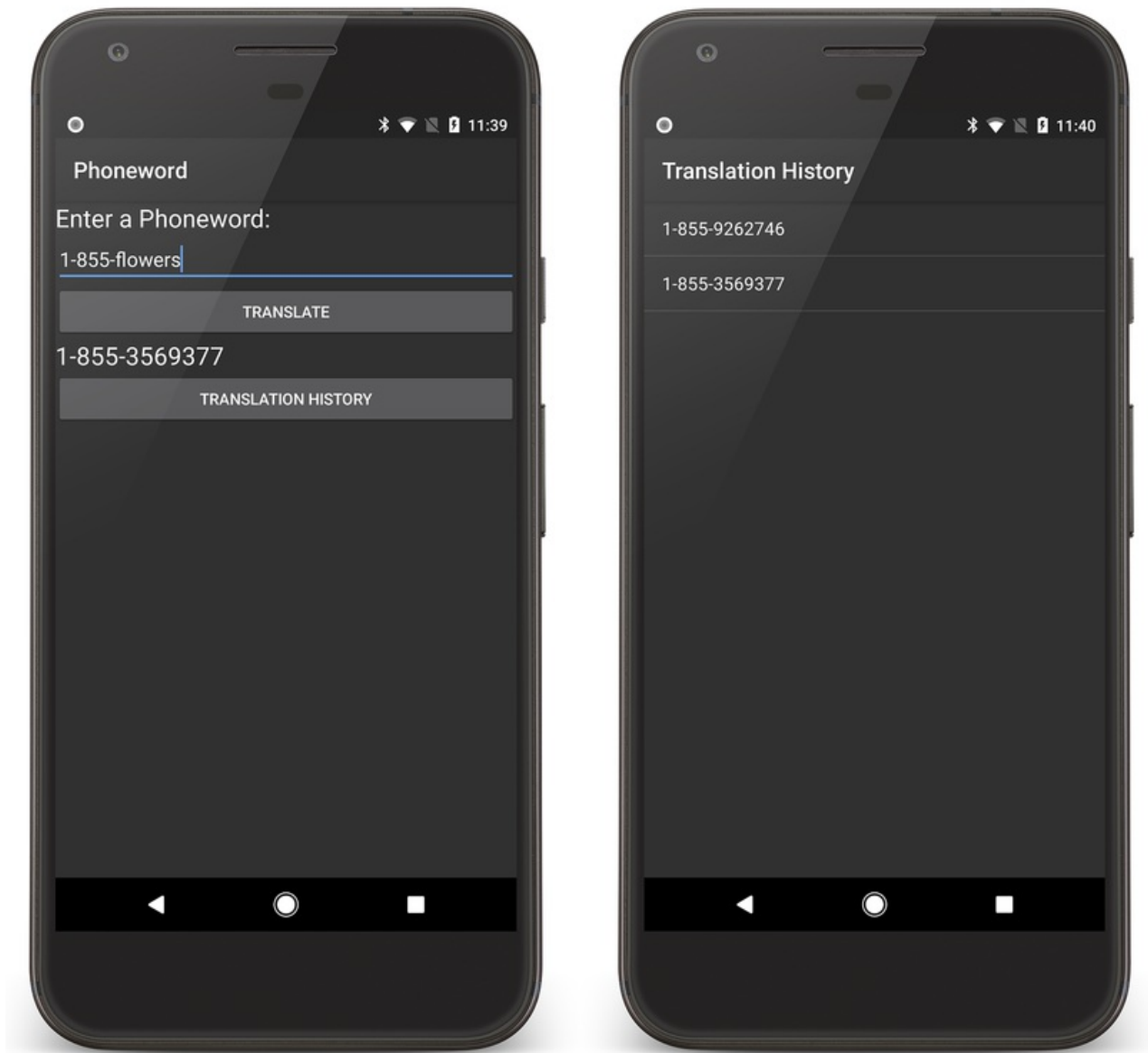
更新“转换”按钮将电话号码添加到 `phoneNumbers` 列表。用于 `TranslateHistoryButton` 的 `Click` 处理程序应与以下代码类似：

```
// Add code to translate number
string translatedNumber = string.Empty;
translateButton.Click += (sender, e) =>
{
    // Translate user's alphanumeric phone number to numeric
    translatedNumber = PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (string.IsNullOrEmpty(translatedNumber))
    {
        translatedPhoneWord.Text = "";
    }
    else
    {
        translatedPhoneWord.Text = translatedNumber;
        phoneNumbers.Add(translatedNumber);
        translationHistoryButton.Enabled = true;
    }
};
```

保存并生成应用程序，确保没有错误。

运行应用程序

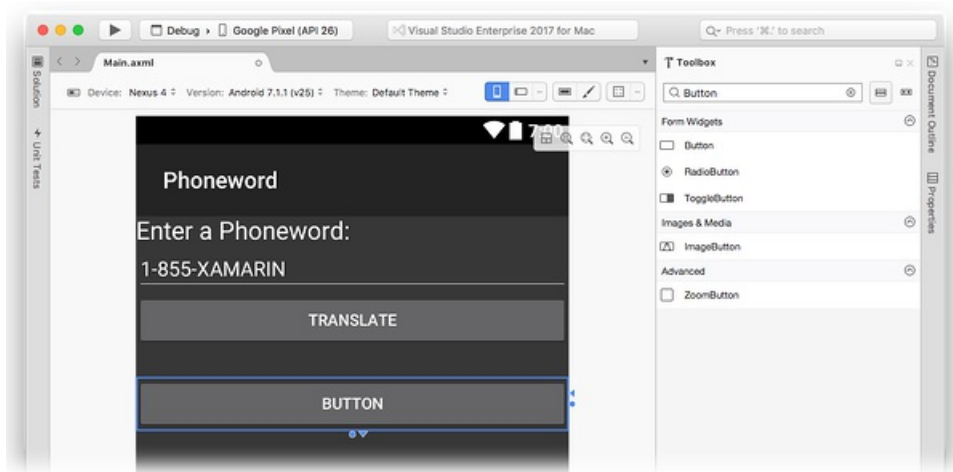
向仿真器或设备部署应用程序。下面的屏幕截图描述了正在运行的 **Phoneword** 应用程序：



首先在 Visual Studio for Mac 中打开 Phoneword 项目，然后从“Solution Pad”中编辑 Main.xml 文件。

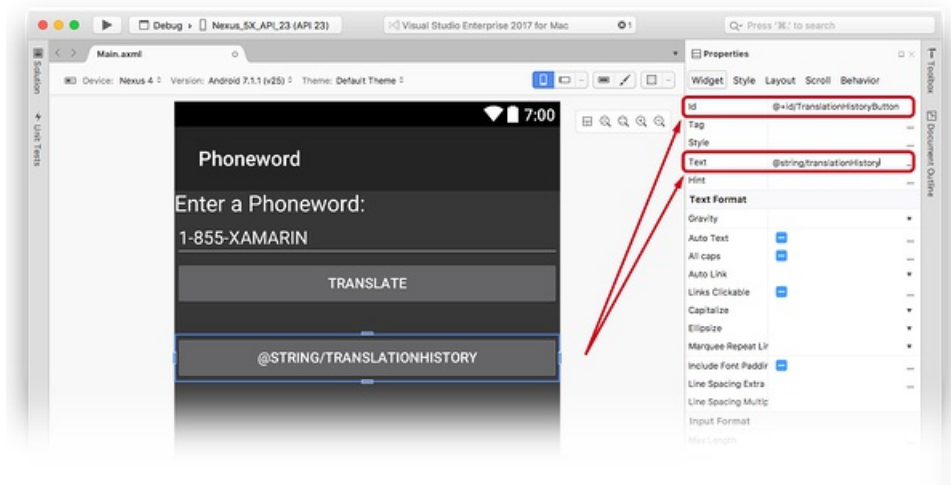
更新布局

从“工具箱”中将“按钮”拖动到 Design Surface 上，然后将其置于“TranslatedPhoneWord”TextView 下方。在“Properties Pad”中，将按钮“ID”更改为 `@+id/TranslationHistoryButton`：

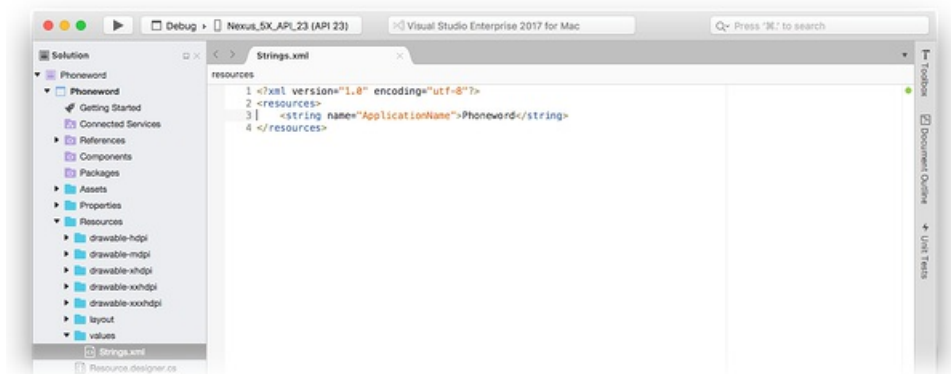


将按钮的 **Text** 属性设为 `@string/translationHistory`。Android 设计器将按字面意思解读此属性，但用户需要做些

更改，使按钮的文本正确显示：



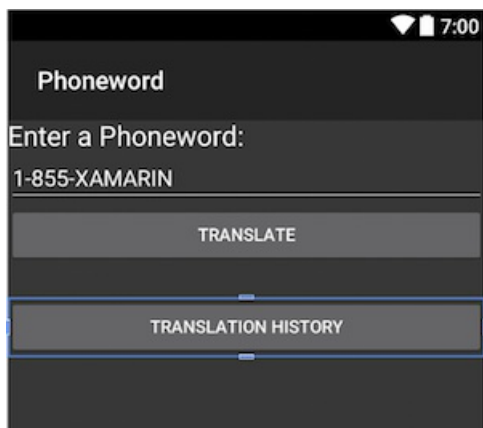
在“Solution Pad”中的“Resources”文件夹下展开“values”节点，然后双击字符串资源文件 Strings.xml：



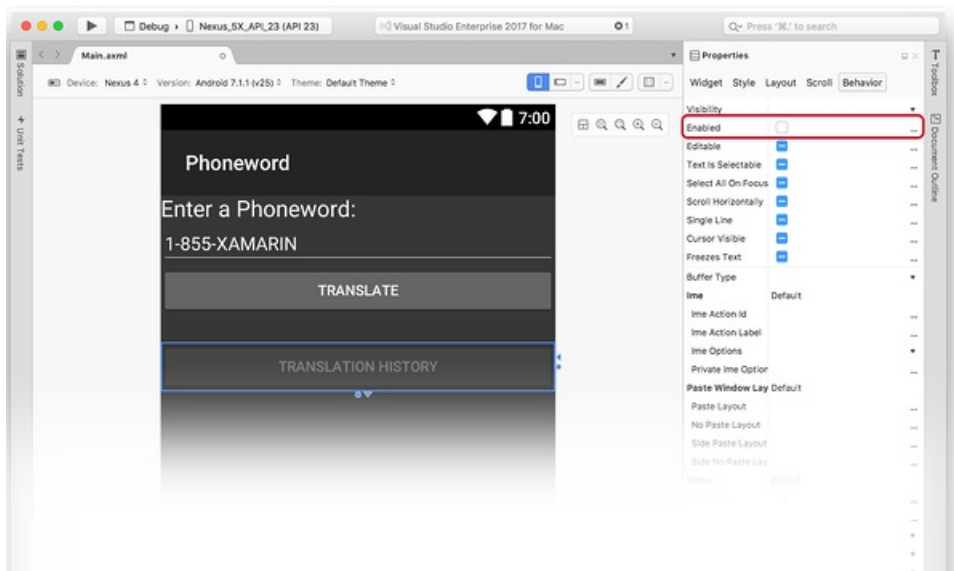
向 **Strings.xml** 文件添加 `translationHistory` 字符串名称和值，然后保存该文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="translationHistory">Translation History</string>
  <string name="ApplicationName">Phoneword</string>
</resources>
```

“转换历史记录”按钮文本应会更新以反映新的字符串值：



在 Design Surface 上选中“转换历史记录”按钮后，在“Properties Pad”中打开“行为”选项卡，然后双击“已启用”复选框以禁用此按钮。这将导致按钮在设计图面上颜色变暗：



创建第二个活动

再创建一个“活动”以支持第二个屏幕。在解决方案面板中，单击 **Phoneword** 项目旁的灰色齿轮图标，然后选择“添加”>“新文件...”：

在“新文件”对话框中，选择“Android”>“活动”，将活动命名为 `TranslationHistoryActivity`，然后单击“添加”。

将 `TranslationHistoryActivity` 中的模板代码替换为以下代码：

```
using System;
using System.Collections.Generic;
using Android.App;
using Android.OS;
using Android.Widget;
namespace Phoneword
{
    [Activity(Label = "@string/translationHistory")]
    public class TranslationHistoryActivity : ListActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            // Create your application here
            var phoneNumbers = Intent.Extras.GetStringArrayList("phone_numbers") ?? new string[0];
            this.ListAdapter = new ArrayAdapter<string>(this, Android.Resource.Layout.SimpleListItem1,
            phoneNumbers);
        }
    }
}
```

在此类中，将按编程方式创建和填充 `ListActivity`，因此无需新建该活动的布局文件。有关更详细的信息，请参阅[了解 Android 多屏显示详述](#)。

添加列表

此应用会收集电话号码(用户已在第一个屏幕上转换的)，然后传递给第二个屏幕。电话号码以字符串列表的形式存储。若要支持列表(和稍后使用的“意向”)，请将以下 `using` 指令添加到 `MainActivity.cs` 顶部：

```
using System.Collections.Generic;
using Android.Content;
```

然后请创建可使用电话号码填充的空白列表。 `MainActivity` 类将如下所示：

```
[Activity(Label = "Phoneword", MainLauncher = true)]
public class MainActivity : Activity
{
    static readonly List<string> phoneNumbers = new List<string>();
    ...// OnCreate, etc.
}
```

在 `MainActivity` 类中，添加以下代码以注册“转换历史记录”按钮（将此行放在 `TranslationHistoryButton` 声明后）：

```
Button translationHistoryButton = FindViewById<Button> (Resource.Id.TranslationHistoryButton);
```

将以下代码添加到 `OnCreate` 方法的末尾，以关联“转换历史记录”按钮：

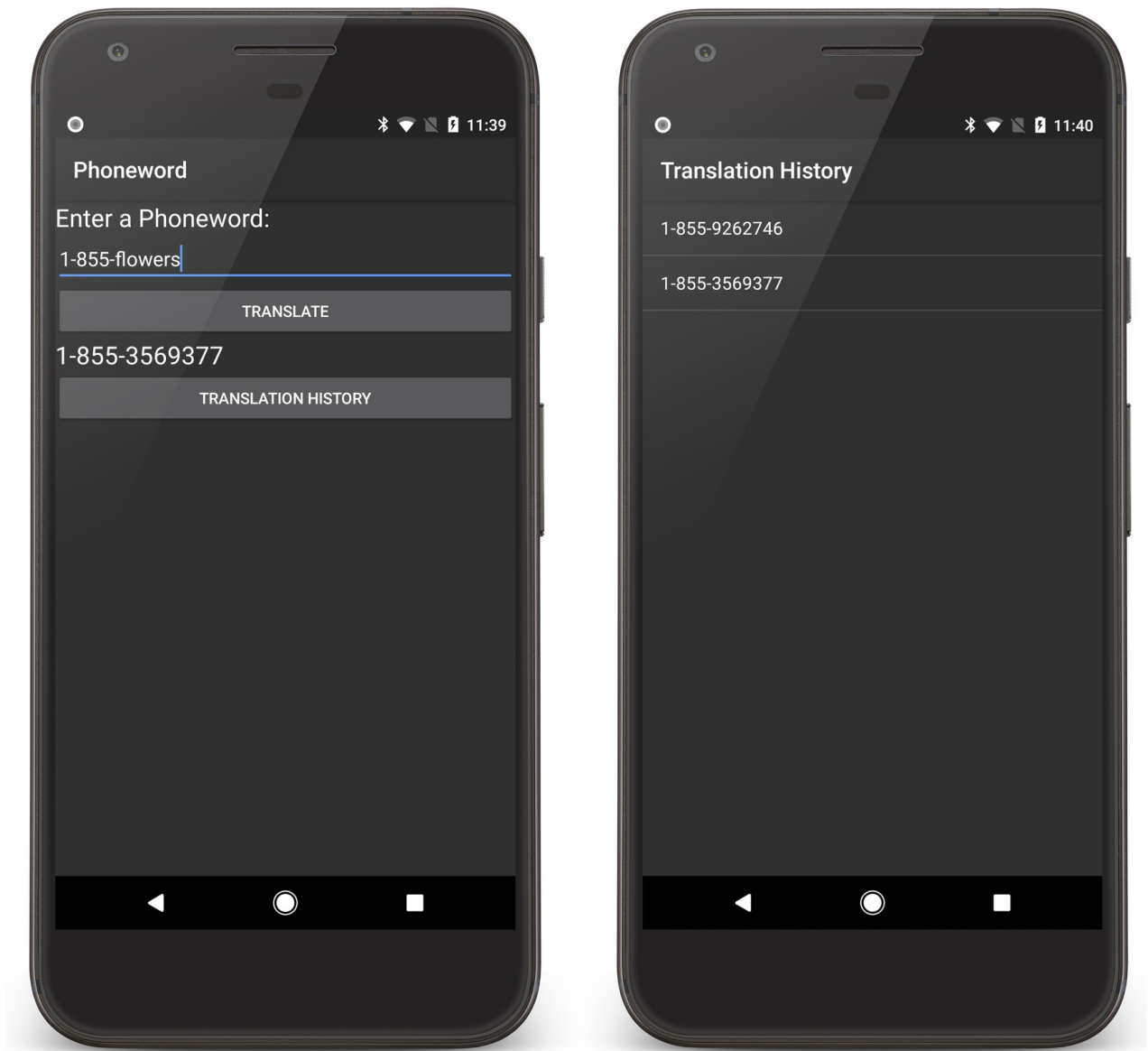
```
translationHistoryButton.Click += (sender, e) =>
{
    var intent = new Intent(this, typeof(TranslationHistoryActivity));
    intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
    StartActivity(intent);
};
```

更新“转换”按钮将电话号码添加到 `phoneNumbers` 列表。用于 `TranslateHistoryButton` 的 `Click` 处理程序应与以下代码类似：

```
translateButton.Click += (sender, e) =>
{
    // Translate user's alphanumeric phone number to numeric
    translatedNumber = PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (string.IsNullOrEmpty(translatedNumber))
    {
        translatedPhoneWord.Text = "";
    }
    else
    {
        translatedPhoneWord.Text = translatedNumber;
        phoneNumbers.Add(translatedNumber);
        translationHistoryButton.Enabled = true;
    }
};
```

运行应用程序

向仿真器或设备部署应用程序。下面的屏幕截图描述了正在运行的 **Phoneword** 应用程序：



恭喜, 你完成了第一个多屏 Xamarin.Android 应用程序! 现在可仔细分析刚才所学的工具和技能 – 接下来介绍了[了解 Android 多屏显示详述](#)。

相关链接

- [Xamarin 应用图标和启动屏幕 \(ZIP\)](#)
- [Phoneword\(示例\)](#)
- [PhonewordMultiscreen\(示例\)](#)

Hello, Android 多屏幕:深入了解

2018/11/2 • [Edit Online](#)

在这个两部分的指南中, 会扩展基本 *Phoneword* 应用程序(在“*Hello, Android*”指南中创建)以便处理第二个屏幕。在此过程中, 引入了基本 *Android* 应用程序构建基块。包括对 *Android* 体系结构的更深入了解, 以帮助你更好地了解 *Android* 应用程序结构和功能。

在 [Hello, Android 多屏幕快速入门](#) 中, 你生成并运行了第一个多屏幕 *Xamarin.Android* 应用程序。

本指南将探讨更高级的 *Android* 体系结构。说明了具有意向的 *Android* 导航, 并探讨了 *Android* 硬件导航选项。剖析了对 *Phoneword* 应用添加的新功能, 同时你会对应用程序与操作系统和其他应用程序之间的关系形成更全面的观点。

Android 体系结构基础知识

在 [Hello, Android 深入了解](#) 中, 你了解到 *Android* 应用程序是独有的程序, 因为它们缺少单一入口点。相反, 操作系统(或其他应用程序)可启动应用程序的任何一个已注册活动, 这进而会启动应用程序的进程。此 *Android* 体系结构深入了解通过介绍 *Android* 应用程序构建基块及其功能, 扩展了你对 *Android* 应用程序的构造原理的了解。

Android 应用程序构建基块

Android 应用程序由特殊 *Android* 类的集合组成, 这些类称为应用程序块, 与任何数量的应用资源(图像、主题、帮助程序类等)捆绑在一起。– 这些类通过称为 *Android* 清单的 XML 文件进行协调。

应用程序块组成 *Android* 应用程序的主干, 因为它们使你可以执行使用常规类通常无法完成的操作。两个最重要的块是 `_活动_` 和 `_服务_`:

- **活动** – 活动与具有用户界面的屏幕对应, 在概念上类似于 Web 应用程序中的网页。例如, 在新闻源应用程序中, 登录屏幕会是第一个活动, 新闻项的可滚动列表会是另一个活动, 而每个项的详细信息页面会是第三个活动。可以在[活动生命周期](#)指南中了解有关活动的详细信息。
- **服务** – *Android* 服务通过接管长时间运行的任务并在后台运行它们来支持活动。服务没有用户界面, 用于处理未绑定到屏幕的任务 – 例如, 在后台播放歌曲或将照片上传到服务器。有关服务的详细信息, 请参阅[创建服务](#)和 [Android 服务](#)指南。

Android 应用程序可能不会使用所有类型的块, 通常具有一种类型的多个块。例如, 来自 [Hello, Android 快速入门](#)的 *Phoneword* 应用程序只由一个活动(屏幕)和一些资源文件组成。简单音乐播放器应用可能具有多个活动以及一个用于在应用处于后台时播放音乐的服务。

意向

Android 应用程序中的另一个基本概念是意向。*Android* 围绕最小特权原则进行设计 – 应用程序只能访问它们正常工作所需的块, 它们对组成操作系统或其他应用程序的块具有有限的访问权限。同样, 块是松散耦合的 – 它们设计为对其他块(甚至是属于同一应用程序的块)知之甚少且仅有有限的访问权限。

为了进行通信, 应用程序块会来回发送异步消息(称为意向)。意向包含有关接收块的信息, 有时还包含一些数据。从一个应用组件发送的意向会触发某个事件在其他应用组件中发生, 从而将两个应用组件绑定在一起并允许它们进行通信。通过来回发送意向, 你可以使块来协调复杂操作(如启动相机应用以进行拍摄和保存、收集位置的信息或从一个屏幕导航到下一个屏幕)。

AndroidManifest.XML

将一个块添加到应用程序时, 它会向称为 **Android** 清单的特殊 XML 文件进行注册。清单会跟踪应用程序中的所有应用程序块, 以及版本要求、权限和链接库 – 操作系统为使应用程序正常运行而需要了解的所有内容。

Android 清单也可与活动和意向配合工作，以控制适合于给定活动的操作。[使用 Android 清单指南](#)中介绍了 Android 清单的这些高级功能。

在单屏幕版本的 Phoneword 应用程序中，只使用了一个活动、一个意向和 `AndroidManifest.xml`，以及其他资源（如图标）。在多屏幕版本的 Phoneword 中，添加了一个其他活动；它使用意向从第一个活动启动。下一部分探讨意向如何帮助在 Android 应用程序中创建导航。

Android 导航

意向过去用于在屏幕之间进行导航。现在可以深入了解此代码，以了解意向的工作原理并了解它们在 Android 导航窗格中的角色。

使用意向启动第二个活动

在 Phoneword 应用程序中，意向用于启动第二个屏幕（活动）。首先创建一个意向，传入当前上下文（`this`，用于引用当前上下文）以及你所查找的应用程序块的类型（`TranslationHistoryActivity`）：

```
Intent intent = new Intent(this, typeof(TranslationHistoryActivity));
```

上下文是与有关应用程序环境的全局信息之间的接口 – 它使新创建的对象可以了解应用程序的进展情况。如果将意向视为消息，则要提供消息收件人的名称（`TranslationHistoryActivity`）和接收方的地址（`Context`）。

Android 提供一个选项，用于将简单数据附加到意向（复杂数据以不同方式进行处理）。在 Phoneword 示例中，`PutStringArrayListExtra` 用于将电话号码列表附加到意向，而 `StartActivity` 会对意向的收件人进行调用。完整代码如下所示：

```
translationHistoryButton.Click += (sender, e) =>
{
    var intent = new Intent(this, typeof(TranslationHistoryActivity));
    intent.PutStringArrayListExtra("phone_numbers", _phoneNumbers);
    StartActivity(intent);
};
```

Phoneword 中引入的其他概念

Phoneword 应用程序引入了多个本指南中未提及的概念。这些概念包括：

字符串资源 – 在 Phoneword 应用程序中，`TranslationHistoryButton` 的文本设置为

`"@string/translationHistory"`。 `@string` 语法表示字符串的值存储在_字符串资源文件_（**Strings.xml**）中。`translationHistory` 字符串的以下值已添加到 **Strings.xml** 中：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="translationHistory">Call History</string>
</resources>
```

有关字符串资源和其他 Android 资源的详细信息，请参阅 [Android 资源指南](#)。

ListView 和 ArrayAdapter – *ListView* 是一个 UI 组件，它提供了显示行的滚动列表的简单方法。`ListView` 实例需要_适配器_，以向它馈送行视图中包含的数据。下面的代码行用于填充 `TranslationHistoryActivity` 的用户界面：

```
this.ListAdapter = new ArrayAdapter<string>(this, Android.Resource.Layout.SimpleListItem1, phoneNumbers);
```

ListView 和适配器不在此文档的讨论范围内，但它们在非常全面的 [ListView 和适配器指南](#)中进行了介绍。[使用数](#)

据填充 `ListView` 专门处理使用内置 `ListActivity` 和 `ArrayAdapter` 类来创建并填充 `ListView`，而无需定义自定义布局(如 Phoneword 示例中所进行的那样)。

总结

祝贺，你已完成第一个多屏 Android 应用程序！本指南介绍了 *Android 应用程序构建基块和意向*，并使用它们生成了一个多屏幕 Android 应用程序。现在，你已具有坚实的基础，可开始开发自己的 Xamarin.Android 应用程序。

接下来，你会在[生成跨平台应用程序指南](#)中了解如何使用 Xamarin 生成跨平台应用程序。

面向 Java 开发人员的 Xamarin

2018/10/26 • [Edit Online](#)

如果你是 Java 开发人员, 则可以在 Xamarin 平台上充分利用你的技能和现有代码, 同时获得 C# 的代码重用优势。你会发现 C# 语法与 Java 语法非常相似, 这两种语言提供非常类似的功能。此外, 你会发现 C# 的特有功能, 这些功能将帮助你轻松进行开发工作。

概述

本文介绍面向 Java 开发人员的 C# 编程, 主要侧重于在开发 Xamarin.Android 应用程序时会遇到的 C# 语言功能。此外, 本文说明了这些功能与其 Java 对应项的区别所在, 并介绍了在 Java 中不可用的重要 C# 功能(与 Xamarin.Android 相关)。包含指向附加参考资料的链接, 因此你可以将本文用作一个“起点”以进一步研究 C# 和 .NET。

如果你熟悉 Java, 那么对于 C# 语法的使用便可以轻松上手。C# 语法与 Java 语法非常相似 – C# 是“大括号”语言, 如 Java、C 和 C++。在许多方面, C# 语法读起来像是 Java 语法的超集, 但是有一些重命名和新增的关键字。

可以在 C# 中找到 Java 的许多主要特征:

- 面向对象的基于类的编程
- 强类型化
- 支持接口
- 泛型
- 垃圾回收
- 运行时编译

Java 和 C# 都被编译为中间语言, 在托管执行环境中运行。C# 和 Java 都是静态类型, 这两种语言将字符串视为不可变类型。这两种语言使用单根类层次结构。和 Java 一样, C# 仅支持单一继承, 不支持全局方法。在这两种语言中, 对象使用 `new` 关键字在堆上创建而成, 当不再使用对象时, 它们会被作为垃圾回收处理。这两种语言提供带 `try` / `catch` 语义的正式异常处理支持。两者都提供线程管理和同步支持。

但是, Java 和 C# 之间存在很多不同。例如:

- Java 不支持隐式类型的局部变量(C# 支持 `var` 关键字)。
- 在 Java 中, 可以仅按值传递参数, 而在 C# 中, 你可以按引用以及值进行传递。(C# 提供 `ref` 和 `out` 关键字, 用于按引用传递参数;Java 中无此类等效项)。
- Java 不支持预处理器指令, 如 `#define`。
- Java 不支持无符号的整数类型, 而 C# 提供无符号的整数类型, 如 `ulong`、`uint`、`ushort` 和 `byte`。
- Java 不支持运算符重载;在 C# 中, 你可以重载运算符和转换。
- 在 Java `switch` 语句中, 代码可以贯穿到下一个 `switch` 部分, 但在 C# 中, 每个 `switch` 部分的结尾必须终止 `switch`(每个部分的结尾必须以 `break` 语句结束)。
- 在 Java 中, 指定由带 `throws` 关键字的方法引发的异常, 但 C# 没有检查异常的概念 – `throws` 关键字在 C# 中不受支持。
- C# 支持语言集成查询 (LINQ), 这样就可以使用保留的字 `from`、`select` 和 `where`, 以便以类似于数据库查

询的方式编写针对集合的查询。

当然，与本文所讨论的相比，C# 和 Java 之间还有更多的区别。此外，Java 和 C# 将继续演进（例如，Android 工具链中尚不存在的 Java 8，它支持 C# 样式的 lambda 表达式），因此这些差异会随时间而变化。本文仅概述了首次使用 Xamarin.Android 的 Java 开发人员当前遇到的最重要的差异。

- [从 Java 到 C# 开发](#)介绍了 C# 和 Java 之间的基本区别。
- [面向对象的编程功能](#)概述了两种语言之间最重要的面向对象的功能区别。
- [关键字差异](#)提供了一个表格，其中包含有效的关键字等效项、仅限 C# 的关键字以及指向 C# 关键字定义[的链接](#)。

C# 为 Xamarin.Android 提供了许多主要功能，Java 开发人员当前尚不可在 Android 上使用这些功能。这些功能可帮助你在更短的时间内编写出更好的代码：

- [属性](#) – 可通过 C# 属性系统直接安全地访问成员变量，而无需编写 setter 和 getter 方法。
- [Lambda 表达式](#) – 在 C# 中，你可以使用匿名方法（也称为“lambda”）更简洁、更高效地表示你的功能。你可以避免编写一次性对象的开销，并且可以向某个方法传递本地状态而无需添加参数。
- [事件处理](#) – C# 为事件驱动的编程提供语言级别支持，对象可以进行注册，以便在出现感兴趣的事件时接收通知。`event` 关键字定义发布服务器类可用于通知事件订阅者的多播广播机制。
- [异步编程](#) – C# 的异步编程功能（`async` / `await`）使应用保持响应状态。此功能的语言级别支持使异步编程轻松实现，且不容易出错。

最后，Xamarin 允许你通过已知的绑定技术来[利用现有的 Java 资产](#)。你可以通过使用 Xamarin 的自动绑定生成器，从 C# 调用现有的 Java 代码、框架和库。若要执行此操作，只需在 Java 中创建静态库，并通过绑定将其公开到 C#。

从 Java 到 C# 开发

以下各节概述了 C# 和 Java 之间的基本“入门”差异；后面的部分将介绍这些语言之间面向对象的差异。

库和程序集

Java 通常将相关类打包到 .jar 文件中。然而，在 C# 和 .NET 中，预编译代码的可重用位将打包到程序集，通常打包为 .dll 文件。程序集是部署 C#/.NET 代码的单位，每个程序集通常与 C# 项目相关联。程序集包含在运行时实时编译的中间代码 (IL)。

有关程序集的详细信息，请参阅 MSDN [程序集和全局程序集缓存](#)主题。

包和命名空间

C# 使用 `namespace` 关键字对相关类型进行分组；这类似于 Java 的 `package` 关键字。通常情况下，Xamarin.Android 应用将驻留在为该应用创建的命名空间中。例如，下面的 C# 代码声明天气报告应用的 `WeatherApp` 命名空间包装器：

```
namespace WeatherApp
{
    ...
}
```

导入类型

在使用外部命名空间中定义的类型时，使用 `using` 语句导入这些类型（近似于 Java `import` 语句）。在 Java 中，你可能使用以下语句导入单个类型：

```
import javax.swing.JButton
```


可能会使用类似于下面的语句导入整个 Java 包：

```
import javax.swing.*
```

C# `using` 语句的工作方式非常类似，但它允许导入整个包而无需指定通配符。例如，在 Xamarin.Android 源文件的开头，通常会看到一系列 `using` 语句，如以下示例所示：

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using System.Net;
using System.IO;
using System.Json;
using System.Threading.Tasks;
```

这些语句从 `System`、`Android.App`、`Android.Content` 等命名空间导入功能。

泛型

Java 和 C# 支持泛型，这是允许你在编译时插入不同类型中的占位符。但是，泛型在 C# 中的工作方式略有不同。在 Java 中，[类型擦除](#)仅在编译时提供类型信息，而不是在运行时。与此相反，.NET 公共语言运行时 (CLR) 对泛型类型提供显式支持，这意味着 C# 有权在运行时访问类型信息。在日常 Xamarin.Android 开发中，这一区别的重要性通常不明显，但如果使用的是[反射](#)，你将依赖此功能在运行时访问类型信息。

在 Xamarin.Android 中，通常会看到泛型方法 `FindViewById` 用于获取对布局控件的引用。此方法接受一个泛型类型参数，指定要查找的控件类型。例如：

```
TextView label = FindViewById<TextView> (Resource.Id.Label);
```

在此代码示例中，`FindViewById` 将布局中定义的 `TextView` 控件作为标签引用，然后将其作为 `TextView` 类型返回。

有关泛型的详细信息，请参阅 MSDN [泛型](#)主题。请注意，Xamarin.Android 对泛型 C# 类的支持存在一些限制；有关详细信息，请参阅[限制](#)。

面向对象的编程功能

Java 和 C# 使用非常相似的面向对象的编程惯用语：

- 所有类最终都派生自单个根对象，所有 Java 对象均派生自 `java.lang.Object`，而所有 C# 对象均派生自 `System.Object`。
- 类实例为引用类型。
- 当你访问某个实例的属性和方法时，可以使用“.”运算符。
- 所有类实例均通过 `new` 运算符在堆上创建。
- 由于这两种语言都使用垃圾回收，因此有一种方法可以显式释放未使用的对象（即，不像 C++ 一样存在 `delete` 关键字）。
- 可以通过继承扩展类，并且这两种语言只允许每种类型有单个基类。
- 可以定义接口，并且一个类可以继承自（即，实现）多个接口定义。

但是,也有一些重要的区别:

- Java 有两个 C# 不支持的强大功能:匿名类和内部类。(但是, C# 允许嵌套类定义 – C# 的嵌套类就像 Java 的静态嵌套类。)
- C# 支持 C 样式结构类型 (`struct`), 而 Java 不支持。
- 在 C# 中, 你可以在单独的源文件中通过使用 `partial` 关键字实现类定义。
- C# 接口不能声明字段。
- C# 使用 C++ 样式析构函数语法来表示终结器。该语法不同于 Java 的 `finalize` 方法, 但语义大致相同。(请注意, 在 C# 中, 析构函数自动调用基类析构函数 – 而在 Java 中则使用对 `super.finalize` 的显式调用。)

类继承

要扩展 Java 中的类, 可以使用 `extends` 关键字。要扩展 C# 中的类, 可以使用冒号 (`:`) 以指示派生。例如, 在 Xamarin.Android 应用中, 通常会看到类似于以下代码段的类派生:

```
public class MainActivity : Activity
{
    ...
}
```

在此示例中, `MainActivity` 继承自 `Activity` 类。

要在 Java 中声明对接口支持, 可以使用 `implements` 关键字。但是, 在 C# 中, 只需将接口名称添加到从中继承的类列表, 如以下代码段所示:

```
public class SensorsActivity : Activity, ISensorEventListener
{
    ...
}
```

在此示例中, `SensorsActivity` 继承自 `Activity`, 并实现 `ISensorEventListener` 接口中声明的功能。请注意, 接口列表必须晚于基类(否则将收到一个编译时错误)。按照约定, C# 接口名称前有一个大写“I”;这样便能确定哪些类是接口, 而无需 `implements` 关键字。

如果想要阻止类在 C# 中进一步子类化, 请在类名前添加 `sealed` – 在 Java 中, 在类名前添加 `final`。

有关 C# 类定义的更多信息, 请参阅 MSDN [类和结构](#) 以及 [继承](#) 主题。

属性

在 Java 中, 赋值函数方法 (setter) 和检查器方法 (getter) 通常用于控制在隐藏和保护类成员不受外部代码影响时如何对这些成员进行更改。例如, Android `TextView` 类提供 `getText` 和 `setText` 方法。C# 提供相似但更直接的机制, 称为“属性”。C# 类用户可以使用与访问字段相同的方式来访问属性, 但是每次访问实际上会导致对调用方透明的方法调用。此“隐秘”方法会导致一些副作用, 如设置其他值、执行转换, 或更改对象状态。

属性通常用于访问和修改 UI(用户界面)对象成员。例如:

```
int width = rulerView.MeasuredWidth;
int height = rulerView.MeasuredHeight;
...
rulerView.DrawingCacheEnabled = true;
```

在此示例中, 可通过访问 `rulerView` 对象的 `MeasuredWidth` 和 `MeasuredHeight` 属性从中读取宽度和高度值。在读取这些属性时, 其关联(但隐藏)字段值中的值会在后台提取, 并返回给调用方。`rulerView` 对象可以在一个度量单位(如像素)中存储宽度和高度的值, 并在访问 `MeasuredWidth` 和 `MeasuredHeight` 属性时将这些值实时转换为不同的度量单位(如毫米)。

`rulerView` 对象还有一个名为 `DrawingCacheEnabled` 的属性 – 示例代码将此属性设置为 `true` 以启用 `rulerView` 中的绘制缓存。在后台, 关联的隐藏字段已更新为新值, `rulerView` 状态的可能其他方面已修改。例如, 当 `DrawingCacheEnabled` 设置为 `false` 时, `rulerView` 还可能清除对象中累积的任何绘制缓存信息。

对属性的访问权限可以是读/写、只读或只写。此外, 可以使用不同的访问修饰符进行读取和写入。例如, 可以定义一个具有公共读取访问权限和专有写入访问权限的属性。

有关 C# 属性的详细信息, 请参阅 MSDN [属性](#) 主题。

调用基类方法

若要在 C# 中调用基类构造函数, 可以使用后跟关键字 `base` 的冒号 (`:`) 和初始化表达式列表; 此 `base` 构造函数调用会被立即置于派生的构造函数参数列表后。在进入派生构造函数时, 将调用基类构造函数; 编译器会在方法主体开头将此调用插入到基构造函数。下面的代码段演示了从 Xamarin.Android 应用中的派生构造函数调用的基构造函数:

```
public class PictureLayout : ViewGroup
{
    ...
    public class PictureLayout (Context context)
        : base (context)
    {
        ...
    }
    ...
}
```

在此示例中, `PictureLayout` 类派生自 `ViewGroup` 类。此示例中所示的 `PictureLayout` 构造函数接受 `context` 参数, 并通过 `base(context)` 调用将其传递给 `ViewGroup` 构造函数。

若要在 C# 中调用基类方法, 请使用 `base` 关键字。例如, Xamarin.Android 应用通常调用基方法, 如下所示:

```
public class MainActivity : Activity
{
    ...
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);
    }
}
```

在这种情况下, 由派生类 (`MainActivity`) 定义的 `OnCreate` 方法调用基类 (`Activity`) 的 `OnCreate` 方法。

访问修饰符

Java 和 C# 都支持 `public`、`private` 和 `protected` 访问修饰符。但是, C# 支持以下两个其他访问修饰符:

- `internal` – 只能在当前程序集内访问类成员。
- `protected internal` – 可在定义程序集、定义类和派生类中访问类成员定义 (程序集内部和外部的派生类都具有访问权限)。

有关 C# 访问修饰符的详细信息, 请参阅 MSDN [访问修饰符](#) 主题。

虚拟和重写方法

Java 和 C# 均支持多形性, 即能够以相同方式处理相关对象。在这两种语言中, 可以使用基类引用来引用派生类对象, 且派生类的方法可以重写基类的方法。两种语言都具有虚拟方法的概念, 此基类方法旨在替换为派生类中的方法。和 Java 一样, C# 支持 `abstract` 类和方法。

但是, Java 和 C# 之间在如何声明和重写虚拟方法方面存在一些差异:

- 默认情况下，在 C# 中，方法是非虚拟的。父类必须显式标记要使用 `virtual` 关键字重写的方法。与此相反，Java 中的所有方法都默认为虚拟方法。
- 要防止在 C# 中重写方法，只需删除 `virtual` 关键字。与此相反，Java 使用 `final` 关键字将方法标记为“不允许重写”。
- C# 派生类必须使用 `override` 关键字显式指示正在重写虚拟基类方法。

有关 C# 对多形性支持的详细信息，请参阅 MSDN [多形性](#) 主题。

Lambda 表达式

可以通过 C# 创建一个闭包：内联匿名方法，可以访问它们包含在其中的方法的状态。使用 lambda 表达式，你可以编写较少的代码行来实现相同功能，你可能已使用更多的代码行在 Java 中实现该功能。

Lambda 表达式使你可以跳过像在 Java 中创建一次性类或匿名类时会涉及到的其他繁琐操作 – 而只需编写方法代码内联的业务逻辑即可。此外，由于 lambda 对周围方法中的变量具备访问权限，因此不必创建用于将状态传递到方法代码的长参数列表。

在 C# 中，lambda 表达式通过 `=>` 运算符创建而成，如下所示：

```
(arg1, arg2, ...) => {  
    // implementation code  
};
```

在 Xamarin.Android 中，lambda 表达式通常用于定义事件处理程序。例如：

```
button.Click += (sender, args) => {  
    clickCount += 1;    // access variable in surrounding code  
    button.Text = string.Format ("Clicked {0} times.", clickCount);  
};
```

在此示例中，lambda 表达式代码（在大括号内的代码）递增点击数，并更新 `button` 文本以显示点击数。此 lambda 表达式在 `button` 对象中注册为点击按钮时要调用的点击事件处理程序。（下文对事件处理程序进行了更详细的说明。）在此简单示例中，lambda 表达式代码不使用 `sender` 和 `args` 参数，但 lambda 表达式需要这些参数来满足事件注册的方法签名要求。实质上，C# 编译器将 lambda 表达式转换为一个匿名方法，在发生按钮点击事件时会调用该方法。

有关 C# 和 lambda 表达式的详细信息，请参阅 MSDN [Lambda 表达式](#) 主题。

事件处理

事件是在对象出现一些有趣的内容时，该对象通知注册订阅者所采用的一种方式。与 Java 不同，订阅者通常在 Java 中实现 `Listener` 接口，其中包含一个回调方法，而 C# 则通过委托为事件处理提供语言级别支持。委托类似于面向对象的类型安全函数指针 – 它封装对象引用和方法令牌。如果客户端对象想要订阅事件，它将创建委托并将该委托传递给通知对象。发生事件时，通知对象将调用委托对象所表示的方法，以通知订阅事件的客户端对象。在 C# 中，事件处理程序实际上就是通过委托调用的方法。

有关委托的详细信息，请参阅 MSDN [委托](#) 主题。

在 C# 中，事件为多播；也就是说，发生某事件时可以通知多个侦听器。当你考虑 Java 和 C# 事件注册之间的语法差异时，将观察到这种差异。在 Java 中，可以调用 `SetXXXListener` 来注册事件通知；在 C# 中，可以使用 `+=` 运算符来注册事件通知，方法是将委托“添加”到事件侦听器列表。在 Java 中，可以调用 `SetXXXListener` 以取消注册，而在 C# 中，则使用 `-=` 将委托从侦听器列表中“去除”。

在 Xamarin.Android 中，事件通常用于在用户对 UI 控件执行某操作时通知对象。通常，UI 控件将具有使用 `event`

关键字定义的成员;将委托附加到这些成员,以便从该 UI 控件中订阅事件。

订阅事件:

1. 创建一个委托对象,在事件发生时引用要调用的方法。
2. 使用 `+=` 运算符将委托附加到你订阅的事件。

下面的示例定义了一个委托(显式使用 `delegate` 关键字)以订阅按钮点击事件。此按钮点击处理程序将启动新的活动:

```
startActivityButton.Click += delegate {  
    Intent intent = new Intent (this, typeof (MyActivity));  
    StartActivity (intent);  
};
```

但是,你还可以使用 lambda 表达式来注册事件,一起跳过 `delegate` 关键字。例如:

```
startActivityButton.Click += (sender, e) => {  
    Intent intent = new Intent (this, typeof (MyActivity));  
    StartActivity (intent);  
};
```

在此示例中, `startActivityButton` 对象具有一个事件,该事件期望带有特定方法签名的委托:该事件接受发件人和事件参数并返回 `void`。但是,因为我们不希望在显式定义此类委托或其方法时遇到问题,我们声明具有 `(sender, e)` 的方法签名并使用 lambda 表达式来实现事件处理程序的主体。请注意,必须声明此参数列表,即使不使用 `sender` 和 `e` 参数。

务必记住,你可以取消订阅委托(通过 `-=` 运算符),但不能取消订阅 lambda 表达式 – 试图执行此操作可能会导致内存泄漏。只有当处理程序不从事件中取消订阅时,才能使用 lambda 形式的事件注册。

通常,lambda 表达式用于声明 Xamarin.Android 代码中的事件处理程序。此用于声明事件处理程序的速记方法乍一看可能比较晦涩,但在你写入和读取代码时它将帮你节省大量的时间。随着熟悉程度的增加,你将习惯于识别此模式(这在 Xamarin.Android 代码中经常出现),可以花更多的时间思考应用程序的业务逻辑,而不需要花太多时间在语法开销上。

异步编程

异步编程是改进应用程序总体响应能力的一种方法。当应用程序的某些部分被冗长的操作所阻塞时,异步编程功能使应用代码的其余部分得以继续运行。如果应用程序未以异步方式编写,访问 Web、处理图像和读取/写入文件等操作示例可能会导致整个应用看上去像冻结了一样。

C# 包含对通过 `async` 和 `await` 关键字进行异步编程的语言级别支持。通过这些语言功能,可以非常方便地编写代码来执行长时间运行的任务,而不会阻止应用的主线程。简而言之,可以在某个方法上使用 `async` 关键字来指示方法中以异步方式运行,且不会阻止调用方线程的代码。在调用标记有 `async` 的方法时使用 `await` 关键字。编译器会将 `await` 解释为一个点,其中方法执行将移至后台线程(会向调用方返回一个任务)。完成此任务后,将在代码中 `await` 点处的调用方线程上恢复代码执行,从而返回 `async` 调用的结果。按照约定,异步运行的方法的名称带有 `Async` 后缀。

在 Xamarin.Android 应用中, `async` 和 `await` 通常用于释放 UI 线程,以便在后台任务中发生长时间运行的操作时,它可以响应用户输入(如点击“取消”按钮)。

在以下示例中,按钮点击事件处理程序将导致异步操作从 Web 下载映像:

```
downloadButton.Click += downloadAsync;
...
async void downloadAsync(object sender, System.EventArgs e)
{
    webClient = new WebClient ();
    var url = new Uri ("http://photojournal.jpl.nasa.gov/jpeg/PIA15416.jpg");
    byte[] bytes = null;

    bytes = await webClient.DownloadDataTaskAsync(url);

    // display the downloaded image ...
}
```

在此示例中，当用户单击 `downloadButton` 控件时，`downloadAsync` 事件处理程序将创建 `WebClient` 对象和 `Uri` 对象，以便从指定 URL 中提取映像。接下来，它使用此 URL 调用 `WebClient` 对象的 `DownloadDataTaskAsync` 方法来检索映像。

请注意，`downloadAsync` 的方法声明以 `async` 关键字开头，指示它将以异步方式运行，并返回一个任务。另请注意，对 `DownloadDataTaskAsync` 的调用前面为 `await` 关键字。应用将事件处理程序执行（从显示 `await` 的点开始）移至后台线程，直到 `DownloadDataTaskAsync` 完成并返回。同时，应用的 UI 线程仍可以响应用户输入，并激发其他控件的事件处理程序。当 `DownloadDataTaskAsync` 完成时（这可能需要几秒钟），执行将恢复，其中 `bytes` 变量设置为调用 `DownloadDataTaskAsync` 的结果，事件处理程序代码的其余部分在调用方的 (UI) 线程上显示下载的映像。

有关 C# 中 `async` / `await` 的说明，请参阅 MSDN [使用 Async 和 Await 的异步编程](#) 主题。有关 Xamarin 对异步编程功能的支持的详细信息，请参阅 [异步支持概述](#)。

关键字差异

Java 中使用的很多语言关键字也在 C# 中使用。还有大量的 Java 关键字在 C# 中具有以不同方式命名的等效对应项，如下表所示：

| JAVA | C# | 描述 |
|-------------------------|------------------------|-------------------------|
| <code>boolean</code> | <code>bool</code> | 用于声明布尔值 true 和 false。 |
| <code>extends</code> | <code>:</code> | 先于要从中继承的类和接口。 |
| <code>implements</code> | <code>:</code> | 先于要从中继承的类和接口。 |
| <code>import</code> | <code>using</code> | 从命名空间导入类型，还可用于创建命名空间别名。 |
| <code>final</code> | <code>sealed</code> | 防止类派生；防止方法和属性在派生类中被重写。 |
| <code>instanceof</code> | <code>is</code> | 评估对象与给定类型是否兼容。 |
| <code>native</code> | <code>extern</code> | 声明外部实现的方法。 |
| <code>package</code> | <code>namespace</code> | 声明一组相关对象的作用域。 |
| <code>T...</code> | <code>params T</code> | 指定采用可变数目的参数的方法参数。 |
| <code>super</code> | <code>base</code> | 用于从派生类中访问父类的成员。 |

| JAVA | C# | 描述 |
|---------------------------|-------------------|--------------------|
| <code>synchronized</code> | <code>lock</code> | 使用锁获取和发布包装代码的关键部分。 |

此外，还有很多关键字是 C# 所特有的，且在 Java 中没有对应项。Xamarin.Android 代码通常使用下面的 C# 关键字(读取 Xamarin.Android 示例代码时，可使用此表作为参考)：

| C# | 描述 |
|-----------------------|-----------------------------------|
| <code>as</code> | 在兼容的引用类型或可以为 null 的类型之间执行转换。 |
| <code>async</code> | 指定方法或 lambda 表达式为异步。 |
| <code>await</code> | 挂起方法执行，直到任务完成。 |
| <code>byte</code> | 无符号的 8 位整数类型。 |
| <code>delegate</code> | 用于封装方法或匿名方法。 |
| <code>enum</code> | 声明枚举，这是一组命名的常数。 |
| <code>event</code> | 声明发布者类中的一个事件。 |
| <code>fixed</code> | 防止变量被重定位。 |
| <code>get</code> | 定义检索属性值的访问器方法。 |
| <code>in</code> | 使一个参数接受泛型接口中派生程度更小的类型。 |
| <code>object</code> | .NET Framework 中 Object 类型的别名。 |
| <code>out</code> | 参数修饰符或泛型类型参数声明。 |
| <code>override</code> | 扩展或修改继承成员的实现。 |
| <code>partial</code> | 声明要拆分为多个文件的定义，或者将一个方法定义从其实现中分离出来。 |
| <code>readonly</code> | 声明只能在声明时分配或由类构造函数分配的类成员。 |
| <code>ref</code> | 通过引用(而非值)来传递参数。 |
| <code>set</code> | 定义设置属性值的访问器方法。 |
| <code>string</code> | .NET Framework 中 String 类型的别名。 |
| <code>struct</code> | 封装一组相关变量的值类型。 |
| <code>typeof</code> | 获取对象类型。 |
| <code>var</code> | 声明一个隐式类型局部变量。 |

| C# | 描述 |
|-------------------------|-------------------|
| value | 引用客户端代码想要分配到属性的值。 |
| virtual | 允许在派生类中重写方法。 |

与现有的 Java 代码交互操作

如果你有不希望转换为 C# 的现有 Java 功能，可以在 Xamarin.Android 应用程序中通过两种技术重复使用现有的 Java 库：

- 创建 Java 绑定库 – 通过此方法，你可以使用 Xamarin 工具生成围绕 Java 类型的 C# 包装器。这些包装器称为“绑定”。因此，Xamarin.Android 应用程序可以通过调用这些包装器来使用 jar 文件。
- Java 本机接口 – Java 本机接口 (JNI) 是一个框架，使 C# 应用可以调用 Java 代码或者由 Java 代码调用。

有关这些技术的详细信息，请参阅 [Java 集成概述](#)。

其他阅读材料

MSDN [C# 编程指南](#)是开始学习 C# 编程语言的绝佳方式，你可以使用 [C# 参考](#)查找特定的 C# 语言功能。

同样，Java 知识要求至少像了解 Java 语言一样了解 Java 类库，C# 实践知识则要求熟悉 .NET framework。Microsoft 的[移动到面向 Java 开发人员的 C# 和 .NET Framework](#) 学习数据包是从 Java 角度了解有关 .NET Framework 详细信息的非常好的途径(同时可以更加深入地了解 C#)。

当你准备好在 C# 中处理第一个 Xamarin.Android 项目时，[Hello, Android](#) 系列可以帮助你生成第一个 Xamarin.Android 应用程序，并进一步加强你对通过 Xamarin 开发 Android 应用程序的基础知识的了解。

总结

本文从 Java 开发人员的角度提供有关 Xamarin.Android C# 编程环境的简介。它指出 C# 和 Java 之间的相似之处，同时解释了它们的实际差异。它介绍了程序集和命名空间，说明如何导入外部类型，并概述了访问修饰符、泛型、类派生、调用基类方法、方法重写以及事件处理方面的区别。它介绍了在 Java 中不可用的 C# 功能，例如属性、`async` / `await` 异步编程、lambda、C# 委托和 C# 事件处理系统。它包含一个涵盖重要 C# 关键字的表格，说明如何与现有的 Java 库进行交互操作，并提供用于进一步研究的相关文档链接。

相关链接

- [Java 集成概述](#)
- [C# 编程指南](#)
- [C# 参考](#)
- [移动到面向 Java 开发人员的 C# 和 .NET Framework](#)

Xamarin.Android 应用程序基础知识

2018/10/26 • [Edit Online](#)

本部分提供有关的更常见的操作任务或概念的开发人员需要开发 Android 应用程序时应注意的一些指南。

辅助功能

此页介绍了如何使用 Android 的辅助功能 Api 来构建应用程序根据[可访问性清单](#)。

了解 Android API 级别

本指南介绍了 Android 如何使用 API 级别来管理跨不同版本的 Android 应用程序兼容性，并介绍如何配置 Xamarin.Android 项目设置以便部署这些应用程序中的 API 级别。此外，本指南介绍如何编写运行时用于处理使用不同的 API 级别，并提供所有的 Android API 级别、版本数字（如 Android 8.0）、Android 代码名称（例如 Oreo）和生成版本代码的引用列表。

在 Android 中的资源

本文介绍了 Android 资源 Xamarin.Android 和文档中的概念及其用法。它介绍了如何在 Android 应用程序中使用的资源以支持应用程序本地化和多个设备，包括不同的屏幕大小和密度。

活动生命周期

活动是 Android 应用程序的基本构造块，它们可以存在于多个不同的状态。活动生命周期以实例化开始和结尾析构，并且包括很多状态之间。当活动更改状态时，会调用相应的生命周期事件方法，通知即将发生的状态更改的活动并使其能够执行的代码以适应所做的更改。本文分析活动的生命周期，并解释了负责该活动具有这些状态更改为良好、可靠的应用程序的一部分的每一阶段。

本地化

此文章介绍了如何将 Xamarin.Android 本地化为其他语言中，通过转换字符串，并提供替代图像。

服务

本文介绍 Android 服务，是 Android 组件，允许在后台完成工作。它说明服务适合于不同方案，并演示如何实现它们同时执行长时间运行后台任务，以及有关为远程过程调用提供一个接口。

广播接收器

本指南介绍了如何创建和使用广播的接收器，响应系统级广播，在 Xamarin.Android 中的 Android 组件。

权限

可以使用内置于 Visual Studio for Mac 或 Visual Studio 的工具支持创建并将权限添加到 Android 清单。本文档介绍如何在 Visual Studio 和 Xamarin Studio 中添加的权限。

图形和动画

Android 提供了非常丰富、不同的框架支持二维图形和动画。本文档介绍这些框架，并讨论如何创建自定义图形和动画和 Xamarin.Android 应用程序中使用它们。

CPU 体系结构

Xamarin.Android 支持多个 CPU 体系结构, 包括 32 位和 64 位设备。本文介绍如何以应用到一个或多个支持 Android 的 CPU 体系结构为目标。

处理旋转

本文介绍如何处理在 Xamarin.Android 中的设备方向更改。它介绍了如何使用 Android 资源系统以自动加载资源的特定设备方向以及如何以编程方式处理方向更改。然后, 介绍在旋转设备时维护状态的方法。

Android 音频

Android OS 提供了广泛支持为多媒体, 其中包含音频和视频。本指南重点介绍在 Android 中的音频, 涵盖播放和录制音频, 使用内置的音频播放器和记录器类, 以及低级别的音频 API。它还介绍如何使用由其他应用程序, 广播音频事件, 以便开发人员可以构建良好的应用程序。

通知

本部分介绍如何在 Xamarin.Android 中实现本地和远程通知。其中介绍了 Android 通知的各种 UI 元素, 并讨论了 API 的涉及创建并显示一条通知。有关远程通知, 说明了同时 Google Cloud Messaging 和 Firebase Cloud Messaging。中包含分步演练和代码示例。

触控

本部分介绍的概念和实现的详细信息上的触控手势 Android。引入和解释跟探索手势识别器的触控 Api。

HttpClient 堆栈和 SSL/TLS

本部分介绍适用于 Android 的 HttpClient 堆栈和 SSL/TLS 实现选择器。这些设置确定将由您的 Xamarin.Android 应用程序的 HttpClient 和 SSL/TLS 实现。

编写响应式应用程序

本文介绍如何使用线程处理来使 Xamarin.Android 应用程序通过将长时间运行任务到后台线程保持响应状态。

在 Android 上的辅助功能

2018/11/13 • [Edit Online](#)

此页介绍了如何使用 Android 的辅助功能 Api 来构建应用程序根据[可访问性清单](#)。请参阅[iOS 可访问性](#)并[OS X 可访问性](#)页的其他平台 Api。

描述 UI 元素

Android 提供 `ContentDescription` 屏幕阅读 Api 用于提供控件的用途的辅助性说明的属性。

内容说明可以设置在 C# 或 AXML 布局文件中。

C#

可以为任何字符串（或字符串资源）的代码中设置的说明：

```
saveButton.ContentDescription = "Save data";
```

AXML 布局

在 XML 中布局使用 `android:contentDescription` 属性：

```
<ImageButton
    android:id="@+id/saveButton"
    android:src="@drawable/save_image"
    android:contentDescription="Save data" />
```

TextView 的 use Hint

有关 `EditText` 并 `TextView` 控制数据输入，使用 `Hint` 属性来提供预期输入内容的说明（而不是 `ContentDescription`）。输入一些文本，文本本身，将“读取”而不是提示。

C#

设置 `Hint` 在代码中的属性：

```
someText.Hint = "Enter some text"; // displays (and is "read") when control is empty
```

AXML 布局

在 XML 布局文件使用 `android:hint` 属性：

```
<EditText
    android:id="@+id/someText"
    android:hint="Enter some text" />
```

LabelFor 链接输入带有标签的字段

若要将标签与数据输入控件相关联，请使用 `LabelFor` 属性

C#

在 C#，设置 `LabelFor` 属性设置为可描述此内容的控件的资源 ID（通常此属性设置上一个标签和引用其他输入的控

件):

```
EditText edit = findViewById<EditText> (Resource.Id.editFirstName);
TextView tv = findViewById<TextView> (Resource.Id.labelFirstName);
tv.LabelFor = Resource.Id.editFirstName;
```

AXML 布局

在布局 XML 使用 `android:labelFor` 属性来引用另一个控件的标识符:

```
<TextView
    android:id="@+id/labelFirstName"
    android:hint="Enter some text"
    android:labelFor="@+id/editFirstName" />
<EditText
    android:id="@+id/editFirstName"
    android:hint="Enter some text" />
```

公布推出适用于可访问性

使用 `AnnounceForAccessibility` 方法在任何视图控件时进行通信的事件或状态更改为用户启用可访问性。此方法不是内置旁白, 提供足够的反馈, 但应额外信息很有帮助的用户使用的大多数操作所必需的。

下面的代码显示了简单的示例调用 `AnnounceForAccessibility`:

```
button.Click += delegate {
    button.Text = string.Format ("{0} clicks!", count++);
    button.AnnounceForAccessibility (button.Text);
};
```

更改焦点设置

可访问的导航依赖于控件具有焦点, 以帮助用户了解哪些操作。Android 提供 `Focusable` 可以标记为专门能够导航期间接收焦点的控件的属性。

C#

若要防止控件获得焦点的C#, 设置 `Focusable` 属性设置为 `false`:

```
label.Focusable = false;
```

AXML 布局

在布局 XML 文件集 `android:focusable` 属性:

```
<android:focusable="false" />
```

您还可以控制与焦点顺序 `nextFocusDown`, `nextFocusLeft`, `nextFocusRight`, `nextFocusUp` 属性, 通常集 AXML 布局中。使用这些属性来确保用户可以轻松地浏览屏幕上的控件。

可访问性和本地化

是的提示和内容说明上面的示例中直接设置为显示值。最好使用中的值是 **Strings.xml** 文件, 如下:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="enter_info">Enter some text</string>
    <string name="save_info">Save data</string>
</resources>
```

使用字符串文件中的文本如下所示在C#和AXML布局文件：

C#

而不是在代码中使用字符串文字，查找已翻译的值从字符串文件 `Resources.GetText`：

```
someText.Hint = Resources.GetText (Resource.String.enter_info);
saveButton.ContentDescription = Resources.GetText (Resource.String.save_info);
```

AXML

在布局XML可访问性属性，如 `hint` 和 `contentDescription` 可以设置为字符串标识符：

```
<TextView
    android:id="@+id/someText"
    android:hint="@string/enter_info" />
<ImageButton
    android:id="@+id/saveButton"
    android:src="@drawable/save_image"
    android:contentDescription="@string/save_info" />
```

将文本存储在单独的文件夹的好处是可以在应用中提供的文件的多个语言翻译。请参阅[Android 本地化指南](#)若要了解如何将已本地化的字符串文件添加到应用程序项目。

测试辅助功能

请按照[这些步骤](#)启用TalkBack和资源管理器通过触摸辅助功能在Android设备上的进行测试。

可能需要安装[TalkBack](#)从Google Play，如果未出现在设置 > 可访问性。

相关链接

- [跨平台可访问性](#)
- [Android 可访问性 Api](#)

了解 Android API 级别

2018/10/26 • [Edit Online](#)

Xamarin.Android 已确定与多个版本的 Android 应用程序的兼容性的多个 Android API 级别设置。本指南介绍了这些设置的含义、如何配置它们，以及有何影响它们对您的应用程序在运行时。

快速入门

Xamarin.Android 提供了三个 Android API 级别的项目设置：

- **目标框架**—指定要在构建您的应用程序使用哪个框架。在使用此 API 级别编译时间通过 Xamarin.Android。
- **最低 Android 版本**—指定您希望在应用程序支持的最旧的 Android 版本。在使用此 API 级别运行 Android 时间。
- **目标 Android 版本**—指定你的应用 Android 版设计用于上运行。在使用此 API 级别运行 Android 时间。

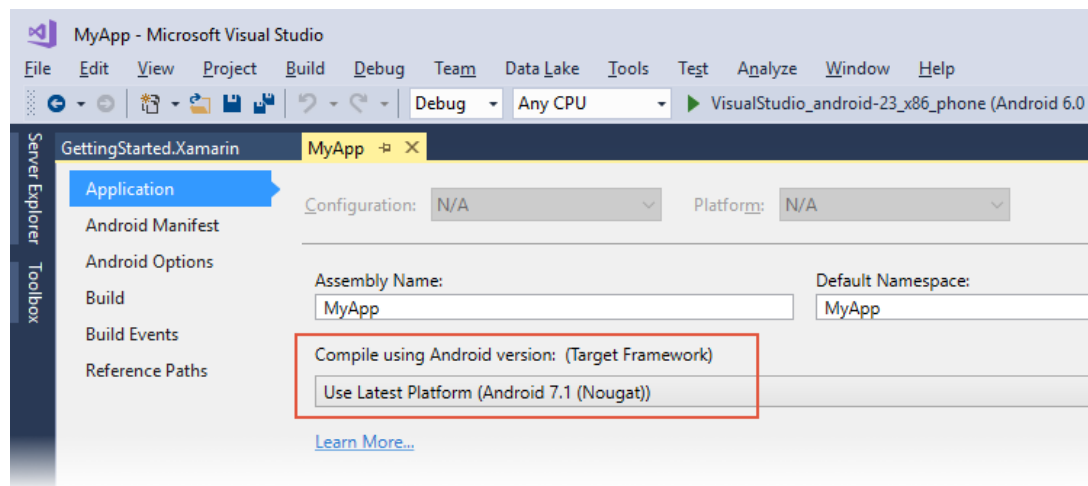
可以为你的项目配置 API 级别之前，必须安装该 API 级别的 SDK 平台组件。有关下载和安装 Android SDK 组件的详细信息，请参阅[Android SDK 安装程序](#)。

NOTE

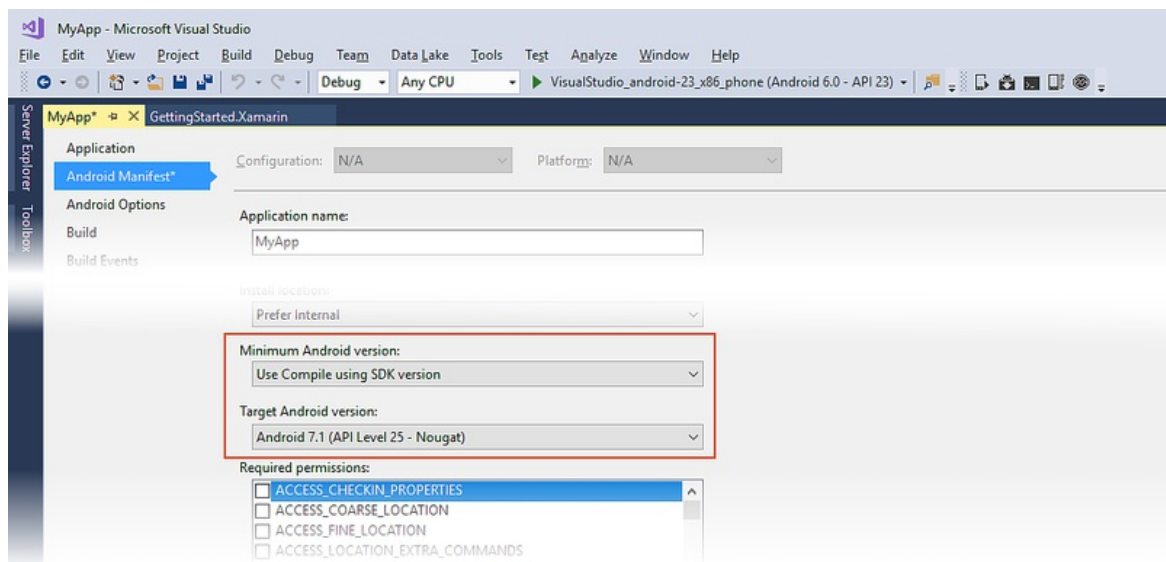
从 2018 年 8 月开始，Google Play 控制台将需要新的应用程序目标 API 级别 26 (Android 8.0) 或更高版本。现有应用程序将需要目标 API 级别 26 或更高版本开始，在 2018 年 11 月。有关详细信息，请参阅[改善应用程序安全性和性能在即将年的 Google Play 上](#)。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

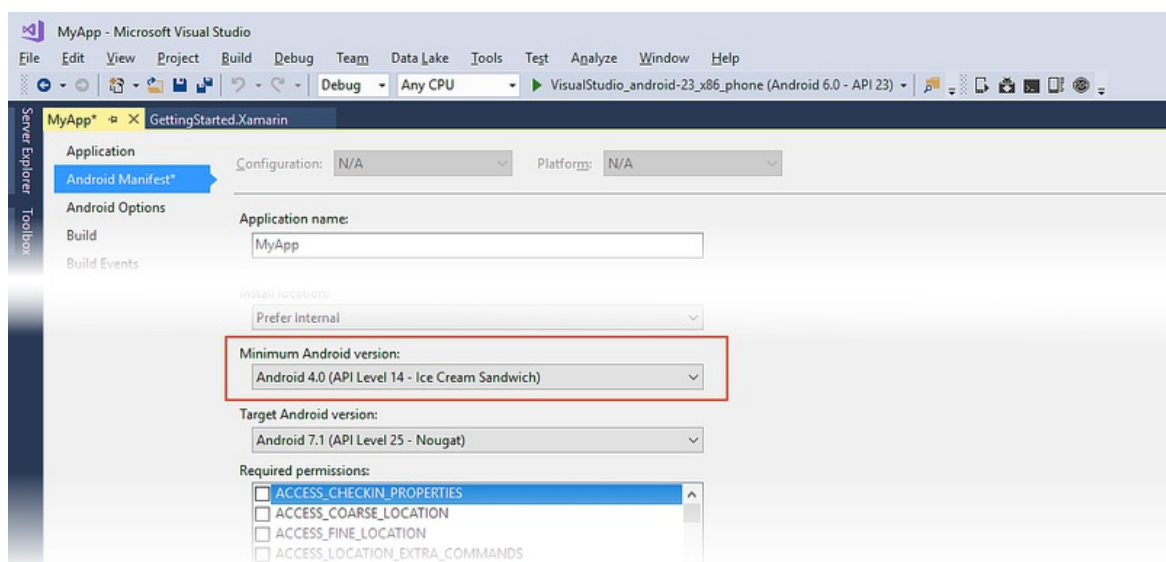
通常情况下，所有三个 Xamarin.Android API 级别设置为相同的值。上应用程序页上，将使用 **Android 版本**（目标框架）编译到最新的稳定 API 版本（或最小值，包含所有所需的功能的 Android 版本）。在以下屏幕截图中，目标框架设置为 **Android 7.1 (API 级别 25-Nougat)**：



上 **Android** 清单页上，将最低 Android 版本设置为使用编译使用 **SDK 版本** 并将目标 Android 版本设置为相同的值（在下面的示例的目标框架版本屏幕截图中，目标 Android 框架设置为 **Android 7.1 (Nougat)**）：



如果你想要保持与早期的 Android 版本的向后兼容性, 设置到目标的最低 **Android 版本**到所需应用程序以支持 Android 的最早版本。(请注意, API 级别 14 是所需的最低 API 级别[Google Play 服务](#)和 [Firebase 支持](#)。)下面的示例配置支持 Android API 级别 14 通过 API 级别 25 的版本:



如果您的应用程序支持多个 Android 版本, 你的代码必须包含运行时检查以确保您的应用程序可使用的最低 Android 版本设置 (请参阅[运行时检查的 Android 版本](#)下面有关详细信息)。如果要使用或创建一个库, 请参阅[API 级别和库](#)下面有关最佳实践配置 API 级别设置的库。

Android 版本和 API 级别

在 Android 平台的发展和新的 Android 版本的发布时, 每个 Android 版本分配唯一整数标识符, 称为 **API 级别**。因此, 每个 Android 版本对应于单个的 Android API 级别。由于用户安装应用程序的较旧也为最新版本的 Android 上, 必须设计实际的 Android 应用程序可以使用多个 Android API 级别。

Android 版本

每个版本的 Android 会由多个名称:

- Android 版本, 如 **Android 9.0**
- 代码 (或餐后甜点) 名称, 例如 `_饼图_`
- 相应的 API 级别, 如 **API 级别 28**

Android 的代码名称可能对应于多个版本和 API 级别 (如在下表中所示), 但每个 Android 版本对应于一个 API 级别。

此外, 定义 `Xamarin.Android生成版本代码`, 它将映射到当前已知的 Android API 级别。下表可以帮助你进行 API 级别、Android 版本、代码名称和 `Xamarin.Android 生成版本代码`之间转换 (生成版本代码中定义 `Android.OS` 命名空间):

| NAME | 版本 | API 级别 | RELEASED(已释放) | 生成版本代码 |
|--------------------|-------------|--------|---------------|------------------------------|
| 饼图 | 9.0 | 28 | 2018 年 8 月 | BuildVersionCodes.P |
| Oreo | 8.1 | 27 | 2017 年 12 月 | BuildVersionCodes.OMr1 |
| Oreo | 8.0 | 26 | 2017 年 8 月 | BuildVersionCodes.O |
| Nougat | 7.1 | 25 | 2016 年 12 月 | BuildVersionCodes.NMr1 |
| Nougat | 7.0 | 24 | 2016 年 8 月 | BuildVersionCodes.N |
| Marshmallow | 6.0 | 23 | 2015 年 8 月 | BuildVersionCodes.M |
| 棒棒糖形 | 5.1 | 22 | 2015 年 3 月 | BuildVersionCodes.LollipopMr |
| 棒棒糖形 | 5.0 | 21 | 2014 年 11 月 | BuildVersionCodes.Lollipop |
| Kitkat 监视 | 4.4W | 20 | Jun 2014 | BuildVersionCodes.KitKatWatc |
| Kitkat | 4.4 | 19 | 2013 年 10 月 | BuildVersionCodes.KitKat |
| Jelly Bean | 4.3 | 18 | 2013 年 7 月 | BuildVersionCodes.JellyBeanV |
| Jelly Bean | 4.2 4.2.2 | 17 | 2012 年 11 月 | BuildVersionCodes.JellyBeanV |
| Jelly Bean | 4.1 4.1.1 | 16 | 2012 年 6 月 | BuildVersionCodes.JellyBean |
| Ice Cream Sandwich | 4.0.3-4.0.4 | 15 | 2011 年 12 月 | BuildVersionCodes.IceCreamSa |
| Ice Cream Sandwich | 4.0 4.0.2 | 14 | 2011 年 10 月 | BuildVersionCodes.IceCreamSa |
| Honeycomb | 3.2 | 13 | 2011 年 6 月 | BuildVersionCodes.HoneyCombV |
| Honeycomb | 3.1.x | 12 | 2011 年 5 月 | BuildVersionCodes.HoneyCombV |
| Honeycomb | 3.0.x | 11 | 2011 年 2 月 | BuildVersionCodes.HoneyComb |
| Gingerbread | 2.3.3-2.3.4 | 10 | 2011 年 2 月 | BuildVersionCodes.GingerBree |
| Gingerbread | 2.3 2.3.2 | 9 | 2010 年 11 月 | BuildVersionCodes.GingerBree |
| Froyo | 2.2.x | 8 | 2010 年 6 月 | BuildVersionCodes.Froyo |
| Eclair | 2.1.x | 7 | 2010 年 1 月 | BuildVersionCodes.EclairMr1 |
| Eclair | 2.0.1 | 6 | 2009 年 12 月 | BuildVersionCodes.Eclair01 |
| Eclair | 2.0 | 5 | 2009 年 11 月 | BuildVersionCodes.Eclair |
| 圆环图 | 1.6 | 4 | 2009 年 9 月 | BuildVersionCodes.Donut |
| Cupcake | 1.5 | 3 | 2009 年 5 月 | BuildVersionCodes.Cupcake |
| Base | 1.1 | 2 | 2009 年 2 月 | BuildVersionCodes.Base11 |

| NAME | 版本 | API 级别 | RELEASED(已释放) | 生成版本代码 |
|------|-----|--------|---------------|------------------------|
| Base | 1.0 | 1 | 2008 年 10 月 | BuildVersionCodes.Base |

此表所示，新的 Android 版本的发布频率—有时多个版本每年。因此，可能运行你的应用的 Android 设备的通用包括的各种较旧和较新的 Android 版本。如何保证您的应用程序将许多不同版本的 Android 上运行一致而可靠地？Android API 级别可帮助你管理此问题。

Android API 级别

每个 Android 设备运行在完全一个API 级别—此 API 级别保证是唯一的每个 Android 平台版本。API 级别精确标识您的应用程序可以调用; 的 API 集的版本它标识作为开发人员的清单元素、权限、编码的对等组合。Android 的系统的 API 级别可帮助确定应用程序是否与在设备上安装应用程序之前的 Android 系统映像兼容的 Android。

当生成应用程序时，它包含以下 API 级别信息：

- *目标*的生成应用上运行的 Android API 级别。
- *最小*Android 设备必须运行你的应用的 Android API 级别。

这些设置用于确保在安装时正常运行应用程序所需的功能也可在 Android 设备上。如果没有，阻止该设备上运行应用。例如，如果 Android 设备的 API 级别低于您指定为你的应用的最低 API 级别，在 Android 设备将阻止用户安装您的应用程序。

项目 API 级别设置

以下各节说明如何使用 SDK 管理器的 API 级别，你想要为目标，跟如何配置的详细说明准备开发环境*目标框架*，*最小值 Android 版本*，并*目标 Android 版本*在 Xamarin.Android 中的设置。

Android SDK 平台

可以在 Xamarin.Android 中选择目标或最低 API 级别之前，必须到该 API 级别安装相对应的 Android SDK 平台版本。目标框架、最低 Android 版本和目标 Android 版本的可用选项的范围被限制为已安装的范围的 Android SDK 版本。可以使用 SDK 管理器以验证安装了所需的 Android SDK 版本，并可用来添加任何新的 API 级别所需的应用程序。如果您不熟悉如何安装 API 级别，请参阅[Android SDK 安装程序](#)。

目标 Framework

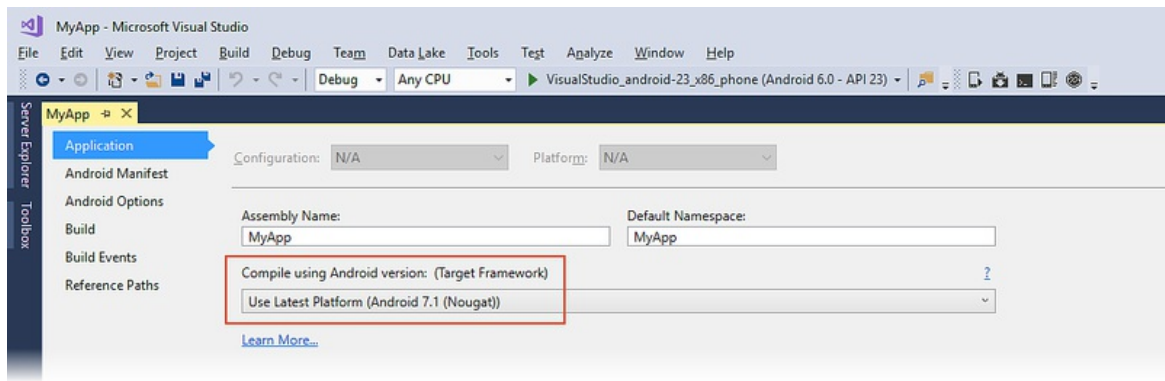
目标框架(也称为 `compileSdkVersion`) 是在生成时的编译你的应用程序的特定 Android 框架版本 (API 级别)。此设置指定哪些 Api 应用需要时运行，但它不起作用的 Api 是实际可供您的应用程序在安装时使用。因此，更改目标框架设置不会更改运行时行为。

目标框架标识你的应用程序针对链接的库版本—此设置确定可以在应用中使用哪些 Api。例如，如果你想要使用 `NotificationBuilder.SetCategory` 在 Android 5.0 Lollipop 中引入的方法，必须将目标框架设置为 **API Level 21 (Lollipop)** 或更高版本。如果将你的项目的目标框架设置为 API 级别等 **API 级别 19 (KitKat)** 尝试调用 `SetCategory` 在代码中的方法，将收到编译错误。

我们建议始终编译与*最新*可用的目标框架版本。执行此操作提供有用的警告消息可能由代码调用任何已弃用的 api。使用最新支持库版本时，使用最新的目标框架版本则尤其—每个库要求您的应用程序进行编译时支持该库的最低 API 级别或更高版本。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

若要访问 Visual Studio 中的目标框架设置，打开项目属性中的[解决方案资源管理器](#)，然后选择[应用程序](#)页：



通过选择下的下拉列表菜单中的 API 级别设置目标框架使用 **Android 版本** 编译如上所示。

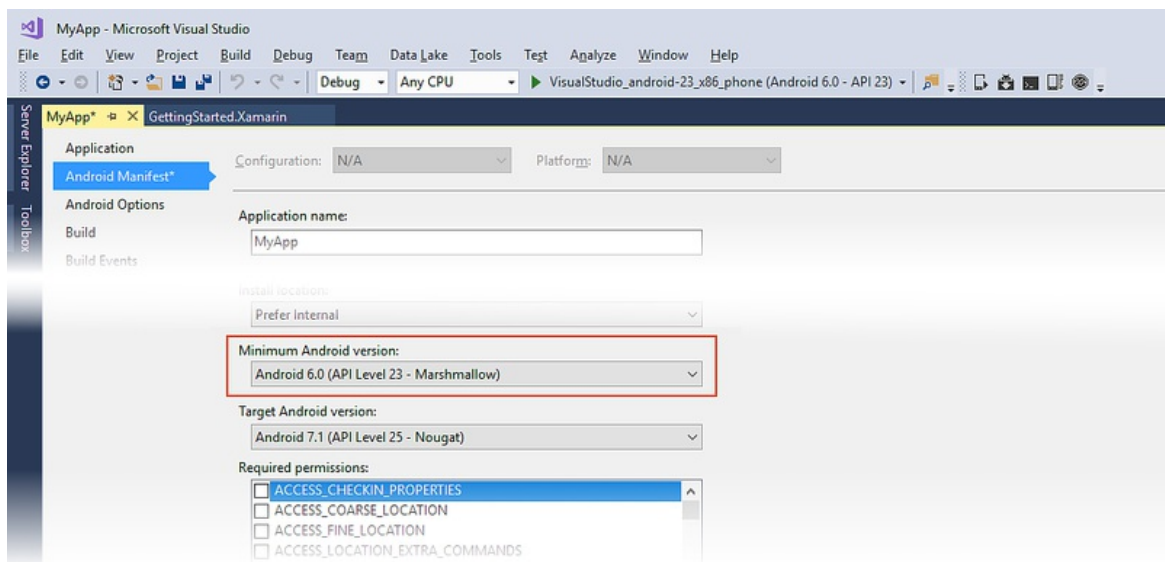
最低 Android 版本

最低 Android 版本(也称为 `minSdkVersion`) 是最早版本的 Android OS (即最低 API 级别), 可以安装和运行你的应用程序。默认情况下, 应用仅可匹配目标框架设置的设备上已安装或更高版本;如果最低 Android 版本设置为**较低**高于目标框架设置中, 您的应用程序还可以在早期版本的 Android 上运行。例如, 如果目标框架设置为**Android 7.1 (Nougat)** 并将最低 Android 版本设置为**Android 4.0.3 (Ice Cream Sandwich)**, 可以在 API 级别 15 从任何平台上安装了应用为 API 级别 25, 非独占。

尽管您的应用程序可能会成功生成, 并在此系列的平台上安装, 但这并不保证将成功运行所有这些平台上。例如, 如果您的应用程序安装在**Android 5.0 (Lollipop)** 和你的代码调用一个 API, 仅适用于**Android 7.1 (Nougat)** 和更高版本, 您的应用程序将获取运行时错误和可能崩溃。因此, 必须确保你的代码-在运行时-, 它调用仅支持的 Android 设备上运行的 Api。换言之, 你的代码必须包含显式运行时检查, 以确保你的应用仅在不够高, 无法支持它们的设备上使用较新的 Api。[运行时检查的 Android 版本](#)本指南中, 将在后面介绍了如何将这些运行时检查添加到你的代码。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

若要访问 Visual Studio 中的最低 Android 版本设置, 打开项目属性中的**解决方案资源管理器**, 然后选择**Android** 清单页。下的下拉列表菜单中**最低 Android 版本**可以为应用程序选择最低 Android 版本:



如果选择**使用编译使用 SDK 版本**, 最低 Android 版本将作为目标框架设置相同。

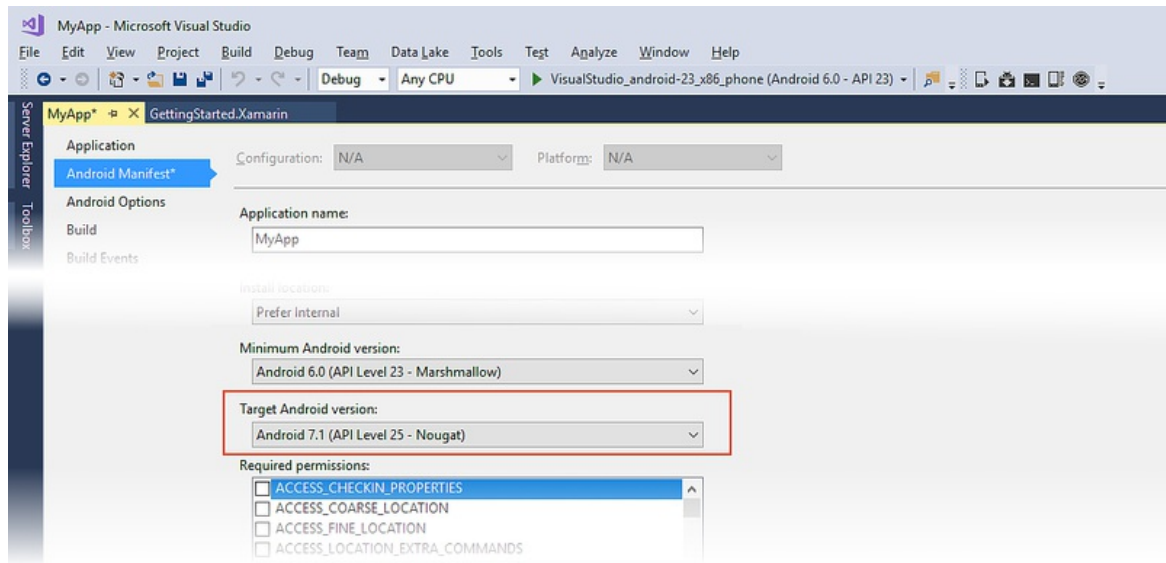
目标 Android 版本

目标 Android 版本(也称为 `targetSdkVersion`) 是 API 级别的 Android 设备的应用应运行在其中。Android 使用此设置来确定是否启用任何兼容性行为-这可确保您的应用程序继续按你期望的方式。Android 使用您的应用程序的目标 Android 版本设置来判断哪些行为更改可应用于您的应用程序而不会破坏它 (这是 Android 如何提供向前兼容性)。

目标框架和目标 Android 版本, 具有非常相似的名称, 而不是相同的操作。目标框架设置目标 API 级别将信息传递给 Xamarin.Android 以供**编译时**, 而目标 Android 版本目标 API 级别将信息传递给 Android 以便在使用**运行时**(应用程序的设备上安装并运行)。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

若要访问 Visual Studio 中的此设置，打开项目属性中的**解决方案资源管理器**，然后选择**Android** 清单页。下的下拉列表菜单中目标 **Android** 版本可以为应用程序中选择目标 Android 版本：



我们建议您显式设置为最新版本的用于测试你的应用的 Android 目标 Android 版本。理想情况下，它应设置为最新的 Android SDK 版本—这样即可使用新的 Api 之前通过行为更改的工作。对于大多数开发人员，我们 **不这样做** 建议目标 Android 版本设置成使用编译使用 **SDK 版本**。

一般情况下，目标 Android 版本应受最低 Android 版本和目标框架。即：

最低 Android 版本 <= 目标 Android 版本 <= 目标框架

有关 SDK 级别的详细信息，请参阅 Android 开发人员[使用 sdk](#)文档。

运行时检查的 Android 版本

发布 Android 每个新版本后，框架 API 会更新以提供新或替代功能。几个例外情况之外，从早期的 Android 版本的 API 功能将会传递到较新的 Android 版本，无需修改即可。因此，如果您的应用程序在特定 Android API 级别上运行，它通常会能够无需修改即可在更高版本的 Android API 级别上运行。但如果你还想要更早版本的 Android 上运行你的应用？

如果您选择的最低 Android 版本 **较低** 与您的目标框架设置，某些 Api 可能不能供您在运行时的应用程序。但是，早期在设备上，但具有降低的功能，仍可运行您的应用程序。对于不是与你的最低 Android 版本设置相对应的 Android 平台上可用的每个 API，你的代码必须显示检查的值 `Android.OS.Build.VERSION.SdkInt` 属性来确定应用程序运行的平台的 API 级别。如果 API 级别 **较低** 支持 API 的最低 Android 版本比你想要调用，则你的代码必须找到一种方法，才能正常运行而无需进行此 API 调用。

例如，让我们假设我们想要使用 `NotificationBuilder.SetCategory` 方法以运行时对通知进行分类 **Android 5.0 Lollipop**（及更高版本），但我们仍然希望对我们的应用程序在早期版本的 Android 上运行，如 **Android 4.1 Jelly Bean**（其中 `SetCategory` 不可用）。在本指南的开始处的 Android 版本表中引用，我们看到的生成版本代码 **Android 5.0 Lollipop** 是 `Android.OS.BuildVersionCodes.Lollipop`。若要支持较旧版本的 Android where `SetCategory` 是不可用，我们的代码可以检测在运行时的 API 级别和有条件地调用 `SetCategory` API 级别时才大于或等于棒棒糖生成版本代码：

```
if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop)
{
    builder.SetCategory(Notification.CategoryEmail);
}
```

在此示例中，我们的应用程序的目标框架设置为 **Android 5.0 (API 级别 21)** 并且其最低 Android 版本设置为 **Android 4.1 (API 级别为 16)**。因为 `SetCategory` 目前在 API 级别 `Android.OS.BuildVersionCodes.Lollipop` 和更高版本，此示例代码将调用 `SetCategory` 仅当有实际可用—它将不尝试调用 `SetCategory` 时 API 级别为 16、17、18、19 或 20。仅的范围内通知不正确（因为它们不按类型分类）排序，但仍发布通知来提醒用户，这些以前的 Android 版本缩减功能。我们的应用仍然正常工作，但其功能会略有降低。

一般情况下，生成版本检查可帮助你决定在运行时之间执行这样的操作与旧方法的新方法的代码。例如：

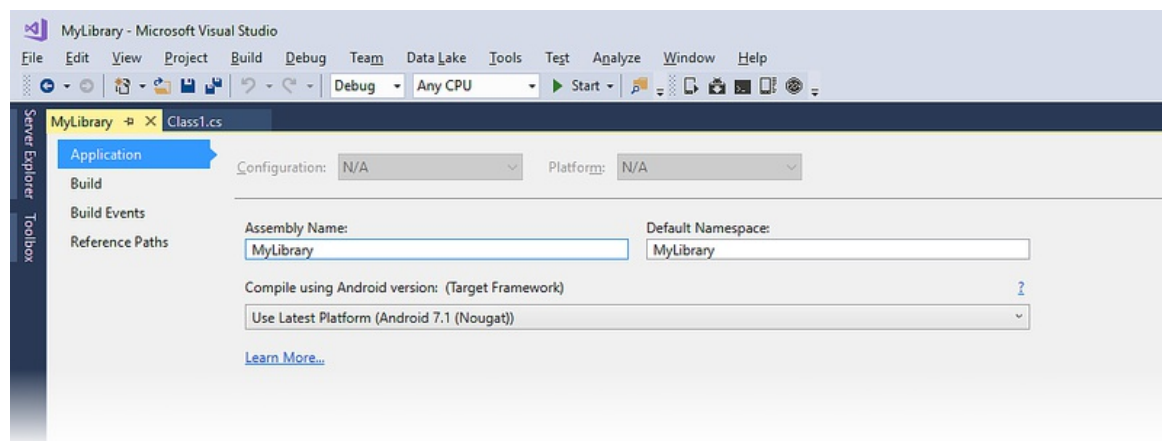
```
if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop)
{
    // Do things the Lollipop way
}
else
{
    // Do things the pre-Lollipop way
}
```

没有快速而简单规则，说明如何减少或缺少一个或多个 API 的较旧 Android 版本上运行时修改应用程序的功能。在某些情况下 (例如，在 `SetCategory` 上面示例中)，只需省略 API 调用时不可用。但是，在其他情况下，您可能需要为时实现替代功能。`Android.OS.Build.VERSION.SdkInt` 被检测出是早于 API 级别，您的应用程序需要以提供其最佳体验。

API 级别和库

- [Visual Studio](#)
- [Visual Studio for Mac](#)

创建 Xamarin.Android 类库项目 (如类库或绑定库) 时，可以配置的目标框架设置-最低 Android 版本和目标 Android 版本设置不可用。这是因为没有任何 **Android** 清单页：



最低 Android 版本和目标 Android 版本设置不可用因为所得的库不是一款独立应用-库可以运行任何 Android 版本，具体取决于它打包在一起的应用。您可以指定如何在库将成为编译，但无法预测哪个平台 API 级别将在上运行库。明确这一点后，使用或创建库时，应遵循以下最佳实践：

- **使用 Android 库时**-如果你正在使用 Android 库应用程序中，请务必设置将设置为 API 级别，它应用的目标框架至少高达目标框架库的设置。
- **创建 Android 库时**-如果您创建 Android 库为使用其他应用程序，请务必将其目标框架设置为编译所需的最低 API 级别。

这些最佳实践建议以帮助防止这种情况，一个库尝试调用不是在运行时（这可能导致应用崩溃）提供的 API。如果您是库的开发人员，您应尽可能限制你使用的 API 调用的总的 API 外围应用的小型 and 获得广泛认可的子集。这样做有助于确保您的库可以安全地用在更广范围的 Android 版本之间。

总结

本指南说明了如何使用 Android API 级别来跨不同版本的 Android 管理应用程序兼容性。它提供的详细的步骤用于配置 Xamarin.Android *目标框架*，*最低 Android 版本*，并 *目标 Android 版本* 项目设置。它提供了有关使用 Android SDK 管理器安装 SDK 包的说明包含有关如何编写代码来处理在运行时，不同 API 级别的示例，并介绍了如何管理创建或使用 Android 库时的 API 级别。它还提供了与 Android 版本数字 (如 Android 4.4)、Android 版本名称 (例如 Kitkat) 和 Xamarin.Android 生成版本代码的 API 级别的完整列表。

相关链接

- [Android SDK 安装](#)

- [SDK CLI 工具更改](#)
- [选取 compileSdkVersion、minSdkVersion 和 targetSdkVersion](#)
- [什么是 API 级别？](#)
- [用、标记和内部版本号](#)

Android 资源

2018/10/26 • [Edit Online](#)

本文介绍了 Android 资源在 Xamarin.Android 中的概念, 并将介绍如何使用它们。它介绍了如何在 Android 应用程序中使用的资源以支持应用程序本地化和多个设备, 包括不同的屏幕大小和密度。

概述

Android 应用程序很少是只需源代码。通常有多个组成应用程序的其他文件: 视频、图像、字体和音频文件, 只是为了仅举几例。总体来说, 这些非源代码文件称为资源和编译 (以及源代码) 在生成过程并打包为分发和安装到设备上的 APK:

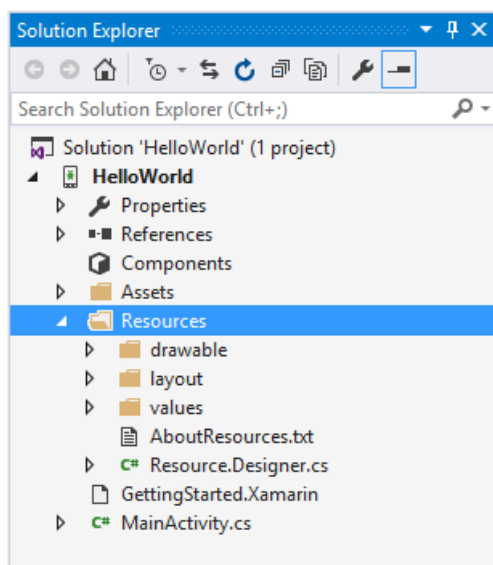


资源到 Android 应用程序有几个优点:

- **代码分离**-将源代码与图像、字符串、菜单、动画、颜色等。这种情况下的资源可帮助显著本地化时。
- **面向多个设备**-提供更简单的无需更改代码的不同设备配置的支持。
- **编译时检查**-资源是静态的编译到应用程序。这允许在编译时, 当它很容易就可以捕获并更正错误, 而不是运行时更难找到并更正而且成本很高时, 要检查的资源的使用情况。

启动新的 Xamarin.Android 项目时, 被创建一个名为资源的特殊目录, 以及一些子目录:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



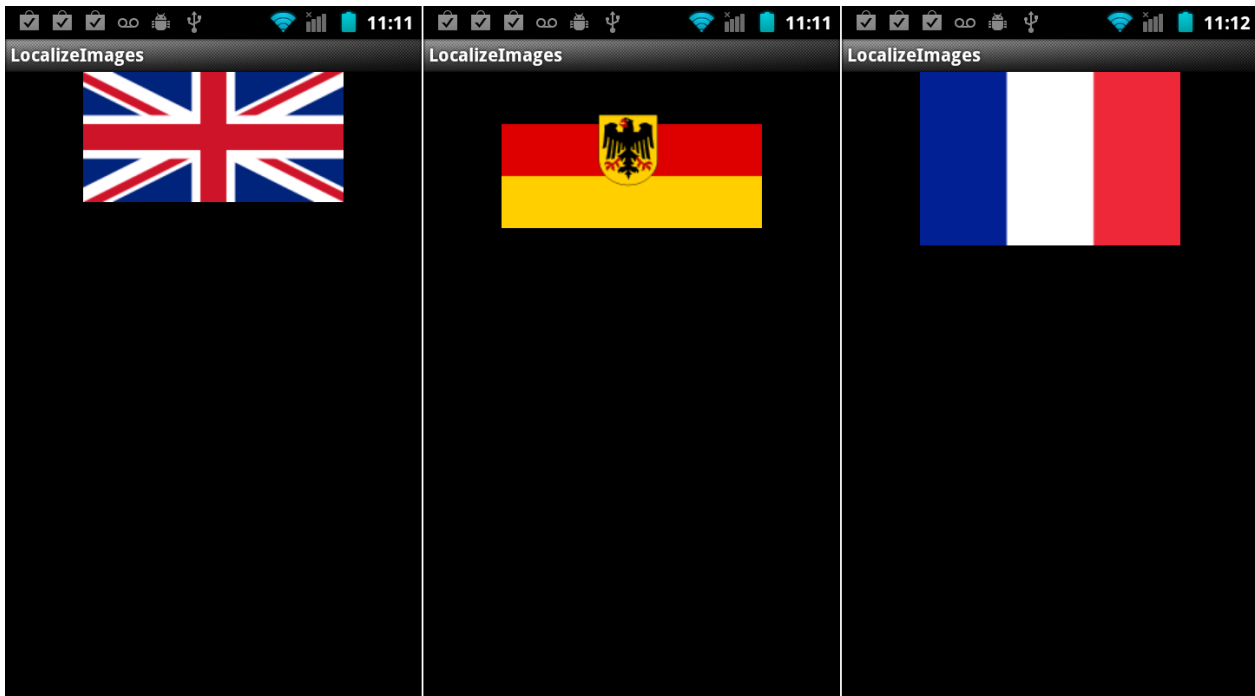
在上图中, 应用程序资源被按类型组织到以下子目录中: 图像进入 **drawable** 目录; 视图进入 **layout** 子目录, 依此类推。

有两种方法访问这些资源在 Xamarin.Android 应用程序中: 以**编程方式**代码中和以**声明方式**中使用特殊的 XML 语法的 XML。

这些资源称为**默认资源**，除非另有指定更具体的匹配项由所有设备。此外，还可以具有每种类型的资源**备用资源**Android 可用于针对特定的设备。例如，可能会提供资源以面向用户的区域设置，屏幕大小或设备是否从纵向向横向，旋转 90 度等。在每个这种情况下，Android 将加载应用程序而无需任何额外编码工作由开发人员使用的资源。

通过添加一个短字符串，调用指定备用资源**限定符**，到末尾的给定的类型的资源的目录。

例如，**resources/drawable-de** 将为设置为德语区域设置的设备指定图像，而 **resources/drawable-fr** 则会为设置为法语区域设置的设备保存图像。下图可以看到一个提供备用资源的示例，其中运行的是同一应用程序，只更改了设备的区域设置：



本文将全面看一下使用资源，并涵盖以下主题：

- **Android 资源基础知识**—使用的默认资源以编程方式或声明的方式，如图像和字体的资源类型添加到应用程序。
- **设备特定的配置**—应用程序中支持的不同屏幕分辨率和密度。
- **本地化**—使用资源来支持应用程序可能使用不同的区域。

相关链接

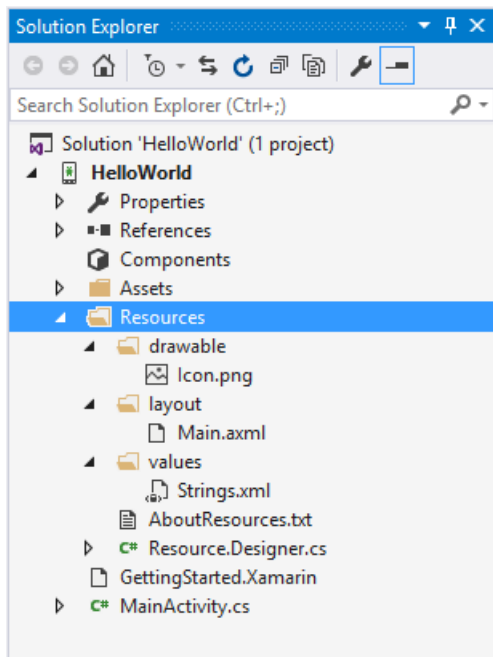
- [使用 Android 资产](#)
- [应用程序基础知识](#)
- [应用程序资源](#)
- [支持多个屏幕](#)

Android 资源基础知识

2018/10/26 • [Edit Online](#)

几乎所有的 Android 应用程序将在其中; 具有某种形式的资源在最低限度它们通常具有 XML 文件的窗体中的用户界面布局。当首次创建 Xamarin.Android 应用程序时, 默认资源是通过 Xamarin.Android 项目模板的安装程序:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



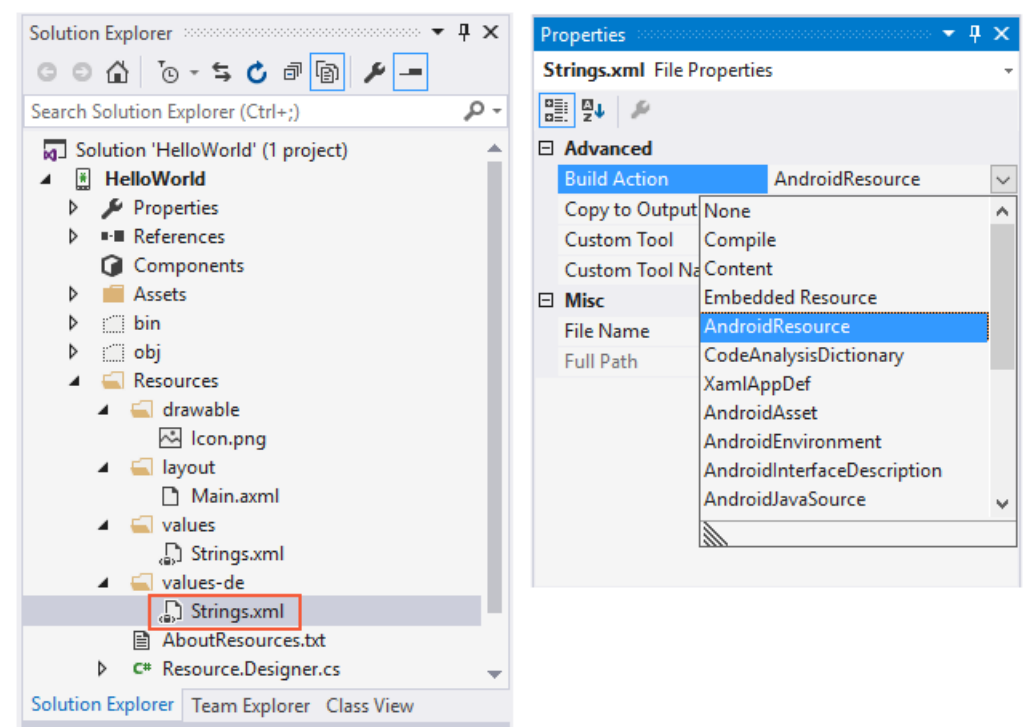
在资源文件夹中创建了五个文件构成默认资源:

- **Icon.png** -应用程序的默认图标
- **Main.axml** -应用程序的默认用户界面布局文件。请注意, Android 时使用 **.xml**文件扩展名, Xamarin.Android 使用 **.axml**文件扩展名。
- **Strings.xml** -字符串表, 以帮助进行本地化的应用程序
- **AboutResources.txt** -这不是必需的并安全地删除。它只提供资源文件夹和文件的高级别概述。
- **Resource.designer.cs** -此文件自动生成并维护的 Xamarin.Android 和保存的唯一 ID 分配给每个资源。这是非常类似, 与在 Java 中编写的 Android 应用程序必须 R.java 文件用途相同。它由 Xamarin.Android 工具自动创建, 并将不时重新生成。

创建和访问资源

创建资源只需将文件添加到所涉及的资源类型的目录。下面的屏幕快照显示了德语区域设置的字符串资源已添加到项目。当**Strings.xml**到文件中, 添加生成操作自动设置为**AndroidResource**由 Xamarin.Android 工具:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



这样,若要正确编译和嵌入的 APK 文件中的资源的 Xamarin.Android 工具。如果由于某种原因生成操作未设置为**Android 资源**、然后将从 APK 中排除文件和任何尝试加载或访问的资源将导致运行时错误,应用程序将崩溃。

此外,务必注意,当时 Android 仅支持的资源项的小写形式的文件名, Xamarin.Android 有点内更能容忍;它将支持大写和小写的文件名。映像名称的约定是使用小写带下划线作为分隔符(例如,我_映像_name.png)。请注意是否使用短划线或空格作为分隔符,不能处理资源名称。

资源添加到项目后,有两种方法在应用程序中使用它们-(内代码)以编程方式或从 XML 文件。

以编程方式引用的资源

若要以编程方式访问这些文件,它们将分配一个唯一的资源 id。此资源 ID 是一个名为的特殊类中定义一个整数 `Resource`, 这在文件找到 **Resource.designer.cs**, 如下所示:

```
public partial class Resource
{
    public partial class Attribute
    {
    }
    public partial class Drawable {
        public const int Icon=0x7f020000;
    }
    public partial class Id
    {
        public const int Textview=0x7f050000;
    }
    public partial class Layout
    {
        public const int Main=0x7f030000;
    }
    public partial class String
    {
        public const int App_Name=0x7f040001;
        public const int Hello=0x7f040000;
    }
}
```

每个资源 ID, 包含在对应于资源类型的嵌套类。例如, 当文件 **Icon.png** 已添加到项目中, Xamarin.Android 更新

`Resource` 类, 创建嵌套的类名为 `Drawable` 常量的内部使用名为 `Icon`。这样, 文件 `Icon.png` 作为代码中引用 `Resource.Drawable.Icon`。 `Resource` 类不应进行手动编辑, 因为 Xamarin.Android 将覆盖对其进行任何更改。

引用时以编程方式(在代码中)的资源, 它们可以访问通过使用以下语法的资源类层次结构:

```
@[<PackageName>].Resource.<ResourceType>.<ResourceName>
```

- **PackageName** – 了提供资源和仅的包时需要其他包中的资源的使用。
- **ResourceType** – 这是上面所述的资源类中的嵌套的资源类型。
- **资源名称** – 这是(不带扩展名)资源的文件名或 XML 元素中的资源使用 `android:name` 属性的值。

从 XML 引用的资源

XML 文件中的资源访问的以下特殊语法:

```
@[<PackageName>:]<ResourceType>/<ResourceName>.
```

- **PackageName** – 了提供资源和仅的包时需要其他包中的资源的使用。
- **ResourceType** – 这是资源类中的嵌套的资源类型。
- **资源名称** – 这是该资源的文件名(而无需文件类型扩展名)的值或 `android:name` 的 XML 元素中的资源的属性。

例如布局文件的内容 `Main.axml`, 如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView android:id="@+id/myImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/flag" />
</LinearLayout>
```

此示例中有 `ImageView` 需要一个名为的可绘制资源标志。 `ImageView` 具有其 `src` 属性设置为 `**@drawable/flag**`。 Android 活动启动时, 将在目录中查找资源/Drawable为名为的文件 `flag.png` (文件扩展名可能是另一种图像格式, 如 `flag.jpg`)加载此文件, 并将其显示在 `ImageView`。运行此应用程序时, 它看起来与以下图像:

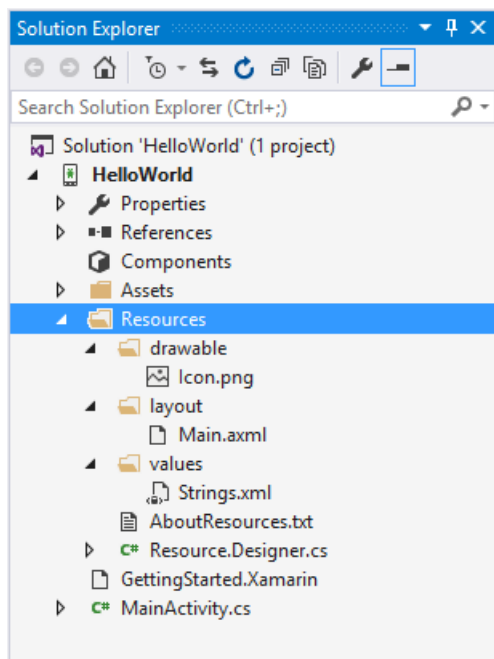


默认资源

2018/10/26 • [Edit Online](#)

默认资源是不特定于任何特定设备或外形规格，因此可以通过 Android OS 的默认选项如果没有更具体的项可以找到资源。在这种情况下，它们是要创建资源的最常见类型。它们到的子目录中的结构资源目录根据它们的资源类型：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



在上图中，该项目具有可绘制资源、布局和价值（包含简单值的 XML 文件）的默认值。

下面提供了资源类型的完整列表：

- **animator**—描述属性动画的 XML 文件。属性动画在 API 级别 11 (Android 3.0) 中引入，并提供用于对某个对象的属性的动画。属性动画是对象的更加灵活和强大的方式来描述在任何类型上的动画。
- **anim**—描述 XML 文件 *tween* 动画。Tween 动画是文本的一系列的动画说明图像或不断增加大小的视图对象或示例中，旋转内容执行的转换。Tween 动画被限制为只能查看对象。
- **color**—描述颜色的状态列表的 XML 文件。若要了解颜色状态列表，请考虑如按钮的 UI 小组件。它可能具有不同的状态，如按下或禁用，和按钮可能会随每个状态更改颜色。列表表示状态列表中。
- **drawable**—可绘制资源是用于图形可编译到应用程序，然后访问 API 调用或引用的其他 XML 资源的常规概念。绘图的一些示例是位图文件 (.png、.gif、.jpg)、名为的特殊可调整大小的位图 [包含 9 个修补程序](#)，列出了状态，泛型在 XML 等中定义的形状。
- **layout**—描述用户界面布局，如活动或列表中的行的 XML 文件。
- **menu**—XML 文件，如描述应用程序菜单 *选项菜单*，*上下文菜单*，和 *子菜单*。菜单的示例，请参阅 [弹出菜单演示](#) 或 [标准控件](#) 示例。
- **raw**—保存在其原始的、二进制形式的任意文件。这些文件编译为二进制格式中的 Android 应用程序。
- **values**—包含简单值的 XML 文件。值目录中的 XML 文件不会定义单个资源，而您可以定义多个资源。例如一个 XML 文件可能持有字符串值的列表，而另一个 XML 文件可能持有的颜色值列表。

- **xml** - 在函数类似于.NET 配置文件的 XML 文件。这些是由应用程序可以在运行时中读取的任意 XML。

备用资源

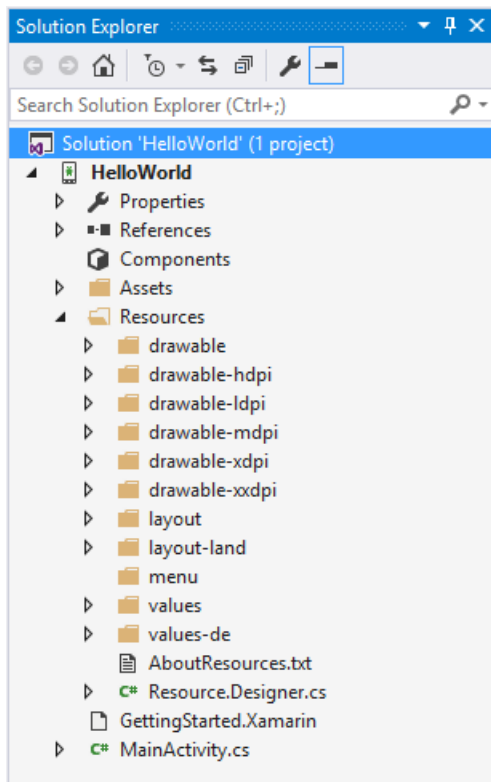
2018/11/13 • [Edit Online](#)

备用资源是针对某个特定设备或如当前语言、特定屏幕大小或像素密度的运行时配置这些资源。如果 Android 可以匹配特定设备或配置默认的资源比更具体的资源，则将改为使用该资源。如果找不到匹配的当前配置的备用资源，则将加载默认资源。Android 决定如何将更详细地，在资源位置部分介绍应用程序将使用哪些资源

备用资源都组织为根据资源类型，就像默认资源资源文件夹中的子目录。替代资源子目录的名称是在窗体中：
ResourceType-限定符

*限定符*是用于标识特定设备配置的名称。在名称每个短划线分隔可能有多个限定符。例如，下面的屏幕截图显示了一个简单的项目具有用于区域设置、屏幕密度、屏幕大小和方向等的各种配置的备用资源：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



将限定符添加到资源类型时，将应用以下规则：

1. 可能有多个限定符，与每个短划线分隔的限定符。
2. 也许只能指定一次限定符。
3. 限定符必须是它们在下表中显示的顺序。

有关参考下面列出了可能的限定符：

- **MCC 和 mnc 是否** – [移动国家/地区代码](#)(MCC) 和 (可选)[移动电话网络代码](#)(mnc 是否)。SIM 卡将提供 MCC，而设备连接到的网络将提供 mnc。虽然可以使用移动国家/地区代码的目标区域设置，但建议方法是使用下面指定的语言限定符。有关示例，对德国，目标资源是限定符 `mcc262`。在美国，T 移动的目标资源限定符是 `mcc310-mnc026`。有关移动国家/地区代码和移动网络代码的完整列表请参阅<http://mcc-mnc.com/>。
- **语言**–两个字母[ISO 639-1 语言代码](#)和后面可跟两个字母[ISO 3166 alpha 2 区域代码](#)。如果提供了这两个限

定符，则它们之间用 `-r`。例如，若目标讲法语的区域设置则的限定符 `fr` 使用。若要面向加拿大法语区域设置中，`fr-rCA` 会使用。语言代码和区域代码的完整列表，请参阅[表示形式的名称的语言代码并国家/地区名称和代码元素](#)。

- **最小宽度**—指定应用程序旨在上执行的最小屏幕宽度。中的更详细地介绍[对于不同的屏幕创建资源](#)。在 API 级别 13 (Android 3.2) 及更高版本可用。例如，限定符 `sw320dp` 到目标设备的高度和宽度至少为使用 320dp。
- **可用宽度**—格式 `w` 中的屏幕的最小宽度 `Ndp`，其中 `N` 是无关的像素密度的宽度。根据用户旋转设备，可能会更改此值。中的更详细地介绍[对于不同的屏幕创建资源](#)。在 API 级别 13 (Android 3.2) 及更高版本可用。示例：使用限定符 `w720dp` 面向的宽度设置为最低 720dp 的设备。
- **可用高度**—格式 `h` 中屏幕的最小高度 `Ndp`，其中 `N` 是在分发点的高度。根据用户旋转设备，可能会更改此值。中的更详细地介绍[对于不同的屏幕创建资源](#)。在 API 级别 13 (Android 3.2) 及更高版本可用。例如，使用限定符 `h720dp` 面向具有的最低 720dp 高度的设备
- **屏幕大小**—此限定符是的则这些资源的屏幕大小的泛化。中更详细地介绍了它[对于不同的屏幕创建资源](#)。可能值为 `small`、`normal`、`large` 和 `xlarge`。在 API 级别 9 (Android 2.3/Android 2.3.1/Android 2.3.2) 中添加
- **屏幕上方面**—基于纵横比，不屏幕方向。长屏是更广。添加在 API 级别 4 (Android 1.6)。可能的值为 `long` 类型的值和 `notlong`。
- **屏幕方向**—纵向或横向屏幕方向。这可能会在应用程序的生存期内更改。可能的值为 `port` 和 `land`。
- **停靠模式**—一辆汽车中的设备停靠或停靠在办公桌前。添加在 API 级别 8 (Android 2.2.x)。可能的值为 `car` 和 `desk`。
- **夜间模式**—是否运行该应用程序在夜间或在一天中。这可能会更改应用程序的生命周期内并且旨在使开发人员有机会在夜间使用接口的较暗版本。添加在 API 级别 8 (Android 2.2.x)。可能的值为 `night` 和 `notnight`。
- **屏幕像素密度 (dpi)**—实际屏幕上的给定区域中的像素数。通常以每英寸点数 (dpi)。可能的值有：
 - `ldpi` — 低密度屏幕。
 - `mdpi` — 中等密度屏幕
 - `hdpi` — 高密度屏幕
 - `xhdpi` — 特高的密度屏幕
 - `nodpi` — 缩放资源
 - `tvdpi` — 在 API 级别 13 (Android 3.2) 屏幕的 `mdpi` 和 `hdpi` 之间中引入。
- **触摸屏类型**—指定触摸屏设备可能具有的类型。可能的值为 `notouch` (没有触摸屏) `stylus` (电阻式触摸屏适用于触笔)，和 `finger` (触摸屏)。
- **键盘可用性**—指定哪种键盘是可用。这可能会在应用程序的生存期内更改—例如当用户在打开的硬件键盘。可能的值有：
 - `keysexposed` — 该设备已使用键盘。如果没有启用的软件键盘，然后才使用此选项时打开硬件键盘。
 - `keyshidden` — 设备已硬件键盘，但其处于隐藏状态，然后不启用任何软件键盘。
 - `keyssoft` — 该设备已启用的软件键盘。
- **主文本输入法**—用于指定哪些类型的硬件密钥为可用于输入。可能的值有：
 - `nokeys` — 不没有输入任何硬件密钥。

- `qwerty` – 没有可用的全键盘。
- `12key` – 12 键硬件键盘
- **导航键可用性** – 5 方式或方向键（光板）导航时可用。这可能会更改你的应用程序的生存期内。可能的值有：
 - `navexposed` – 导航键是可供用户使用
 - `navhidden` – 导航键将不可用。
- **主要的非触摸导航方法** – 导航可在设备上的类型。可能的值有：
 - `nonav` – 可用的唯一导航设施是触摸屏
 - `dpad` – d-pad（光板）是可用于导航
 - `trackball` – 设备具有用于导航轨迹球
 - `wheel` – 其中有一个或多个方向轮可用不常见的情况
- **平台版本（API 级别）** – 格式 `v` 设备支持的 API 级别 `N`，其中 `N` 是目标的 API 级别。例如，`v11` 将目标 API 级别 11 (Android 3.0) 设备。

有关更多详细信息资源限定符请参阅[提供资源](#)Android 开发人员网站上。

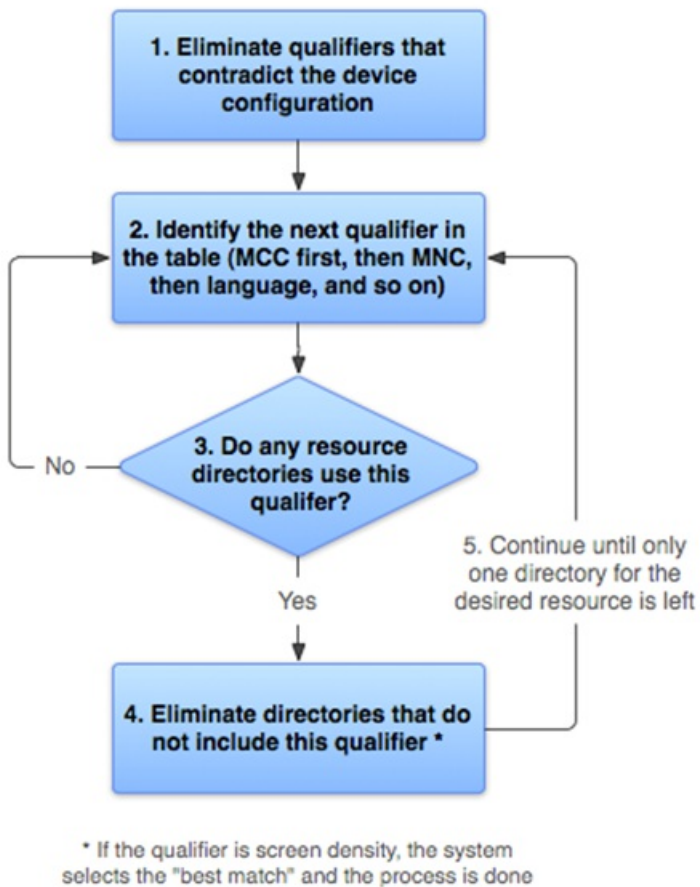
Android 如何确定使用哪些资源

它是非常可能和可能的 Android 应用程序将包含多个资源。请务必了解在该设备上运行 Android 选择应用程序的资源的方式。

Android 通过遍历以下测试的规则确定基本的资源：

- **消除相互矛盾的限定符** – 例如，如果设备方向为纵向，然后所有横向资源目录将被都拒绝。
- **忽略不支持的限定符** – 并非所有的限定符可供所有 API 级别。如果资源目录包含的设备不支持的限定符，则将忽略该资源目录。
- **确定下一步的最高优先级限定符** – 指上表选择（从上到下）的下一步最高优先级限定符。
- **保持限定符任何资源目录** – 是否存在任何匹配到上表限定符的资源目录选择（从上到下）的下一步最高优先级限定符。

下面的流程图也说明了这些规则：



当系统正在寻找特定于密度的资源，但找不到它们时，它将尝试查找其他密度的特定资源并将其扩展。Android 不一定使用默认资源。例如时寻找低密度资源和它不可用，Android 可能对默认或中等密度资源选择高密度的资源版本。这是 0.5 的因为高密度的资源可以按比例缩小到原来，这会导致比减少需要 0.75 倍的中等密度资源较少的可见性问题。

作为示例，请考虑具有以下可绘制资源目录的应用程序：

```
drawable
drawable-en
drawable-fr-rCA
drawable-en-port
drawable-en-notouch-12key
drawable-en-port-ldpi
drawable-port-ldpi
drawable-port-notouch-12key
```

和现在使用以下配置设备上运行应用程序：

- 区域设置 – en GB
- 方向–端口
- 屏幕密度 – hdpi
- 触摸屏类型 – notouch
- 主要的输入的法 – 12key

因为它们会发生冲突的区域设置中的法语资源开始时，消除 `en-GB`，我们使用保留：

```
drawable
drawable-en
drawable-en-port
drawable-en-notouch-12key
drawable-en-port-ldpi
drawable-port-ldpi
drawable-port-notouch-12key
```

接下来, 从上的表限定符选择第一个限定符: MCC 和 mnc 是否。不没有包含此限定符, 因此忽略 MCC/mnc 代码任何资源目录。

选择下一步的限定符, 则这是语言。没有匹配的语言代码的资源。不匹配的语言代码的所有资源目录 `en` 被拒绝, 以便资源的列表现在为:

```
drawable-en-port
drawable-en-notouch-12key
drawable-en-port-ldpi
```

下一步是存在的限定符是为屏幕方向, 因此所有不匹配的屏幕方向的资源目录 `port` 消除了:

```
drawable-en-port
drawable-en-port-ldpi
```

接下来是屏幕密度的限定符 `ldpi`, 这会导致一个排除多个资源目录:

```
drawable-en-port-ldpi
```

由于此过程中, Android 将使用可绘制资源的资源目录中 `drawable-en-port-ldpi` 设备。

NOTE

屏幕大小限定符提供此选择过程的一个例外。很可能适用于 Android 选择专为比当前设备提供了一个较小屏幕的资源。例如, 大屏幕设备可能使用的资源为正常大小屏幕提供。但是此反之不成立: 同一个大屏幕设备将使用为加大屏幕提供的资源。如果 Android 找不到与给定的屏幕大小匹配的资源集, 该应用程序将崩溃。

为不同屏幕创建资源

2018/10/26 • [Edit Online](#)

Android 本身在设备上运行多个不同，每个都拥有各种分辨率、屏幕大小和屏幕密度。Android 将执行缩放和调整大小以使应用程序在这些设备上运行，但这可能会导致非最佳用户体验。例如，图像可能会显示模糊，或者它们可能会按预期在视图上定位。

概念

几个术语和概念非常重要，若要了解为支持多个屏幕。

- **屏幕大小**—用于显示你的应用程序的物理空间量
- **屏幕密度**—在屏幕上任何给定区域中的像素数。典型单位是度量的每英寸点数 (dpi)。
- **解决方法**—在屏幕上的像素为单位的总数。开发应用程序时，解析不可与屏幕大小和密度一样重要。
- **密度无关像素 (dp)**—虚拟单位度量值以允许布局设计为独立的密度。此公式用于将分发点转换成屏幕像素：
$$px = dp \times dpi \div 160$$
- **方向**—屏幕的方向被视为可横向宽度大于高度时。与此相反，纵向方向为时屏幕的高度大于宽度。如用户旋转设备，可以在应用程序的生存期内更改方向。

请注意，这些概念的前三个相互关联—提高分辨率不增加密度会增加屏幕大小。但是如果增加的密度和解决方法，那么屏幕大小可以保持不变。屏幕大小、密度和解决方法之间的关系使变得复杂屏幕支持快速。

为了帮助应对这种复杂性，Android 框架倾向于使用*密度无关的像素 (dp)* 对于屏幕的布局。通过使用密度无关的像素，UI 元素将显示以用户在具有不同密度的屏幕上具有相同的物理大小。

支持不同屏幕大小和密度

Android 处理大部分工作来呈现的每个屏幕配置正确的布局。但是，有一些执行的操作，可以帮助解决问题的系统。

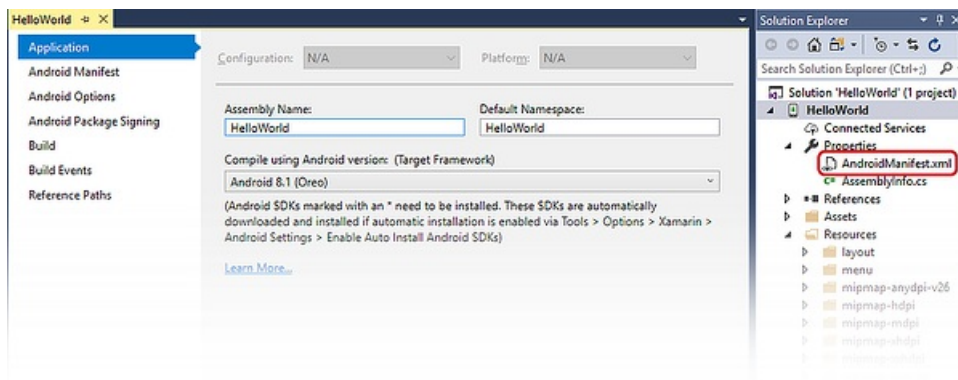
使用而不是在布局中的实际像素密度无关像素已足够在大多数情况下，以确保密度独立性。Android 将在运行时为适当大小缩放绘图。但是，有可能，缩放将导致位图，以显示模糊。若要解决此问题，提供针对不同密度的备用资源。当设计用于多个分辨率和屏幕密度的设备，它将证明管理更轻松开始使用更高的分辨率或密度映像然后再减少。

声明支持的屏幕大小

声明的屏幕大小可确保仅受支持的设备可以下载应用程序。这可以通过设置[支持屏幕](#)中的元素 **AndroidManifest.xml** 文件。此元素用于指定应用程序支持何种屏幕大小。如果在应用程序可以正确地放置其布局以填满屏幕支持被视为给定的屏幕。通过使用此清单元素，该应用程序将不会显示在 [Google Play](#) 适用于不符合屏幕规范的设备。但是，应用程序仍将具有不受支持的屏幕，设备上运行，但两种布局可能出现的模糊和像素化。

中的声明支持的屏幕 sixes **Properites/AndroidManifest.xml** 解决方案的文件：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



编辑**AndroidManifest.xml**包括支持屏幕:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="HelloWorld.HelloWorld">
    <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="27" />
    <supports-screens android:resizable="true"
        android:smallScreens="true"
        android:normalScreens="true"
        android:largeScreens="true" />
    <application android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true" android:theme="@style/AppTheme">

    </application>
</manifest>
```

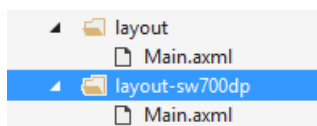
针对不同屏幕大小提供备用布局

备用布局可自定义视图的特定的屏幕大小、更改的位置或组件 UI 元素的大小。

从 API 级别 13 (Android 3.2) 开始, 屏幕大小将不推荐使用而改为使用 **swNdp** 限定符。此新限定符声明需要给定布局空间量。建议用于运行在 Android 3.2 或更高版本的应用程序应使用这些较新的限定符。

例如, 如果布局所需的最小 700 屏幕宽度的分发点, 备用布局会转的文件夹中布局 **sw700dp**:

- Visual Studio
- Visual Studio for Mac



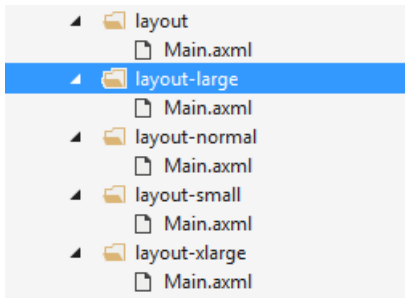
作为一般准则, 下面是一些数字为各种设备:

- 典型的电话 – 320 dp: 典型的电话
- 5"平板电脑 /"tweener"设备 – 480 dp: 例如 Samsung 记录
- 7"平板电脑 – 600 dp: 如 Barnes & Noble 四处
- 10 个"平板电脑 – 720 dp: 如 Motorola Xoom

应用程序的目标 API 级别最多 12 (Android 3.1) 布局应放入目录使用限定符**小型/正常/大型 /加大**作为不同屏幕大小所提供的大多数设备的泛化。例如, 在下图中, 有四个不同的屏幕大小的备用资源:

- Visual Studio

- [Visual Studio for Mac](#)

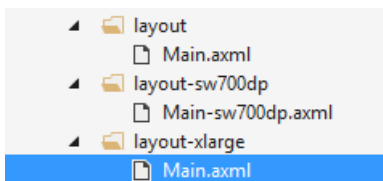


以下是较旧的 pre-API 级别 13 屏幕大小限定符如何与密度无关的像素进行比较的比较：

- 426 dp x 320 dp 是小
- 470 dp x 320 dp 是正常
- 640 dp x 480 dp 是大型
- 960 dp x 720 dp 是加大

较新屏幕的大小在 API 级别 13 中的限定符并设置具有优先权要高于 API 级别 12 和更低的较旧屏幕限定符。对于跨应用程序，将旧的和新的 API 级别，它可能有必要创建备用资源使用这两个集的限定符，如以下屏幕截图中所示：

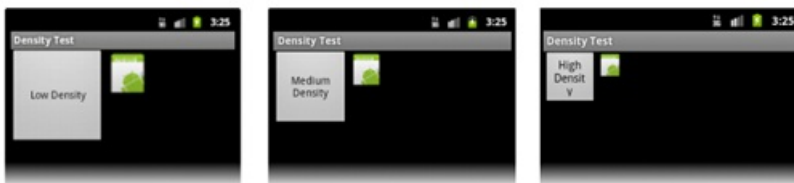
- [Visual Studio](#)
- [Visual Studio for Mac](#)



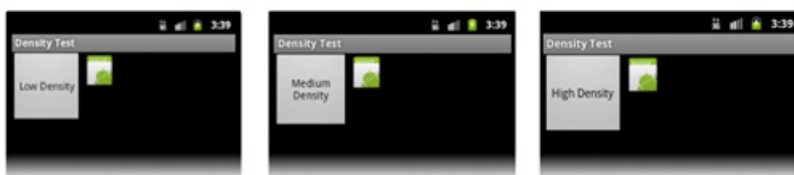
针对不同屏幕密度提供不同的位图

虽然 Android 缩放根据设备的位图，位图本身可能不完美地增加或减少：它们可能会变得模糊。提供适用于屏幕密度的位图将缓解此问题。

例如如下，图是可能发生的布局和外观问题的示例时密度-指定未提供资源。



这与使用特定于密度的资源设计的布局：

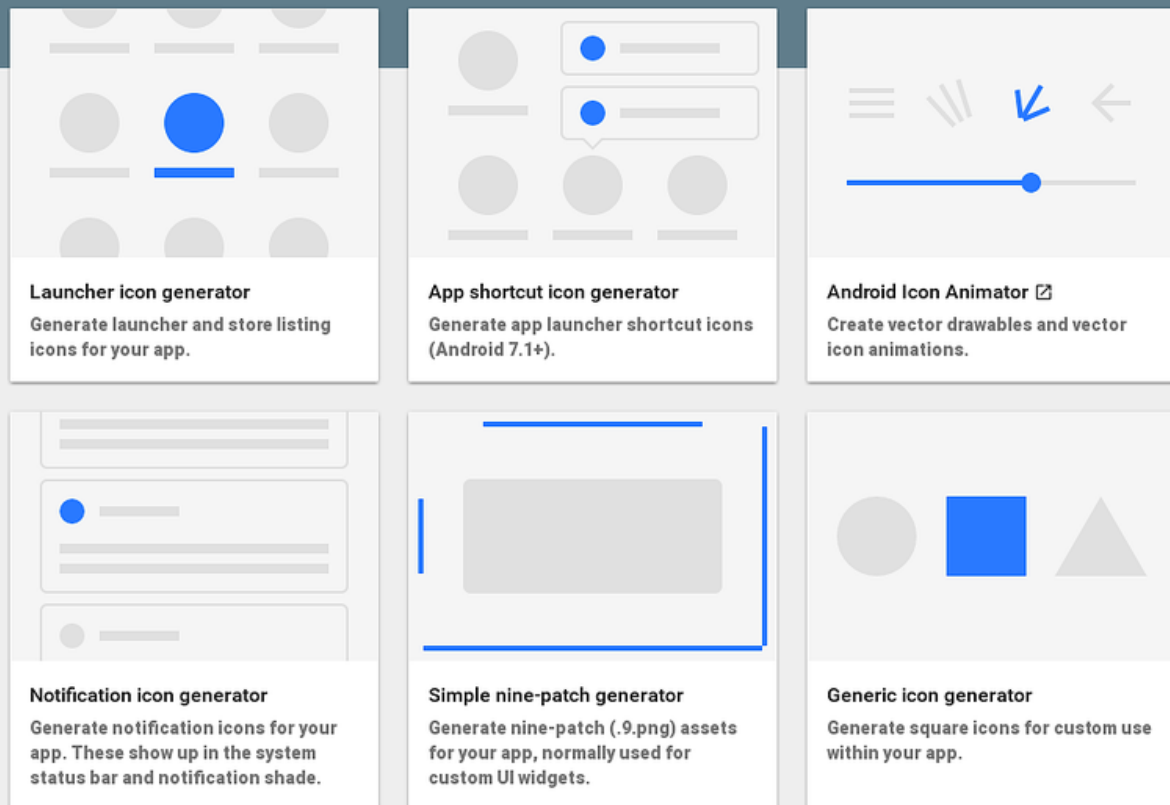


Android Asset Studio 创建不同的密度资源

这些位图不同密度的创建可能会有点令人厌倦。在这种情况下，Google 已创建一个在线实用程序，这样可以减少一些涉及与名为这些位图创建内容的麻烦 [Android Asset Studio](#)。

Android Asset Studio

A collection of tools to easily generate assets such as launcher icons for your Android app.



此网站将创建一个映像，从而针对四个通用屏幕密度的位图的帮助。Android Asset Studio 将然后创建一些自定义位图，并可以稍后下载为 zip 文件。

多个屏幕的提示

Android 上创作多个设备上，运行，屏幕大小和屏幕密度的组合可能会感到不知所措。以下提示可帮助最大程度减少支持各种设备所需的工作量：

- 仅设计和开发为所需-有许多不同设备，但某些极少数外观造型，可能需要花费大量精力来设计和开发的存在。**屏幕大小和密度**仪表板是提供数据的屏幕大小/屏幕密度矩阵分解的 Google 提供的页。此明细提供有关如何支持屏幕上的开发工作的见解。
- 使用分发点而不是像素-像素会变得麻烦作为屏幕密度的更改。请不要对像素值。避免以 dp（密度无关像素为单位）支持的像素为单位。
- 避免 **AbsoluteLayout** 只要可能-它在 API 级别 3 (Android 1.5) 中已弃用，并且会导致脆弱的布局。不应使用。相反，尝试使用如下所示更灵活的布局小组件 **LinearLayout**, **RelativeLayout**, 或新**GridLayout**。
- 选取一个为默认布局方向-例如，而不是提供的备用资源布局 **land**并布局端口，放置的资源在横向布局，以及为纵向的资源布局端口。
- 使用高度和宽度的 **LayoutParams** -在 XML 布局文件中，定义 UI 元素时 Android 应用程序中使用**wrap_content**和**fill_parent**值将为获得更大成功确保正确查看跨不同设备比使用像素或密度无关的单位。这些维度值导致 Android 来相应地缩放位图资源。鉴于相同原因，密度无关的单位是最适合保留有关何时指定边距和填充的 UI 元素。

测试多个屏幕

Android 应用程序必须针对所有受支持的配置进行测试。理想情况下应在实际设备自身上测试设备，但在许多情况下这是不可能或可行。在这种情况下，每个设备配置为使用仿真程序和 Android 虚拟设备安装程序将很有用。

Android SDK 提供了一些仿真器外观可用于创建 Avd 将复制大小、密度和多个设备的分辨率。许多硬件供应商同样提供为其设备的外观。

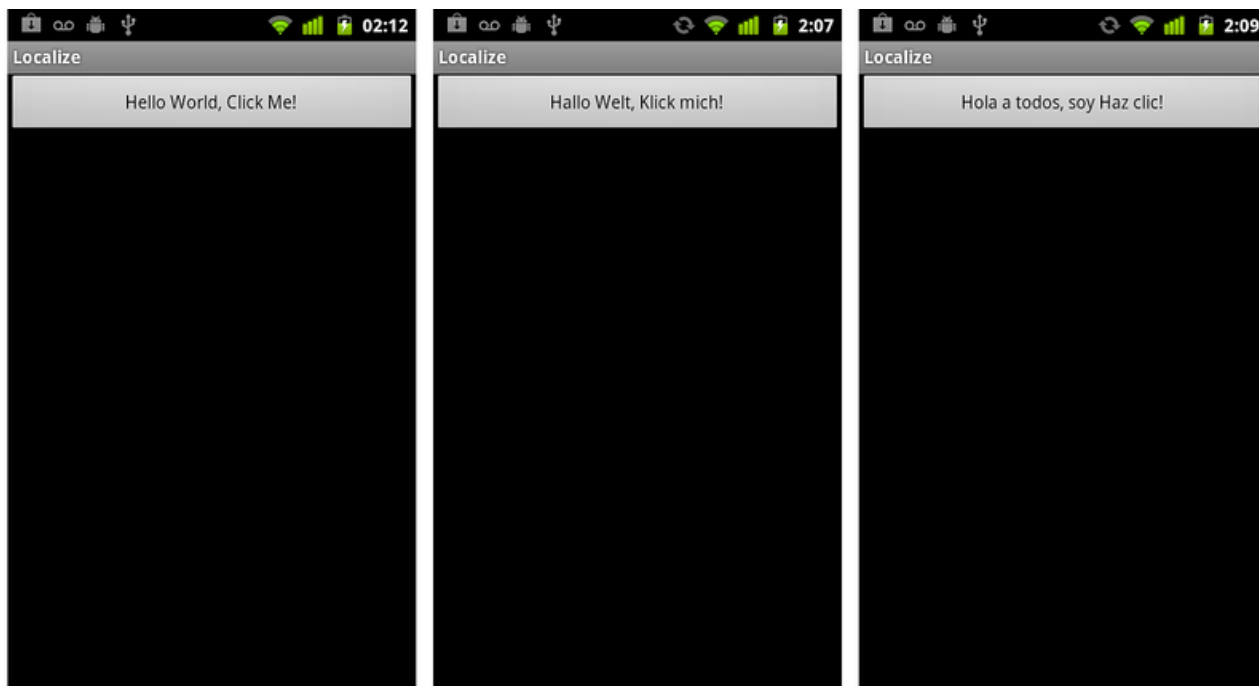
另一种方法是使用第三方测试服务的服务。这些服务将需要对 apk 进行，许多不同设备上运行，然后提供应用程序的工作方式的反馈。

应用程序本地化和字符串资源

2018/10/26 • [Edit Online](#)

应用程序本地化是提供备用资源，以针对某个特定区域设置的操作。例如，你可能会对于各种国家/地区，提供本地化的语言字符串或可能会更改颜色或布局以匹配特定区域性。Android 将加载并在运行时无需更改源代码时使用适用于设备的区域设置的资源。

例如下，图显示了在三个不同的设备区域设置中，运行的相同应用程序，但每个按钮中显示的文本是特定于每个设备设置为的区域设置：

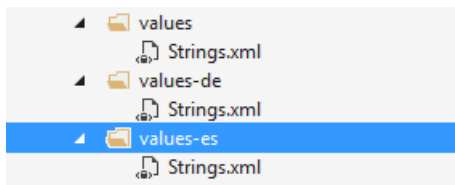


在此示例中，布局文件的内容**Main.xml**外观如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button
        android:id="@+id/myButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
</LinearLayout>
```

在上面的示例中，为按钮的字符串从资源加载通过提供的资源 ID 的字符串：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



本地化的 Android 应用

读取[简介本地化](#)的提示和本地化移动应用的指南。

[本地化 Android 应用](#)指南包含有关如何将字符串翻译和本地化使用 Xamarin.Android 的映像更具体的示例。

相关链接

- [本地化的 Android 应用](#)
- [跨平台本地化概述](#)

使用 Android 资产

2018/10/26 • [Edit Online](#)

`_资产_`提供一种方法要包含在你的应用程序中的任意文件，例如文本、xml、字体、音乐和视频。如果尝试以包括这些文件作为"资源"，Android 将处理它们，到其资源的系统和将无法获取原始数据。如果你想要访问数据保持不变，资产是一种方式执行此操作。

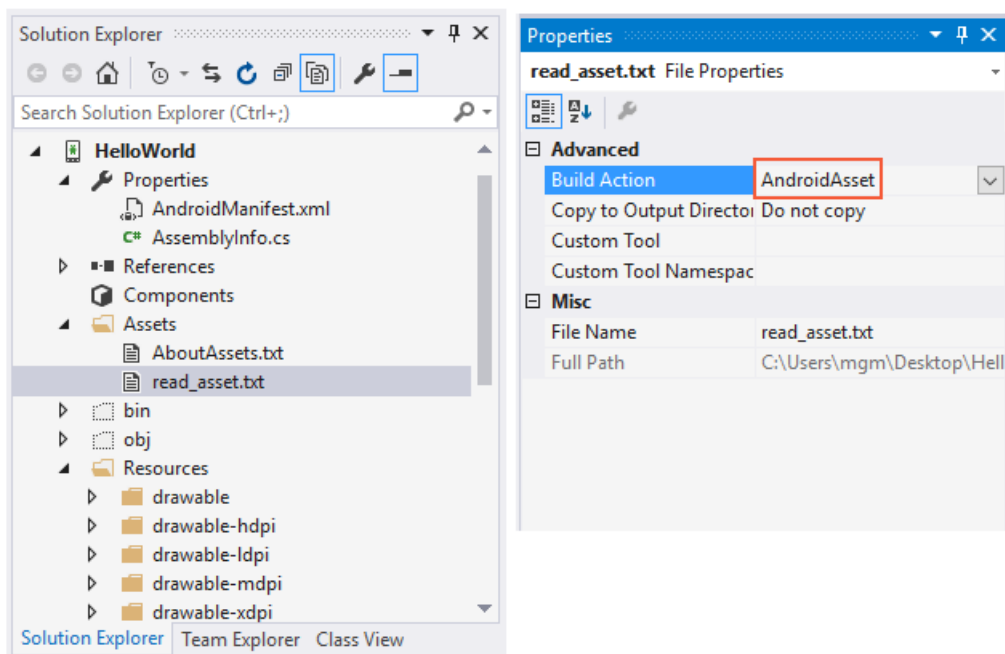
资产添加到你的项目将显示就像从你的应用程序使用可以读取的文件系统一样 `AssetManager`。在此简单演示中，我们要将文本文件资产添加到我们的项目，使用读取该 `AssetManager`，并将其显示在一个 `TextView`。

将资产添加到项目

资产，请转 `Assets` 项目文件夹。将新的文本文件添加到此文件夹名为 `read_asset.txt`。将一些文本放入其中，像"我来自资产！"。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

Visual Studio 应已设置生成操作到此文件 `AndroidAsset`:



选择正确 **BuildAction** 可确保，将在编译时到 APK 中打包该文件。

读取资产

使用读取资产 `AssetManager`。实例 `AssetManager` 即可访问 `资产属性` `Android.Content.Context`，例如活动。在下面的代码中，我们打开我们 `read_asset.txt` 资产，读取内容，并将其使用一个 `TextView` 显示。

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Create a new TextView and set it as our view
    TextView tv = new TextView (this);

    // Read the contents of our asset
    string content;
    AssetManager assets = this.Assets;
    using (StreamReader sr = new StreamReader (assets.Open ("read_asset.txt")))
    {
        content = sr.ReadToEnd ();
    }

    // Set TextView.Text to our asset content
    tv.Text = content;
    SetContentView (tv);
}
```

运行应用程序

运行应用程序, 你应该会看到如下:



相关链接

- [AssetManager](#)
- [上下文](#)

字体

2018/10/26 • [Edit Online](#)

概述

从 API 级别 26 开始, Android SDK 允许字体作为资源, 就像布局一样来处理或绘图。[Android 支持库 26 NuGet](#)将向后移植到应用程序的目标 API 级别 14 或更高版本的新字体 API 的。

针对 API 26 或之后安装 Android 支持库 v26, 有两种方法中的 Android 应用程序使用的字体:

1. **打包为 Android 资源字体**—这可确保字体始终可供该应用程序, 但会增加 APK 的大小。
2. **下载字体** – Android 还支持下载从字体_字体提供程序_。字体提供程序会检查该字体是否已在设备上。如有必要, 将下载并缓存在设备上的字体。此字体可以在多个应用程序之间共享。

相似的字体(或可能有几种不同样式的字体)可能会分组到_字体系列_。这允许开发人员指定的字体, 例如它的权重, 某些属性和 Android 将自动选择合适的字体的字体系列。

Android 支持库 v26 会将字体的支持向后移植到 API 级别 26。如果目标是较旧的 API 级别, 则需声明 `app` XML 命名空间, 并使用 `android:` 命名空间和 `app:` 命名空间来命名各种字体属性。如果仅使用 `android:` 命名空间, 则字体不会显示在运行 API 25 或更低级别的设备上。例如, 以下 XML 代码片段声明一个新的[字体系列](#)资源, 该资源可以在 API 14 及更高级别中使用:

```
<?xml version="1.0" encoding="utf-8"?>
<font-family
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <font android:font="@font/sourcesanspro_regular"
        android:fontStyle="normal"
        android:fontWeight="400"
        app:font="@font/sourcesanspro_regular"
        app:fontStyle="normal"
        app:fontWeight="400" />

</font-family>
```

只要字体以正确方式提供给 Android 应用程序, 它们可应用于 UI 小组件通过设置 `fontFamily` 属性。例如, 以下代码片段演示了如何在一个 TextView 中显示字体:

```
<TextView
    android:text="The quick brown fox jumped over the lazy dog."
    android:fontFamily="@font/sourcesanspro_regular"
    app:fontFamily="@font/sourcesanspro_regular"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

本指南将首先讨论如何使用字体为 Android 资源, 并接着讨论如何在运行时的字体下载。

字体作为资源

将字体打包进 Android APK 中可确保它对应用程序始终可用。字体文件(.TTF 或 .OTF 文件)可以像任何其他资源一样添加到 Xamarin.Android 应用程序中, 只需将文件复制到 Xamarin.Android 项目的 **Resources** 文件夹中的子目录即可。字体资源保存在项目 **Resources** 文件夹的 **font** 子目录中。

NOTE

字体应有生成操作的**AndroidResource**或它们将不打包到最终 APK。生成操作应由 IDE 自动设置。

有许多类似字体文件（例如，使用不同的权重或样式相同字体）时，可能将它们分组到一个字体系列。

字体系列

字体系列是一套有不同粗细和样式的字体。例如，粗体和斜体可能有不同的字体文件。字体系列由 XML 文件（保留在 **Resources/font** 目录中）中的 `font` 元素定义。每个字体系列都应该有其自己的 XML 文件。

若要创建字体系列，请首先添加到所有字体资源/字体文件夹。然后，在字体系列的字体文件夹中创建新的 XML 文件。XML 文件的名称与所引用的字体没有关联或关系；资源文件可以采用任何合法的 Android 资源文件名称。此 XML 文件会有一个根 `font-family` 元素，其中包含一个或多个 `font` 元素。每个 `font` 元素声明一个字体的特性。

以下 XML 是 *Sources Sans Pro* 字体的字体系列示例，定义多个不同的字体粗细。它在 **Resources/font** 文件夹中另存为名为 **sourcesanspro.xml** 的文件：

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <font android:font="@font/sourcesanspro_regular"
        android:fontStyle="normal"
        android:fontWeight="400"
        app:font="@font/sourcesanspro_regular"
        app:fontStyle="normal"
        app:fontWeight="400" />
    <font android:font="@font/sourcesanspro_bold"
        android:fontStyle="normal"
        android:fontWeight="800"
        app:font="@font/sourcesanspro_bold"
        app:fontStyle="normal"
        app:fontWeight="800" />
    <font android:font="@font/sourcesanspro_italic"
        android:fontStyle="italic"
        android:fontWeight="400"
        app:font="@font/sourcesanspro_italic"
        app:fontStyle="italic"
        app:fontWeight="400" />
</font-family>
```

`fontStyle` 属性具有两个可能值：

- **normal** – 普通字体
- **italic** – 倾斜字体

`fontWeight` 特性对应于 CSS `font-weight` 特性，是指字体粗细。值的范围为 100 - 900。以下列表介绍了常见字体粗细值及其名称：

- **Thin** – 100
- **Extra Light** – 200
- **Light** – 300
- **Normal** – 400
- **Medium** – 500
- **Semi Bold** – 600
- **Bold** – 700
- **Extra Bold** – 800
- **Black** – 900

一旦定义字体系列，它可以是以声明方式使用通过设置 `fontFamily`，`textStyle`，和 `fontWeight` 布局文件中的属性。例如 600 权重字体（普通）和斜体文本样式，将设置以下 XML 代码片段：

```
<TextView
    android:text="Sans Source Pro semi-bold italic, 600 weight, italic"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fontFamily="@font/sourcesanspro"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_horizontal"
    android:fontWeight="600"
    android:textStyle="italic"
/>
```

以编程方式指定字体

可以通过使用以编程方式设置字体 `Resources.getFont` 方法来检索 `Typeface` 对象。许多视图具有 `TypeFace` 可用于将字体分配给该小组件的属性。此代码片段演示如何以编程方式将字体设置上一个 `TextView`：

```
Android.Graphics.Typeface typeface = this.Resources.GetFont(Resource.Font.caveat_regular);
textView1.Typeface = typeface;
textView1.Text = "Changed the font";
```

`GetFont` 方法将自动加载字体系列内的第一种字体。若要加载与特定样式相符的字体，请使用 `Typeface.Create` 方法。此方法会尝试加载与指定样式相符的字体。例如，以下代码片段会尝试从 **Resources/font** 中定义的字体系列加载一个粗体 `Typeface` 对象：

```
var typeface = Typeface.Create("<FONT FAMILY NAME>", Android.Graphics.TypefaceStyle.Bold);
textView1.Typeface = typeface;
```

下载字体

而不是将字体打包为应用程序资源，Android 可以从远程源下载字体。这会减少 APK 的大小的理想效果。

字体下载的帮助下_字体提供程序_。这是专用的内容提供程序管理的设备上的所有应用程序的字体下载和缓存。Android 8.0 包括字体提供程序来下载从字体Google 字体存储库。此默认字体提供程序是向后的移植到使用 Android 支持库 v26 API 级别 14。

当应用程序发出请求的一种字体时，字体提供程序将首先检查以查看该字体是否已在设备上。如果没有，则它将尝试下载字体。如果字体不能为已下载，然后 Android 将使用默认系统字体。一旦下载字体，可供所有应用程序在设备上，而不仅仅是发出初始请求的应用。

当发出请求以下载字体时，应用程序不会直接查询字体提供程序。相反，应用将使用的实例 `FontsContract` API (或 `FontsContractCompat` 如果正在使用支持库 26)。

Android 8.0 支持两种不同方式下载字体：

1. **声明为资源的可下载字体**—应用可能会声明到 Android 通过 XML 资源文件的可下载字体。这些文件将包含的所有 Android 需要字体时应用启动并在设备上对其进行缓存以异步方式下载的元数据。
2. **以编程方式**—Android API 级别 26 中的 `Api` 允许应用程序时运行该应用程序，以编程方式下载字体。应用程序将创建 `FontRequest` 对象为给定的字体，并将传递到此对象 `FontsContract` 类。`FontsContract` 采用 `FontRequest`，并检索从字体_字体提供程序_。Android 将以同步方式下载该字体。创建的示例 `FontRequest` 将在本指南后面所示。

无论使用哪种方法，都必须先将资源文件添加到 Xamarin.Android 应用程序，然后才能下载字体。首先，必须在 **Resources/font** 目录的 XML 文件中声明字体系列中的字体。以下代码片段是一个示例，演示了如何使用默认的字

体提供程序从 [Google 字体开源集合](#) 下载字体，该提供程序是 Android 8.0 (或支持库 v26) 附带的：

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:fontProviderAuthority="com.google.android.gms.fonts"
    android:fontProviderPackage="com.google.android.gms"
    android:fontProviderQuery="VT323"
    android:fontProviderCerts="@array/com_google_android_gms_fonts_certs"
    app:fontProviderAuthority="com.google.android.gms.fonts"
    app:fontProviderPackage="com.google.android.gms"
    app:fontProviderQuery="VT323"
    app:fontProviderCerts="@array/com_google_android_gms_fonts_certs"
>
</font-family>
```

`font-family` 元素包含以下属性，声明 Android 需要下载字体的信息：

1. **fontProviderAuthority** – 字体提供程序颁发机构，用于进行请求。
2. **fontPackage** – 请求使用的字体提供程序的包。这用于验证提供程序的标识。
3. **fontQuery** – 这是一个字符串，将帮助定位所请求的字体的字体提供程序。字体查询的详细信息是特定于字体提供程序。`QueryBuilder` 类 [可下载字体](#) 示例应用程序提供了一些有关查询格式对于字体 Google 字体开放源集合中。
4. **fontProviderCerts** – 的提供程序应使用签名的证书的哈希集的列表的资源数组。

字体定义后，可能需要提供有关的信息_字体证书_涉及的下载。

字体证书

如果字体提供程序没有预装在设备上，或者应用使用的是 `Xamarin.Android.Support.Compat` 库，则 Android 需要字体提供程序的安全证书。这些证书会列在数组资源文件中，该文件保留在 **Resources/values** 目录中。

例如，名为下面的 XML **Resources/values/fonts_cert.xml**，并将证书存储 Google 字体提供程序：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="com_google_android_gms_fonts_certs">
        <item>@array/com_google_android_gms_fonts_certs_dev</item>
        <item>@array/com_google_android_gms_fonts_certs_prod</item>
    </array>
    <string-array name="com_google_android_gms_fonts_certs_dev">
        <item>

MIEEqDCCA5CgAwIBAgIJANWfUGx90071MA0GCSqGSIb3DQEBBAUAMIGUMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcn5pYTEwMBQGA1UEBxMNTW91bnRhaw4gVmllldzEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECXMhQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDEiMCAGCSqGSIb3DQEJARYTYW5kcm9pZEBhbmRyb2lkLmNvbTAeFw0wODAwMTUyMzMNTZaFw0zNTA5MDEyMzMNTZaMIGUMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcn5pYTEwMBQGA1UEBxMNTW91bnRhaw4gVmllldzEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECXMhQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDEiMCAGCSqGSIb3DQEJARYTYW5kcm9pZEBhbmRyb2lkLmNvbTCCASAwDQYJKoZIhvcNAQEBBQADggENADCCAQgCggEBANbOLggKv+IxTDG
Ns8/TGFy0PTP6DHTvbbr24kT9ixcOd9W+EaBPWW+wPPKQmsHxajtWjmQwWfna8mZuSeJS48LIgAZlKkpFeVyxw0QMBujb8X8ETrWY550NaFtI6
t9+u7hZeTfHwqNvacKhp1RbE6dBRGWynwMVX8XW8N1+UjFaq6GCJukT4qmpN2afb8sCjUiqqGuMwYXrFVee74bQgLHWGJwPmvmLHC69EH6kWr2
Zijx40KXlSIx2xT1AsShee70w5iDBiK4aph27YH3TxkXy9V89TDdexAcKk/cVHYNnDBapcav17y0RiQ4biu8ymM8Ga/nmzhRKya6G0cGw8CAQOj
gfwwgfkWHDYVR00BBYEFi0cxb6VTEM8YYY6FbBMvAPyT+CyMIHJBgNVHSMegcEwgb6AFi0cxb6VTEM8YYY6FbBMvAPyT+CyoYGapIGXMIGUMQs
wCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcn5pYTEwMBQGA1UEBxMNTW91bnRhaw4gVmllldzEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECx
MHQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDEiMCAGCSqGSIb3DQEJARYTYW5kcm9pZEBhbmRyb2lkLmNvbYIJANWfUGx90071MAWGA1UdEwQFM
AMBAF8wDQYJKoZIhvcNAQEEBQADggEBABnTDPEF+3iSP0wNfdIjZiZ1A1nrPzgAIHVvXxunW7SBrdhEglQZBbKJEk5kT0mtKoOD1JMRSu1xuTKEB
ahWRbqHsXclaxJ0BADb0kkjVEJu/Lh5hgYZn0jv1ba8Ld7HCKePCVePoTJBdI4fvugnL8TsgK05aIskyY0hKI9L8KFqfGTl1lzOv2KowD0KwWtA
WPoGChZxmQ+nBli+gWYmZM1vAkP+aayLe0a1EQiml0a10762r0GX00ks+UeXde2Z4e+8S/pf7pITEI/tP+MxJTLw9QUUEv9lKtk+jkbqxbsh8n
fBUapfKqYn0eidpqw2AzVp3juYl7//fKnaPhJD9gs=

        </item>
    </string-array>
    <string-array name="com_google_android_gms_fonts_certs_prod">
        <item>

MIEQZCCAYugAwIBAgIJAMLgh0ZkSjCNMA0GCSqGSIb3DQEBBAUAMHQCzAJBgNVBAYTA1VTMRMwEQYDVQQQIEwpcDYwZm9ybm1hMRYwFAYDVQQ
HEw1Nb3VudGFpb1BwW3MRQWEGYDVQQKEwtHb29nbGUGSW5jLjEjEQMA4GA1UECXMhQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDAeFw0wODAwMj
EyMzEzEzRaFw0zNjAxMDcyMzEzEzRaMHQXcCzAJBgNVBAYTA1VTMRMwEQYDVQQQIEwpcDYwZm9ybm1hMRYwFAYDVQQHEw1Nb3VudGFpb1BwW3M
RQWEGYDVQQKEwtHb29nbGUGSW5jLjEjEQMA4GA1UECXMhQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDCCASAwDQYJKoZIhvcNAQEBBQADggENADCC
AQgCggEBAktwLGDY06IIRgqWbxJOKdoR8qtW0I9Y4sypEwPpt1TtcvZApxsdyxMJZ2J0Rland2qSGT2y5b+3JkKedxiLDmpHpDs2WCbdxgRcz
fey5YZnTJ4VZbH0xqVWw/8lGmPav5xVwnIiJS6HXk+BVKZF+JcWjAsb/GEUq/efdpuzSqeYTCfi6idkyugwfyWxFU1+5fZKUaRKYCwkFQVfcAs
1fXA5V+++FGfvjJ/CxURaSxaBvGdGdhfXE28LWuT9ozC15xw4Yq50GazvV24mZVs000yZ31j7kYvtwYK6NeADwbSxDdJEq04k//0zOHKrUiGYX
tqw/A0LFFtqoZKFjnkCAQOjgdkwgdYwHQYDVR00BBYEFMD9jMIhF1Ylmn/Tgt9r45jk14alMIGmBgNVHSMegZ4wgZuAFMD9jMIhF1Ylmn/Tgt9r
45jk14al0XikdjB0MQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcn5pYTEwMBQGA1UEBxMNTW91bnRhaw4gVmllldzEUMBIGA1UEChMLR29
vZ2x1IEluYy4xEDA0BgNVBASTB0FuZHJvaWQxEADA0BgNVBAMTB0FuZHJvaW5SCCQDC4IdGZEowjTAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBA
UAA4IBAQBt0L074UwLDYKqs6Tm8/yzKkEu116FmH4rkaymUIE0P9KaMftG1MexFlaYjzmB20xZyl6euNXEsQH8jgwyxCUKRJNexBiGcCEyj6z+
a1fuHHvkiaai+KL8W1EyNmgjmyy8AW7P+LLlkr+ho5zEHatRbM/YANqGcFh5iZBqpnHf1SKMXFh4dd239FJ1jWYfbMDMy3NS5CTMQ2XF11Mvcy
UTdZPERjzQfTbQe3aDQsQcafEQPD+nqActiFKZ0Np0IS9L9kr/wbNvyz6ENwPiTrjV2KRKEjH78ZMcUQXg0L3BYHJ31c69Vs5Ddf9uUGGMY1dX3W
fMBEmh/9iFBDAaTCK

        </item>
    </string-array>
</resources>
```

这些资源中的文件的位置，该应用是能够下载字体。

声明可下载字体作为资源

通过列出中的可下载字体**AndroidManifest.XML**，Android 将应用程序首次启动时以异步方式下载字体。该字体的本身列出在数组资源文件中，与以下类似：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="downloadable_fonts" translatable="false">
        <item>@font/vt323</item>
    </array>
</resources>
```

若要下载这些字体，他们需要在中声明**AndroidManifest.XML**通过添加 `meta-data` 的小孩 `application` 元素。例如，如果在一个资源文件中声明的可下载字体**Resources/values/downloadable_fonts.xml**，则必须添加到清单中此代码片段：


```
<meta-data android:name="downloadable_fonts" android:resource="@array/downloadable_fonts" />
```

下载字体具有字体 Api

可以以编程方式通过实例化下载字体 `FontRequest` 对象并传递给 `FontContractCompat.RequestFont` 方法。

`FontContractCompat.RequestFont` 方法会首先检查以查看在设备上是否存在该字体，然后如有必要将以异步方式查询字体提供程序并尝试下载应用程序的字体。如果 `FontRequest` 是无法下载字体，则 Android 将使用默认系统字体。

一个 `FontRequest` 对象包含将使用的字体提供程序查找和下载字体信息。一个 `FontRequest` 需要四个条信息：

1. **字体提供程序颁发机构**–字体提供程序颁发机构，用于进行请求。
2. **字体包**–请求使用的字体提供程序的包。这用于验证提供程序的标识。
3. **字体查询**–这是一个字符串，将帮助定位所请求的字体的字体提供程序。字体查询的详细信息是特定于字体提供程序。字符串的详细信息是特定于字体提供程序。`QueryBuilder` 类[可下载字体](#)示例应用程序提供了一些有关查询格式对于字体 Google 字体开放源集合中。
4. **字体提供程序证书**–的提供程序应使用签名的证书的哈希集的列表的资源数组。

此代码片段示范了实例化一个新的 `FontRequest` 对象：

```
FontRequest request = new FontRequest("com.google.android.gms.fonts", "com.google.android.gms",  
<FontToDownload>, Resource.Array.com_google_android_gms_fonts_certs);
```

在上一代代码段 `FontToDownload` 是一个查询，可帮助在 Google 字体开放源集合中的字体。

然后才传递 `FontRequest` 到 `FontContractCompat.RequestFont` 方法，必须创建的两个对象：

- `FontContractCompat.FontRequestCallback` – 这是一个抽象类，后者必须扩展。它是将一个回调时调用 `RequestFont` 已完成。Xamarin.Android 应用程序必须子类 `FontContractCompat.FontRequestCallback` 并重写 `OnTypefaceRequestFailed` 和 `OnTypefaceRetrieved`，提供时下载失败或成功分别执行的操作。
- `Handler` – 这是 `Handler` 其将通过使用 `RequestFont` 下载在线程上的字体，如有必要。字体应不 UI 线程上下载。

此代码片段示范了C#以异步方式将一种字体下载 Google 字体开放源集合中的类。它实现 `FontRequestCallback` 接口，并引发C#事件时 `FontRequest` 已完成。

```

public class FontDownloadHelper : FontsContractCompat.FontRequestCallback
{
    // A very simple font query; replace as necessary
    public static readonly String FontToDownload = "Courgette";

    Android.OS.Handler Handler = null;

    public event EventHandler<FontDownloadEventArgs> FontDownloaded = delegate
    {
        // just an empty delegate to avoid null reference exceptions.
    };

    public void DownloadFonts(Context context)
    {
        FontRequest request = new FontRequest("com.google.android.gms.fonts",
"com.google.android.gms", FontToDownload , Resource.Array.com_google_android_gms_fonts_certs);
        FontsContractCompat.RequestFont(context, request, this, GetHandlerThreadHandler());
    }

    public override void OnTypefaceRequestFailed(int reason)
    {
        base.OnTypefaceRequestFailed(reason);
        FontDownloaded(this, new FontDownloadEventArgs(null));
    }

    public override void OnTypefaceRetrieved(Android.Graphics.Typeface typeface)
    {
        base.OnTypefaceRetrieved(typeface);
        FontDownloaded(this, new FontDownloadEventArgs(typeface));
    }

    Handler GetHandlerThreadHandler()
    {
        {
            if (Handler == null)
            {
                HandlerThread handlerThread = new HandlerThread("fonts");
                handlerThread.Start();
                Handler = new Handler(handlerThread.Looper);
            }
            return Handler;
        }
    }
}

public class FontDownloadEventArgs : EventArgs
{
    public FontDownloadEventArgs(Android.Graphics.Typeface typeface)
    {
        Typeface = typeface;
    }
    public Android.Graphics.Typeface Typeface { get; private set; }
    public bool RequestFailed
    {
        get
        {
            return Typeface != null;
        }
    }
}

```

若要使用此帮助器，一个新 `FontDownloadHelper` 创建和分配事件处理程序：

```
var fontHelper = new FontDownloadHelper();

fontHelper.FontDownloaded += (object sender, FontDownloadEventArgs e) =>
{
    //React to the request
};
fontHelper.DownloadFonts(this); // this is an Android Context instance.
```

总结

本指南介绍 Android 8.0 支持可下载的字体和字体作为资源中的新 Api。它介绍了如何将现有的字体嵌入在 APK 中以及在布局中使用它们。它还将讨论如何 Android 8.0 支持下载字体从字体提供程序，以编程方式或通过声明字体元数据资源文件中。

相关链接

- [fontFamily](#)
- [FontConfig](#)
- [FontRequest](#)
- [FontsContractCompat](#)
- [Resources.GetFont](#)
- [Typeface](#)
- [Android 支持库 26 NuGet](#)
- [在 Android 中使用字体](#)
- [CSS 字体粗细规范](#)
- [Google 字体开放源集合](#)
- [San Pro 源](#)

活动生命周期

2018/10/26 • [Edit Online](#)

活动是 Android 应用程序的基本构造块，它们可以存在于多个不同的状态。活动生命周期以实例化开始和结尾析构，并且包括很多状态之间。当活动更改状态时，会调用相应的生命周期事件方法，通知即将发生的状态更改的活动并使其能够执行的代码以适应所做的更改。本文分析活动的生命周期，并解释了负责该活动具有这些状态更改为良好、可靠的应用程序的一部分的每一阶段。

活动生命周期概述

活动是一个不寻常的特定于 Android 编程概念。在传统的应用程序开发通常会有一个静态 main 方法，执行以启动应用程序。与 Android，但是，情况就不同了；Android 应用程序可以通过任何已注册的活动应用程序中启动。在实践中，大多数应用程序只能指定为应用程序入口点的特定活动。但是，如果应用程序崩溃，或终止由操作系统，操作系统可以尝试重新启动应用程序在打开的最后一个活动或以前的活动堆栈内的其他任何位置。此外，操作系统可能会暂停活动，当它们不处于活动状态，并回收它们是否内存不足。若要允许应用程序的情况活动重新启动时，尤其是当活动依赖于以前的活动中的数据的数据的正确还原其状态，必须进行仔细考虑。

活动生命周期实现为一系列操作系统的方法调用的生命周期的活动。这些方法允许开发人员实现，才可满足他们的应用程序的状态和资源管理要求的功能。

它是非常重要的应用程序开发人员分析每个活动的要求，以确定需要实现公开的活动生命周期的方法。如果不这样做可能会导致应用程序不稳定、崩溃、资源膨胀和甚至基础操作系统不稳定。

本章介绍活动生命周期中详细信息，包括：

- 活动状态
- 生命周期方法
- 保留应用程序的状态

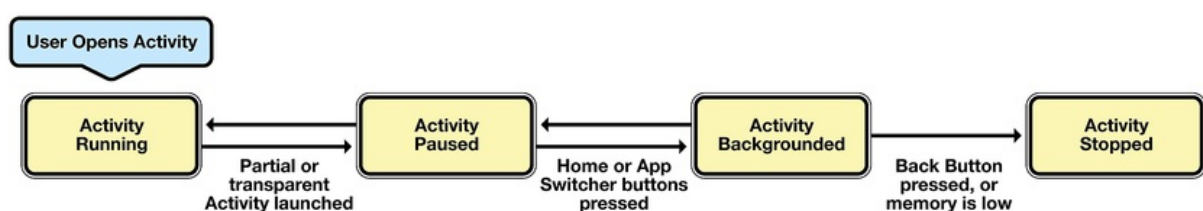
本部分还包括[演练](#)，提供了有关如何有效地将状态保存在活动生命周期中的实际示例。本章结束时了解活动生命周期和如何支持 Android 应用程序中。

活动生命周期

Android 活动生命周期包括开发人员提供资源管理框架的方法中的 Activity 类公开的集合。此框架允许开发人员以满足应用程序中的每个活动的唯一的状态管理要求并正确处理资源管理。

活动状态

Android OS 进行仲裁基于其状态的活动。这有助于确定不再使用，从而允许操作系统以回收内存和资源的活动的 Android。下图说明了活动可能在其生存期内要经历的状态：



这些状态可分为 4 个主要的组，如下所示：

1. **活动或正在运行**-活动都被视为处于活动状态或运行它们是否在前台，也称为活动堆栈的顶部。这被视为在 Android 中，最高优先级的活动，这种情况下将仅在终止由操作系统在极端情况下，此类像该活动将尝

试使用更多的内存比是设备上的可用因为这样可能会导致 UI 无响应。

2. **暂停**-当设备进入睡眠状态, 或活动是仍然可见, 但通过新的、非全尺寸或透明的活动部分隐藏时, 活动将被视为已暂停。暂停的活动是仍保持活动状态, 也就是说, 维护状态和成员的所有信息, 并保持附加到窗口管理器。这被视为第二个如果取消此活动将满足保持稳定且高度可响应的活动/正在运行活动所需的资源要求最高优先级的活动在 Android 中, , 因此, 将仅由操作系统被终止。
3. **已停止/Backgrounded**-完全遮盖的另一个活动的活动被视为已停止或在后台。请尝试停止的活动仍保留其状态和成员的信息, 以便只要有可能, 但已停止活动都被认为是最低优先级的三种状态, 并且在这种情况下, 操作系统将终止此状态, 首先以满足资源中的活动要求的更高优先级的活动。
4. **重新启动**-可能会从任何位置的活动已暂停为停止在要从内存中删除由 Android 的生命周期中。如果用户导航回必须重新启动的活动将还原到以前保存的状态, 且然后向用户显示。

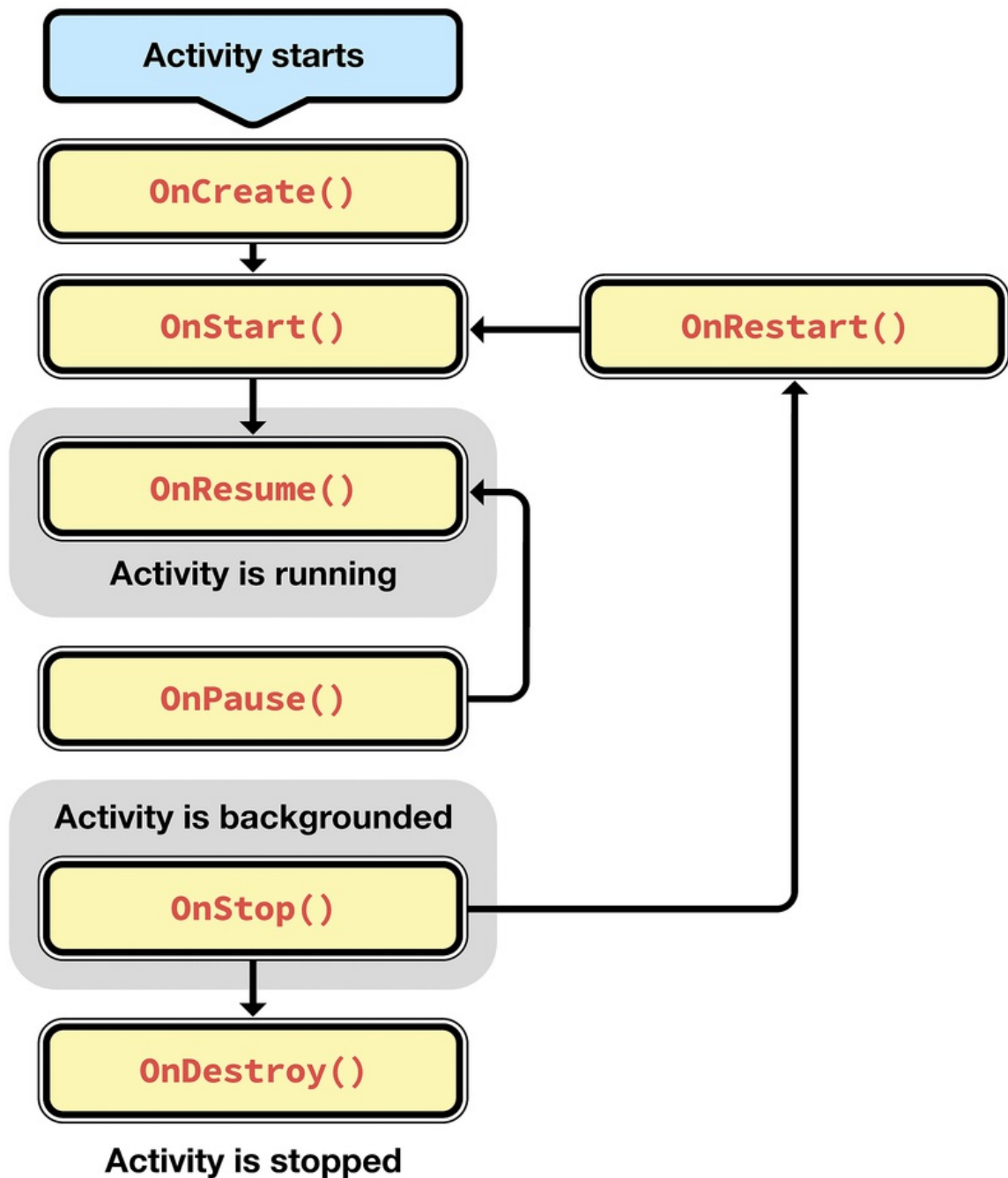
活动重新创建以响应配置更改

更复杂得多, Android 将一个详细扳手引发调用配置更改的组合中。配置更改会快速活动析构/重新 creation 周期发生活动的配置更改时, 例如, 当设备是[旋转](#)(和该活动需要重新生成在横向或纵向模式下), 当显示键盘 (和活动提供的机会来调整自身大小), 或当设备处于停靠, 等等。

配置更改仍会导致相同的活动状态更改会停止并重新启动活动过程中发生的。但是, 为了确保应用程序感觉响应, 并且还在配置更改过程中执行, 十分重要, 它们会尽可能快地处理。正因为如此, Android 有一个特定的 API, 可用于在配置更改过程中保留状态。我们将介绍这更高版本中[Lifecycle 整个管理状态](#)部分。

活动生命周期方法

Android SDK 和扩展插件, Xamarin.Android framework 所提供的用于管理应用程序中的活动的状态的功能强大的模型。当更改活动的状态时, 活动将通知由操作系统, 对该活动调用特定的方法。下图说明了这些方法中与活动生命周期的关系:



作为开发人员，您可以通过重写这些方法对在活动中的处理状态更改。请务必注意，但是，所有生命周期方法在 UI 线程上调用，且将阻止执行下的一个 UI 工作，例如隐藏当前活动中，操作系统显示一个新的活动，等等。在这种情况下，这些方法中的代码应该越简单越好感觉良好执行使用应用程序。应在后台线程上执行任何长时间运行的任务。

让我们检查每个这些生命周期方法和它们的使用：

OnCreate

OnCreate是第一种方法来创建活动调用。`OnCreate` 始终重写以执行任何启动初始化，则可能如要求的活动：

- 创建视图
- 初始化变量
- 静态数据绑定到列表

`OnCreate` 采用**捆绑**参数，它是用于存储和活动，如果绑定不是 null 之间传递状态信息和对象的字典，这表示该活动正在重新启动，并且它应还原其状态从在前一个实例。下面的代码演示如何从捆绑检索值：

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    string intentString;
    bool intentBool;

    if (bundle != null)
    {
        intentString = bundle.GetString("myString");
        intentBool = bundle.GetBoolean("myBool");
    }

    // Set our view from the "main" layout resource
    SetContentView(Resource.Layout.Main);
}
```

一次 `OnCreate` 已完成后，将调用 Android `OnStart`。

OnStart

`OnStart`始终在调用后系统 `OnCreate` 已完成。如果它们需要活动变得可见例如刷新活动中的视图的当前值之前执行特定任务的任何权利，活动可能会重写此方法。Android 将调用 `OnResume` 紧跟此方法。

OnResume

系统调用 `OnResume` 时该活动已准备好开始与用户进行交互。活动应重写此方法以执行以下任务：

- 帧速率（游戏生成中的常见任务）的提升效应
- 启动动画
- 侦听 GPS 更新
- 显示任何相关的警报或对话框
- 绑定外部事件处理程序

例如，下面的代码段演示如何初始化照相机：

```
public void OnResume()
{
    base.OnResume(); // Always call the superclass first.

    if (_camera==null)
    {
        // Do camera initializations here
    }
}
```

`OnResume` 很重要，因为这是任何操作中完成 `OnPause` 应在未完成 `OnResume`，因为它是唯一的生命周期方法，保证操作之后执行 `OnPause` 时将返回到生命的活动。

OnPause

`OnPause`系统以将活动放到背景或将成为部分遮住活动时调用。在需要时，活动应重写此方法：

- 未保存的更改提交到持久性数据
- 销毁或清理消耗资源的其他对象
- 负载增加帧速率和暂停动画
- 取消注册通知处理程序（即绑定到服务的那些）或外部事件处理程序。这必须进行以防止活动内存泄漏。
- 同样，如果活动具有显示任何对话框或警报，它们必须先清除与 `.Dismiss()` 方法。

例如，下面的代码段即会发布照相机，活动不能进行使用它暂停时：

```

public void OnPause()
{
    base.OnPause(); // Always call the superclass first

    // Release the camera as other activities might need it
    if (_camera != null)
    {
        _camera.Release();
        _camera = null;
    }
}

```

有两种可能的生命周期方法将调用之后的 `OnPause` :

1. `OnResume` 如果活动是要返回到前台, 将会调用。
2. `OnStop` 如果该活动放入后台, 将会调用。

OnStop

`OnStop`不再对用户可见的活动时调用。发生以下情况之一时, 将发生这种情况:

- 新的活动启动, 并介绍了此活动。
- 将现有的活动置于前台。
- 活动已被破坏。

`OnStop` 可能不会始终会调用在内存不足的情况下, 例如 Android 时可用的资源以及不能正确后台活动。出于此原因, 最好是不依赖于 `OnStop` 获取准备销毁活动时调用。下一步后将在可能调用的生命周期方法 `OnDestroy` 如果该活动将消失, 或 `OnRestart` 如果返回传入活动与用户进行交互。

onDestroy

`OnDestroy`是之前它已被销毁并从内存中完全删除活动实例调用的最后一个方法。在极端情况下 Android 会严重影响应用程序过程在承载活动, 这将导致 `OnDestroy` 不调用。大多数活动将不会实现此方法, 因为大多数清理并完成了关机 `OnPause` 和 `OnStop` 方法。 `OnDestroy` 方法通常被重写清理长时间运行的资源可能会泄漏资源。出现这种可能是后台线程中启动 `OnCreate` 。

将名为该活动已被销毁后没有生命周期方法。

OnRestart

`OnRestart`后您的活动已停止, 在它重新启动之前调用。在用户按应用程序中的上一个活动的主页按钮时, 会是很好的例子。当发生这种情况 `OnPause`, 然后 `OnStop` 调用方法, 并将活动移动到后台, 但不是会被销毁。如果用户是使用任务管理器或类似的应用程序还原应用程序, Android 将调用 `OnRestart` 活动的方法。

有哪种逻辑应在实现通用准则 `OnRestart`。这是因为 `OnStart` 总是调用而不考虑是否正在创建活动或正在重新启动, 因此应在初始化活动所需的任何资源 `OnStart`, 而非 `OnRestart`。

下一个生命周期方法之后调用 `OnRestart` 将为 `OnStart`。

备份 vs. 主页

许多 Android 设备都有两个不同的按钮:"上一步"按钮和"主页"按钮。可以为 Android 4.0.3 下面的屏幕截图中看到此示例:



即使它们看上去拥有相同的效果将应用程序放在后台的没有在两个按钮之间略有不同。当用户单击后退按钮时，它们在告诉 Android 与活动完成。Android 将销毁该活动。与此相反，当用户单击主页按钮该活动只是放入后台-Android 将终止该活动。

管理整个生命周期状态

已停止或销毁活动时系统将提供的机会保存更高版本的解除冻结的活动的状态。已保存状态，这称为实例状态。Android 提供了用于在活动生命周期过程中存储实例状态的三个选项：

1. 存储中的基元值 `Dictionary` 称为**捆绑** Android 将用于保存状态。
2. 创建自定义类将保存位图等复杂值。Android 将使用此自定义类来保存状态。
3. 绕过配置更改生命周期中，并假设完成负责维护活动中的状态。

本指南介绍了前两个选项。

捆绑包状态

用于保存实例状态的首要选项是使用键/值字典对象，称为**捆绑**。请记住，创建一个活动时 `onCreate` 方法作为参数传递一个捆绑包，则此捆绑包可用于还原实例状态。不建议用于捆绑包进行更复杂的数据，不会快速或轻松地序列化键/值对（如位图）；相反，它应该用于简单的值，例如字符串。

活动提供方法来帮助进行保存和检索捆绑中的实例状态：

- `onSaveInstanceState` - 这由 Android 调用的活动已被破坏。活动可以实现此方法，如果它们需要保存任何键/值状态项。
- `onRestoreInstanceState` - 之后调用此 `onCreate` 方法完成，并且提供了另一种可能用于初始化完成后恢复其状态的活动。

下图说明了如何使用这些方法：



`onSaveInstanceState`

`onSaveInstanceState`正在停止活动都会调用。它将接收该活动可以存储在其状态的捆绑包参数。活动时设备体验的配置更改，可以使用 `Bundle` 传入的要通过重写保留的活动状态的对象 `onSaveInstanceState`。例如，考虑以下代码：

```

int c;

protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    this.SetContentView (Resource.Layout.SimpleStateView);

    var output = this.FindViewById<TextView> (Resource.Id.outputText);

    if (bundle != null) {
        c = bundle.GetInt ("counter", -1);
    } else {
        c = -1;
    }

    output.Text = c.ToString ();

    var incrementCounter = this.FindViewById<Button> (Resource.Id.incrementCounter);

    incrementCounter.Click += (s,e) => {
        output.Text = (++c).ToString();
    };
}

```

上面的代码增加名为一个整数 `c` 名为的按钮时 `incrementCounter` 单击时，显示在结果 `TextView` 名为 `output`。当配置更改发生-例如，当将设备旋转-上面的代码将丢失的值 `c` 因为 `bundle` 是 `null`，如下图中所示：



若要保留的值 `c` 活动可以在此示例中，重写 `OnSaveInstanceState`，捆绑中保存值，如下所示：

```

protected override void OnSaveInstanceState (Bundle outState)
{
    outState.PutInt ("counter", c);
    base.OnSaveInstanceState (outState);
}

```

现在当将设备与新方向旋转的整数部分将为绑定中保存和检索的行：

```
c = bundle.GetInt ("counter", -1);
```

NOTE

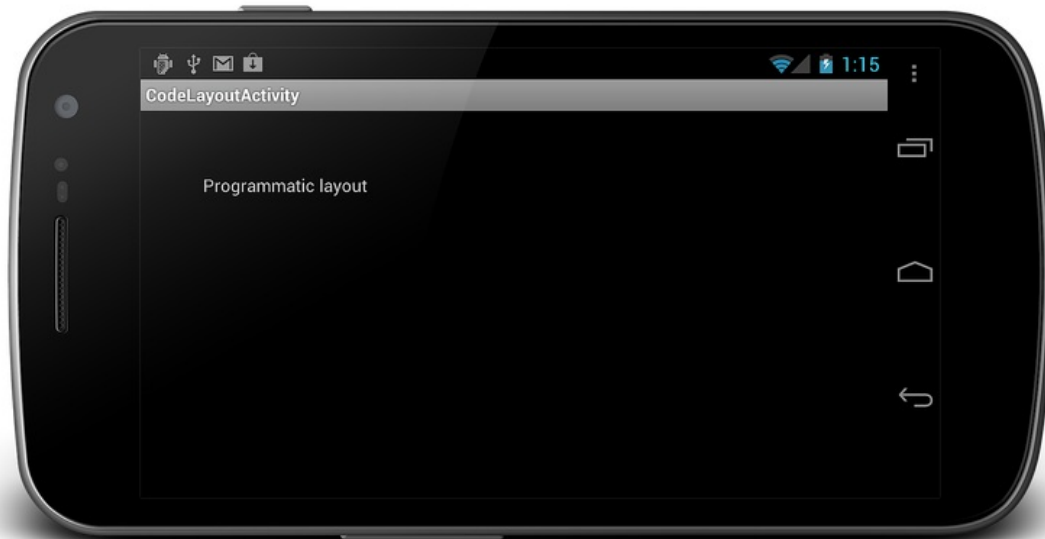
它是重要始终调用基实现 `OnSaveInstanceState` , 以便也可以保存的视图层次结构的状态。

视图状态

重写 `OnSaveInstanceState` 是用于在活动中的暂时性数据保存在方向更改, 如上面的示例中的计数器之间适当机制。但是的默认实现 `OnSaveInstanceState` 将负责的每个视图中, 在 UI 中保存暂时性数据, 只要每个视图都有分配的 ID。例如, 假设应用程序具有 `EditText` 元素在 XML 中定义, 如下所示:

```
<EditText android:id="@+id/myText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

由于 `EditText` 控件具有 `id` 分配, 在用户输入一些数据并旋转设备时, 数据仍显示, 如下所示:



OnRestoreInstanceState

`OnRestoreInstanceState`后将调用 `OnStart` 。它使活动能够还原以前保存在上一个过程绑定到任何状态 `OnSaveInstanceState` 。这是提供给在同一个包 `OnCreate` , 但是。

下面的代码演示如何在还原状态 `OnRestoreInstanceState` :

```
protected override void OnRestoreInstanceState(Bundle savedInstanceState)
{
    base.OnRestoreSaveInstanceState(savedInstanceState);
    var myString = savedInstanceState.GetString("myString");
    var myBool = savedInstanceState.GetBoolean("myBool");
}
```

此方法存在是为了提供一些灵活地时应还原状态。有时是等待, 直到还原实例状态之前完成所有初始化更为合适。此外, 将现有的活动的子类可能只想要从实例状态还原的某些值。在许多情况下, 它不需要重写 `OnRestoreInstanceState` , 因为大多数活动可以还原使用提供给此捆绑包的状态 `OnCreate` 。

有关保存状态使用的示例 `Bundle`，请参阅[演练-保存活动状态](#)。

捆绑包限制

尽管 `OnSaveInstanceState` 可以轻松地保存暂时性数据，它具有一些限制：

- 它不是在所有情况下调用。例如，按主页或回退出活动不会导致 `OnSaveInstanceState` 被调用。
- 捆绑包传递到 `OnSaveInstanceState` 不专为大型对象，如图像。对于大型对象，保存该对象从 `OnRetainNonConfigurationInstance` 非常可取，如下所述。
- 通过使用此捆绑包来保存的数据进行序列化，这可能会导致延迟。

捆绑包状态是适用于不使用太多内存的简单数据，而非配置实例数据很有用的更复杂的数据或检索成本较高的数据，如 web 服务调用或很复杂数据库查询。根据需要在对象中获取保存非配置实例数据。下一部分将介绍 `OnRetainNonConfigurationInstance` 作为保留通过配置更改的更复杂数据类型的一种方法。

保存的复杂数据

捆绑包中保存数据，除了 Android 还支持将数据保存通过重写 `OnRetainNonConfigurationInstance`，并返回的实例 `Java.Lang.Object`，其中包含要保留的数据。有两个主要好处使用 `OnRetainNonConfigurationInstance` 保存状态：

- 从返回的对象 `OnRetainNonConfigurationInstance` 性能也具有更大、更复杂的数据类型，因为内存保留此对象。
- `OnRetainNonConfigurationInstance` 方法是按需、被调用，并且只在需要时。这是比使用手动缓存更经济。

使用 `OnRetainNonConfigurationInstance` 是适用于方案是相当昂贵检索多个时间的数据，如 web 服务调用。有关示例，请搜索 Twitter 的以下代码：

```

public class NonConfigInstanceActivity : ListActivity
{
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);
        SearchTwitter ("xamarin");
    }

    public void SearchTwitter (string text)
    {
        string searchUrl = String.Format("http://search.twitter.com/search.json?" + "q={0}&rpp=10&include_entities=false&" + "result_type=mixed", text);

        var httpReq = (HttpWebRequest)HttpWebRequest.Create (new Uri (searchUrl));
        httpReq.BeginGetResponse (new AsyncCallback (ResponseCallback), httpReq);
    }

    void ResponseCallback (IAsyncResult ar)
    {
        var httpReq = (HttpWebRequest)ar.AsyncState;

        using (var httpRes = (HttpWebResponse)httpReq.EndGetResponse (ar)) {
            ParseResults (httpRes);
        }
    }

    void ParseResults (HttpWebResponse httpRes)
    {
        var s = httpRes.GetResponseStream ();
        var j = (JsonObject)JsonObject.Load (s);

        var results = (from result in (JsonArray)j ["results"] let jResult = result as JsonObject select jResult ["text"].ToString ().ToArray ();

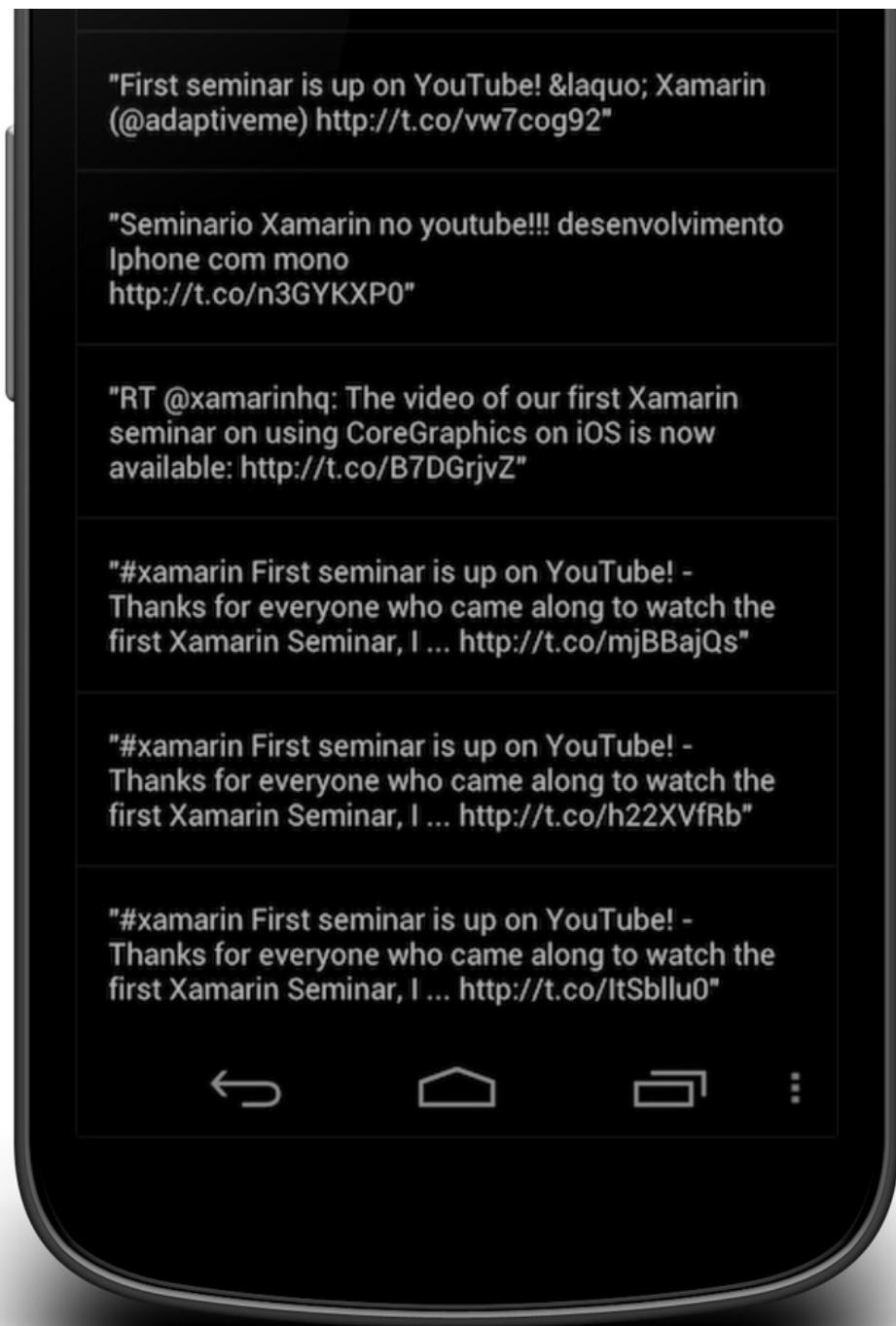
        RunOnUiThread (() => {
            PopulateTweetList (results);
        });
    }

    void PopulateTweetList (string[] results)
    {
        ListAdapter = new ArrayAdapter<string> (this, Resource.Layout.ItemView, results);
    }
}

```

此代码检索从格式化为 JSON web 结果、分析它们，然后在列表中，显示结果，如以下屏幕截图中所示：





配置发生更改时-例如, 在旋转设备-时代码重复该过程。若要重复使用最初检索到的结果并不会导致不必要的冗余网络调用, 我们可以使用 `OnRetainNonconfigurationInstance` 来保存这些结果, 如下所示:

```

public class NonConfigInstanceActivity : ListActivity
{
    TweetListWrapper _savedInstance;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        var tweetsWrapper = LastNonConfigurationInstance as TweetListWrapper;

        if (tweetsWrapper != null) {
            PopulateTweetList (tweetsWrapper.Tweets);
        } else {
            SearchTwitter ("xamarin");
        }

        public override Java.Lang.Object OnRetainNonConfigurationInstance ()
        {
            {
                base.OnRetainNonConfigurationInstance ();
                return _savedInstance;
            }

            ...

            void PopulateTweetList (string[] results)
            {
                ListAdapter = new ArrayAdapter<string> (this, Resource.Layout.ItemView, results);
                _savedInstance = new TweetListWrapper{Tweets=results};
            }
        }
    }
}

```

现在, 当将设备旋转, 从检索原始结果 `LastNonConfiguartionInstance` 属性。在此示例中, 结果组成 `string[]` 包含推文。由于 `OnRetainNonConfigurationInstance` 要求 `Java.Lang.Object` 将返回 `string[]` 子类包装在类中 `Java.Lang.Object`, 如下所示:

```

class TweetListWrapper : Java.Lang.Object
{
    public string[] Tweets { get; set; }
}

```

例如, 尝试使用 `TextView` 返回的对象作为 `OnRetainNonConfigurationInstance` 将导致泄露活动, 如下面的代码所示:

```
TextView _textView;

protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    var tv = LastNonConfigurationInstance as TextViewWrapper;

    if(tv != null) {
        _textView = tv;
        var parent = _textView.Parent as FrameLayout;
        parent.RemoveView(_textView);
    } else {
        _textView = new TextView (this);
        _textView.Text = "This will leak.";
    }

    SetContentView (_textView);
}

public override Java.Lang.Object OnRetainNonConfigurationInstance ()
{
    base.OnRetainNonConfigurationInstance ();
    return _textView;
}
```

在本部分中，我们已了解如何保留与简单的状态数据 `Bundle`，并保留更复杂的数据类型与 `OnRetainNonConfigurationInstance`。

总结

Android 活动生命周期为状态管理的应用程序中的活动提供强大的框架，但它可能难以理解和实现。这一章介绍了活动可能会经历其生存期内，以及与这些状态相关联的生命周期方法期间的不同状态。接下来，指南提供有关应在每个这些方法中执行逻辑的类型。

相关链接

- [处理旋转](#)
- [Android 活动](#)

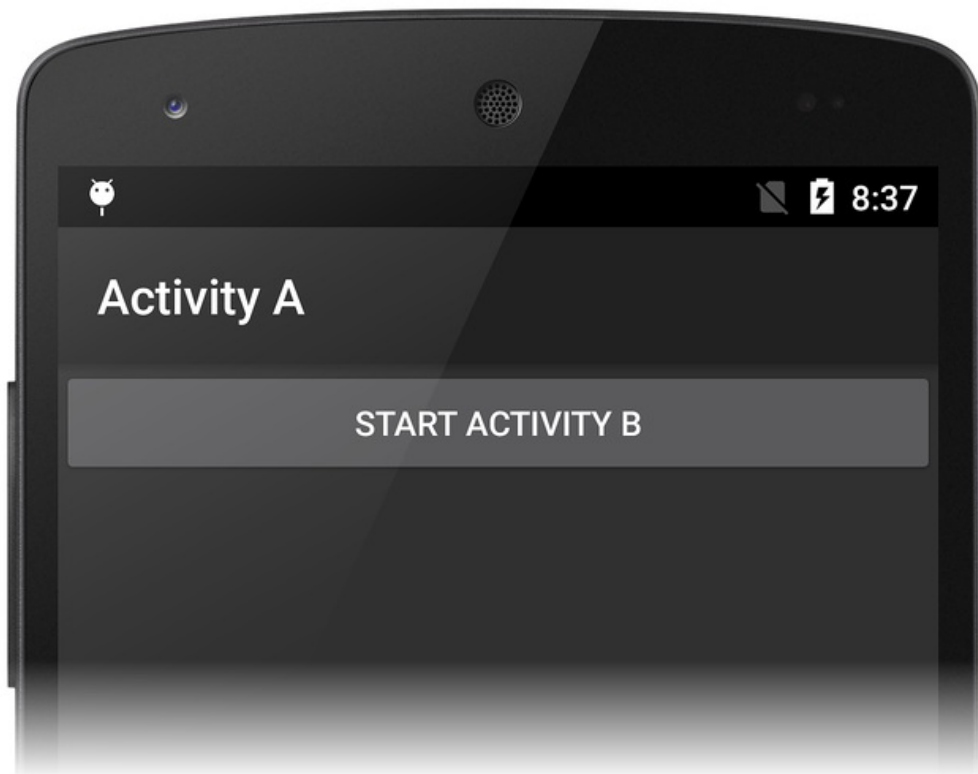
演练-保存活动状态

2018/10/26 • [Edit Online](#)

我们已介绍其理论依据活动生命周期指南, 中保存状态现在, 让我们看一下示例。

活动状态演练

让我们打开 **ActivityLifecycle_Start** 项目 (在 [ActivityLifecycle](#) 示例), 生成它, 并运行它。这是一个非常简单的项目具有两个活动以演示活动生命周期和各种生命周期方法的调用方式。当启动应用程序的屏幕 `MainActivity` 显示:



查看状态转换

在此示例中的每个方法将写入到 IDE 应用程序输出窗口以指示活动状态。(若要在 Visual Studio 中打开输出窗口, 键入 **CTRL ALT O**; 若要将在 Visual Studio for Mac 中打开输出窗口中, 单击视图 > 板 > 应用程序输出。)

当首次启动该应用程序时, 输出窗口会显示的状态更改活动 A:

```
[ActivityLifecycle.MainActivity] Activity A - onCreate
[ActivityLifecycle.MainActivity] Activity A - onStart
[ActivityLifecycle.MainActivity] Activity A - onResume
```

当我们单击 **启动活动 B** 按钮, 我们看到活动 A 暂停和停止时活动 B 经历其状态更改:

```
[ActivityLifecycle.MainActivity] Activity A - onPause
[ActivityLifecycle.SecondActivity] Activity B - onCreate
[ActivityLifecycle.SecondActivity] Activity B - onStart
[ActivityLifecycle.SecondActivity] Activity B - onResume
[ActivityLifecycle.MainActivity] Activity A - onStop
```

因此，*活动 B* 已启动并显示来代替 *活动 A*：



当我们单击回按钮，*活动 B* 销毁并 *活动 A* 恢复：

```
[ActivityLifecycle.SecondActivity] Activity B - onPause
[ActivityLifecycle.MainActivity] Activity A - onRestart
[ActivityLifecycle.MainActivity] Activity A - onStart
[ActivityLifecycle.MainActivity] Activity A - onResume
[ActivityLifecycle.SecondActivity] Activity B - onStop
[ActivityLifecycle.SecondActivity] Activity B - onDestroy
```

添加一个单击计数器

接下来，我们将更改应用程序，它使我们能够计算并显示单击次数的按钮。首先，让我们添加 `_counter` 实例的变量 `MainActivity`：

```
int _counter = 0;
```

接下来，让我们编辑 `Resource/layout/Main.xml` 布局文件，并添加一个新 `clickButton` 显示的用户单击按钮的次数。得到 `Main.xml` 应如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/myButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/mybutton_text" />
    <Button
        android:id="@+id/clickButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/counterbutton_text" />
</LinearLayout>
```

让我们将以下代码添加到末尾`OnCreate`中的方法`MainActivity`—此代码处理单击事件从`clickButton`：

```
var clickbutton = FindViewById<Button> (Resource.Id.clickButton);
clickbutton.Text = Resources.GetString (
    Resource.String.counterbutton_text, _counter);
clickbutton.Click += (object sender, System.EventArgs e) =>
{
    _counter++;
    clickbutton.Text = Resources.GetString (
        Resource.String.counterbutton_text, _counter);
};
```

当我们生成并再次运行该应用，一个新按钮显示的递增，并显示的值`_counter`上每次单击：



但当我们旋转为横向模式的设备，此计数将丢失：



检查应用程序输出，我们看到活动A已暂停、停止、销毁、重新创建、重新启动，然后恢复期间从纵向到横向模式下的旋转：

```
[ActivityLifecycle.MainActivity] Activity A - onPause
[ActivityLifecycle.MainActivity] Activity A - onStop
[ActivityLifecycle.MainActivity] Activity A - onDestroy

[ActivityLifecycle.MainActivity] Activity A - onCreate
[ActivityLifecycle.MainActivity] Activity A - onStart
[ActivityLifecycle.MainActivity] Activity A - onResume
```

因为活动A即被损坏和将设备旋转时，再次重新创建其实例状态将丢失。接下来，我们将添加代码以保存并还原实例状态。

将代码添加到保留实例状态

让我们将方法添加到 `MainActivity` 保存实例状态。之前活动A是销毁，Android 会自动调用 `onSaveInstanceState`，并在将传递捆绑，我们可以使用它来存储我们实例状态。让我们使用它来将我们单击计数另存为一个整数值：

```
protected override void onSaveInstanceState (Bundle outState)
{
    outState.PutInt ("click_count", _counter);
    Log.Debug(GetType().FullName, "Activity A - Saving instance state");

    // always call the base implementation!
    base.OnSaveInstanceState (outState);
}
```

当活动A重新创建和恢复后，Android 将此传递 `Bundle` 返回到我们 `onCreate` 方法。让我们将代码添加到 `onCreate` 还原 `_counter` 从传入的值 `Bundle`。添加以下代码行之前只是其中 `clickbutton` 定义：

```

if (bundle != null)
{
    _counter = bundle.GetInt ("click_count", 0);
    Log.Debug(GetType().FullName, "Activity A - Recovered instance state");
}

```

生成和应用程序再次运行，然后单击第二个按钮几次。当我们旋转为横向模式的设备时，将被保留计数！



让我们看看输出窗口以查看发生了什么情况：

```

[ActivityLifecycle.MainActivity] Activity A - onPause
[ActivityLifecycle.MainActivity] Activity A - Saving instance state
[ActivityLifecycle.MainActivity] Activity A - onStop
[ActivityLifecycle.MainActivity] Activity A - onDestroy

[ActivityLifecycle.MainActivity] Activity A - onCreate
[ActivityLifecycle.MainActivity] Activity A - Recovered instance state
[ActivityLifecycle.MainActivity] Activity A - onStart
[ActivityLifecycle.MainActivity] Activity A - onResume

```

之前[OnStop](#)调用方法，我们的新 `OnSaveInstanceState` 调用方法以保存 `_counter` 中的值 `Bundle`。Android 传递这 `Bundle` 告诉我们调用时我们 `OnCreate` 方法，并且我们能够用它来还原 `_counter` 到我们从离开的位置的值。

总结

在本演练中，我们使用了我们的活动生命周期的知识来保留状态数据。

相关链接

- [ActivityLifecycle \(示例\)](#)
- [活动生命周期](#)
- [Android 活动](#)

创建 Android 服务

2018/11/13 • [Edit Online](#)

本指南介绍 *Xamarin.Android 服务*，是 *Android* 组件，允许以不包含活动的用户界面完成的工作。服务非常通常用于在较长时间计算，下载文件、播放音乐，如在后台中执行任务等。它介绍了服务适合于不同方案，并演示如何实现它们用于执行长时间运行后台任务以及提供接口的远程过程调用。

Android 服务概述

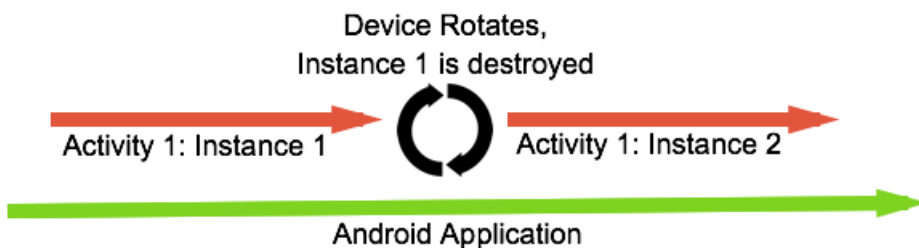
移动应用不像桌面应用程序。桌面有喝许多屏幕空间、内存、存储空间和已连接的电源等资源，移动设备不这样做。这些约束强制执行移动应用不同的行为。例如，在小屏幕上移动设备通常意味着一次只有一个应用（即活动）可见。其他活动都移到后台，推送到挂起状态，它们不能执行任何工作。但是，只是因为 *Android* 应用程序在后台并不意味着就无法应用以继续工作。

Android 应用程序由至少一个以下四个主要组件组成：*活动*，*广播接收器*，*内容提供商*，以及 *_Services_*。活动是许多出色的 *Android* 应用程序的基础，因为它们提供 UI，它允许用户与应用程序进行交互。但是，当涉及到执行并发或后台工作时，活动并不总是最佳选择。

在 *Android* 中的后台工作的主要机制是 *_服务_*。*Android 服务* 是用于执行某项操作而无需用户界面的组件。服务可能会下载文件、播放音乐，或将筛选器应用于映像。服务还可用于进程间通信 (*IPC*) 之间的 *Android* 应用程序。例如，一个 *Android* 应用程序可能会使用从另一个应用的音乐播放器服务或应用程序可能会公开到通过服务的其他应用数据（例如个人的联系信息）。

服务和其执行后台工作的能力至关重要提供平滑流畅的用户界面。所有 *Android* 应用程序具有 *_主线程_*（也称为 *_UI 线程_*）上的活动的运行。若要使设备保持响应状态，*Android* 必须能够更新用户界面的每秒 60 帧速率。如果 *Android* 应用程序主线程上执行过多的工作，则 *Android* 将丢弃帧，这又会导致出现显得很很不稳定的 UI（有时也称为 *_janky_*）。这意味着在 UI 线程上执行任何操作应完成在两个帧，大约 16 毫秒（1，第二，个每隔 60 帧）之间的时间范围。

为了解决此问题，开发人员可能会使用线程在活动中来执行某些工作会阻止 UI。但是，这可能导致问题。它是很有可能，*Android* 将销毁并重新创建活动的多个实例。但是，*Android* 将自动销毁线程，这可能导致内存泄漏。最好的例子是何时 [设备旋转](#) – *Android* 会尝试销毁该活动的实例，然后重新创建一个新：



这是可能的内存泄漏–仍在运行该活动的第一个实例所创建的线程。如果线程可以对活动的第一个实例的引用，这将防止垃圾收集该对象从 *Android*。但是，该活动的第二个实例仍将创建（这又可能会创建一个新线程）。旋转几个场合中快速而连续的设备可能会耗尽所有 RAM 和强制 *Android* 终止整个应用程序以回收内存。

根据经验，如果要执行的工作应生存期长于活动，则服务应创建用于完成该工作。但是，如果工作只是适用于活动的上下文，然后创建线程来执行的工作可能是更合适。例如，创建缩略图照片刚才添加到照片库应用程序应可能会出现服务中。但是，一个线程可能更适合播放活动位于前台时，应仅可以听到一些音乐。

后台工作可以分为两个宽泛的分类：

1. **长时间运行任务** – 这是正在进行，直到显式停止工作。举例 *_长时间运行的任务_* 是流式传输音乐一款应用或，必须监视从传感器收集的数据。即使应用程序没有可视用户界面，必须运行这些任务。

2. **定期任务** – (有时称为_作业_) 定期任务是指属于相对较短的持续时间(几秒钟之内), 按计划运行(即一天一次一周或可能的只是一次在下一步是 60 秒)。出现这种不可从 internet 下载文件, 或生成图像的缩略图。

有四种不同类型的 Android 服务:

- **绑定服务** – A_绑定服务_是具有某些其他组件(通常活动)绑定到它的服务。绑定的服务提供可绑定的组件和服务彼此交互的接口。一旦绑定到的服务没有更多客户端, Android 将关闭该服务。
- `IntentService` – `IntentService` 是一个专门的子类 `Service` 类, 用于简化服务创建和使用情况。`IntentService` 旨在处理各个自治的调用。与一个服务, 它可以同时处理多个调用, 不同 `IntentService` 更像是_工作队列处理器_工作排队等候和 `IntentService` 在单个工作线程上一次处理一个每个作业。通常情况下, `IntentService` 未绑定到的活动或片段。
- **已启动服务** – A_已启动服务_是已启动一些其他 Android 组件(如活动), 直到内容显式告知在后台持续运行的服务若要停止的服务。不同于绑定的服务, 已启动的服务不具有直接绑定到它的任何客户端。出于此原因, 务必设计, 以便它们可以正常重新启动在必要时启动的服务。
- **混合服务** – A_混合服务_是一项服务具有的特征_已启动服务_和一个_绑定服务_。组件绑定到它或它可能由某些事件启动时, 可以通过启动一种混合服务。客户端组件可能会或可能未绑定到混合服务。一种混合服务将继续运行, 直到明确指示若要停止, 或者没有任何绑定到它的多个客户端。

要使用的类型是服务的非常依赖于应用程序的要求。根据经验, `IntentService` 或绑定的服务是足以满足大多数任务都必须执行的 Android 应用程序, 因此首选项应授予给服务这两种类型之一。`IntentService` 是"单稳"任务, 例如下载文件, 而需要与活动/片段的频繁交互时, 绑定的服务也可能适用的一个不错选择。

尽管大多数服务在后台运行, 还有一个特殊的子类别, 称为_前景服务_。这是一种服务, 提供更高的优先级(与普通服务相比)来为用户(如播放音乐)执行某些任务。

也可以在同一设备上其自己的进程中运行服务, 这有时称为_远程服务_或是_进程外服务_。这需要更多工作来创建, 但也可用于应用程序需要共享功能与其他应用程序, 并可以在某些情况下, 提高应用程序的用户体验。

Android 8.0 中的后台执行限制

启动 Android 8.0 (API 级别为 26) 在 Android 应用程序不再能够自由地在后台运行。当在前台, 应用可以启动和运行不受限制的服务。当应用程序移动到后台时, Android 将授予该应用程序一定的时间启动和使用服务。后经过指定的时间, 则应用无法再启动任何服务和启动的任何服务将被终止。此时不能执行任何工作的应用。Android 要考虑的应用程序位于前台, 如果满足以下条件之一:

- 没有可见的活动(启动或暂停)。
- 应用程序已开始的前景色服务。
- 另一个应用程序位于前台, 并使用从应用程序都将成为在后台的组件。此示例是如果应用程序 A, 位于前台, 绑定到所提供的服务应用程序 b。应用程序 B 然后也将是被视为在前台, 但在后台正在终止 android。

有某些情况下, 其中, 即使应用是在后台, Android 将唤醒应用和放宽这些限制等几分钟, 可通过应用程序来执行某些工作:

- 高优先级应用接收 Firebase 云消息。
- 应用程序将收到一个广播。
- 应用程序收到执行 `PendingIntent` 以响应一条通知。

现有 Xamarin.Android 应用程序可能需要更改其执行后台工作以避免可能出现在 Android 8.0 上的任何问题的方式。下面是一些实用的替代方案到 Android 服务:

- 计划在使用 **Android 作业计划程序**在后台中运行的工作或**Firebase 作业调度程序** –这两个库提供一个框架, 用于应用程序分离到中的后台工作_作业_, 离散的工作单位。可以在运行作业时, 应用程序然后可以有计划操作系统以及某些条件的作业。
- **启动该服务在前台**–前景服务可用于应用程序时必须执行某些任务在后台, 用户可能需要定期与该任务进行交互。前景服务将显示持久通知, 以便用户可以知道应用正在运行的后台任务, 且还提供了一种方法来监视或与

任务交互。此示例将向用户播放播客或可能下载播客一集中，以便可以更高版本感到满意的主发挥播客应用。

- 使用高优先级 **Firebase 云消息 (FCM)** – 时 Android 接收应用程序的高优先级 FCM，它将允许该应用将在后台服务运行的时间短时间。这将是具有后台服务，可以轮询在后台中的应用的良好替代方法。
- 推迟工作的应用程序时进入前台 – 如果先前的解决方案不可行的则应用必须开发其自己的方法来暂停和继续工作时应用程序转入前台。

相关链接

- [Android Oreo 后台执行限制](#)

创建服务

2018/10/26 • [Edit Online](#)

Xamarin.Android 服务必须遵循 Android 服务的两个侵犯的隔离规则：

- 它们必须扩展 `Android.App.Service` 。
- 它们必须使用修饰 `Android.App.ServiceAttribute` 。

Android 服务的另一个要求是必须注册它们 **AndroidManifest.xml** 和给定的唯一名称。Xamarin.Android 将自动注册该服务清单中在生成时使用必需的 XML 特性。

此代码段是在 Xamarin.Android 中满足这两个要求创建服务的最简单的示例：

```
[Service]
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things.
}
```

在编译时，Xamarin.Android 将注册该服务的注入到下面的 XML 元素 **AndroidManifest.xml**（注意，Xamarin.Android 生成服务的随机名称）：

```
<service android:name="md5a0cbbf8da641ae5a4c781aaf35e00a86.DemoService" />
```

可以通过其他 Android 应用程序与共享服务_导出_它。这可以通过设置 `Exported` 属性上的 `ServiceAttribute`。一种服务，在导出时 `ServiceAttribute.Name` 还应设置属性以提供有意义的公共名称的服务。此代码段演示了如何导出和命名服务：

```
[Service(Exported=true, Name="com.xamarin.example.DemoService")]
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things.
}
```

AndroidManifest.xml 此服务的元素然后如下所示：

```
<service android:exported="true" android:name="com.xamarin.example.DemoService" />
```

服务具有其自己使用创建服务时调用的回调方法的生命周期。完全调用哪个方法取决于服务的类型。已启动的服务必须实现不同的生命周期方法比绑定的服务，而混合服务必须实现已启动的服务和绑定的服务的回调方法。这些方法包括的所有成员 `Service` 类；该服务已启动将确定哪些生命周期方法将在调用。将稍后更详细地讨论这些生命周期方法。

默认情况下，服务将作为 Android 应用程序在同一进程中启动。可以通过设置其自身进程中启动服务

`ServiceAttribute.IsolatedProcess` 属性设为 true：

```
[Service(IsolatedProcess=true)]
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things, in it's own process!
}
```

下一步是了解如何启动服务，然后继续研究如何实现三个不同类型的服务。

NOTE

服务在运行在 UI 线程，因此，如果任何工作是为执行可阻止将 UI，该服务必须使用线程来执行的工作。

启动服务

若要在 Android 中启动服务的最基本方法是调度 `Intent` 其中包含元数据来帮助确定哪个服务应已启动。有两种不同样式的意向，可用于启动服务：

- **目的在于明确指示** – `Intent` 将确定完全什么服务应该用于完成给定的操作。目的在于明确指示可以看作具有特定的地址；以字母 Android 将路由到显式标识的服务的目的。此代码段是使用显式意向来启动调用服务的一个示例 `DownloadService`：

```
// Example of creating an explicit Intent in an Android Activity
Intent downloadIntent = new Intent(this, typeof(DownloadService));
downloadIntent.data = Uri.Parse(fileToDownload);
```

- **隐式 Intent** – 这种类型的意向松散标识操作，用户想要执行，但若要完成该操作的确切服务是未知。隐式 Intent 可以被视为是以字母解决“To Whom It 可能需考虑...”。Android 将检查的目的，内容并确定是否符合要求的现有服务。

`IntentFilter` 用来帮助查找与已注册服务的隐式目的。意向筛选器是添加到一个 XML 元素 **AndroidManifest.xml** 其中包含必需元数据来帮助查找具有隐式 intent 的服务。

```
Intent sendIntent = new Intent("common.xamarin.DemoService");
sendIntent.Data = Uri.Parse(fileToDownload);
```

如果 Android 具有隐式 intent 的多个可能的匹配项，它可能会要求用户选择要处理操作的组件：

Open with



Recipe App



Chrome

JUST ONCE

ALWAYS

IMPORTANT

从 Android 5.0 (AP 级别 21) 开始隐式 intent 不能用于启动服务。

在可能的情况下, 应用程序应使用显式意向来启动服务。隐式 Intent 不要求提供要启动的特定服务—它是在设备上安装某些服务的请求来处理该请求。此二义性请求可能导致错误的服务处理请求或不必要地启动 (这会增加设备上的资源的压力) 的另一个应用。

如何调度目的取决于服务的类型, 并将更高版本中为每种类型的服务的特定指南的更详细地讨论。

为隐式意向创建意向筛选器

若要将服务与隐式 Intent 相关联, Android 应用程序必须提供某些元数据来标识该服务的功能。此元数据由提供 `IntentFilter`。意向筛选器包含一些信息, 例如操作或意向来启动服务中必须存在的数据类型。在 Xamarin.Android 中, 意向筛选器中注册 **AndroidManifest.xml** 修饰具有的服务 `IntentFilterAttribute`。例如, 下面的代码添加相应的操作的意向筛选器 `com.xamarin.DemoService` :

```
[Service]
[IntentFilter(new String[]{"com.xamarin.DemoService"})]
public class DemoService : Service
{
}
```

这会导致中包含的条目 **AndroidManifest.xml** 文件—打包在一起的方式类似于下面的示例应用程序的条目:

```
<service android:name="demoservice.DemoService">
  <intent-filter>
    <action android:name="com.xamarin.DemoService" />
  </intent-filter>
</service>
```

使用 Xamarin.Android 服务开的基础知识, 让我们检查更多详细信息中的服务不同子的类型。

相关链接

- [Android.App.Service](#)
- [Android.App.ServiceAttribute](#)
- [Android.App.Intent](#)
- [Android.App.IntentFilterAttribute](#)

在 Xamarin.Android 中绑定服务

2018/11/13 • [Edit Online](#)

绑定的服务是 Android 提供客户端（例如 Android 活动）可与之交互的客户端-服务器接口的服务。本指南介绍了与创建绑定的服务以及如何在 Xamarin.Android 应用程序中使用它所涉及的关键组件。

绑定服务概述

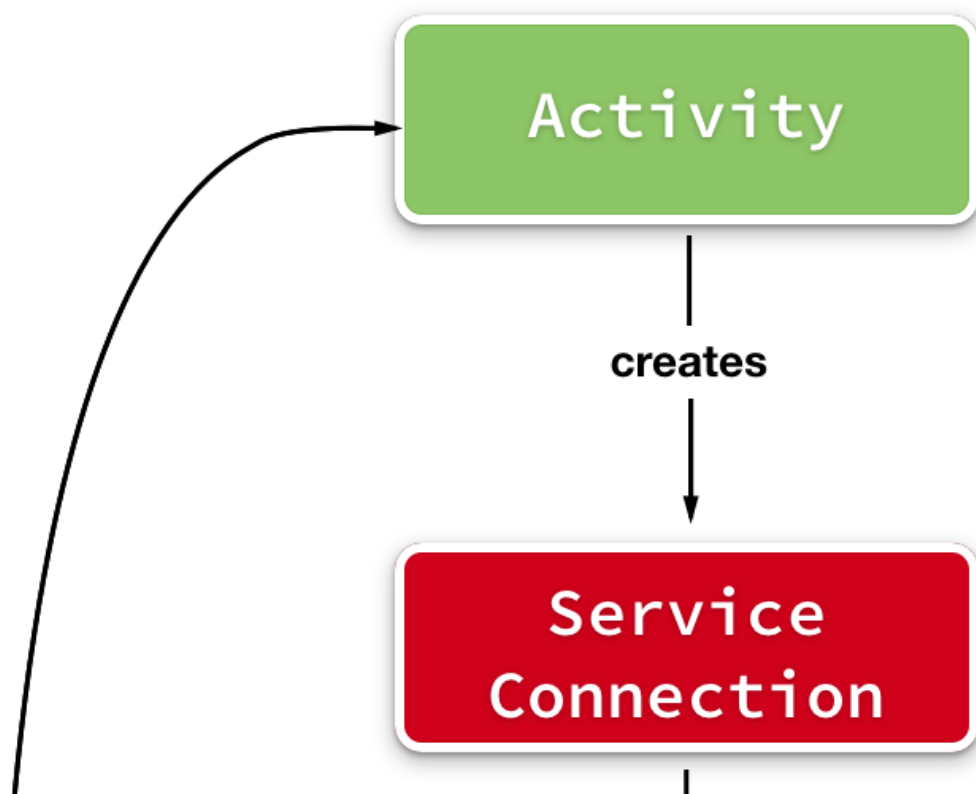
向客户端进行直接交互的客户端-服务器接口提供服务的服务被称为_绑定服务_。可以有多个客户端同时连接到服务的单一实例。绑定的服务和客户端是相互隔离的。相反，Android 提供了一系列管理这两个之间的连接状态的中间对象。此状态由对象实现 `Android.Content.IServiceConnection` 接口。客户端创建此对象并将其作为参数传递给 `BindService` 方法。`BindService` 是出现在任何 `Android.Content.Context` 对象（如活动）。它是对 Android 操作系统启动该服务并绑定到该客户端的请求。有三种方式向客户端可能会将绑定到服务使用 `BindService` 方法：

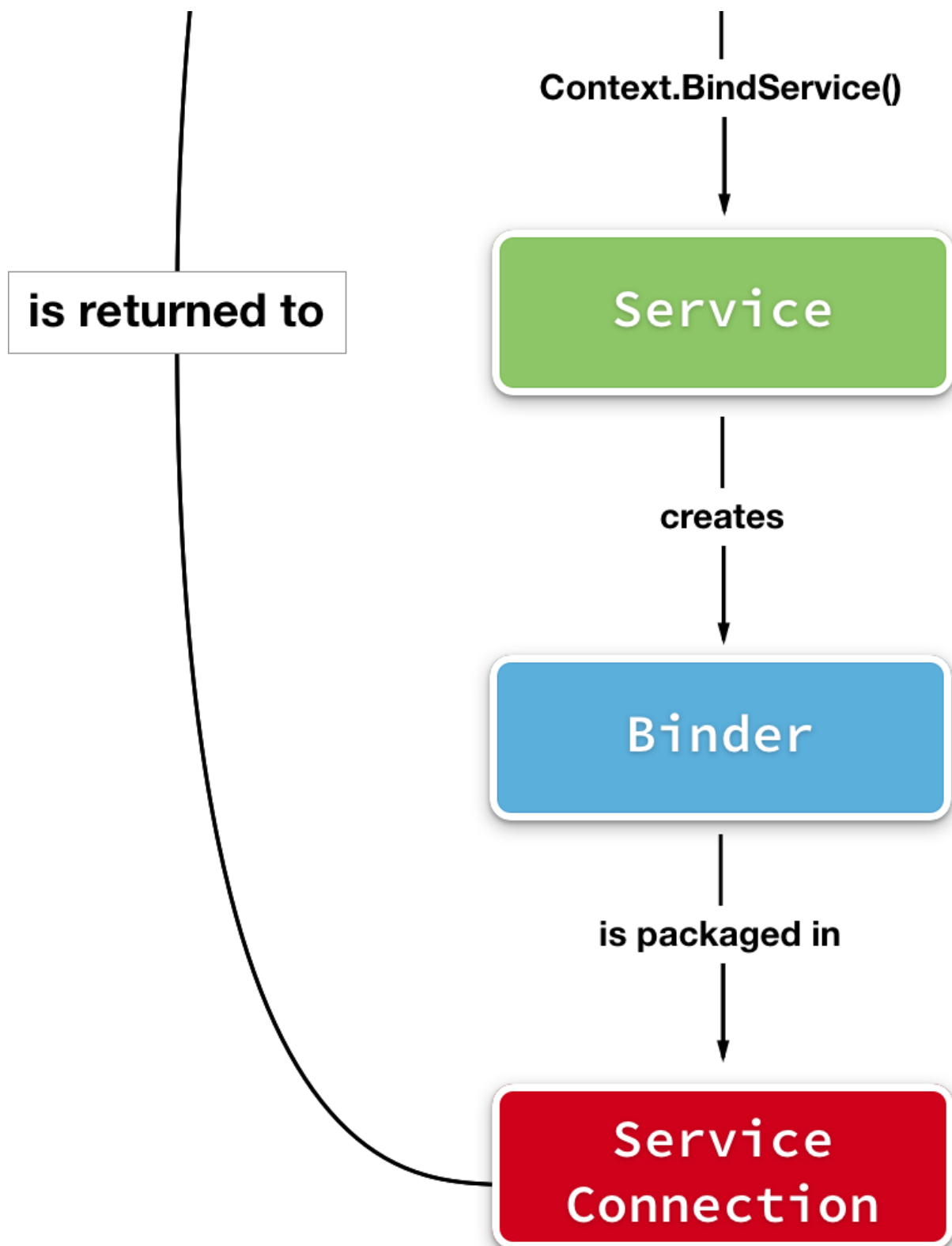
- **服务活页夹-服务联编程序**是一个类，实现 `Android.OS.IBinder` 接口。大多数应用程序将不会直接实现此接口，它们将改为扩展 `Android.OS.Binder` 类。这是最常用的方法，适用于在同一进程中时存在服务和客户端。
- **使用 Messenger**—这种技术适合时该服务可能存在于单独的进程中。相反，服务请求客户端和服务通过之间封送 `Android.OS.Messenger`。`Android.OS.Handler` 将要用于处理在服务中创建 `Messenger` 请求。另一指南中包含此文件。
- **使用 Android 接口定义语言 (AIDL)**—AIDL 是一种高级的技术，将此指南中未提及。

一旦客户端具有已绑定到的服务，二者之间的通信是通过时 `Android.OS.IBinder` 对象。此对象负责将允许客户端与服务进行交互的接口。没有必要为每个 Xamarin.Android 应用程序以实现此接口从零开始，Android SDK 提供 `Android.OS.Binder` 负责大部分所需的对象之间的封送处理代码的类客户端和服务。

完成客户端与服务操作后，它必须通过调用从它取消绑定 `UnbindService` 方法。一旦从一种服务，最后一个客户端具有未绑定，Android 将停止并释放绑定服务。

此图描述了如何将活动、服务连接、联编程序和服务所有彼此相关的：





本指南介绍了如何扩展 `Service` 类，以实现绑定的服务。它还将介绍实现 `IServiceConnection` 和扩展 `Binder` 以允许客户端与服务进行通信。示例应用程序附带本指南中，其中包含一个名为 `Xamarin.Android` 项目使用的一种解决方案**`BoundServiceDemo`**。这是非常基本的应用程序，该示例演示了如何实现的服务以及如何将活动绑定到它。绑定的服务具有只有一个方法，使用非常简单的 API `GetFormattedTimestamp`，这会返回一个字符串，告知用户已启动该服务时，多长时间运行。应用程序还允许用户手动解除绑定，并将绑定到该服务。



Bound Services Demo

This app is a demonstration of a bound service in Xamarin.Android. This example will bind to a service, which will return a timestamp telling how long the service has been running.

GET TIMESTAMP FROM SERVICE

UNBIND FROM SERVICE

BIND TO SERVICE

Service started at 12/9/2016 4:57:17 PM (00:00:08.7910440 ago).

实现和使用绑定的服务

有三个组件必须实现中使用的绑定的服务的 Android 应用程序的顺序：

1. 扩展 `Service` 类并实现生命周期回调方法-此类将包含将执行将服务的请求的工作的代码。下面更详细地包含此文件。
2. 创建一个类实现 `IServiceConnection` -回叫方法将调用此接口提供通过 Android 通知客户端与服务连接发生更改时，即客户端已连接或断开连接到服务。服务连接还会提供对客户端可用来直接与服务交互的对象的引用。此引用名为 `_联编程序_`。
3. 创建一个类实现 `IBinder` - `A_联编程序_`实现提供的 API 的客户端用于与服务通信。此联编程序可以提供对绑定的服务中，引用允许直接调用方法或联编程序可以提供一个客户端 API 的封装并隐藏该应用程序中的绑定的服务。`IBinder` 必须为远程过程调用提供所需的代码。不需要（或不建议）若要实现 `IBinder` 直接接口。应用程序应改为扩展 `Binder` 提供了大部分所需的基本功能的类型 `IBinder`。
4. 启动并绑定到服务 - Android 应用程序的服务连接，联编程序和服务创建后是负责启动服务并将绑定到它。

将以下各节更详细地讨论每个步骤。

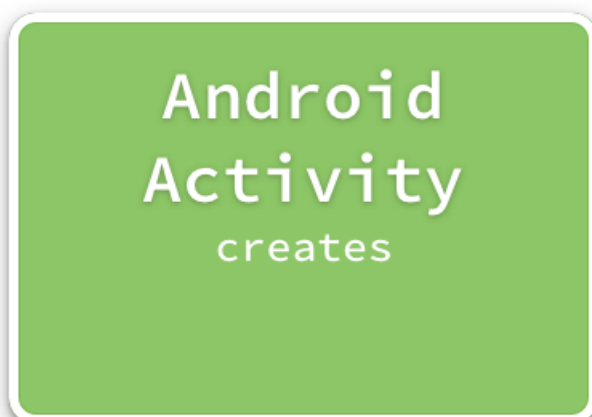
扩展 `Service` 类

若要创建使用 Xamarin.Android 的服务，有必要为子类 `Service` 并进行修饰的类 `ServiceAttribute`。

Xamarin.Android 生成工具使用该属性来正确注册该服务在应用中 `AndroidManifest.xml` 文件非常类似活动中，绑定的服务具有其自己的生命周期和回叫方法与相关联在其生命周期中的重要事件。以下列表是某些较常用服务将实现的回调方法的示例：

- `OnCreate` - 通过 Android 调用此方法，因为它实例化服务。它用于初始化的任何变量或对象的生存期内所需的服务。此方法是可选的。
- `OnBind` - 所有绑定的服务必须实现此方法。第一个客户端尝试连接到服务时，调用此操作。它将返回的实例 `IBinder`，以便客户端可能会与服务交互。只要服务正在运行，`IBinder` 对象将用于满足绑定到服务的任何将来的客户端请求。
- `OnUnbind` - 具有未绑定的绑定的所有客户端时，调用此方法。通过返回 `true` 该服务将更高版本调用此方法中，从 `OnRebind` 传递到意向 `OnUnbind` 时将新的客户端绑定到它。服务继续运行之后未绑定时将执行此操作。如果最近未绑定的服务也已启动的服务，会发生此情况并 `StopService` 或 `StopSelf` 尚未调用。在此类方案中，`OnRebind` 允许检索的意图。默认值返回 `false`，这不执行任何操作。可选。
- `OnDestroy` - Android 销毁该服务时，调用此方法。在这种方法，应执行任何必要的清理，释放资源，例如。可选。

绑定服务的密钥生命周期事件显示在此图中：



BindService()



**Service.
onCreate()**



First client r



**Service.
onBind()**



**Service is
Running**



**Client is
unexpected
UnbindService()**

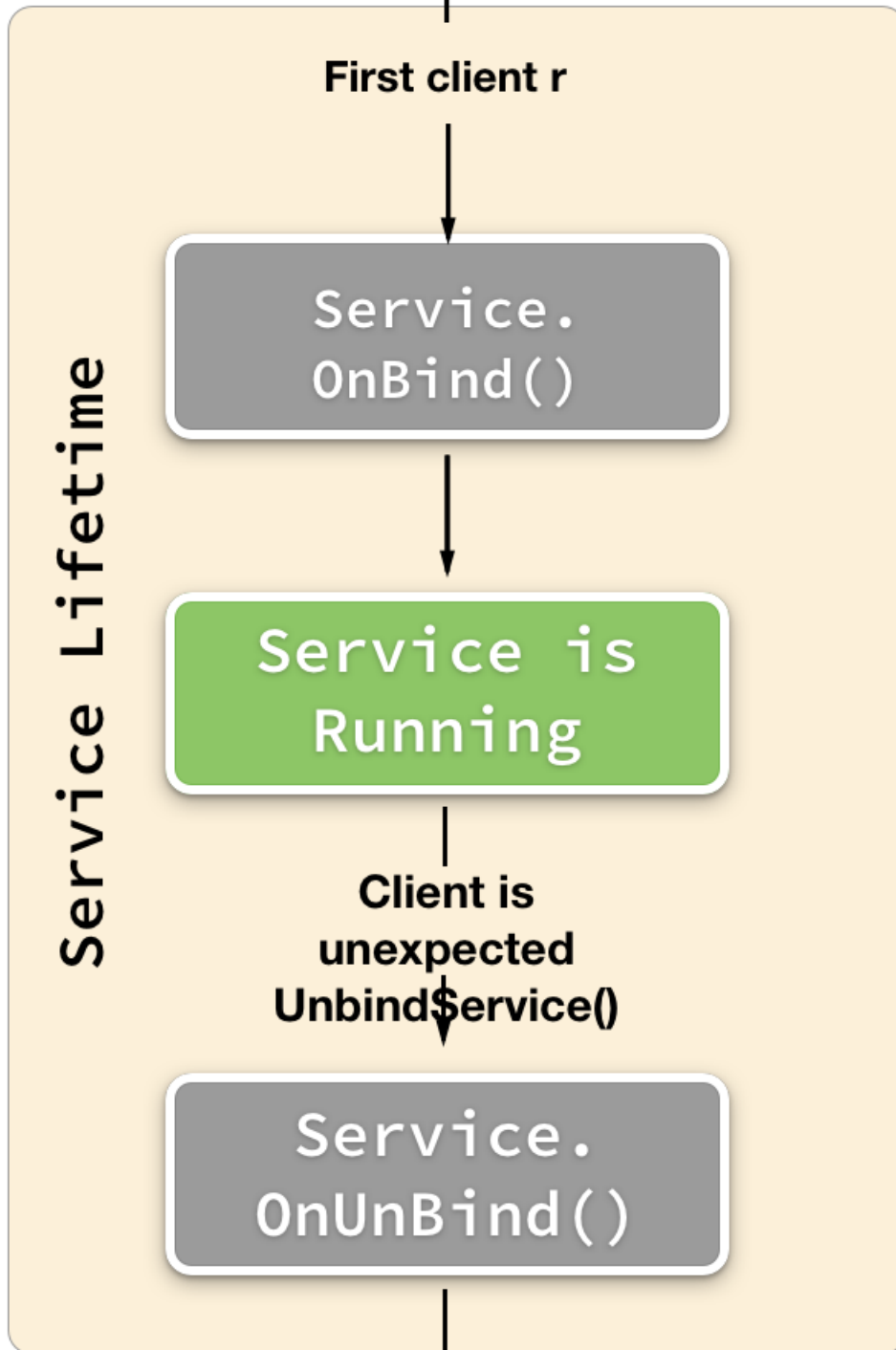


**Service.
onUnBind()**



**Last Client
unbinds from**

Service Lifetime



Service



**Service.
Destroy()**

附带本指南的配套应用程序中的以下的代码段显示了如何在 Xamarin.Android 中实现的绑定的服务：

```

using Android.App;
using Android.Util;
using Android.Content;
using Android.OS;

namespace BoundServiceDemo
{
    [Service(Name="com.xamarin.ServicesDemo1")]
    public class TimestampService : Service, IGetTimestamp
    {
        static readonly string TAG = typeof(TimestampService).FullName;
        IGetTimestamp timestampper;

        public IBinder Binder { get; private set; }

        public override void OnCreate()
        {
            // This method is optional to implement
            base.OnCreate();
            Log.Debug(TAG, "OnCreate");
            timestampper = new UtcTimestampper();
        }

        public override IBinder OnBind(Intent intent)
        {
            // This method must always be implemented
            Log.Debug(TAG, "OnBind");
            this.Binder = new TimestampBinder(this);
            return this.Binder;
        }

        public override bool OnUnbind(Intent intent)
        {
            // This method is optional to implement
            Log.Debug(TAG, "OnUnbind");
            return base.OnUnbind(intent);
        }

        public override void OnDestroy()
        {
            // This method is optional to implement
            Log.Debug(TAG, "OnDestroy");
            Binder = null;
            timestampper = null;
            base.OnDestroy();
        }

        /// <summary>
        /// This method will return a formatted timestamp to the client.
        /// </summary>
        /// <returns>A string that details what time the service started and how long it has been running.
        </returns>
        public string GetFormattedTimestamp()
        {
            return timestampper?.GetFormattedTimestamp();
        }
    }
}

```

在示例中，`OnCreate` 方法初始化一个对象，保存用于检索和格式设置客户端将请求的时间戳的逻辑。当第一个客户端尝试绑定到服务时，将调用 Android `OnBind` 方法。此服务将实例化 `TimestampBinder` 将允许客户端访问正在运行的服务的此实例的对象。`TimestampBinder` 类下一节中所述。

实现 IBinder

如前文所述，`IBinder` 对象提供客户端和服务之间的通信通道。Android 应用程序不应实现 `IBinder` 接口，

`Android.OS.Binder` 应进行扩展。`Binder` 类提供了许多必要的基础结构这是必要的封送到客户端从服务（这可能会在单独的进程中运行）联编程序对象。在大多数情况下，`Binder` 子类是只有少量的代码行，并将包装对服务的引用。在此示例中，`TimestampBinder` 具有一个属性，它公开 `TimestampService` 到客户端：

```
public class TimestampBinder : Binder
{
    public TimestampBinder(TimestampService service)
    {
        this.Service = service;
    }

    public TimestampService Service { get; private set; }
}
```

这 `Binder` 就可以调用该服务; 上的公共方法为例：

```
string currentTimestamp = serviceConnection.Binder.Service.GetFormattedTimestamp()
```

一次 `Binder` 已被扩展，它是实现所需 `IServiceConnection` 将所有内容连接在一起。

创建服务连接

`IServiceConnection` 将显示 | 引入 | 公开 | 连接 `Binder` 到客户端对象。除了实现 `IServiceConnection` 接口，类必须扩展 `Java.Lang.Object`。服务连接还应提供客户端可以访问某些方式 `Binder`（并因此与其绑定的服务）。

此代码摘自随附的示例项目是一种实现 `IServiceConnection`：

```

using Android.Util;
using Android.OS;
using Android.Content;

namespace BoundServiceDemo
{
    public class TimestampServiceConnection : Java.Lang.Object, IServiceConnection, IGetTimestamp
    {
        static readonly string TAG = typeof(TimestampServiceConnection).FullName;

        MainActivity mainActivity;
        public TimestampServiceConnection(MainActivity activity)
        {
            IsConnected = false;
            Binder = null;
            mainActivity = activity;
        }

        public bool IsConnected { get; private set; }
        public TimestampBinder Binder { get; private set; }

        public void OnServiceConnected(ComponentName name, IBinder service)
        {
            Binder = service as TimestampBinder;
            IsConnected = this.Binder != null;

            string message = "onServiceConnected - ";
            Log.Debug(TAG, $"OnServiceConnected {name.ClassName}");

            if (IsConnected)
            {
                message = message + " bound to service " + name.ClassName;
                mainActivity.UpdateUiForBoundService();
            }
            else
            {
                message = message + " not bound to service " + name.ClassName;
                mainActivity.UpdateUiForUnboundService();
            }

            Log.Info(TAG, message);
            mainActivity.timestampMessageTextView.Text = message;
        }

        public void OnServiceDisconnected(ComponentName name)
        {
            Log.Debug(TAG, $"OnServiceDisconnected {name.ClassName}");
            IsConnected = false;
            Binder = null;
            mainActivity.UpdateUiForUnboundService();
        }

        public string GetFormattedTimestamp()
        {
            if (!IsConnected)
            {
                return null;
            }

            return Binder?.GetFormattedTimestamp();
        }
    }
}

```

在绑定过程的一部分，将调用 Android `OnServiceConnected` 方法，提供 `name` 要绑定的服务和 `binder`，保存对服务本身的引用。在此示例中，服务连接具有两个属性，如果客户端连接到服务，或不保存对此联编程序和为一个布尔型标志的引用的其中一个。

`OnServiceDisconnected` 客户端和服务之间的连接意外丢失或损坏时，才会调用方法。此方法允许客户端有机会响应对的服务中断。

启动并绑定到具有目的在于明确指示服务

若要使用的绑定的服务，客户端（例如活动）必须实例化实现的对象 `Android.Content.IServiceConnection` 并调用 `BindService` 方法。`BindService` 将返回 `true` 如果该服务所绑定到 `false` 如果不是。`BindService` 方法采用三个参数：

- `Intent` -意图应显式标识要连接到的服务。
- `IServiceConnection` 对象-此对象是提供回调方法以在绑定的服务启动和停止时通知客户端的中介。
- `Android.Content.Bind` 枚举-此参数是一组标志用于系统时，用于绑定对象。最常使用的值是 `Bind.AutoCreate`，这将自动启动服务，如果它尚未运行。

以下代码片段示范了如何在使用目的在于明确指示活动中启动绑定的服务：

```
protected override void OnStart ()
{
    base.OnStart ();

    if (serviceConnection == null)
    {
        this.serviceConnection = new TimestampServiceConnection(this);
    }

    Intent serviceToStart = new Intent(this, typeof(TimestampService));
    BindService(serviceToStart, this.serviceConnection, Bind.AutoCreate);
}
```

IMPORTANT

启动 Android 5.0 (API 级别 21) 它才可以绑定到的目的在于明确指示服务。

有关服务连接和联编程序的体系结构说明。

某些 OOP 纯粹主义者可能会否决的以前的实现 `TimestampBinder` 类，如不符合 [Demeter 定律](#) 其中，最简单窗体中指出“不与陌生人;仅与您的朋友”。此特定的实现公开具体 `TimestampService` 给所有客户端类。

严格地说，不需要客户端了解的有关 `TimestampService` 和公开给客户端的具体类可能会导致应用程序更加脆弱且难以维护它的生存期内。另一种方法是使用接口公开 `GetFormattedTimestamp()` 方法，并对通过该服务的代理调用 `Binder` (或可能的服务连接类)：

```
public class TimestampBinder : Binder, IGetTimestamp
{
    TimestampService service;
    public TimestampBinder(TimestampService service)
    {
        this.service = service;
    }

    public string GetFormattedTimestamp()
    {
        return service?.GetFormattedTimestamp();
    }
}
```

此特定示例允许用于在服务本身上调用方法的活动：

```
// In this example the Activity is only talking to a friend, i.e. the IGetTimestamp interface provided by the
Binder.
string currentTimestamp = serviceConnection.Binder.GetFormattedTimestamp()
```

相关链接

- [Android.App.Service](#)
- [Android.Content.Bind](#)
- [Android.Content.Context](#)
- [Android.Content.IServiceConnection](#)
- [Android.OS.Binder](#)
- [Android.OS.IBinder](#)
- [BoundServiceDemo \(示例\)](#)

在 Xamarin.Android 中的意向服务

2018/10/26 • [Edit Online](#)

意向服务概述

两者都已启动并绑定意味着，若要使性能保持平滑，服务需要异步执行工作的主线程上运行的服务。若要解决此问题的最简单方式之一是使用_辅助队列处理器模式_、完成的工作由单个线程提供服务的队列中的放置位置。

`IntentService` 是一个的子类 `Service` 提供这种模式的特定 Android 实现的类。将要管理的工作排入队列，正在启动工作线程来服务队列，并提取请求关闭要在辅助线程上运行的队列。`IntentService` 会默默地自行停止并删除工作线程队列中的没有更多工作时。

通过创建将工作提交到队列 `Intent`，然后将其传递 `Intent` 到 `StartService` 方法。

不能停止或中断 `OnHandleIntent` 方法 `IntentService` 而正在处理。此设计中，由于 `IntentService` 应保持无状态-它不应依赖于活动的连接或应用程序的其余部分通信。`IntentService` 旨在 statelessly 处理工作请求。

有两个要求子类化 `IntentService`：

1. 新类型(通过子类化创建 `IntentService`) 仅重写 `OnHandleIntent` 方法。
2. 新类型的构造函数需要一个字符串，它用于命名将处理请求的工作线程。调试应用程序时，主要使用此工作线程的名称。

下面的代码演示 `IntentService` 实现，使用重写 `OnHandleIntent` 方法：

```
[Service]
public class DemoIntentService: IntentService
{
    public DemoIntentService () : base("DemoIntentService")
    {
    }

    protected override void OnHandleIntent (Android.Content.Intent intent)
    {
        Console.WriteLine ("perform some long running work");
        ...
        Console.WriteLine ("work complete");
    }
}
```

将任务发送至 `IntentService` 通过实例化 `Intent`，然后再调用 `StartService` 作为参数的意向的方法。其目的将作为参数传递给服务 `OnHandleIntent` 方法。此代码片段是发送到意向的工作请求的示例：

```
// This code might be called from within an Activity, for example in an event
// handler for a button click.
Intent downloadIntent = new Intent(this, typeof(DemoIntentService));

// This is just one example of passing some values to an IntentService via the Intent:
downloadIntent.Put
("file_to_download", "http://www.somewhere.com/file/to/download.zip");

StartService(downloadIntent);
```

`IntentService` 可以提取值从意图，此代码片段中所示：


```
protected override void OnHandleIntent (Android.Content.Intent intent)
{
    string fileToDownload = intent.GetStringExtra("file_to_download");

    Log.Debug("DemoIntentService", $"File to download: {fileToDownload}.");
}
```

相关链接

- [IntentService](#)
- [StartService](#)

使用 Xamarin.Android 启动的服务

2018/10/26 • [Edit Online](#)

启动的服务概述

启动的服务通常执行工作的单元，而无需向客户端提供的任何直接反馈或结果。一个工作单元的示例是一种服务，将文件上载到服务器。客户端将请求服务上传从设备到网站文件。该服务将安静地上载文件（即使应用程序在前台中有任何活动），并上传完成时终止自身。请务必意识到已启动的服务将在应用程序的 UI 线程上运行。这意味着如果一项服务将执行会阻塞 UI 线程的工作，它必须创建并根据需要释放的线程。

不同于绑定的服务，没有“纯”已启动的服务及其客户端之间的通信渠道。这意味着已启动的服务将执行绑定的服务比一些不同的生命周期方法。下表列出了已启动的服务中常见的生命周期方法：

- `OnCreate` – 调用一次当首次启动该服务。这是应实现初始化代码的位置。
- `OnBind` – 必须由所有服务类，实现此方法，但已启动的服务通常无需绑定到它的客户端。因此，已启动的服务只返回 `null`。一种混合服务（这是绑定的服务和已启动的服务的组合）与此相反，必须实现并返回 `Binder` 客户端。
- `OnStartCommand` – 为每个请求以启动服务，以响应对的调用的任何一个调用 `StartService` 或由系统重新启动。这是该服务可以开始任何长时间运行的任务的位置。该方法将返回 `StartCommandResult` 值，该值指示如何或如果系统应处理由于内存不足导致关闭后重新启动服务。此调用会在主线程上的发生。下面更详细地介绍了这些方法。
- `OnDestroy` – 当正在销毁该服务时，调用此方法。它用于执行任何最终清理所需。

已启动的服务的重要方法是 `OnStartCommand` 方法。它将调用每个时间服务收到的请求执行一些操作。以下代码片段示范了 `OnStartCommand`：

```
public override StartCommandResult OnStartCommand (Android.Content.Intent intent, StartCommandFlags flags, int
startId)
{
    // This method executes on the main thread of the application.
    Log.Debug ("DemoService", "DemoService started");
    ...
    return StartCommandResult.Sticky;
}
```

第一个参数是 `Intent` 对象，其中包含有关要执行的元数据。第二个参数包含 `StartCommandFlags` 提供了有关方法调用的一些信息的值。此参数具有两个可能值之一：

- `StartCommandFlag.Redelivery` – 这意味着 `Intent` 是在前一次重新交付 `Intent`。此值时返回的服务提供 `StartCommandResult.RedeliverIntent` 但已停止，但它无法正确关闭。
- `StartCommandFlag.Retry` - 在上一次时收到此值，则 `OnStartCommand` 调用失败，Android 尝试使用相同目的为以前的失败尝试再次启动该服务。

最后，第三个参数是一个整数值，是唯一的标识请求的应用程序。有可能多个调用方可能会调用同一服务对象。此值用于将请求以停止与给定的请求来启动服务的服务相关联。它将在部分更详细地讨论[停止服务](#)。

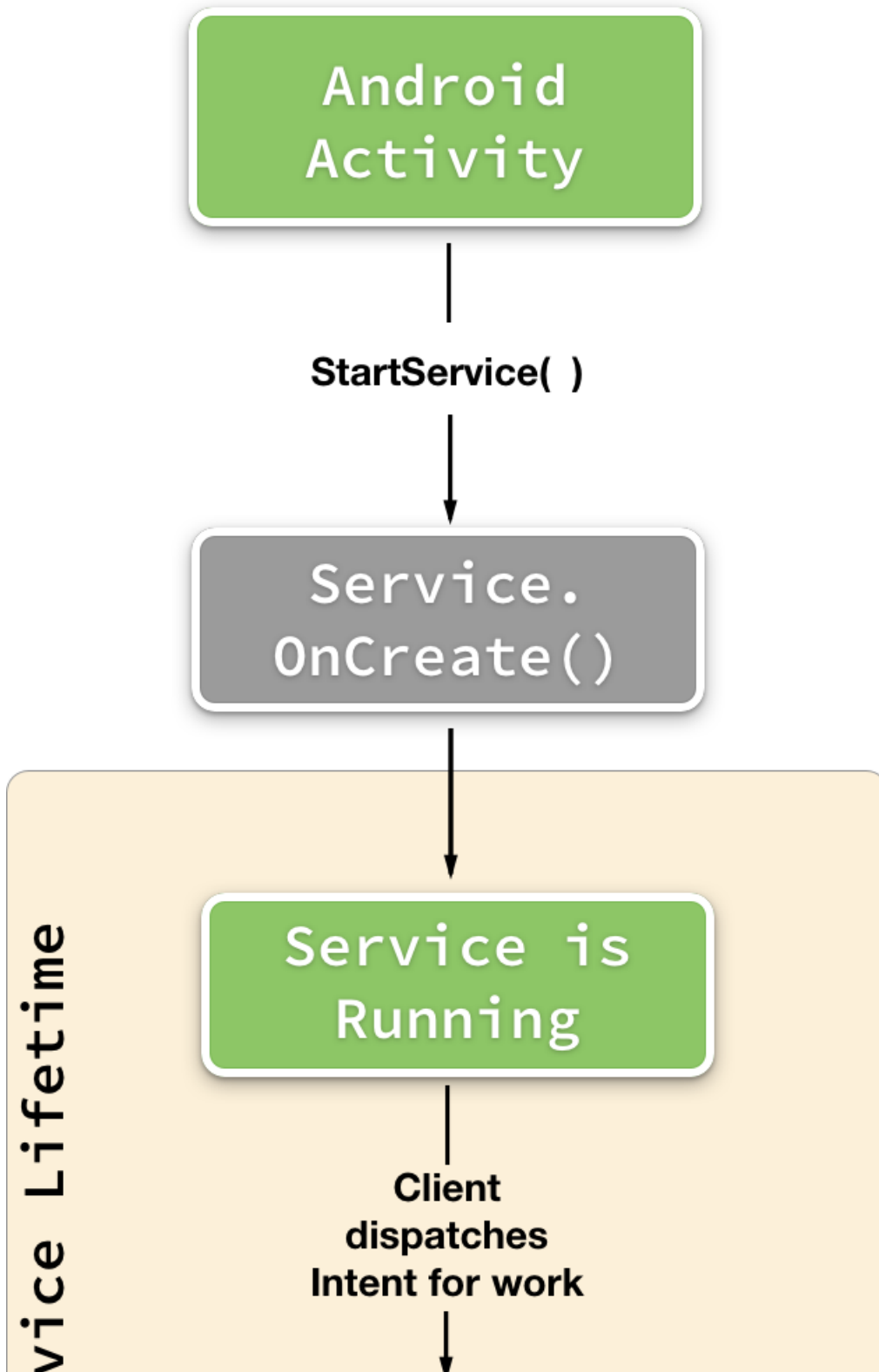
值 `StartCommandResult` 服务返回作为建议向 Android 上要执行的操作如果由于资源约束而终止服务。有三个可能值为 `StartCommandResult`：

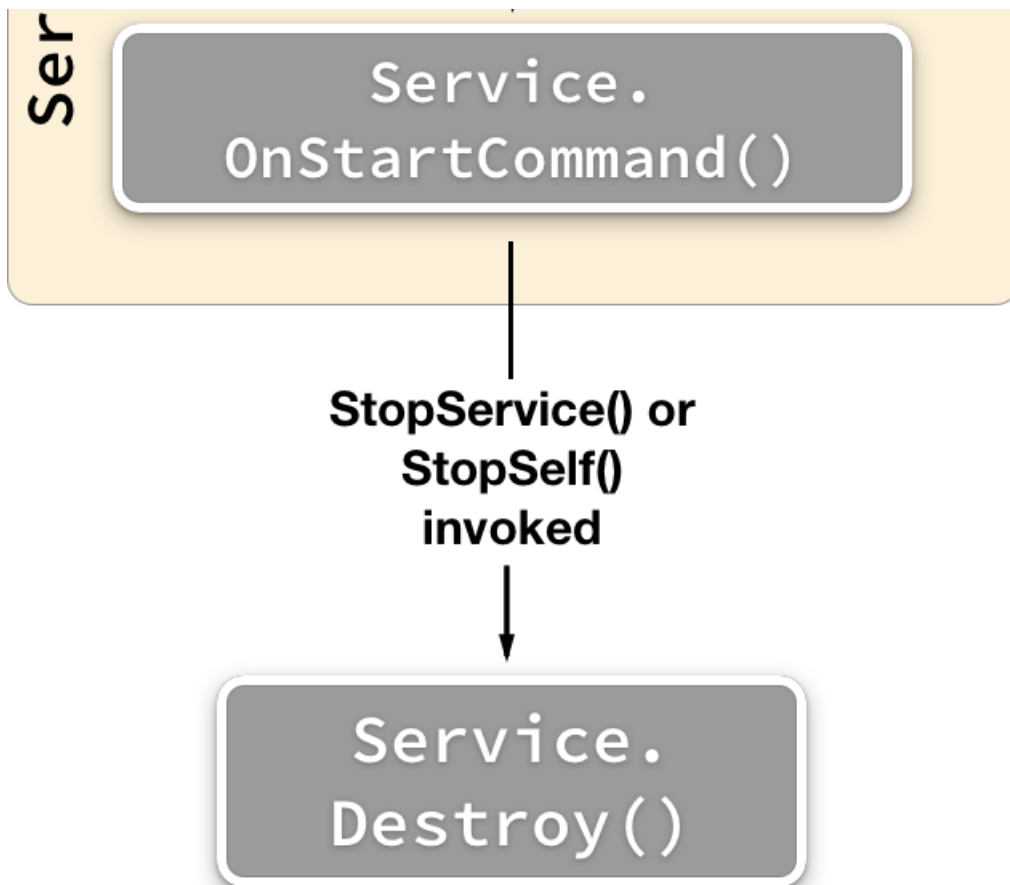
- **`StartCommandResult.NotSticky`** – 此值会指出不需要重新启动已终止该服务的 Android。作为此示例，请考虑在应用中生成库的缩略图的服务。如果终止该服务，它不是重要立即重新创建缩略图-缩略图可以重新创建在下次运行应用程序。

- **StartCommandResult.Sticky** –这告知 Android 在重新启动服务，但不是能提供最初启动该服务的最后一个目的。如果不有任何挂起的意向，若要处理，则 `null` 将意向参数提供。出现这种可能是音乐播放器应用程序；该服务将重启准备好播放音乐，但它会播放最后一首歌曲。
- **StartCommandResult.RedeliverIntent** –此值是将告知 Android，重新启动服务并重新提供上次 `Intent`。此示例是下载数据文件的应用服务。如果终止该服务，数据文件仍需要下载。通过返回 `StartCommandResult.RedeliverIntent`，当 Android 到服务，重新启动的服务还会提供意图（其中包含要下载的文件 URL）。这将允许您下载后，重新启动或恢复（具体取决于代码的切实实现）。

还有第四个值 `StartCommandResult` – `StartCommandResult.ContinuationMask`。返回此值 `OnStartCommand` 并介绍了如何 Android 将继续该服务已终止。通常，此值不用于启动服务。

已启动的服务的密钥生命周期事件显示在此图中：





停止服务

已启动的服务将继续无限期; 运行只要有足够的系统资源, android 会使服务正在运行。客户端必须停止该服务, 或者在完成其工作时, 服务可能停止本身。有两种方法来停止服务:

- **Android.Content.Context.StopService()** -客户端 (例如活动) 可以请求通过停止服务调用 `StopService` 方法:

```
StopService(new Intent(this, typeof(DemoService)));
```

- **Android.App.Service.StopSelf()** -服务可能会自行关闭前通过调用 `StopSelf` :

```
StopSelf();
```

使用 `startId` 来停止服务

多个调用方可以请求启动服务。如果没有未完成的启动请求, 服务可以使用 `startId` 传递到 `OnStartCommand` 将阻止服务被过早停止。 `startId` 将对应于最新调用 `StartService`, 并将递增每次调用它。因此, 如果对后续请求 `StartService` 尚不导致调用 `OnStartCommand`, 该服务可以调用 `StopSelfResult`, 并向其传递的最新值 `startId` 已收到 (而不是只需调用 `StopSelf`)。如果调用 `StartService` 尚不导致相应地调用 `OnStartCommand`, 系统不会停止该服务, 因为 `startId` 中使用 `StopSelf` 调用将不对应于最新 `StartService` 调用。

相关链接

- [StartedServicesDemo \(示例\)](#)
- [Android.App.Service](#)
- [Android.App.StartCommandFlags](#)
- [Android.App.StartCommandResult](#)

- [Android.Content.BroadcastReceiver](#)
- [Android.Content.Intent](#)
- [Android.OS.Handler](#)
- [Android.Widget.Toast](#)
- 状态栏图标

前景服务

2018/10/26 • [Edit Online](#)

前景服务是一种特殊类型的绑定的服务或已启动的服务。有时服务将执行任务，用户必须是主动了解，这些服务称为_前景服务_。前景服务的一个示例是一个应用，会向用户提供说明驾车或步行时。即使应用是在后台，仍然很重要的服务具有足够的资源才能正常工作，并且用户具有的快速而方便的方法来访问应用。对于 Android 应用程序，这意味着前景服务应该接收优先级高于"regular"服务和前景服务必须提供 `Notification` Android 将显示，只要服务正在运行。

若要开始的前景色服务，该应用程序必须调度告知 Android 在启动该服务将意向。然后该服务必须注册为与 Android 前景服务本身。Android 8.0（或更高版本）上运行的应用程序应使用 `Context.StartForegroundService` 方法以启动服务，而应使用具有较旧版本的 Android 设备运行的应用程序 `Context.StartService`

此 C# 扩展方法是举例说明如何启动前景服务。在 Android 8.0 及更高版本会将 `StartForegroundService` 方法，否则为较旧 `StartService` 将使用方法。

```
public static void StartForegroundServiceCompat<T>(this Context context, Bundle args = null) where T : Service
{
    var intent = new Intent(context, typeof(T));
    if (args != null)
    {
        intent.PutExtras(args);
    }

    if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.O)
    {
        context.StartForegroundService(intent);
    }
    else
    {
        context.StartService(intent);
    }
}
```

作为前景色服务进行注册

前景服务启动后，它必须注册其自身与 Android 一起通过调用 `StartForeground`。如果该服务已启动与 `Service.StartForegroundService` 方法，但不注册自身，则 Android 将停止该服务并将标记为不响应的应用。

`StartForeground` 使用这两者都是必需的两个参数：

- 要标识该服务的应用程序中是唯一的的一个整数值。
- 一个 `Notification` Android 将显示为状态栏中，只要服务正在运行的对象。

Android 将为状态栏中显示通知，只要服务正在运行。通知最小值，将向该服务正在运行的用户提供视觉提示。理想情况下，通知应为用户提供应用程序或可能是某些操作按钮来控制应用程序的快捷方式。此示例是音乐播放器—显示通知可能会有所按钮来暂停/播放音乐，可使退回到上一首歌曲，或可以跳到下一首歌曲。

此代码片段是作为前台服务注册服务的示例：

```
// This is any integer value unique to the application.
public const int SERVICE_RUNNING_NOTIFICATION_ID = 10000;

public override StartCommandResult OnStartCommand(Intent intent, StartCommandFlags flags, int startId)
{
    // Code not directly related to publishing the notification has been omitted for clarity.
    // Normally, this method would hold the code to be run when the service is started.

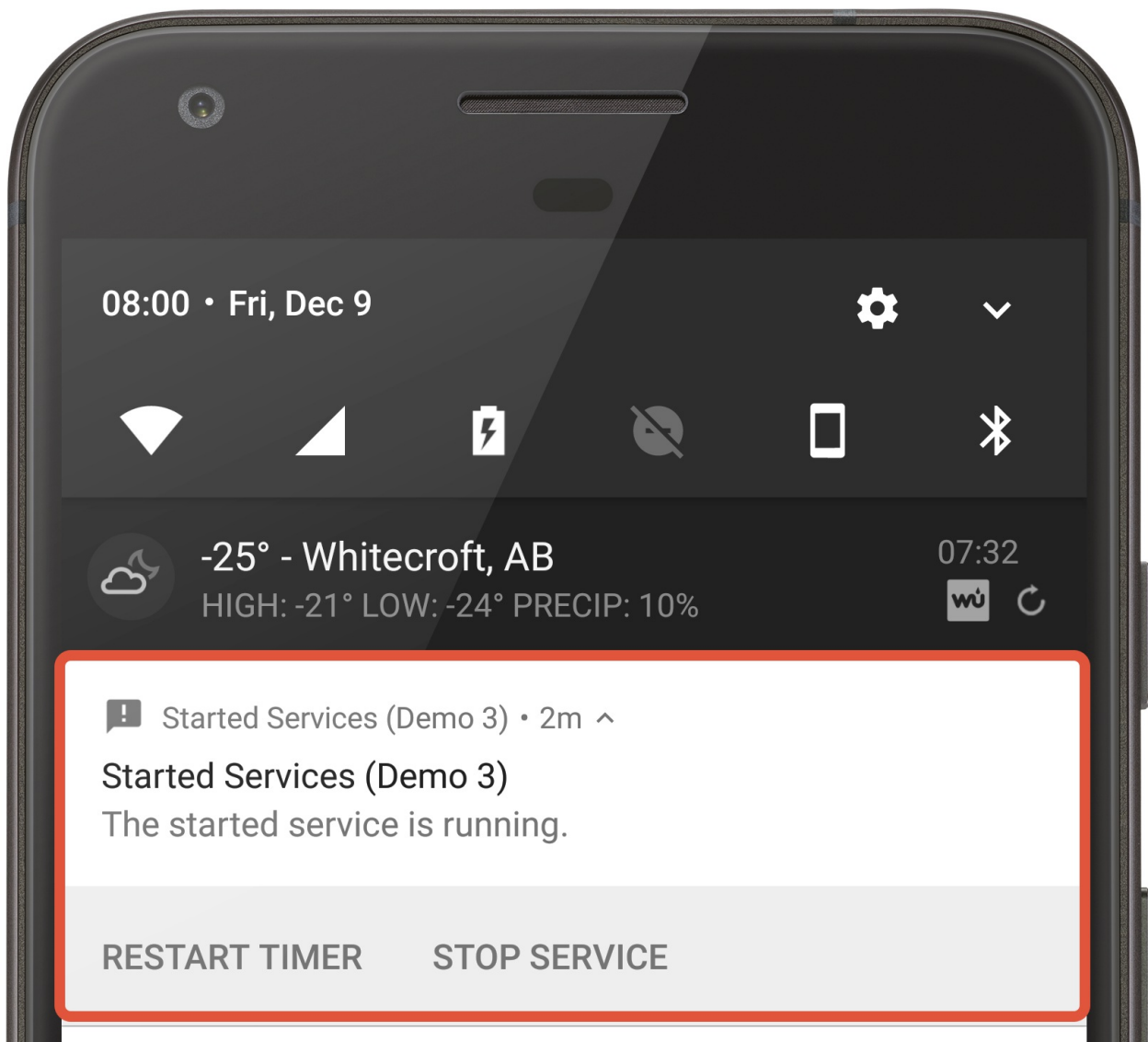
    var notification = new Notification.Builder(this)
        .SetContentTitle(Resources.GetString(Resource.String.app_name))
        .SetContentText(Resources.GetString(Resource.String.notification_text))
        .SetSmallIcon(Resource.Drawable.ic_stat_name)
        .SetContentIntent(BuildIntentToShowMainActivity())
        .SetOngoing(true)
        .AddAction(BuildRestartTimerAction())
        .AddAction(BuildStopServiceAction())
        .Build();

    // Enlist this instance of the service as a foreground service
    StartForeground(SERVICE_RUNNING_NOTIFICATION_ID, notification);
}
```

前一次通知将显示类似于下面的状态栏通知：



此屏幕截图显示包含允许用户控制的服务的两个操作的通知送纸器中展开的通知：



有关通知的详细信息现已推出[本地通知](#)一部分[Android 通知](#)指南。

作为前景色服务正在注销

服务可以取消列出本身作为前台服务通过调用方法 `StopForeground`。 `StopForeground` 不会停止该服务，但它将删除的通知图标和信号可以向下关闭此服务，如有必要的 Android。

此外可以通过传递中删除状态栏通知显示 `true` 方法：

```
StopForeground(true);
```

如果该服务将停止通过调用 `StopSelf` 或 `StopService`，将删除状态栏通知。

相关链接

- [Android.App.Service](#)
- [Android.App.Service.StartForeground](#)
- [本地通知](#)
- [ForegroundServiceDemo](#)（示例）

远程进程中的运行 Android 服务

2018/11/13 • [Edit Online](#)

通常情况下, Android 应用程序中的所有组件将在同一进程中都运行。Android 服务是一个值得注意的例外的可以配置为在其自身的进程中运行并与其他应用程序, 包括来自其他 Android 开发人员共享。本指南介绍了如何创建和使用使用 Xamarin Android 远程服务。

不足的进程服务概述

应用程序启动时, Android 创建要在其中运行应用程序的进程。通常情况下, 所有组件应用程序将在都运行此进程。Android 服务是一个值得注意的例外的可以配置为在其自身的进程中运行并与其他应用程序, 包括来自其他 Android 开发人员共享。这些类型的服务嘿 远程服务_或_进程外服务。这些服务的代码将包含在与主应用程序; 相同的 APK 但是, 当服务启动 Android 将创建只是该服务的新进程。与此相反, 在与应用程序的其余部分相同的进程中运行的服务有时称为_本地服务_。

一般情况下, 不应用程序来实现远程服务。本地服务是足够(和好)在许多情况下的应用程序的要求。扩展的进程具有其自己的内存空间必须管理的 Android。虽然整个应用程序, 这确实产生更多的开销, 但在某些情况下, 它可以在其自己的进程中运行服务更有利:

1. **共享功能**-某些应用程序开发人员可能有多个应用程序和所有应用程序之间共享的功能。打包 Android 服务在其自己的进程中运行可以简化应用程序维护中该功能。还有可能在其自己独立的 APK 中的服务打包并将其单独部署应用程序的其余部分。
2. **改善用户体验**-有两种方法, 进程外服务可以提高应用程序的用户体验。第一种方式处理内存管理。当垃圾回收 (GC) 周期时, Android 会暂停进程中的所有活动, 直到 GC 已完成。用户可能会发现已作为"出现断断续续的问题"或"jank"暂停。当服务在运行时是自己的过程, 它是服务进程已暂停, 不是应用程序进程。暂停将用户得很明显, 因为未暂停应用程序进程(和用户界面)。

其次, 如果一个进程的内存要求变得太大, Android 可能终止该进程以释放设备资源。如果服务有很大的内存需求量, 用作 UI 在同一进程中运行, 然后当 Android 强制回收这些资源时用户界面将关闭, 迫使用户在启动应用程序。但是, 如果在其自己的进程中运行的服务关闭的情况下通过 Android, UI 过程保持不受影响。用户界面可以绑定(和重新启动)服务, 透明的用户, 并恢复正常功能。
3. **提高应用程序性能**-的 UI 进程可能会终止或关闭的情况下独立于服务进程。通过将较长的启动任务移动到进程外服务, 就可以 UI 的启动时间可能改进(假定服务进程保持活动状态的时间启动 UI 之间)。

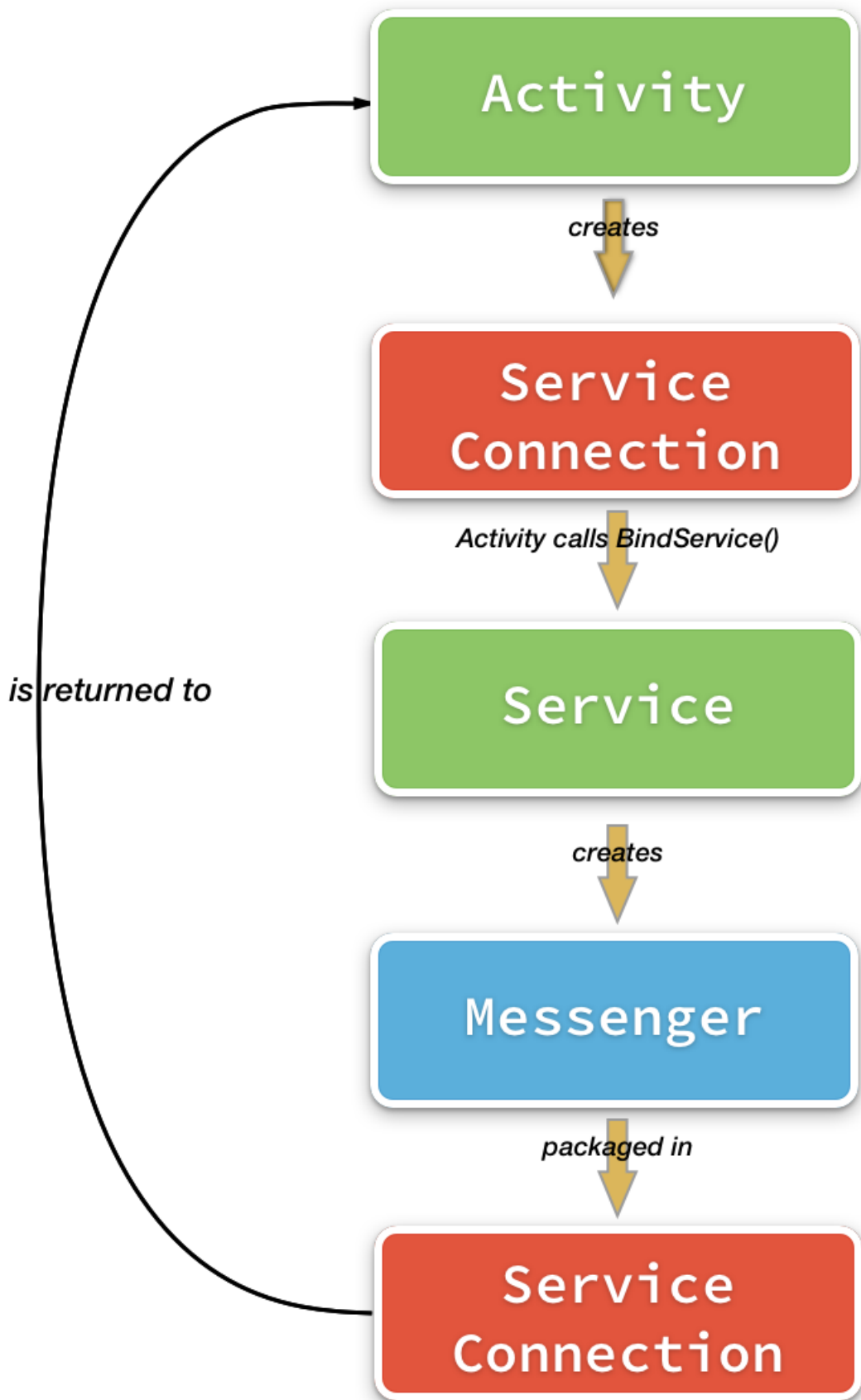
在许多方面, 绑定到另一个进程中运行的服务是与相同[绑定到本地服务](#)。客户端将调用 `BindService` 绑定(和启动, 如有必要)服务。 `Android.OS.IServiceConnection` 将创建对象来管理客户端和服务之间的连接。如果客户端成功绑定到的服务, 则 Android 将返回的对象通过 `IServiceConnection` 可用于在服务上调用方法。客户端然后使用此对象与服务进行交互。若要查看, 下面是绑定到服务的步骤:

- **创建意向**-目的在于明确指示必须使用与绑定到该服务。
- **实现和实例化** `IServiceConnection` 对象 - `IServiceConnection` 对象充当客户端和服务之间的中介。它负责监视客户端和服务之间的连接。
- **调用** `BindService` 方法-调用 `BindService` 将调度意图和 Android 将负责启动服务并建立通信的上一步骤中创建的服务连接客户端和服务。

跨进程边界的需求会引入额外的复杂性: 的通信是单向(客户端到服务器)和客户端不能直接调用服务类上的方法。回想一下, 当服务作为客户端运行时相同的过程, 提供了 Android `IBinder` 对象, 这可能会让为双向通信。这不是与在其自己的进程中运行服务的情况。客户端通信与远程服务的帮助 `Android.OS.Messenger` 类。

当客户端请求与远程服务绑定时, 将调用 Android `Service.OnBind` 生命周期方法, 它将返回内部 `IBinder` 对象, 它封

装由 `Messenger`。`Messenger` 是通过一种特殊的薄包装 `IBinder` Android SDK 提供的实现。`Messenger` 负责的两个不同进程之间的通信。开发人员是关心的序列化消息跨进程边界封送消息，然后在客户端上对它进行反序列化的详细信息。这项工作由 `Messenger` 对象。此图显示了客户端启动绑定到进程外服务时，会涉及的客户端的 Android 组件：



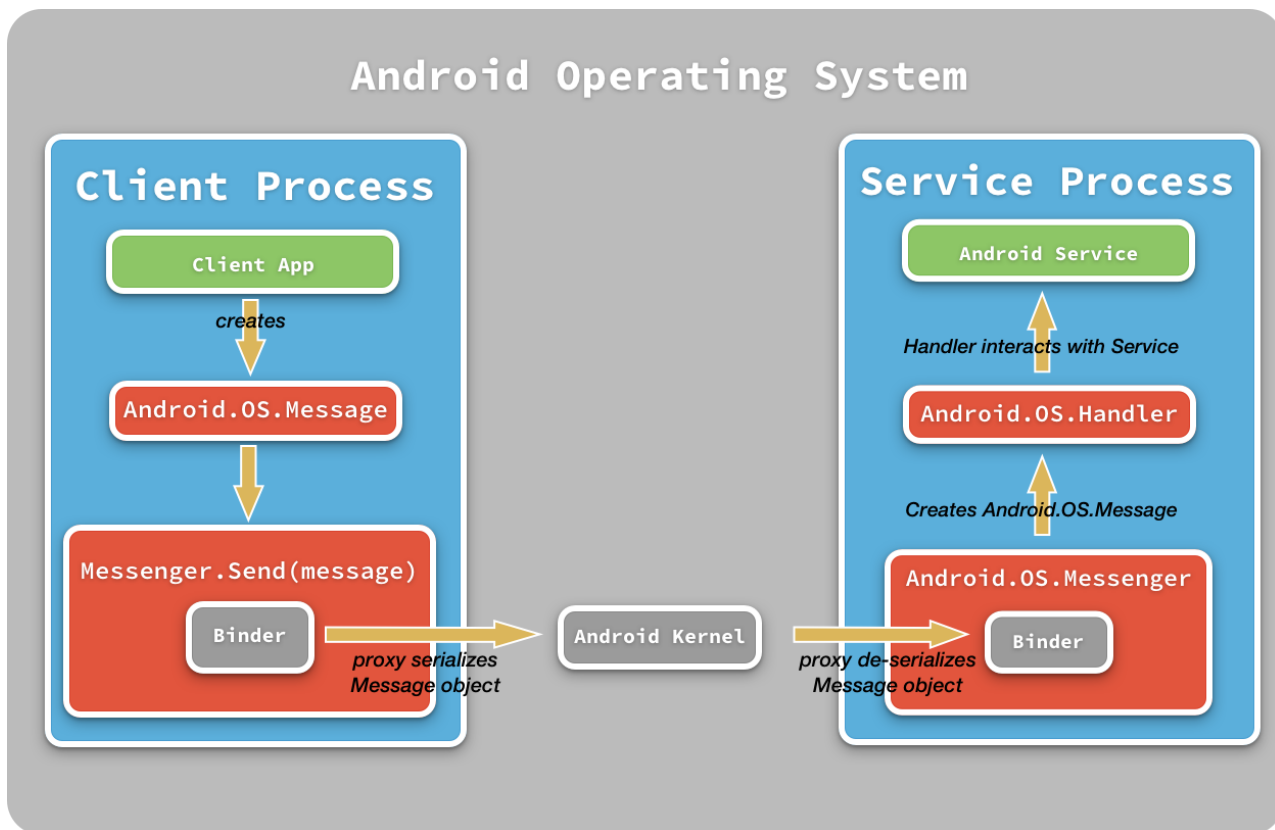
`Service` 远程进程中的类将经历中的本地进程的绑定的服务将经历相同生命周期回调和许多相关的 Api 是相同的。`Service.onCreate` 用于初始化 `Handler`，并插入到 `Messenger` 对象。同样，`onBind` 重写，但而不是返回 `IBinder` 对象，该服务将返回 `Messenger`。此图描述了时会发生什么情况在服务客户端绑定到它：



当 `Message` 收到的服务，它由处理的实例中 `Android.OS.Handler`。该服务将实现其自己 `Handler` 子类必须重写 `HandleMessage` 方法。通过调用此方法 `Messenger`，并接收 `Message` 作为参数。`Handler` 会检查 `Message` 元数据和使用这些信息来调用该服务的方法。

当客户端创建单向通信时发生 `Message` 对象，并将其调度到服务使用 `Messenger.Send` 方法。`Messenger.Send` 将序列化 `Message` 和手动关闭序列化数据，会将消息路由跨进程边界的 `Android` 和服务。`Messenger`，它由承载该服务将创建 `Message` 从传入数据的对象。这 `Message` 放入队列，其中的消息是到一次提交的一个 `Handler`。`Handler` 将检查

中包含的元数据 `Message` 上调用适当的方法和 `Service`。下图说明了这些操作中的各种概念：



本指南介绍了实现进程外服务的详细信息。本文将讨论如何实现用于在其自己的进程中运行的服务和客户端与该服务使用可能通信的方式 `Messenger` 框架。它将还简要讨论的双向通信：客户端将消息发送到服务和将消息发送回客户端服务。服务可以在不同的应用程序之间共享，因为本指南还将讨论使用 Android 权限来限制对服务的客户端访问的一项技术。

IMPORTANT

[Bugzilla 51940/GitHub 1950](#)-使用隔离的进程和应用程序的自定义类的服务不能正确解析重载Xamarin.Android 服务将不会启动正确的报表时 `IsolatedProcess` 设置为 `true`。本指南提供的引用。Xamarin.Android 应用程序仍应能够与用 Java 编写的进程外服务进行通信。

要求

本指南假定你熟悉创建服务。

尽管可以使用与应用程序面向较旧的隐式的意向，但 Android Api，本指南将重点介绍以独占方式使用显式的意图。面向 Android 5.0（API 级别 21）的应用程序或更高版本必须使用目的在于明确指示要绑定的服务;这是将本指南中演示的技术...

创建单独的进程中运行的服务

上文所述，在其自己的进程中运行服务的事实意味着一些不同的 Api 是涉及。作为快速概述，下面是使用绑定和使用远程服务的步骤：

- 创建 `Service` 子类-子类 `Service` 键入和实现绑定的服务的生命周期方法。还有必要设置会通知该服务是在其自己的进程中运行的 Android 的元数据。
- 实现 `Handler` - `Handler` 负责分析客户端请求、提取已从客户端，传递任何参数并调用服务上的相应方法。
- 实例化 `Messenger` -如上所述，每个 `Service` 必须保留的实例 `Messenger` 类将客户端将请求路由到 `Handler` 上一步中创建。

服务旨在在其自己的进程中运行的是，从根本上说，仍绑定的服务。服务类将扩展基 `Service` 类，并使用修饰 `ServiceAttribute` 包含 Android 需要 Android 清单中捆绑的元数据。若要开始进行的以下属性 `ServiceAttribute` 对进程外服务很重要的：

1. `Exported` – 此属性必须设置为 `true` 以允许其他应用程序与服务交互。此属性的默认值为 `false`。
2. `Process` – 必须设置此属性。它用于指定该服务将在运行进程的名称。
3. `IsolatedProcess` – 此属性将启用额外的安全性，告知 Android 在与系统其余部分进行交互的最小权限与独立的沙盒中运行服务。请参阅[Bugzilla 51940-服务使用的隔离的进程和应用程序的自定义类无法正确解析重载](#)。
4. `Permission` – 就可以通过指定客户端必须请求（并被授予）的权限来控制对服务的客户端访问。

若要运行其自己的进程的服务 `Process` 属性上的 `ServiceAttribute` 必须设置为服务的名称。与外部应用程序进行交互 `Exported` 属性应设置为 `true`。如果 `Exported` 是 `false`，然后在相同的 APK（即相同应用程序）中的唯一客户端和运行在同一进程中的将能够与服务交互。

该服务将在运行的进程的种类取决于的值 `Process` 属性。Android 标识三个不同类型的过程：

- **专用流程**–专用流程都是一个且仅可用于启动该应用程序。若要标识为专用的进程，其名称必须启动：（用分号）。在上一代码段和屏幕截图中所示的服务是专用流程。以下代码片段示范了 `ServiceAttribute`：

```
[Service(Name = "com.xamarin.TimestampService",
        Process=":timestampservice_process",
        Exported=true)]
```

- **全局进程**–全局进程中运行的服务是在设备上运行的所有应用程序可以访问。全局进程必须是小写字母开头的完全限定的类名。（除非采取步骤来保护服务，其他应用程序可能会将绑定并与之交互。针对未经授权的使用服务的安全将讨论在本指南中的更高版本。）

```
[Service(Name = "com.xamarin.TimestampService",
        Process="com.xamarin.xample.messengerservice.timestampservice_process",
        Exported=true)]
```

- **隔离进程**–隔离的进程是在独立于其他系统的并具有其自己的任何特殊权限自己沙箱中运行的进程。若要在一个独立的进程中运行服务 `IsolatedProcess` 的属性 `ServiceAttribute` 设置为 `true` 中此代码片段所示：

```
[Service(Name = "com.xamarin.TimestampService",
        IsolatedProcess= true,
        Process="com.xamarin.xample.messengerservice.timestampservice_process",
        Exported=true)]
```

IMPORTANT

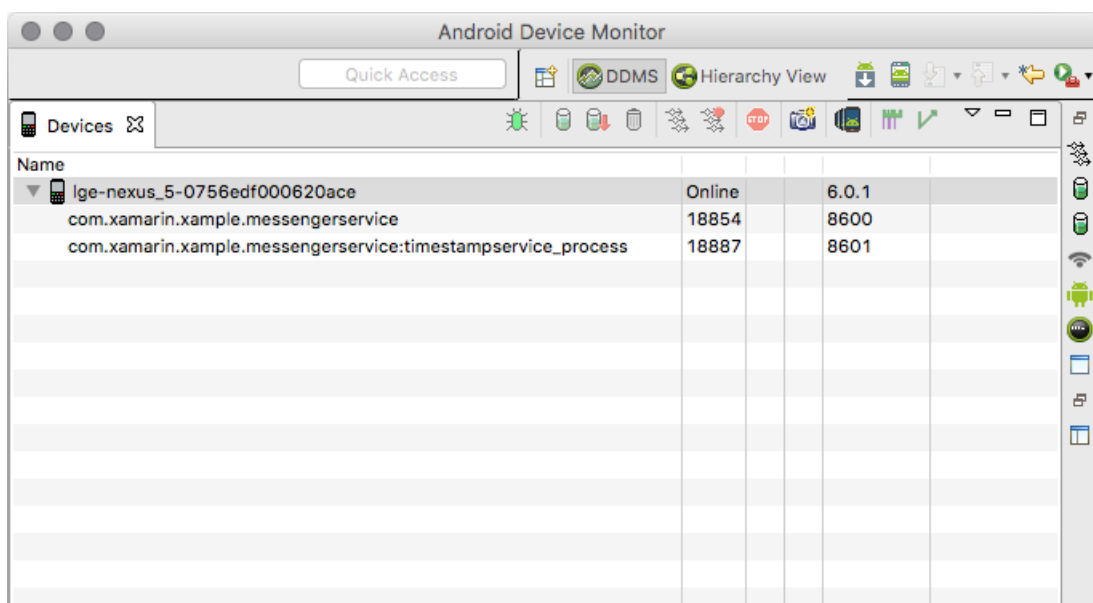
请参阅[Bugzilla 51940-服务使用的隔离的进程和应用程序的自定义类无法正确解析重载](#)

独立的服务是安全的应用程序和设备的不受信任代码的简单方法。例如，应用可能会下载并从网站中执行脚本。在这种情况下，在一个独立的进程中执行这提供了额外的安全性，以防止破坏 Android 设备不受信任的代码。

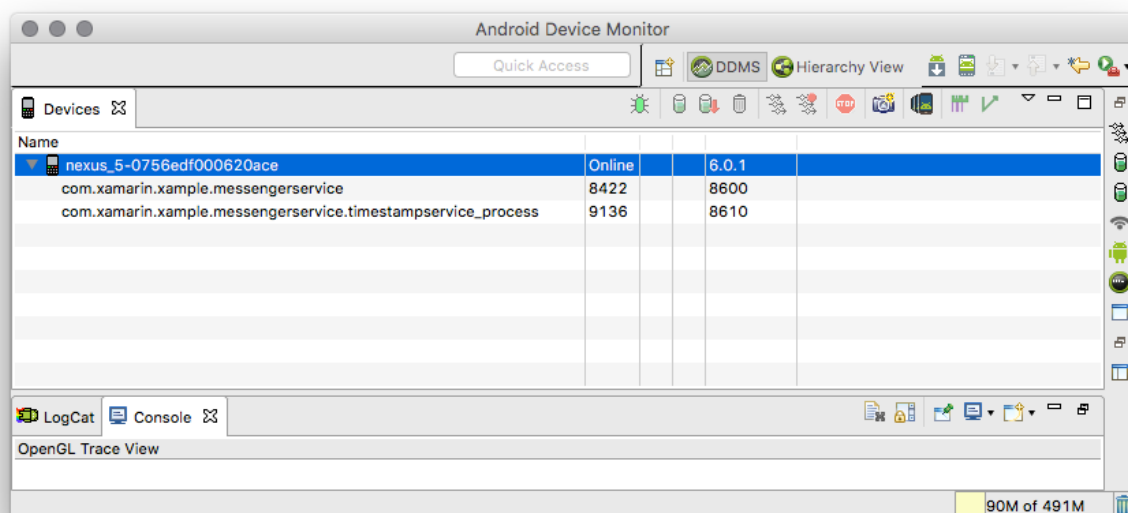
IMPORTANT

一旦被导出服务，不应更改服务的名称。更改服务的名称可能会破坏其他应用程序使用的服务。

若要查看效果的 `Process` 属性具有，下面的屏幕截图显示了在其自己专用的进程中运行的服务：



此下一步的屏幕截图显示了 `Process="com.xamarin.xample.messengerservice.timestampservice_process"` 和全局进程中运行的服务：



一次 `ServiceAttribute` 已设置，需要实现该服务 `Handler`。

实现一个处理程序

若要处理客户端请求，该服务必须实现 `Handler` 并重写 `HandleMessage` methodThis 是方法采用 `Message` 实例，其中封装的方法调用从客户端，并将转换到某些操作或任务调用该服务将执行。 `Message` 对象会公开一个名为属性 `What` 这是一个整数值，其含义客户端和服务之间共享以及与该服务是针对客户端执行某些任务。

示例应用程序中的以下代码段说明的一个示例 `HandleMessage`。在此示例中，有服务的客户端可以请求两个操作：

- 第一个操作是 `_Hello, World_` 消息，客户端已发送一个简单的消息，到服务。
- 第二个操作将调用服务上的方法和检索字符串，在这种情况下的字符串是一条消息，返回已运行的服务已启动并在多长时间的什么时间：

```

public class TimestampRequestHandler : Android.OS.Handler
{
    // other code omitted for clarity

    public override void HandleMessage(Message msg)
    {
        int messageType = msg.What;
        Log.Debug(TAG, $"Message type: {messageType}.");

        switch (messageType)
        {
            case Constants.SAY_HELLO_TO_TIMESTAMP_SERVICE:
                // The client as sent a simple Hello, say in the Android Log.
                break;

            case Constants.GET_UTC_TIMESTAMP:
                // Call methods on the service to retrieve a timestamp message.
                break;
            default:
                Log.Warn(TAG, $"Unknown messageType, ignoring the value {messageType}.");
                base.HandleMessage(msg);
                break;
        }
    }
}

```

此外，还可以为包参数中的服务 `Message`。这将更高版本在本指南中讨论。创建要考虑的下一个主题 `Messenger` 对象来处理传入 `Message`。

实例化 Messenger

如前所述，反序列化 `Message` 对象并调用 `Handler.HandleMessage` 负责 `Messenger` 对象。`Messenger` 类还提供了 `IBinder` 对象在客户端将用于将消息发送到服务。

服务启动时，它将实例化 `Messenger` 注入和 `Handler`。执行此初始化的好时机是在 `OnCreate` 服务的方法。此代码片段是初始化其自己的服务的一个示例 `Handler` 和 `Messenger`：

```

private Messenger messenger; // Instance variable for the Messenger

public override void OnCreate()
{
    base.OnCreate();
    messenger = new Messenger(new TimestampRequestHandler(this));
    Log.Info(TAG, $"TimestampService is running in process id {Android.OS.Process.MyPid()}.");
}

```

此时，最后一步是为 `Service` 重写 `OnBind`。

实现 Service.OnBind

所有绑定的服务，无论是，在它们自己的进程中运行而必须实现 `OnBind` 方法。此方法的返回值是某些客户端可用来与服务交互的对象。完全该对象是什么取决于该服务是本地服务或远程服务。虽然本地服务将返回一个自定义 `IBinder` 实现中，远程服务将返回 `IBinder` 的封装，但 `Messenger` 上一节中的已创建：

```

public override IBinder OnBind(Intent intent)
{
    Log.Debug(TAG, "OnBind");
    return messenger.Binder;
}

```

一旦完成这三个步骤，远程服务可被视为完成。

使用服务

所有客户端必须实现一些代码，以便能够将绑定和使用远程服务。从概念上讲，从客户端的角度来看，很少之间有差异到本地服务或远程服务绑定。客户端调用 `BindService` 方法，并传递显式意向来标识该服务和一个 `IServiceConnection` 可帮助管理客户端和服务之间的连接。

此代码片段示范了如何创建目的在于明确指示向远程服务绑定。其目的必须标识包含服务和服务的名称的包。若要设置此信息的一种方法是使用 `Android.Content.ComponentName` 对象，并提供的到意向。此代码片段是一个示例：

```
// This is the package name of the APK, set in the Android manifest
const string REMOTE_SERVICE_COMPONENT_NAME = "com.xamarin.TimestampService";
// This is the name of the service, according the value of ServiceAttribute.Name
const string REMOTE_SERVICE_PACKAGE_NAME   = "com.xamarin.xample.messengerservice";

// Provide the package name and the name of the service with a ComponentName object.
ComponentName cn = new ComponentName(REMOTE_SERVICE_PACKAGE_NAME, REMOTE_SERVICE_COMPONENT_NAME);
Intent serviceToStart = new Intent();
serviceToStart.SetComponent(cn);
```

当绑定服务时，`IServiceConnection.OnServiceConnected` 方法调用，并且提供了 `IBinder` 到客户端。但是，客户端将不直接使用 `IBinder`。相反，它将实例化 `Messenger` 中的对象 `IBinder`。这是 `Messenger` 客户端将用来与远程服务进行交互。

以下是非常基本的示例 `IServiceConnection` 演示客户端将如何处理连接到和从服务断开连接的实现。请注意，`OnServiceConnected` 方法接收并 `IBinder`，并在客户端创建 `Messenger` 从的 `IBinder`：


```

public class TimestampServiceConnection : Java.Lang.Object, IServiceConnection
{
    static readonly string TAG = typeof(TimestampServiceConnection).FullName;

    MainActivity mainActivity;
    Messenger messenger;

    public TimestampServiceConnection(MainActivity activity)
    {
        IsConnected = false;
        mainActivity = activity;
    }

    public bool IsConnected { get; private set; }
    public Messenger Messenger { get; private set; }

    public void OnServiceConnected(ComponentName name, IBinder service)
    {
        Log.Debug(TAG, $"OnServiceConnected {name.ClassName}");

        IsConnected = service != null
        Messenger = new Messenger(service);

        if (IsConnected)
        {
            // things to do when the connection is successful. perhaps notify the client? enable UI features?
        }
        else
        {
            // things to do when the connection isn't successful.
        }
    }

    public void OnServiceDisconnected(ComponentName name)
    {
        Log.Debug(TAG, $"OnServiceDisconnected {name.ClassName}");
        IsConnected = false;
        Messenger = null;

        // Things to do when the service disconnects. perhaps notify the client? disable UI features?
    }
}

```

一旦创建服务连接和意图的方式，就可以为客户端调用 `BindService` 和启动绑定过程：

```

IServiceConnection serviceConnection = new TimestampServiceConnection(this);
BindActivity(serviceToStart, serviceConnection, Bind.AutoCreate);

```

客户端已顺利绑定到该服务后，`Messenger` 是可用，就可以为客户端发送 `Messages` 到服务。

将消息发送到服务

只要客户端连接，并具有 `Messenger` 对象，就可以与服务通信的调度 `Message` 对象通过 `Messenger`。此通信为单向，客户端发送消息但没有向客户端从服务返回消息。在这方面，`Message` 是即发即弃的机制。

若要创建的首选的方式 `Message` 对象是使用 `Message.obtain` 工厂方法。此方法将提取 `Message` 从由 Android 维护的全局池中的对象。`Message.obtain` 也有一些重载的方法，允许 `Message` 要使用该服务所需的参数和值初始化对象。一次 `Message` 是实例化，其调度到该服务通过调用 `Messenger.Send`。此代码段是一种创建和调度 `Message` 到服务进程：

```
Message msg = Message.Obtain(null, Constants.SAY_HELLO_TO_TIMESTAMP_SERVICE);
try
{
    serviceConnection.Messenger.Send(msg);
}
catch (RemoteException ex)
{
    Log.Error(TAG, ex, "There was a error trying to send the message.");
}
```

有几种不同形式的 `Message.Obtain` 方法。上面的示例使用 `Message.Obtain(Handler h, Int32 what)`。由于这是向进程外服务; 的异步请求将无响应服务, 因此 `Handler` 设置为 `null`。第二个参数, `Int32 what`, 将存储在 `.What` 属性的 `Message` 对象。`.What` 服务进程中的代码使用属性对服务调用方法。

`Message` 类还公开给接收方使用的两个附加属性: `Arg1` 和 `Arg2`。这两个属性都可能有一些特殊达成具有客户端和服务之间的含义的值的整数值。例如, `Arg1` 可能包含客户 ID 和 `Arg2` 可能保存该客户采购订单号。

`Method.Obtain(Handler h, Int32 what, Int32 arg1, Int32 arg2)` 可用于设置两个属性时 `Message` 创建。另一种方法来填充这两个值是设置 `.Arg` 并 `.Arg2` 属性是直接在 `Message` 对象后已创建。

将其他值传递给该服务

可以通过将更复杂的数据传递给该服务 `Bundle`。在这种情况下, 可以额外值放入 `Bundle` 和发送连同 `Message` 通过设置 `.Data` 属性之前发送的属性。

```
Bundle serviceParameters = new Bundle();
serviceParameters.

var msg = Message.Obtain(null, Constants.SERVICE_TASK_TO_PERFORM);
msg.Data = serviceParameters;

messenger.Send(msg);
```

NOTE

一般情况下, `Message` 不应具有的有效负载大小超过 1 MB。大小限制可能会有所不同, 根据 Android 版本和任何专有供应商可能具有进行更改到其实现的 Android 开放源项目 (AOSP) 的设备与捆绑在一起。

从服务返回值

到目前为止讨论过的消息传递体系结构是单向的客户端向服务发送一条消息。如有必要服务无法将值返回给客户然后到目前为止讨论过的所有内容被反转。该服务必须创建 `Message` 打包任何返回值和调度 `Message` 通过 `Messenger` 到客户端。但是, 该服务不会创建其自己 `Messenger`; 相反, 它依赖于客户端实例化和包 `Messenger` 初始请求的一部分。该服务会 `Send` 使用此客户端提供的消息 `Messenger`。

双向通信的事件的顺序是:

1. 客户端绑定到的服务。当该服务和客户端连接时, `IServiceConnection` 由维护客户端将具有对引用 `Messenger` 对象, 用于传输 `Message` 到该服务。为避免混淆, 这将成为 `_服务 Messenger_`。
2. 客户端实例化 `Handler` (称为 `_客户端处理程序_`), 并使用其初始化其自身 `Messenger` (`客户端 Messenger`)。请注意, 服务 `Messenger` 和客户端 `Messenger` 是两个不同的对象处理在两个不同的方向的流量。服务 `Messenger` 处理到服务, 客户端的消息, 而客户端 `Messenger` 将处理从服务向客户端的消息。
3. 客户端创建 `Message` 对象, 并设置 `ReplyTo` 与客户端 `Messenger` 的属性。然后, 消息是发送到使用服务 `Messenger` 服务。
4. 该服务从客户端, 接收消息并执行请求的工作。
5. 该服务发送到客户端的响应时间时, 它将使用 `Message.Obtain` 若要创建一个新 `Message` 对象。

6. 若要将此消息发送到客户端，该服务将提取从客户端 Messenger `.ReplyTo` 属性的客户端消息，并使用它向 `.Send``Message` 返回给客户端。
7. 当客户端收到响应时，它具有自己 `Handler`，将处理 `Message` 通过检查 `.What` 属性（如有必要，提取所包含的任何参数和 `Message`）。

此代码示例演示如何在客户端将实例化 `Message` 并将其打包 `Messenger` 服务应使用用于响应：

```
Handler clientHandler = new ActivityHandler();
Messenger clientMessenger = new Messenger(activityHandler);

Message msg = Message.obtain(null, Constants.GET_UTC_TIMESTAMP);
msg.ReplyTo = clientMessenger;

try
{
    serviceConnection.Messenger.Send(msg);
}
catch (RemoteException ex)
{
    Log.Error(TAG, ex, "There was a problem sending the message.");
}
```

该服务必须进行一些更改到其自身 `Handler` 提取 `Messenger` 并使用它来发送到客户端的答复。此代码片段示范了如何在服务的 `Handler` 会创建 `Message` 并将其发送回客户端：

```
// This is the message that the service will send to the client.
Message responseMessage = Message.obtain(null, Constants.RESPONSE_TO_SERVICE);
Bundle dataToReturn = new Bundle();
dataToReturn.putString(Constants.RESPONSE_MESSAGE_KEY, "This is the result from the service.");
responseMessage.Data = dataToReturn;

// The msg object here is the message that was received by the service. The service will not instantiate a
// client,
// It will use the client that is encapsulated by the message from the client.
Messenger clientMessenger = msg.ReplyTo;
if (clientMessenger!= null)
{
    try
    {
        clientMessenger.Send(responseMessage);
    }
    catch (Exception ex)
    {
        Log.Error(TAG, ex, "There was a problem sending the message.");
    }
}
```

请注意，在上面的代码示例 `Messenger` 客户端创建的实例是不服务接收的相同对象。这些是两个不同 `Messenger` 表示通信通道的两个单独进程中运行的对象。

保护服务与 Android 权限

该 Android 设备上运行的所有应用程序可访问的全局进程中运行的服务。在某些情况下，这种开放性和可用性是不可取，并需要从未经授权的安全客户端的安全对访问服务。若要限制对远程服务的访问的一种方法是使用 Android 权限。

可以通过标识的权限 `Permission` 的属性 `ServiceAttribute` 修饰 `Service` 子类别。这将命名为绑定到该服务时，必须授予客户端的权限。如果客户端不具有相应的权限，则将引发 Android `Java.Lang.SecurityException` 客户端尝试绑定到服务。

有四个 Android 提供的不同权限级别：

- **正常**—这是默认的权限级别。它用于标识可以将自动授予 android 请求它的客户端的低风险权限。用户无需显式授予这些权限，但可以在应用设置中查看权限。
- **签名**—这是授予的权限将自动由 Android 进行签名的相同证书的应用程序的特殊类别。此权限旨在使其可轻松地应用程序开发人员共享组件或自己的应用而无需求助于常量的审批用户之间的数据。
- **signatureOrSystem**—这是非常类似于签名上面所述的权限。除了自动授予对由相同的证书签名的应用，此权限还将向授予应用进行签名的相同用于对应用进行签名的证书安装的 Android 系统映像。此权限通常仅用于 Android ROM 开发人员通过允许其使用第三方应用的应用程序。它不常使用的意为广大公众，常规分发的应用。
- **危险**—危险的权限是指那些可能会导致用户的问题。出于此原因，**危险权限**必须明确批准的用户。

因为 `signature` 并 `normal` 自动授予权限在安装时通过 Android 至关重要，在安装 APK 托管服务之前 APK 包含客户端。如果首次安装客户端，则 Android 将授予的权限。在这种情况下，它将需要卸载客户端的 APK，安装服务 APK，并重新安装客户端的 APK。

有两种常见的方式与 Android 权限的服务安全：

1. **实现签名级别安全性**—签名级别安全性意味着，权限将自动授予对这些应用程序使用包含该服务对 APK 进行签名的同一密钥进行签名的。这是开发人员保护其服务，但保留其可从自己的应用程序访问的简单方法。通过设置声明签名级权限 `Permission` 的属性 `ServiceAttribute` 到 `signature`：

```
[Service(Name = "com.xamarin.TimestampService",
        Process="com.xamarin.TimestampService.timestampservice_process",
        Permission="signature")]
public class TimestampService : Service
{
}
```

2. **创建自定义权限**—可以服务的开发人员用来创建该服务的自定义权限。这是最好的开发人员希望与其他开发人员的应用程序共享其服务。自定义权限需要更多工作才能实现，在下面将讨论。

创建自定义的简化的示例 `normal` 权限将在下一节中所述。有关 Android 权限的详细信息，请参阅 Google 的文档[最佳做法和安全](#)。有关 Android 权限的详细信息，请参阅[权限部分](#)的 Android 权限有关的详细信息的[应用程序清单的 Android 文档](#)。

NOTE

一般情况下，Google 不鼓励使用自定义权限因为它们可能会给用户造成混淆。

创建自定义权限

若要使用的自定义权限，来声明它的服务时的客户端显式请求该权限。

若要创建 APK，在服务中的权限 `permission` 元素添加到 `manifest` 中的元素 **AndroidManifest.xml**。此权限必须具有 `name`，`protectionLevel`，和 `label` 属性集。`name` 属性必须设置为一个字符串，唯一标识该权限。名称将显示在应用信息视图 **Android** 设置（如在下一节中所示）。

`protectionLevel` 属性必须设置为上面所述的四个字符串值之一。`label` 和 `description` 必须引用字符串资源，用于向用户提供的用户友好名称和说明。

此代码段是声明一个自定义的一个示例 `permission` 中的属性 **AndroidManifest.xml** 包含服务的 apk：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.xamarin.xample.messengerservice">

    <uses-sdk android:minSdkVersion="21" />

    <permission android:name="com.xamarin.xample.messengerservice.REQUEST_TIMESTAMP"
        android:protectionLevel="signature"
        android:label="@string/permission_label"
        android:description="@string/permission_description"
        />

    <application android:allowBackup="true"
        android:icon="@mipmap/icon"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

    </application>
</manifest>

```

然后, 将**AndroidManifest.xml**客户端的 APK 必须显式请求此新权限。这是通过添加 `users-permission` 归于**AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.xamarin.xample.messengerclient">

    <uses-sdk android:minSdkVersion="21" />

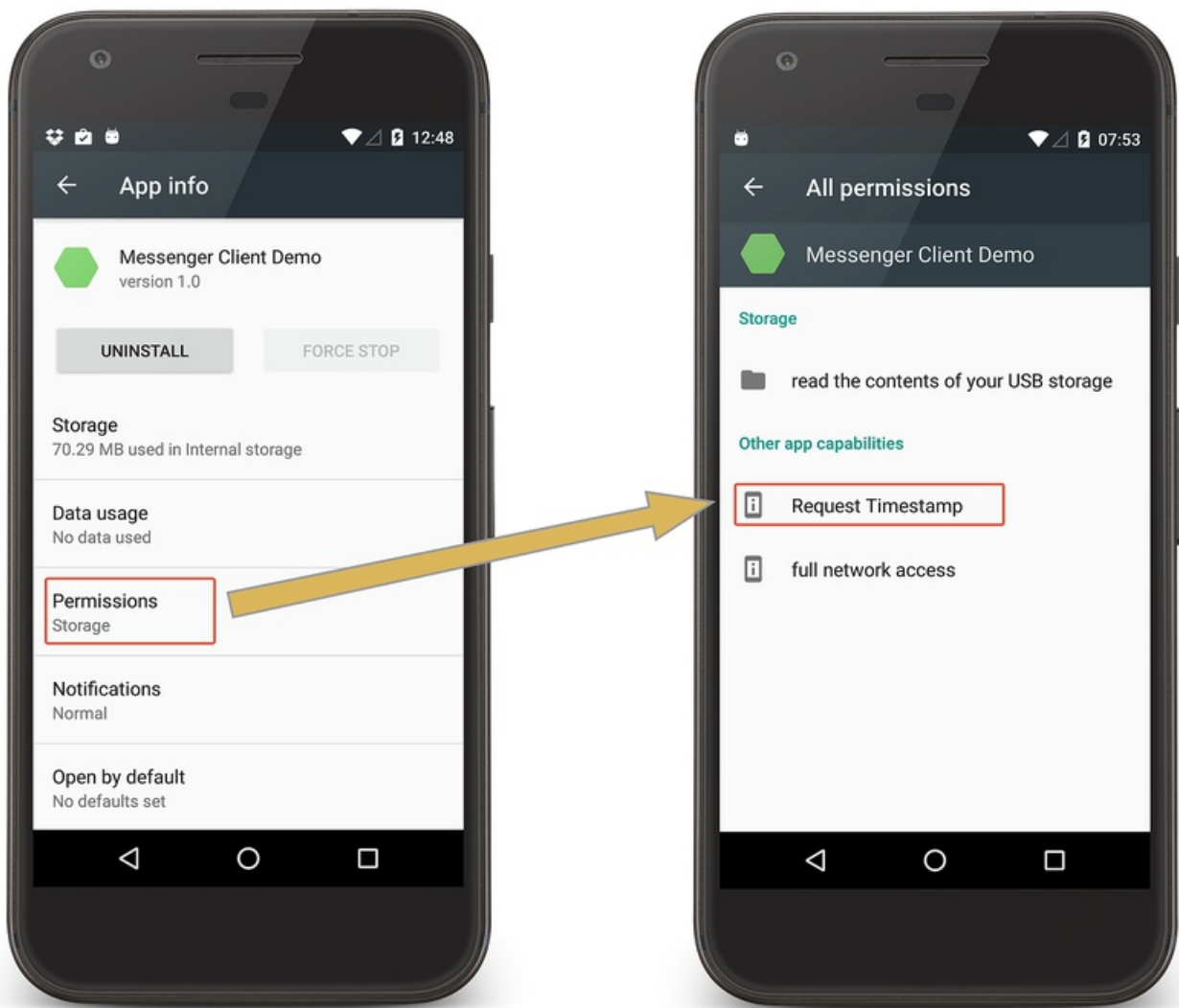
    <uses-permission android:name="com.xamarin.xample.messengerservice.REQUEST_TIMESTAMP" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/icon"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
    </application>
</manifest>

```

查看向应用授予的权限

若要查看应用程序已被授予的权限, 打开 Android 设置应用, 然后选择应用。查找和选择列表中的应用程序。从应用信息屏幕上, 点击**权限**这将显示一个视图, 显示所有应用程序授予的权限:



总结

本指南是有关如何在远程进程中运行的 Android 服务的高级的讨论。本地计算机和远程服务之间的差异介绍过的方法, 以及为什么远程服务可以是对稳定性和性能的 Android 应用程序很有帮助的一些原因。介绍如何实现远程服务和客户端可以在与服务进行通信后, 指南往提供一种方法来限制从只有经过授权的客户端访问该服务。

相关链接

- [处理程序](#)
- [消息](#)
- [Messenger](#)
- [ServiceAttribute](#)
- [导出属性](#)
- [使用隔离的进程和应用程序的自定义类的服务不能正确解析重载](#)
- [进程和线程](#)
- [Android 清单的权限](#)
- [安全提示](#)
- [MessengerServiceDemo \(示例\)](#)

服务通知

2018/10/26 • [Edit Online](#)

本指南介绍如何 Android 服务可能使用本地通知调度到用户的信息。

服务通知概述

服务通知允许用于向用户显示信息的应用，即使 Android 应用程序不在前景中。很可能的通知，以便提供用于用户，例如，显示从应用程序的活动操作。下面的代码示例演示了一项服务可能会如何调度到用户的通知：

```
[Service]
public class MyService: Service
{
    // A notification requires an id that is unique to the application.
    const int NOTIFICATION_ID = 9000;

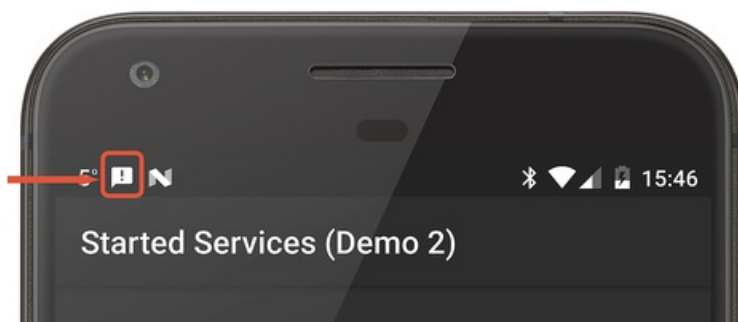
    public override StartCommandResult OnStartCommand(Intent intent, StartCommandFlags flags, int startId)
    {
        // Code omitted for clarity - here is where the service would do something.

        // Work has finished, now dispatch a notification to let the user know.
        Notification.Builder notificationBuilder = new Notification.Builder(this)
            .SetSmallIcon(Resource.Drawable.ic_notification_small_icon)
            .SetContentTitle(Resources.GetString(Resource.String.notification_content_title))
            .SetContentText(Resources.GetString(Resource.String.notification_content_text));

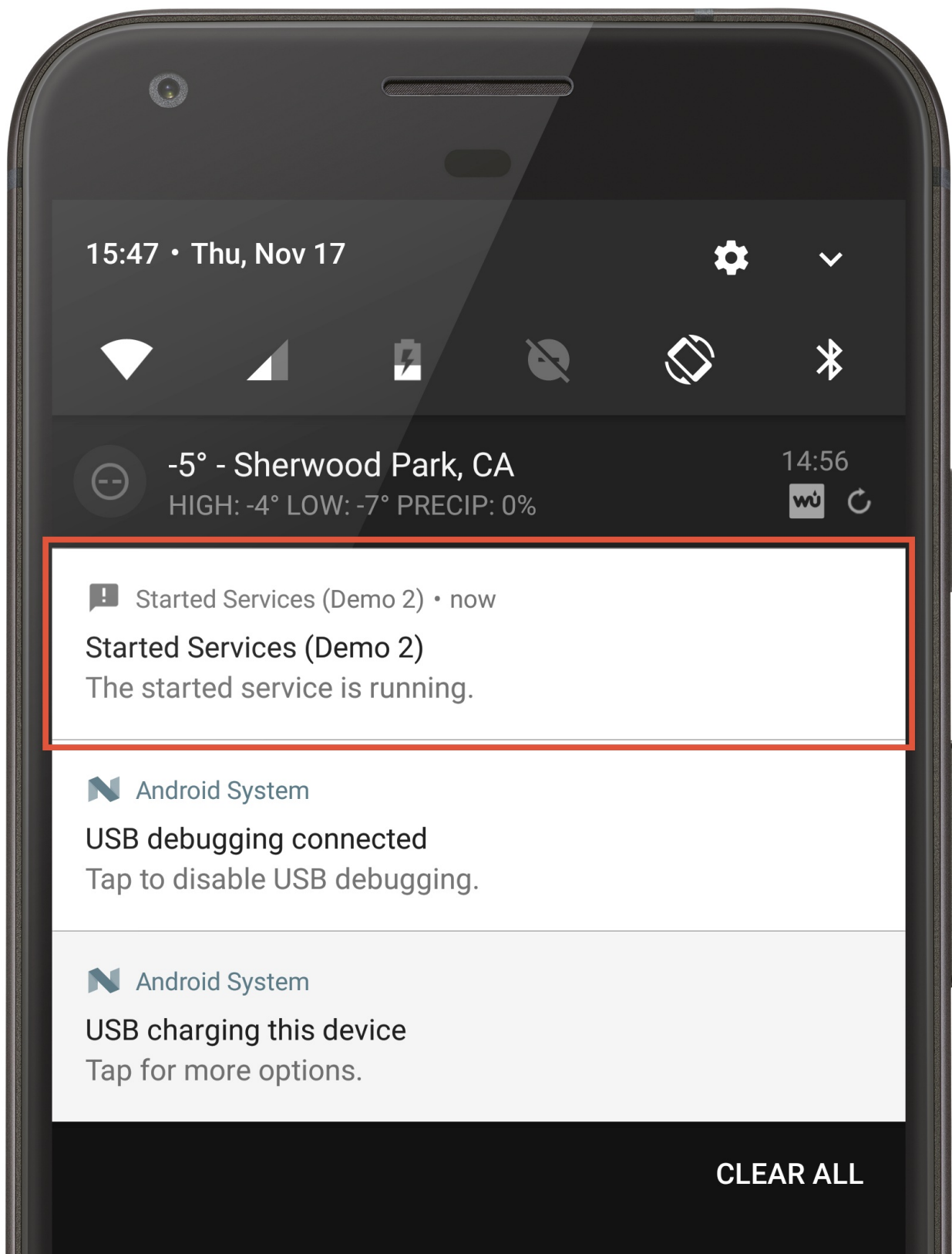
        var notificationManager = (NotificationManager)GetSystemService(NotificationService);
        notificationManager.Notify(NOTIFICATION_ID, notificationBuilder.Build());
    }
}
```

这是屏幕截图显示通知的示例：

Notification from
Service



当用户滑向下的顶部的通知屏幕时，将显示完整的通知：



正在更新通知

若要更新通知，该服务将重新发布使用相同的通知 id。Android 将显示或更新状态栏中根据需要的通知。


```
void UpdateNotification(string content)
{
    var notification = GetNotification(content, pendingIntent);

    NotificationManager notificationManager =
(NotificationManager)GetSystemService(Context.NotificationService);
    notificationManager.Notify(NOTIFICATION_ID, notification);
}

Notification GetNotification(string content, PendingIntent intent)
{
    return new Notification.Builder(this)
        .SetContentTitle(tag)
        .SetContentText(content)
        .SetSmallIcon(Resource.Drawable.NotifyLg)
        .SetLargeIcon(BitmapFactory.DecodeResource(Resources, Resource.Drawable.Icon))
        .SetContentIntent(intent).Build();
}
```

有关通知的详细信息现已推出[本地通知](#)一部分[Android 通知指南](#)。

相关链接

- [在 Android 中的本地通知](#)

在 Xamarin.Android 中的广播的接收器

2018/11/13 • [Edit Online](#)

本部分讨论如何使用广播接收器。

广播的接收器概述

一个_广播的接收器_是允许对消息作出响应的应用程序的 Android 组件 (Android `Intent`) 的广播 Android 操作系统或应用程序。请按照广播_发布-订阅_模型-导致广播发布和接收感兴趣事件的那些组件的事件。

Android 标识两种类型的广播：

- **显式广播**-广播这些类型面向特定的应用程序。显式广播的最常见用途是启动活动。举例说明了当应用需要拨打的电话号码; 显式广播它会调度意向面向 Android 和电话号码沿传递拨打的手机应用。Android 然后将路由到 Phone 应用程序的意图。
- **隐式广播**-这些广播调度到设备上的所有应用。隐式广播的一个示例是 `ACTION_POWER_CONNECTED` 意向。Android 检测到设备上的电池正在充电每次发布此目的。Android 将路由到已注册此事件的所有应用此目的。

广播的接收器是一个的子类 `BroadcastReceiver` 类型和它必须重写 `OnReceive` 方法。Android 将执行 `OnReceive` 上主线程, 因此此方法应设计为可执行速度快。在生成中的线程时应小心 `OnReceive` 因为 Android 可能会终止进程, 在方法结束时。如果广播的接收器必须执行长时间运行的工作, 则我们建议安排_作业_使用 `JobScheduler` 或 `Firestore` 作业调度程序_。计划作业的工作将在单独的指南中讨论。

*意向筛选器_用于注册, 以便 Android 能够正确地将消息广播的接收器。*意向筛选器可以指定在运行时 (这有时称为_上下文注册接收方_或是_动态注册), 或在 Android 清单 (可以以静态方式定义_清单注册接收方_)。Xamarin.Android 提供了 C# 属性, `IntentFilterAttribute`, , 将以静态方式注册意向筛选器 (这将在本指南后面的更详细地讨论)。从 Android 8.0, 它不能进行隐式广播静态注册的应用程序。

清单注册接收方和上下文注册接收方之间的主要区别是到将仅针对广播响应上下文注册接收方运行时应用程序, 而清单注册的接收方可以响应即使可能未在运行该应用将广播。

有两组 Api 用于管理广播接收方和发送广播：

1. `Context` - `Android.Content.Context` 类可用于注册广播的接收器将响应的系统范围内的事件。 `Context` 还用于发布系统范围内广播。
2. `LocalBroadcastManager` - 这是一个 API, 可通过 [Xamarin 支持库 v4 NuGet 包](#)。此类用于保留广播和在使用它们的应用程序的上下文中隔离的广播的接收器。此类可用于阻止其他应用程序响应仅应用程序广播或将消息发送到专用的接收方。

广播的接收器可能无法显示对话框, 并且强烈建议不要使用, 以后启动从广播接收器中的某个活动。如果广播的接收器必须通知用户, 它应发布通知。

不能将绑定到或开始从广播接收器中的服务。

本指南将介绍如何创建广播的接收器以及如何注册, 以便它可能会收到广播。

创建广播的接收器

若要创建广播的接收器在 Xamarin.Android 中, 应用程序应子类 `BroadcastReceiver` 类中, 与它修饰 `BroadcastReceiverAttribute`, 并替代 `OnReceive` 方法：

```
[BroadcastReceiver(Enabled = true, Exported = false)]
public class SampleReceiver : BroadcastReceiver
{
    public override void OnReceive(Context context, Intent intent)
    {
        // Do stuff here.

        String value = intent.GetStringExtra("key");
    }
}
```

当 Xamarin.Android 编译类时，它还将使用必要的元数据注册接收器更新 AndroidManifest。静态注册广播接收器中，对于 `Enabled` 正确必须设置为 `true`，否则 Android 将无法再创建接收方的实例。

`Exported` 属性控制广播的接收器是否可以接收来自外部应用程序消息。如果未显式设置属性，如果有与广播接收器关联的意向筛选器基于的 android 确定属性的默认值。如果至少一个意向筛选器的广播接收方则 Android 将假定 `Exported` 属性是 `true`。如果没有与广播接收器关联的意向筛选器，则 Android 将认为此值越 `false`。

`OnReceive` 方法接收到的引用 `Intent`，被调度到广播接收器。这样就可以发送方的意图以将值传递到广播接收器。

广播接收器以静态方式注册意向筛选器

当 `BroadcastReceiver` 用修饰 `IntentFilterAttribute`，Xamarin.Android 将添加必需 `<intent-filter>` 元素 android 清单在编译时。以下代码片段是广播接收器将时设备完成启动（如果用户授予了适当的 Android 权限）运行的示例：

```
[BroadcastReceiver(Enabled = true)]
[IntentFilter(new[] { Android.Content.Intent.ActionBootCompleted })]
public class MyBootReceiver : BroadcastReceiver
{
    public override void OnReceive(Context context, Intent intent)
    {
        // Work that should be done when the device boots.
    }
}
```

还有可能创建意向筛选器将响应的自定义的目的。请看下面的示例：

```
[BroadcastReceiver(Enabled = true)]
[IntentFilter(new[] { "com.xamarin.example.TEST" })]
public class MySampleBroadcastReceiver : BroadcastReceiver
{
    public override void OnReceive(Context context, Intent intent)
    {
        // Do stuff here
    }
}
```

面向 Android 8.0（API 级别 26）的应用，或更高版本不能以静态方式注册隐式广播。显式广播可能仍以静态方式注册应用。没有此限制不受约束的隐式广播的简短列表。这些异常中所述[隐式广播异常](#)Android 文档中的指南。对于有兴趣隐式广播的应用程序必须这样做因此动态使用 `RegisterReceiver` 方法。这在下一部分介绍。

上下文注册广播的接收器

通过调用执行上下文（也称为动态注册）的注册为接收方 `RegisterReceiver` 方法和广播的接收器必须通过调用注销 `UnregisterReceiver` 方法。若要防止泄漏资源，务必之后取消注册接收方已不再相关的上下文（活动或服务）。例如，服务可能将广播通知有可用来向用户显示更新的活动的意图。当在活动开始时，它将注册为这些意图。当活动移动到后台，并且不再对用户可见，它应注销接收方，因为用于显示更新的 UI 不再可见。下面的代码段是活动的如何注册和注销广播的接收器上下文中的示例：

```
[Activity(Label = "MainActivity", MainLauncher = true, Icon = "@mipmap/icon")]
public class MainActivity: Activity
{
    MySampleBroadcastReceiver receiver;

    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        receiver = new MySampleBroadcastReceiver()

        // Code omitted for clarity
    }

    protected override OnResume()
    {
        base.OnResume();
        RegisterReceiver(receiver, new IntentFilter("com.xamarin.example.TEST"));
        // Code omitted for clarity
    }

    protected override OnPause()
    {
        UnregisterReceiver(receiver);
        // Code omitted for clarity
        base.OnPause();
    }
}
```

在上一示例中，当活动进入前台，它将注册将侦听的自定义意向使用广播的接收器 `OnResume` 生命周期方法。因为该活动将移到后台，`OnPause()` 方法将取消注册接收方。

发布广播

广播可能会发布到创建的意向对象和调度其与在设备上安装的所有应用 `SendBroadcast` 或 `SendOrderedBroadcast` 方法。

1. **Context.SendBroadcast** 方法—有几个此方法的实现。这些方法将广播到整个系统的目的。将收到以不确定的顺序意向的广播的接收器。这提供了大量的灵活性，但意味着其他应用程序可以注册并接收意图。这可能会造成潜在的安全风险。应用程序可能需要实现添加安全性以防止未经授权的访问。一个可能的解决方案是使用 `LocalBroadcastManager` 这会仅分发应用程序的专用空间中的消息。此代码片段是举例说明如何分派使用其中一个意向 `SendBroadcast` 方法：

```
Intent message = new Intent("com.xamarin.example.TEST");
// If desired, pass some values to the broadcast receiver.
message.PutExtra("key", "value");
SendBroadcast(message);
```

此代码段是另一个示例通过使用发送广播 `Intent.SetAction` 的方法来确定该操作：

```
Intent intent = new Intent();
intent.SetAction("com.xamarin.example.TEST");
intent.PutExtra("key", "value");
SendBroadcast(intent);
```

2. **Context.SendOrderedBroadcast**—这是方法非常类似于 `Context.SendBroadcast`，区别在于目的将已发布的一个时，接收方是否注册了接收方的顺序。

LocalBroadcastManager

[Xamarin 支持库 v4](#)提供了一个名为的帮助器类 `LocalBroadcastManager`。 `LocalBroadcastManager` 适用于不希望发送或接收广播从其他应用在设备上的应用。 `LocalBroadcastManager` 时才会发布消息上下文内的应用程序，并且仅向注册到这些广播接收器 `LocalBroadcastManager`。此代码片段是注册广播的接收器使用的示例

`LocalBroadcastManager`：

```
Android.Support.V4.Content.LocalBroadcastManager.GetInstance(this). RegisterReceiver(receiver, new
IntentFilter("com.xamarin.example.TEST"));
```

在设备上的其他应用程序不能接收通过发布消息 `LocalBroadcastManager`。此代码片段演示如何调度意向使用

`LocalBroadcastManager`：

```
Intent message = new Intent("com.xamarin.example.TEST");
// If desired, pass some values to the broadcast receiver.
message.PutExtra("key", "value");
Android.Support.V4.Content.LocalBroadcastManager.GetInstance(this).SendBroadcast(message);
```

相关链接

- [BroadcastReceiver API](#)
- [Context.RegisterReceiver API](#)
- [Context.SendBroadcast API](#)
- [Context.UnregisterReceiver API](#)
- [意向 API](#)
- [IntentFilter API](#)
- [LocalBroadcastManager \(Android 文档\)](#)
- [在 Android 中的本地通知指南](#)
- [Android 支持库 v4 \(NuGet\)](#)

Android 本地化

2018/11/14 • [Edit Online](#)

本文档介绍了本地化功能的 Android SDK 以及如何使用 Xamarin 访问它们。

Android 平台功能

本部分介绍 Android 的主要本地化功能。请跳到[下一节](#)若要查看特定的代码和示例。

区域设置

用户选择在其语言设置 > 语言和输入。选择此选项控制显示的语言和区域设置使用（例如。适用于日期和数字格式设置）。

可通过当前上下文的查询的当前区域设置 `Resources`：

```
var lang = Resources.Configuration.Locale; // eg. "es_ES"
```

此值将为包含语言代码和区域设置代码，以下划线分隔的区域设置标识符。有关参考，下面是[Java 区域设置列表](#)并[StackOverflow 通过 Android 支持区域设置](#)。

常见示例包括：

- `en_US` 英语（美国）
- `es_ES` 西班牙语（西班牙）
- `ja_JP` 日语（日本）
- `zh_CN` 中文（中国）
- `zh_TW` 中文（台湾）
- `pt_PT` 葡萄牙语（葡萄牙）
- `pt_BR` 葡萄牙语（巴西）

LOCALE_CHANGED

Android 生成 `android.intent.action.LOCALE_CHANGED` 当用户更改其语言选择。

可以选择活动来处理此情况设置 `android:configChanges` 属性上的活动，如下：

```
[Activity (Label = "@string/app_name", MainLauncher = true, Icon="@drawable/launcher",  
    ConfigurationChanges = ConfigChanges.Locale | ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
```

在 Android 中的国际化基础知识

Android 的本地化策略具有以下主要部分：

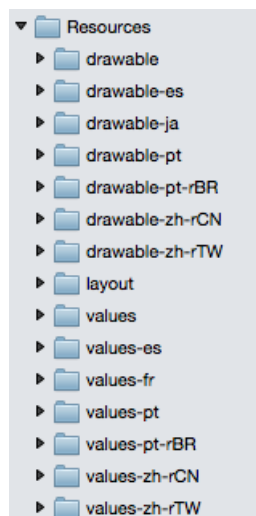
- 若要包含本地化的字符串、图像和其他资源的资源文件夹。
- `getText` 方法，用于检索代码中的已本地化的字符串
- `@string/id` 在 AXML 文件，以自动放置在布局中的已本地化的字符串。

资源文件夹

Android 应用程序管理资源文件夹中的大多数内容，例如：

- 布局-包含 AXML 布局文件。
- 可绘制-包含图像和其他可绘制资源。
- 值-包含字符串。
- 原始-包含数据文件。

大多数开发人员已熟悉使用 **dpi** 上后缀 **drawable** directory 来提供多个版本的映像, 让 Android 选择每个设备的正确版本。相同的机制用于后缀与语言和区域性标识符的资源目录中提供多个语言翻译。



NOTE

指定顶级语言时 **es** 只有两个字符是必需的; 但在指定完整的区域设置时, 目录名称格式需要短划线和小写 **r** 来分隔两个部分, 例如 **pt rBR** 或 **zh rCN**。这与在代码中, 其中包含下划线 (例如返回的值。 **pt_BR**) 格式模式中出现的位置生成。这两种是不同的值.NET **CultureInfo** 类使用, 其中包含短划线仅 (例如。 **pt-BR**) 格式模式中出现的位置生成。在 Xamarin 平台上工作时, 请记住这些差异。

Strings.xml 文件格式

本地化值目录 (例如。值 **es** 或值 **pt rBR**) 应包含名为的文件 **Strings.xml** 将包含对该区域设置翻译后的文本。

每个翻译的字符串是具有 ID 指定为资源的 XML 元素 **name** 属性和已翻译的字符串的值:

```
<string name="app_name">TaskyL10n</string>
```

需要根据正常 XML 规则, 转义和 **name** 必须是有效的 Android 资源 ID (没有空格或短划线)。下面是该示例的默认 (英语) 字符串文件的示例:

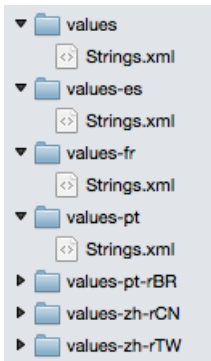
values/Strings.xml

```
<resources>
  <string name="app_name">TaskyL10n</string>
  <string name="taskadd">Add Task</string>
  <string name="taskname">Name</string>
  <string name="tasknotes">Notes</string>
  <string name="taskdone">Done</string>
  <string name="taskcancel">Cancel</string>
</resources>
```

西班牙语目录值 **es** 包含具有相同名称的文件 (**Strings.xml**), 其中包含翻译:

values-es/Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TaskyLeon</string>
    <string name="taskadd">agregar tarea</string>
    <string name="taskname">Nombre</string>
    <string name="tasknotes">Notas</string>
    <string name="taskdone">Completo</string>
    <string name="taskcancel">Cancelar</string>
</resources>
```



与字符串文件设置后，可以在布局和代码中引用已翻译的值。

AXML 布局文件

若要引用布局文件中的已本地化的字符串，使用 `@string/id` 语法。此 XML 代码段的示例所示 `text` 属性所设置的本地化资源 Id（已省略某些其他属性）：

```
<TextView
    android:id="@+id/NameLabel"
    android:text="@string/taskname"
    ... />
<CheckBox
    android:id="@+id/chkDone"
    android:text="@string/taskdone"
    ... />
```

GetText 方法

若要检索已翻译的字符串在代码中，使用 `getText` 方法并传入的资源 ID：

```
var cancelText = Resources.getText (Resource.String.taskcancel);
```

数量字符串

Android 字符串资源，您还可以创建 *数量字符串* 这使翻译人员提供不同的翻译，对于不同的数量，例如：

- "没有剩余 1 个任务。"
- "有 2 任务仍将执行操作。"

（而不是泛型"有 n 个任务左"）。

在 Strings.xml


```
<plurals name="numberOfTasks">
  <!--
    As a developer, you should always supply "one" and "other"
    strings. Your translators will know which strings are actually
    needed for their language.
  -->
  <item quantity="one">There is %d task left.</item>
  <item quantity="other">There are %d tasks still to do.</item>
</plurals>
```

若要呈现完整字符串使用 `GetQuantityString` 方法, 传递的资源 ID 以及要显示的值 (其中传递两次)。Android 使用第二个参数来确定 `quantity` 要使用的字符串, 第三个参数是实际代入 (两者都是必需的) 的字符串值。

```
var translated = Resources.GetQuantityString (
    Resource.Plurals.numberOfTasks, taskcount, taskcount);`
```

有效 `quantity` 开关是:

- 零
- one
- 两个
- 几个
- many
- 其他

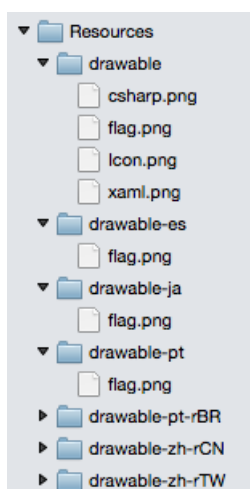
中的更多详细信息中所述 [Android 文档](#)。如果给定的语言不需要特殊处理时, 那些 `quantity` 字符串将被忽略 (例如, 英语仅使用 `one` 并 `other`; 指定 `zero` 字符串会产生任何效果, 将不使用它)。

图像

本地化的映像遵循相同的规则作为字符串文件: 应用程序中引用的所有图像应都置于 **drawable** 目录, 因此没有回退。

特定于区域设置的映像应随后可以放置在限定可绘制文件夹等 **drawable-es** 或 **drawable-ja** (还可以添加 dpi 说明符)。

在此屏幕截图, 四个映像保存在 **drawable** 目录中, 但只有一个分区 **flag.png**, 已本地化其他目录中的副本。



其他资源类型

您还可以提供其他类型的替代方法, 包括布局、动画和原始文件的特定于语言的资源。这意味着, 您可以提供特定的屏幕布局的一个或多个目标语言, 例如你可以创建一个布局专门为德语, 允许很长的文本标签。

Android 4.2 引入了对支持 [从右到左 \(RTL\) 语言](#) 如果你设置的应用程序设置 `android:supportsRtl="true"`。资源限

定符 "ldrtl" 可以包含在要包含专为从右向左显示的自定义布局目录名称。

有关资源目录命名和故障回复的详细信息，请参阅 Android 文档[提供替代资源](#)。

应用名称

应用程序名称很容易通过使用本地化 `@string/id` 中为 `MainLauncher` 活动：

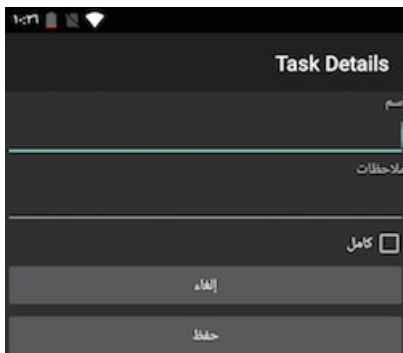
```
[Activity (Label = "@string/app_name", MainLauncher = true, Icon="@drawable/launcher",  
    ConfigurationChanges = ConfigChanges.Orientation | ConfigChanges.Locale)]
```

右到左 (RTL) 语言

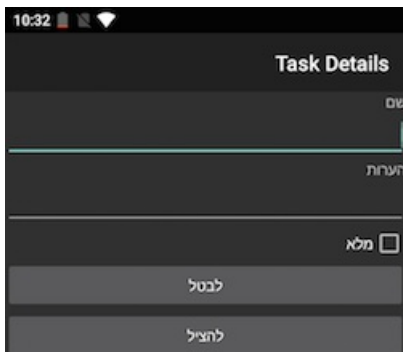
Android 4.2 及更高版本提供完全支持对于 RTL 布局，详见[本机 RTL 支持博客](#)。

使用 Android 4.2 (API 级别 17) 时和更高版本，可以使用指定值的对齐方式 `start` 并 `end` 而不是 `left` 并 `right` (例如 `android:paddingStart`)。此外，还有新的 Api，如 `LayoutDirection`，`TextDirection`，和 `TextAlignment` 以帮助
您构建适应屏幕从右向左读取器。

以下屏幕截图显示[本地化Tasky示例](#)在阿拉伯语中：



下一步的屏幕截图显示[本地化Tasky示例](#)希伯来语版本：



使用本地化的文本按 RTL **Strings.xml** LTR 文本的方式相同的文件。

正在测试

请务必全面测试的默认区域设置。如果出于某种原因（即它们是缺少）无法加载默认资源，你的应用程序将崩溃。

模拟器测试

请参阅 Google [Android 模拟器上测试](#)部分，了解如何将模拟器设置为使用 ADB 命令程序的特定区域设置的说明。

```
adb shell setprop persist.sys.locale fr-CA;stop;sleep 5;start
```

设备测试

若要在设备上测试，更改所使用的语言设置应用。提示：记下图标和菜单项的位置，以便可以还原到原始设置的

语言。

总结

本文介绍如何本地化使用内置的资源处理的 Android 应用程序的基础知识。针对 iOS、Android 和跨平台（包括 Xamarin.Forms）中的应用程序，可以了解更多有关 i18n 和 L10n[此跨平台指南](#)。

相关链接

- [Tasky（在代码中已本地化）（示例）](#)
- [Android 资源与本地化](#)
- [跨平台本地化概述](#)
- [Xamarin.Forms 本地化](#)
- [iOS 本地化](#)

在 Xamarin.Android 中的权限

2018/11/13 • [Edit Online](#)

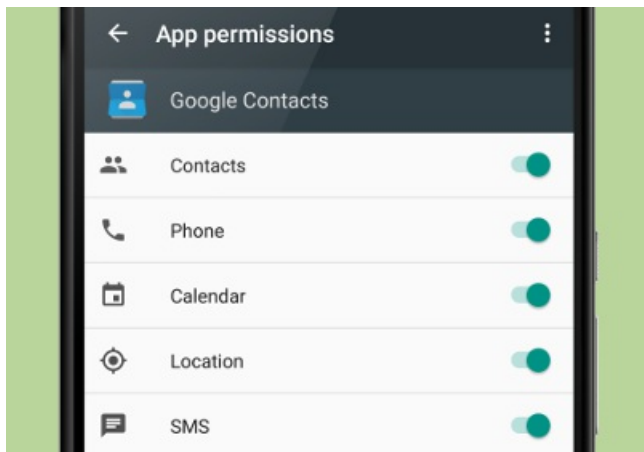
概述

在其自己的沙盒中运行的 android 应用程序和安全原因不能访问某些系统资源或硬件设备上。它可能会使用这些资源之前, 用户必须显式授予应用权限。例如, 应用程序不能从用户访问而无需显式权限的设备上 GPS。

Android 将引发 `Java.Lang.SecurityException` 如果应用程序尝试访问受保护的资源而无需权限。

权限声明中 **AndroidManifest.xml** 由应用开发的应用程序开发。Android 有两个不同的工作流以获取这些权限的用户的同意:

- 对于针对 Android 5.1 (API 级别 22) 或更低版本的应用程序, 安装了应用时, 将出现的权限请求。如果用户未授予的权限, 将不安装该应用程序。安装应用后, 没有办法通过卸载应用程序撤销权限除外。
- 从 Android 6.0 (API 级别 23) 开始, 用户提供更好地控制权限;在可授予或撤消的权限, 只要在设备上安装应用。此屏幕截图显示的权限设置的 Google 联系人应用。它列出了不同的权限, 并允许用户启用或禁用的权限:



Android 应用程序必须在运行时以查看它们是否具有访问受保护的资源的权限检查。如果应用不具有权限, 则它必须进行请求使用新的 Api 提供的 Android SDK 的用户授予的权限。权限划分为两个类别:

- **正常的权限**-这些是这会带来很少的安全风险, 用户的安全或隐私的权限。在安装时, android 6.0 将自动授予正常的权限。请参阅 Android 文档[普通权限的完整列表](#)。
- **危险权限**-与普通权限相比危险的权限是指那些保护用户的安全或隐私。这些必须显式授予用户。发送或接收短信是操作的需要非常危险的权限的示例。

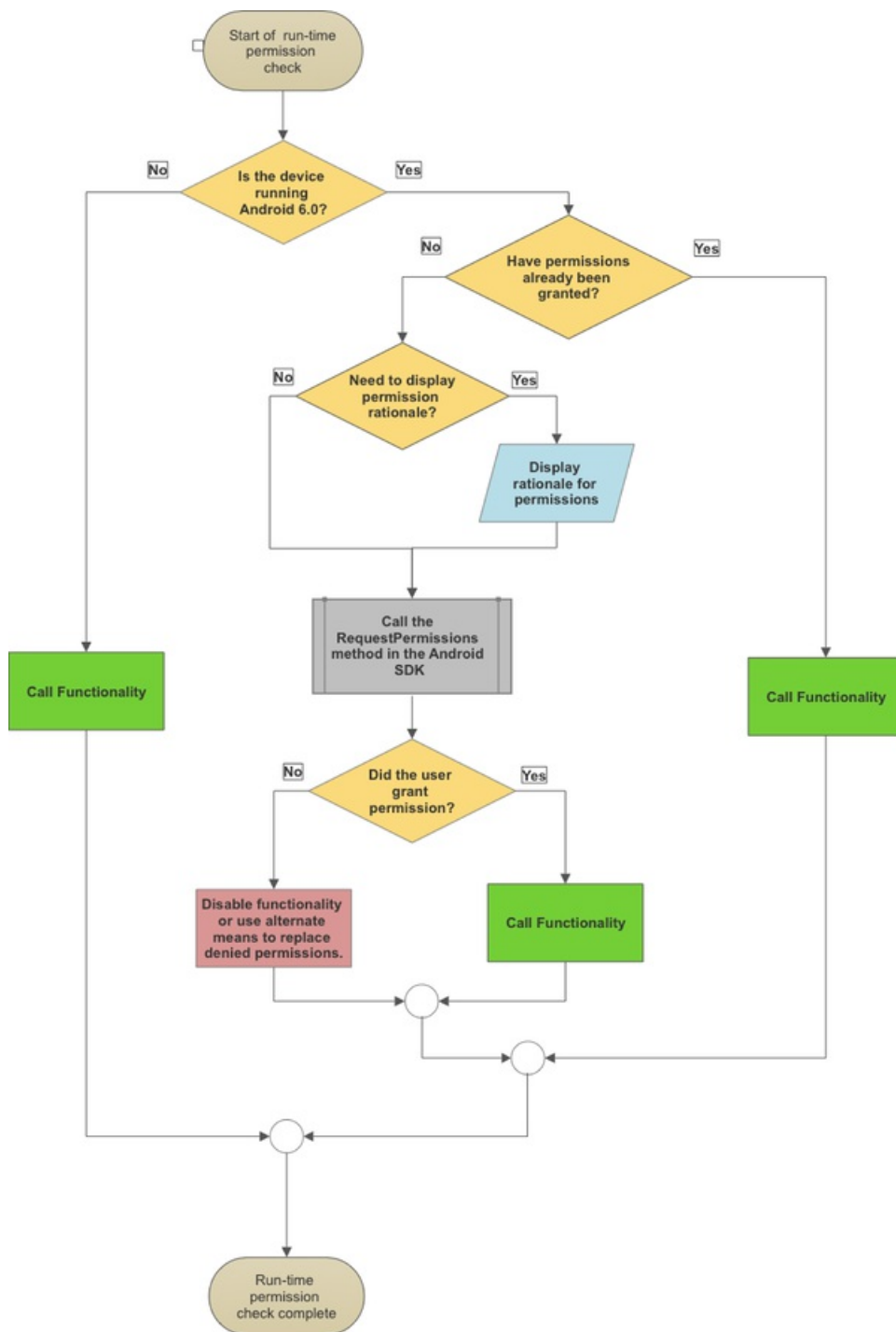
IMPORTANT

随着时间的推移可能会更改权限所属的类别。有可能, 此分类为“正常”权限就是将权限提升的将来 API 级别非常危险的权限。

危险权限进一步细分为多个 [权限组](#)。权限组将保存在逻辑上关联的权限。时用户授予权限的权限组的一个成员, Android 会自动为该组的所有成员授予权限。例如, `STORAGE` 权限组保留着两 `WRITE_EXTERNAL_STORAGE` 和 `READ_EXTERNAL_STORAGE` 权限。如果用户授予权限 `READ_EXTERNAL_STORAGE`, 则 `WRITE_EXTERNAL_STORAGE` 同时自动授予权限。

请求一个或多个权限之前, 它是最佳做法提供关于为何应用需要请求权限前的权限了合理依据。一旦用户理解基本原理, 该应用程序可从用户请求权限。通过了解基本原理, 用户可以做出明智的决策, 如果他们想要授予权限, 并了解负面影响, 如果它们不匹配。

检查和请求权限的整个工作流程被称为_运行时权限_检查, 并可以汇总成以下关系图:



Android 支持库 backports 某些较旧版本的 Android 权限的新 Api。这些向后移植 Api 将自动检查设备上的 Android 版本, 因此不需要执行 API 级别检查每个时间。

本文档将讨论如何将权限添加到 Xamarin.Android 应用程序以及如何面向 Android 6.0 (API 级别 23) 的应用, 或更高版本应该执行运行时权限检查。

NOTE

就可以对硬件的权限可能会影响通过 Google Play 筛选应用的方式。例如, 如果应用需要用于相机的权限, 然后 Google Play 不会显示应用不具有安装摄像头的设备上 Google Play 商店中。

要求

强烈建议包括的 Xamarin.Android 项目 [Xamarin.Android.Support.Compat](#) NuGet 包。而且无需不断地与旧版本

的 Android, 提供一种常见的特定 Api 接口。此包将向后移植权限检查应用程序运行的 Android 的版本。

请求系统权限

使用 Android 权限的第一步是声明的权限在 Android 清单文件。这必须完成而不考虑 API 级别应用所面向。

面向 Android 6.0 或更高版本不能假定, 因为用户授予在过去, 在某个时间点的权限, 权限将为有效的下一步的时间的应用程序。面向 Android 6.0 的应用必须始终执行运行时权限检查。面向 Android 5.1 或更低的应用, 不需要执行运行时权限检查。

NOTE

应用程序应只请求所需的权限。

声明在清单中的权限

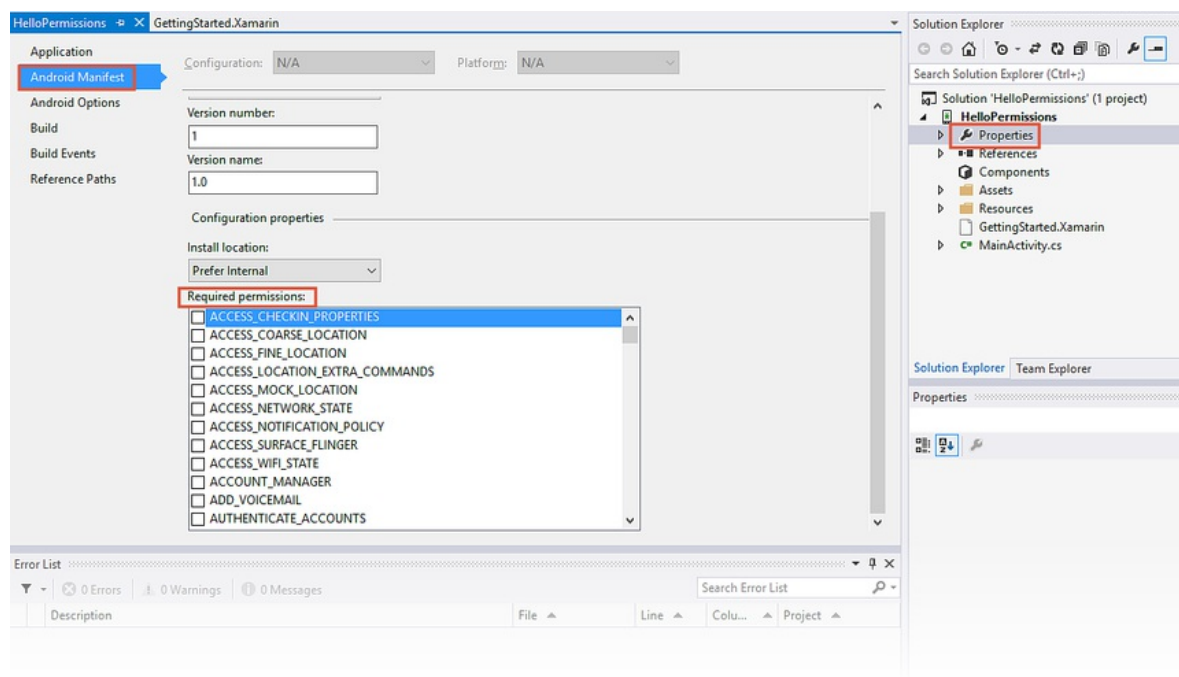
权限添加到**AndroidManifest.xml**与 `uses-permission` 元素。例如, 如果应用程序是查找该设备的位置, 它需要正常运行, 并且课程位置权限。以下两个元素添加到清单中:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

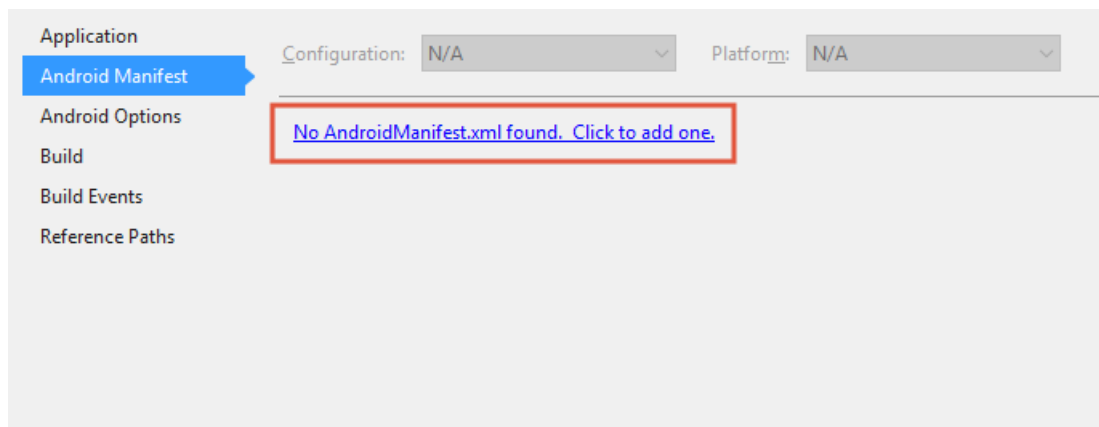
- [Visual Studio](#)
- [Visual Studio for Mac](#)

它是可以声明使用 Visual Studio 中内置的工具支持的权限:

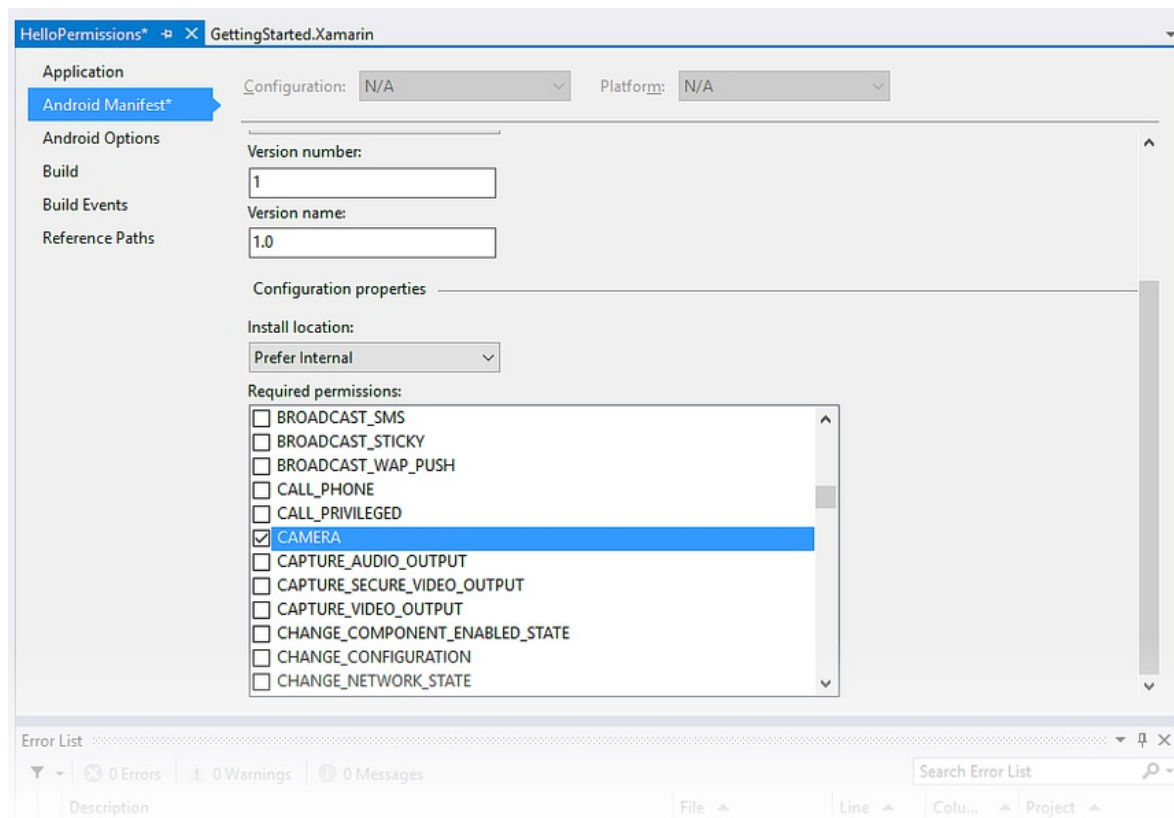
1. 双击**属性**中**解决方案资源管理器**, 然后选择**Android 清单**属性窗口中的选项卡:



2. 如果应用程序尚不包含 **AndroidManifest.xml**, 单击**找到任何 AndroidManifest.xml**. 单击此项可添加一个, 如下所示:



3. 选择你的应用程序需要从任何权限所需的权限列出并保存：



Xamarin.Android 将自动添加某些权限在生成时的调试版本。这将使调试更轻松的应用程序。具体而言，两个值得注意权限是 `INTERNET` 和 `READ_EXTERNAL_STORAGE`。这些自动设置的权限不会在中启用所需的权限列表。发行版本，但是，请使用仅中显式设置的权限所需的权限列表。

对于面向 Android 5.1 (API 级别 22) 或更低版本的应用程序，没有任何更多，需要完成。将运行 Android 6.0 (API 23 级别 23) 或更高版本的应用程序应继续进行到下一节介绍了如何执行权限检查的运行的时。

在 Android 6.0 运行时权限检查

`ContextCompat.CheckSelfPermission` (适用于 Android 支持库) 的方法用于检查是否已授予特定权限。此方法将返回 `Android.Content.PM.Permission` 枚举具有两个值之一：

- `Permission.Granted` – 已授予指定的权限。
- `Permission.Denied` – 尚未授予指定的权限。

此代码片段示范了如何在活动中的照相机权限检查：

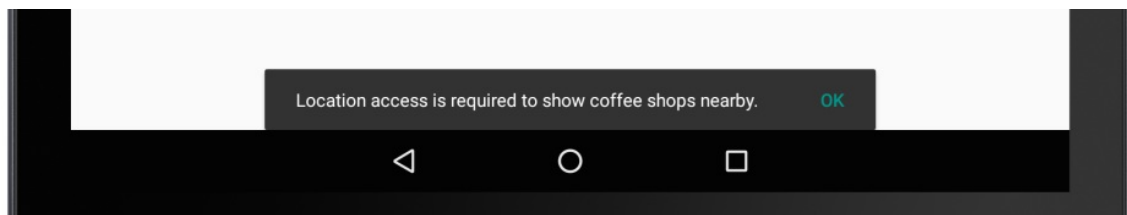
```

if (ContextCompat.CheckSelfPermission(this, Manifest.Permission.Camera) == (int)Permission.Granted)
{
    // We have permission, go ahead and use the camera.
}
else
{
    // Camera permission is not granted. If necessary display rationale & request.
}

```

它是最佳做法以告知用户为何权限是应用程序，以便可以进行明智的决策要授予的权限。此示例将应用采用的照片和地域标记它们。它是明确告知用户照相机权限是必需的但可能不会清除应用也需要该设备的位置的原因。要弄清楚为何应显示一条消息，以帮助用户了解为什么说的位置权限是比较理想和照相机权限是必需。

`ActivityCompat.ShouldShowRequestPermissionRationale` 方法用于确定是否应为用户显示基本原理。此方法将返回 `true` 如果应显示给定的权限的基本原理。此屏幕截图显示解释为什么应用程序需要知道设备的位置的应用程序会显示 Snackbar 的示例：



如果用户授予权限，

`ActivityCompat.RequestPermissions(Activity activity, String[] permissions, int requestCode)` 应调用方法。此方法需要以下参数：

- **活动**—这是请求的权限，并且是通过 Android 结果的通知的活动。
- **权限**—正在请求权限的列表。
- **requestCode**—一个整数值，用于匹配对的权限请求的结果 `RequestPermissions` 调用。此值应大于零。

此代码片段是讨论了两种方法的示例。首先，进行检查以确定是否应显示权限基本原理。如果要弄清楚为何将显示，则 Snackbar 显示与基本原理。如果用户单击确定中 Snackbar，然后应用将请求的权限。如果用户不接受基本原理，然后该应用程序应无法继续以请求权限。如果未显示基本原理，该活动将请求的权限：

```

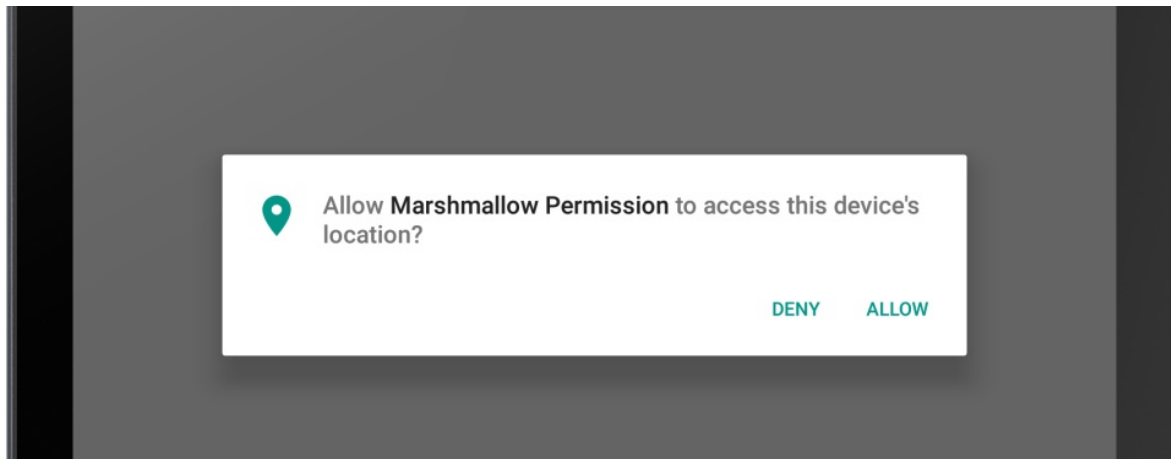
if (ActivityCompat.ShouldShowRequestPermissionRationale(this, Manifest.Permission.AccessFineLocation))
{
    // Provide an additional rationale to the user if the permission was not granted
    // and the user would benefit from additional context for the use of the permission.
    // For example if the user has previously denied the permission.
    Log.Info(TAG, "Displaying camera permission rationale to provide additional context.");

    var requiredPermissions = new String[] { Manifest.Permission.AccessFineLocation };
    Snackbar.Make(layout,
        Resource.String.permission_location_rationale,
        Snackbar.LengthIndefinite)
        .SetAction(Resource.String.ok,
            new Action<View>(delegate(View obj) {
                ActivityCompat.RequestPermissions(this, requiredPermissions, REQUEST_LOCATION);
            })
        )
        .Show();
}
else
{
    ActivityCompat.RequestPermissions(this, new String[] { Manifest.Permission.Camera }, REQUEST_LOCATION);
}

```

`RequestPermission` 可以调用，即使用户已授予的权限。后续调用不是必需的但用户提供机会来确认（或吊销）权

限。当 `RequestPermission` 是名为，控件将转移到操作系统，将显示一个用户界面的接受权限：



用户已完成，Android 将返回结果的活动的回调方法，通过 `OnRequestPermissionsResult`。此方法是接口的一部分 `ActivityCompat.IOnRequestPermissionsResultCallback` 活动必须实现的。此接口具有单个方法 `OnRequestPermissionsResult`，这将调用由 Android 通知的活动的用户的选择。如果用户已授予的权限，该应用程序可以继续操作并使用受保护的资源。举例说明如何实现 `OnRequestPermissionsResult` 如下所示：

```
public override void OnRequestPermissionsResult(int requestCode, string[] permissions, Permission[]
grantResults)
{
    if (requestCode == REQUEST_LOCATION)
    {
        // Received permission result for camera permission.
        Log.Info(TAG, "Received response for Location permission request.");

        // Check if the only required permission has been granted
        if ((grantResults.Length == 1) && (grantResults[0] == Permission.Granted)) {
            // Location permission has been granted, okay to retrieve the location of the device.
            Log.Info(TAG, "Location permission has now been granted.");
            Snackbar.Make(layout, Resource.String.permission_available_camera, Snackbar.LengthShort).Show();
        }
        else
        {
            Log.Info(TAG, "Location permission was NOT granted.");
            Snackbar.Make(layout, Resource.String.permissions_not_granted, Snackbar.LengthShort).Show();
        }
    }
    else
    {
        base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

总结

本指南介绍如何添加和查看有在 Android 设备的权限。权限旧的 Android 应用（API 级别 < 23）和新的 Android 应用程序之间的工作方式的差异（API 级别 > 22）。它介绍了如何在 Android 6.0 中执行运行时权限检查。

相关链接

- [正常的权限的列表](#)
- [运行时权限示例应用](#)
- [在 Xamarin.Android 中的处理权限](#)

图形和动画

2018/10/26 • [Edit Online](#)

Android 提供了非常丰富、不同的框架支持二维图形和动画。本主题介绍这些框架，并介绍了如何创建自定义图形和动画使用 Xamarin.Android 应用程序中。

概述

尽管是传统意义上的有限能力的设备上运行，最高的评级移动应用程序通常具有复杂用户体验 (UX)，具有高质量的图形和动画提供直观的响应式动态外观。在移动应用程序获取的详细信息并更复杂，用户已经开始时会出现的情况越来越多的应用程序。

幸运的是对我们而言，新式移动平台都具有用于保留的易用性的同时创建复杂的动画和自定义图形非常功能强大的框架。这使开发人员能够添加不费吹灰之力丰富的交互功能。

在 Android 中的 UI API 框架大致分为两类：图形和动画。

图形进一步分为不同的方法进行二维和三维图形。3D 图形是可通过许多内置的框架，例如 OpenGL ES（移动特定版本的 OpenGL）和 MonoGame（一个跨平台工具包与 XNA 工具包兼容）等第三方框架。虽然 3D 图形不在本文的讨论范围内，我们将查看内置的 2D 绘制技术。

Android 提供了两个不同的 API 的创建 2D 图形。一个是高级别的声明性方法和其他编程的低级别 API：

- **可绘制资源**—这些用于在 XML 文件中嵌入绘制指令（更通常）或以编程方式创建自定义图形。可绘制资源通常定义为包含说明或适用于 Android 呈现 2D 图形操作的 XML 文件。
- **画布**—这是一个较低级别 API，包括直接在基础位图上绘制。它提供了非常精细地控制显示的内容。

除了这些 2D 图形技术，Android 还提供了多种不同的方式来创建动画：

- **可绘制动画**—Android 还支持名为帧的帧动画 *可绘制动画*。这是最简单的动画 API。Android 按顺序加载，并显示可绘制资源（类似于卡通）的序列中。
- **查看动画**—*视图动画*是原始动画 API 的 Android 中并且在所有版本的 Android 中可用。此 API 的局限性在于它只适用于视图对象，并仅可以对这些视图执行简单转换。视图动画通常定义在 XML 文件中找到 `/Resources/anim` 文件夹。
- **属性动画**—Android 3.0 引入了一组新的动画 API 的称为 *属性动画*。这些新的 API 引入了可用于对任何对象的属性进行动画处理，而不仅仅是查看对象的可扩展性和灵活性系统。这种灵活性允许动画封装在不同的类，可让代码共享变得更容易。

视图动画是更适合应用程序必须支持较旧预 Android 3.0 API（API 级别为 11）。否则应用程序应使用较新属性动画 API 的上面提到的原因。

所有这些框架都是可行选项，但是在可能的情况下，首选项应授予给属性的动画，因为它是一个更灵活的 API，可使用。属性动画允许动画逻辑封装在不同的类，可以使代码变得更容易共享并简化了代码维护。

可访问性

图形和动画帮助使 Android 应用程序更具吸引力和使用；乐趣但是，务必要记住一些交互发生阅读器，备用输入设备通过或协助缩放。此外，一些交互可能没有音频功能。

如果它们在设计时考虑到中的辅助功能应用是在这些情况下更易于使用：提供提示和导航用户界面的帮助，并确保没有文本内容或 UI 的图形化元素说明。

请参阅[Google 的可访问性指南](#)有关如何使用 Android 的辅助功能 Api 的详细信息。

2D 图形

可绘制资源是 Android 应用程序中的一种常用方法。因为与其他资源，可绘制资源是声明性-在 XML 文件中定义。此方法允许从资源的代码完全分离。这可以简化开发和维护，因为不需要更改代码以更新或更改 Android 应用程序中的图形。但是，可绘制资源可用于许多简单和常见图形要求，而它们缺乏 power 和控制的画布 API。

另一个方法，使用[画布](#)对象，则非常类似于其他传统的 API 框架，如 System.Drawing 或 iOS 的核心绘图。使用画布对象提供了创建如何 2D 图形的最大控制。它是适用于以下情况，可绘制资源不起作用，或将很难使用。例如，可能需要绘制自定义滑块控件更改其外观将根据滑块的值与相关的计算。

让我们首先检查可绘制资源。它们是更简单且包括最常见的自定义绘制用例。

可绘制资源

在目录中的 XML 文件中定义可绘制资源 `/Resources/drawable`。与嵌入 PNG 或 JPEG 的不需要提供特定于密度的可绘制资源版本。在运行时，Android 应用程序将加载这些资源，并使用包含在这些 XML 文件中的说明创建 2D 图形。Android 定义多种不同类型的可绘制资源：

- [ShapeDrawable](#) -这是一个可绘制对象的绘制基本几何形状，并应用一组有限的该形状上的图形效果。它们是等自定义按钮，或设置背景的 TextViews 非常有用。我们会举例说明如何使用 `ShapeDrawable` 这篇文章中更高版本。
- [StateListDrawable](#) -这是一个可绘制资源，将更改基于状态的小组件/控件的外观。例如，一个按钮可能会更改其外观，具体取决于是否按下或未。
- [LayerDrawable](#) -将层叠在另一个的多个其他绘图的此可绘制资源。举例[LayerDrawable](#)以下屏幕截图所示：



- [TransitionDrawable](#) -这是[LayerDrawable](#)但有一处不同。一个[TransitionDrawable](#)能够进行动画处理另一个显示基础上的一个层。
- [LevelListDrawable](#) -这是非常类似于[StateListDrawable](#)，它将显示基于特定条件的映像。但是，与不同[StateListDrawable](#)，则[LevelListDrawable](#)显示基于整数值的映像。举例[LevelListDrawable](#)是显示 WiFi 信号的强度。作为 WiFi 信号更改的强度，可绘制显示将相应地更改。
- [ScaleDrawable/ClipDrawable](#) -这些绘图如其名称所示，提供缩放和剪辑的功能。[ScaleDrawable](#)将扩展另一个可绘制，while [ClipDrawable](#)将剪辑另一个 Drawable。
- [InsetDrawable](#) -此可绘制将应用嵌入另一个可绘制资源两端。使用视图需要小于该视图的实际边界的背景。
- XML [BitmapDrawable](#) -此文件是一组的说明进行操作，在 XML 中，若要在实际位图上执行。Android 可以执行某些操作是拼贴、抖动，和抗锯齿。这很常见用途之一是在布局的背景磁贴位图。

可绘制示例

让我们看看如何 2D 图形使用以下工具创建一个快速示例 `ShapeDrawable`。一个 `ShapeDrawable` 可以定义四个基本形状之一：矩形、椭圆、线条和环。还有可能要应用基本效果，如渐变、颜色和大小。以下 XML 是 `ShapeDrawable` 中可能找到[AnimationsDemo](#)配套项目（文件中 `Resources/drawable/shape_rounded_blue_rect.xml`）。它定义一个具有紫色渐变背景的矩形和圆形角：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" android:shape="rectangle">
<!-- Specify a gradient for the background -->
<gradient android:angle="45"
    android:startColor="#55000066"
    android:centerColor="#00000000"
    android:endColor="#00000000"
    android:centerX="0.75" />

<padding android:left="5dp"
    android:right="5dp"
    android:top="5dp"
    android:bottom="5dp" />

<corners android:topLeftRadius="10dp"
    android:topRightRadius="10dp"
    android:bottomLeftRadius="10dp"
    android:bottomRightRadius="10dp" />
</shape>
```

我们可以引用此可绘制资源以声明方式中的布局和其他 Drawable 以下 XML 中所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#33000000">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="@drawable/shape_rounded_blue_rect"
        android:text="@string/message_shapedrawable" />
</RelativeLayout>
```

此外可以以编程方式应用可绘制资源。下面的代码段演示如何以编程方式设置的一个 TextView 背景：

```
TextView tv = findViewById<TextView>(Resource.Id.shapeDrawableTextView);
tv.setBackgroundResource(Resource.Drawable.shape_rounded_blue_rect);
```

若要查看此将类似的内容，请运行 *AnimationsDemo* 项目，然后从主菜单中选择形状可绘制的项。我们应看到类似于以下屏幕截图的内容：



有关 XML 元素和语法的可绘制资源的更多详细信息，请查阅[Google 文档](#)。

使用画布绘图 API

绘图功能非常强大，但有其局限性。某些操作是不可能或非常复杂（例如：将筛选器应用于通过设备相机拍摄的图片）。它会很难通过使用可绘制资源应用红眼。相反，画布 API 允许应用程序能够非常精细地控制要有选择地更改图片的特定部分中的颜色。

一种常用于画布的是 [画图](#) 类。此类包含有关如何绘制颜色和样式信息。它用于提供内容，此类的颜色和透明度。

使用画布 API *绘画器的模型* 绘制 2D 图形。操作将应用在彼此的连续层中。每个操作将覆盖基础位图的某些区域。区域重叠先前绘制的区域，新画图将部分或完全遮盖旧。这是 System.Drawing 和 iOS 的核心图形等许多其他图形 Api 的工作原理的相同方法。

有两种方法来获取 `Canvas` 对象。第一种方法涉及到定义 [位图](#) 对象，然后实例化 `Canvas` 对象与之。例如，下面的代码段创建一个新的画布与基础位图：

```
Bitmap bitmap = Bitmap.CreateBitmap(100, 100, Bitmap.Config.Argb8888);
Canvas canvas = new Canvas(b);
```

若要获取的其他方式 `Canvas` 对象是通过 `OnDraw` 提供的回调方法 [视图](#) 基类。Android 调用此方法，在决定视图需要自我绘制，并且传入 `Canvas` 要使用的视图对象。

画布类公开方法，以编程方式提供的绘制说明。例如：

- `Canvas.DrawPaint` – 用指定绘制填充整个画布的位图。
- `Canvas.DrawPath` – 绘制指定的几何形状使用指定的画图。
- `Canvas.DrawText` – 指定实心圆画布上绘制文本。在位置绘制文本 `x,y`。

使用画布 API 绘图

让我们查看操作中的画布 API 的示例。下面的代码段显示了如何绘制一个视图：

```
public class MyView : View
{
    protected override void OnDraw(Canvas canvas)
    {
        base.OnDraw(canvas);
        Paint green = new Paint {
            AntiAlias = true,
            Color = Color.Rgb(0x99, 0xcc, 0),
        };
        green.SetStyle(Paint.Style.FillAndStroke);

        Paint red = new Paint {
            AntiAlias = true,
            Color = Color.Rgb(0xff, 0x44, 0x44)
        };
        red.SetStyle(Paint.Style.FillAndStroke);

        float middle = canvas.Width * 0.25f;
        canvas.DrawPaint(red);
        canvas.DrawRect(0, 0, middle, canvas.Height, green);
    }
}
```

上面这段代码首先创建红色画图和绿色绘制对象。它使用红色，填充画布的内容，然后指示画布以绘制为画布宽度的 25% 的绿色矩形。此示例所示通过 `AnimationsDemo` 包含在本文的源代码的项目。通过启动应用程序并从主菜单中选择绘制项，我们应类似于以下屏幕：

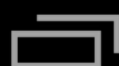
-8°



13:58



Drawing



动画

用户喜欢在其应用程序中移动的内容。动画是改善应用程序的用户体验，并帮助其脱颖而出的好办法。最佳的动画是用户未注意到，因为他们觉得自然的。Android 提供以下三个 API 用于动画：

- **查看动画**—这是原始 API。这些动画绑定到特定的视图，并可以对视图的内容执行简单转换。由于它的简单性，此 API 仍可用于操作如 alpha 动画、旋转等等。
- **属性动画**—Android 3.0 中引入了属性动画。它们使应用程序能够对几乎所有内容进行动画处理。可以使用属性动画若要更改的任何属性的任何对象，即使该对象不在屏幕上可见。
- **可绘制动画**—这用于将应用非常简单的动画的特殊可绘制资源影响布局。

一般情况下，属性动画是首选的系统，以用作更为灵活，提供了更多的功能。

视图动画

视图动画限制为视图，并且只能对值，例如开始和结束点、大小、旋转和透明度执行动画。这些类型的动画通常称为 *tween 动画*。可以通过两种方式定义视图动画—以编程方式在代码或通过使用 XML 文件。XML 文件是声明视图动画的首选的方法，因为它们是更具可读性且更易于维护。

动画 XML 文件将存储在 `/Resources/anim` Xamarin.Android 项目的目录。此文件必须具有以下元素之一作为根元素：

- `alpha` — 淡入或淡出动画。
- `rotate` — 旋转动画。
- `scale` — 调整大小的动画。
- `translate` — 水平和/或垂直移动。
- `set` — 可能包含一个或多个其他动画元素的容器。

默认情况下，将同时应用到 XML 文件中的所有动画。若要使动画按顺序发生，请设置 `android:startOffset` 上面定义的元素之一上的属性。

可以通过使用影响中动画的行为更改的速率 *内插器*。内插器实现自定义动画加速、重复出现，或 decelerated。Android 框架提供的多个内的插器默认情况下，例如（但不是限于）：

- `AccelerateInterpolator` / `DecelerateInterpolator` — 这些内插器增加或减少中动画的行为更改的速率。
- `BounceInterpolator` — 更改退回的末尾。
- `LinearInterpolator` — 更改的速率保持不变。

以下 XML 显示了将合并这些元素的一些动画文件的示例：

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android=http://schemas.android.com/apk/res/android
    android:shareInterpolator="false">

    <scale android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillEnabled="true"
        android:fillAfter="false"
        android:duration="700" />

    <set android:interpolator="@android:anim/accelerate_interpolator">
        <scale android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:fillEnabled="true"
            android:fillBefore="false"
            android:fillAfter="true"
            android:startOffset="700"
            android:duration="400" />

        <rotate android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:fillEnabled="true"
            android:fillBefore="false"
            android:fillAfter="true"
            android:startOffset="700"
            android:duration="400" />

    </set>
</set>

```

此动画将所有动画同时执行。第一个缩放动画将水平拉伸图像和垂直收缩，然后将映像将同时逆时针旋转 45 度和收缩，请从屏幕消失。

动画可以以编程方式应用于视图都动画，然后将它应用到一个视图。Android 提供的帮助器类

`Android.Views.Animations.AnimationUtils`，将放大量动画资源并返回的一个实例

`Android.Views.Animations.Animation`。此对象通过调用应用于视图 `StartAnimation` 并传递 `Animation` 对象。下面的代码段显示了此示例：

```

Animation myAnimation = AnimationUtils.LoadAnimation(Resource.Animation.MyAnimation);
ImageView myImage = FindViewById<ImageView>(Resource.Id.imageView1);
myImage.StartAnimation(myAnimation);

```

现在，我们已基本了解工作方式查看动画，允许将移到属性动画。

属性动画

属性动画制作人员是 Android 3.0 中引入了一个新的 API。它们提供一个更具扩展性 API，可以使用任何对象上的任何属性进行动画处理。

实例创建所有属性动画动画器子类。应用程序不直接使用此类，而是使用它的子类之一：

- `ValueAnimator` – 此类是在整个属性动画 API 中最重要的类。它会计算需要更改的属性的值。 `ViewAnimator`

不会直接更新这些值; 相反, 它会引发可用于更新经过动画处理的对象的事件。

- **ObjectAnimator** –此类是一个的子类 `ValueAnimator`。它旨在通过接受的目标对象和要更新属性来简化对象制作动画的过程。
- **AnimationSet** –此类负责协调动画到另一个运行在关系中的方式。动画它们之间可能会运行同时, 按顺序, 或指定的延迟。

评估器是动画制作人员用于在动画期间计算新值的特殊类。默认情况下, Android 提供了下列计算器:

- **IntEvaluator** –计算整数属性的值。
- **FloatEvaluator** –计算 float 属性值。
- **ArgbEvaluator** –计算的颜色属性的值。

如果要进行动画处理的属性不是 `float`, `int` 或颜色, 应用程序可能会创建其自己的计算器, 通过实现 `ITypeEvaluator` 接口。(实现自定义评估者是超出了本主题的范围。)

使用 ValueAnimator

有两个部分的任何动画: 计算动画的值, 然后某些对象上的属性上设置这些值。`ValueAnimator` 仅计算值, 但它将不会运行对象直接。相反, 在动画生命周期过程中将调用的事件处理程序将更新对象。此设计允许多个属性以从一个经过动画处理的值进行更新。

获取的实例 `ValueAnimator` 通过调用以下工厂方法之一:

- `ValueAnimator.OfInt`
- `ValueAnimator.OfFloat`
- `ValueAnimator.OfObject`

完成后执行的操作, `ValueAnimator` 实例都必须具有其持续时间设置, 然后启动它。下面的示例演示如何设置一个介于 0 和 1 之间的动画效果的 1000 毫秒范围内:

```
ValueAnimator animator = ValueAnimator.OfInt(0, 100);
animator.SetDuration(1000);
animator.Start();
```

本身, 上面的代码段不是很有用, 但动画器将运行, 但并没有针对更新后的值。`Animator` 类将引发时决定是通知的新值的侦听器所需的更新事件。应用程序可能会提供一个事件处理程序以响应此事件, 如下面的代码段中所示:

```
MyCustomObject myObj = new MyCustomObject();
myObj.SomeIntegerValue = -1;

animator.Update += (object sender, ValueAnimator.AnimatorUpdateEventArgs e) =>
{
    int newValue = (int) e.Animation.AnimatedValue;
    // Apply this new value to the object being animated.
    myObj.SomeIntegerValue = newValue;
};
```

现在, 我们已了解 `ValueAnimator`, 详细了解如何允许 `ObjectAnimator`。

使用 ObjectAnimator

ObjectAnimator 是一个的子类 `ViewAnimator` 组合的计时引擎和值计算 `ValueAnimator` 与绑定事件处理程序所需的逻辑。`ValueAnimator` 要求应用程序以显式关联的事件处理程序 – `ObjectAnimator` 将为我们处理的此步骤。

有关 API `ObjectAnimator` 非常类似于用于 API `ViewAnimator`, 但需要您提供对象和要更新的属性的名称。下面的示例演示使用的示例 `ObjectAnimator`:

```
MyCustomObject myObj = new MyCustomObject();
myObj.SomeIntegerValue = -1;

ObjectAnimator animator = ObjectAnimator.OfFloat(myObj, "SomeIntegerValue", 0, 100);
animator.SetDuration(1000);
animator.Start();
```

您可以看到从上一代码段中，`ObjectAnimator` 可以减少并简化对对象进行动画处理所需的代码。

可绘制动画

最后一个动画 API 是可绘制动画 API。可绘制动画在它们加载一系列可绘制资源—并将其按顺序显示类似于翻转 it 卡通。

可绘制资源具有的 XML 文件中定义 `<animation-list>` 元素作为根元素和一系列 `<item>` 定义每个帧动画中的元素。此 XML 文件存储在 `/Resource/drawable` 的应用程序文件夹。以下 XML 是可绘制动画的示例：

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/asteroid01" android:duration="100" />
  <item android:drawable="@drawable/asteroid02" android:duration="100" />
  <item android:drawable="@drawable/asteroid03" android:duration="100" />
  <item android:drawable="@drawable/asteroid04" android:duration="100" />
  <item android:drawable="@drawable/asteroid05" android:duration="100" />
  <item android:drawable="@drawable/asteroid06" android:duration="100" />
</animation-list>
```

此动画将运行六个帧。`android:duration` 属性声明将显示每个帧的时间。下一步的代码段演示创建可绘制动画，然后启动它，当用户单击按钮在屏幕上的示例：

```
AnimationDrawable _asteroidDrawable;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    _asteroidDrawable = (Android.Graphics.Drawables.AnimationDrawable)
        Resources.GetDrawable(Resource.Drawable.spinning_asteroid);

    ImageView asteroidImage = FindViewById<ImageView>(Resource.Id.imageView2);
    asteroidImage.SetImageDrawable((Android.Graphics.Drawables.Drawable) _asteroidDrawable);

    Button asteroidButton = FindViewById<Button>(Resource.Id.spinAsteroid);
    asteroidButton.Click += (sender, e) =>
    {
        _asteroidDrawable.Start();
    };
}
```

现在我们已经介绍动画中的 Android 应用程序提供 Api 的基础知识。

总结

本文介绍了许多新概念和 API 以帮助将一些图形添加到 Android 应用程序。首先它讨论了各种二维图形 API 的并演示了如何 Android, 应用程序可以直接使用画布对象在屏幕上绘制。我们还了解到一些备用的技术, 允许以声明方式创建使用 XML 文件的图形。然后我们接着讨论的旧的和新的 API 用于在 Android 中创建动画。

相关链接

- [动画演示（示例）](#)
- [动画和图形](#)
- [使用动画将转换为现实的移动应用](#)
- [AnimationDrawable](#)
- [画布](#)
- [对象动画器](#)
- [值动画器](#)

CPU 体系结构

2018/10/26 • [Edit Online](#)

Xamarin.Android 支持多个 CPU 体系结构, 包括 32 位和 64 位设备。本文介绍如何以应用到一个或多个支持 Android 的 CPU 体系结构为目标。

CPU 体系结构概述

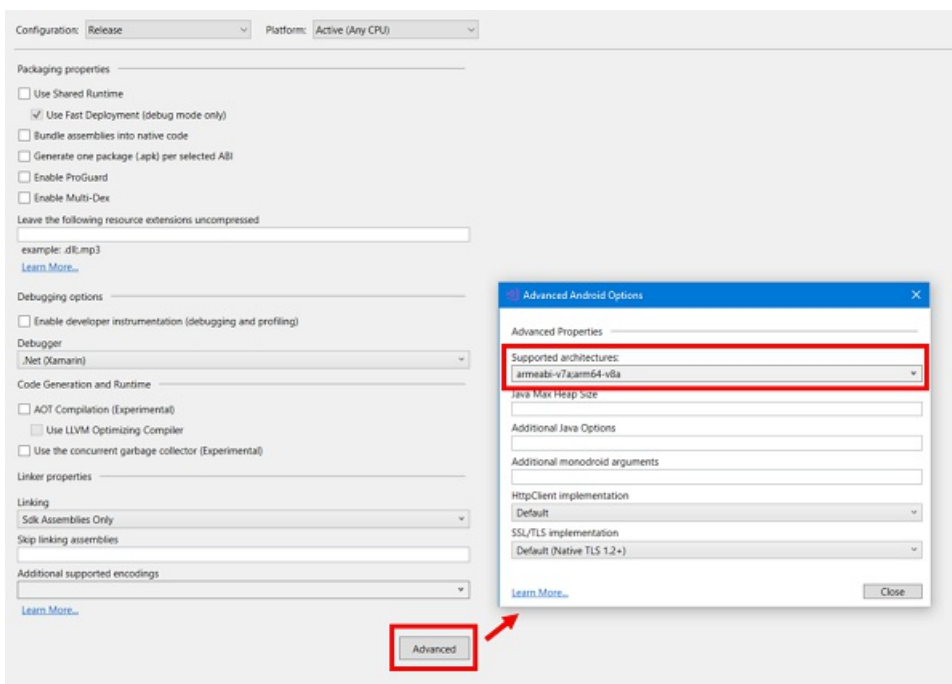
当你准备你的应用版本时, 必须指定哪些平台 CPU 体系结构应用支持。单个 APK 可包含计算机代码, 以支持多个不同的体系结构。特定于体系结构的代码的每个集合相关联 *应用程序二进制接口*(ABI)。每个 ABI 定义应如何与 Android 进行交互, 在运行时此机器代码。有关此方面的详细信息, 请参阅 [多核设备&Xamarin.Android](#)。

如何指定支持的体系结构

- [Visual Studio](#)
- [Visual Studio for Mac](#)

通常情况下, 显式选择一种体系结构 (或体系结构) 如果您的应用程序配置为版本。如果您的应用程序配置为调试, 则使用共享运行时并使用快速部署选项处于启用状态, 其中禁用选择显式体系结构。

在 Visual Studio 中, 右键单击您的项目解决方案资源管理器, 然后选择属性。下 **Android** 选项页上, 确保打包属性部分并验证使用共享运行时已禁用 (关闭此选项, 您可以显式选择哪些 Abi 支持)。单击高级按钮并在支持的体系结构, 检查你想要支持的体系结构:



Xamarin.Android 支持以下体系结构:

- **armeabi** –至少支持 ARMv5TE 指令集的基于 ARM 的 Cpu。请注意, `armeabi` 不是线程安全的不应在多 CPU 的设备上使用。
- **armeabi-v7a** –基于 ARM 的 Cpu 使用硬件浮点操作和多个 CPU (SMP) 设备。请注意, `armeabi-v7a` 计算机代码不会在 ARMv5 设备上运行。
- **arm64-v8a** –基于 64 位 ARMv8 体系结构的 Cpu。

- **x86** –支持 x86 (或 IA-32) 的 Cpu 指令集。此指令集是等效的 Pentium Pro, 包括 MMX、SSE、SSE2 和 SSE3 指令。
- **x86_64**支持的 64 位 x86 的 Cpu (也称为x64并AMD64) 指令集。

Xamarin.Android 将默认为 `armeabi-v7a` 有关发行生成。此设置可以提供性能明显优于 `armeabi`。如果你面向的 64 位 ARM 平台 (如 Nexus 9 中), 选择 `arm64-v8a`。如果你将应用部署到 x86 设备, 选择 `x86`。如果面向 x86 设备使用 64 位 CPU 体系结构, 选择 `x86_64`。

面向多个平台

若要针对多个 CPU 体系结构, 可以选择多个 ABI (但代价是 APK 文件较大)。可以使用生成一个包 (**.apk**) 每个选定 **ABI**选项 (中所述[设置打包属性](#)) 以创建每个单独的 APK 支持的体系结构。

无需选择**arm64-v8a**或**x86_64**面向 64 位设备; 64 位支持不需要 64 位硬件上运行你的应用。例如, 64 位 ARM 设备 (如Nexus 9) 可以运行应用程序配置为 `armeabi-v7a`。启用 64 位支持的主要优点是使你的应用以解决更多的内存。

NOTE

64 位运行时支持目前试验性功能。请记住, 64 位运行时不64 位设备上运行您的应用程序所必需。

其他信息

在某些情况下, 您可能需要创建单独的 APK 的每个体系结构 (可减小 APK, 或者是因为你的应用已共享库的特定于特定的 CPU 体系结构)。有关此方法的详细信息, 请参阅[生成特定于 ABI 的 Apk](#)。

处理旋转

2018/10/26 • [Edit Online](#)

本主题介绍如何处理在 *Xamarin.Android* 中的设备方向更改。它介绍了如何使用 *Android* 资源系统以自动加载资源的特定设备方向以及如何以编程方式处理方向更改。

概述

由于移动设备轻松地旋转，内置旋转是移动操作系统中的标准功能。*Android* 提供复杂的框架来处理旋转中的应用程序，而不管在 XML 中或以编程方式在代码中以声明方式创建的用户界面。在自动处理旋转的设备上的声明性布局更改时，应用程序可以受益于与 *Android* 资源系统的紧密集成。对于以编程方式布局，则必须手动处理更改。这样可以更好地控制在运行时，但代价更多的工作是开发人员。应用程序还可以选择以选择退出活动重新启动并手动控制的方向更改。

本指南将检查以下有针对性的主题：

- **声明性布局旋转**—如何使用 *Android* 资源系统生成能够识别方向的应用程序，包括如何加载布局和特定方向的绘图。
- **以编程方式布局旋转**—如何以编程方式添加控件，以及如何手动处理方向的更改。

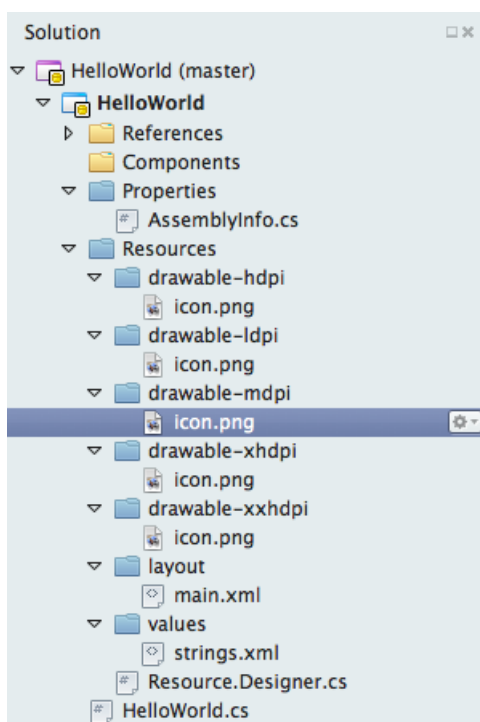
处理旋转以声明方式与布局

通过遵循命名约定的文件夹中包括的文件，*Android* 会自动加载相应的文件时在方向更改。这包括对支持：

- **布局资源**—指定哪些布局文件针对每个方向进行扩充。
- **可绘制资源**—指定哪个绘图加载针对每个方向。

布局资源

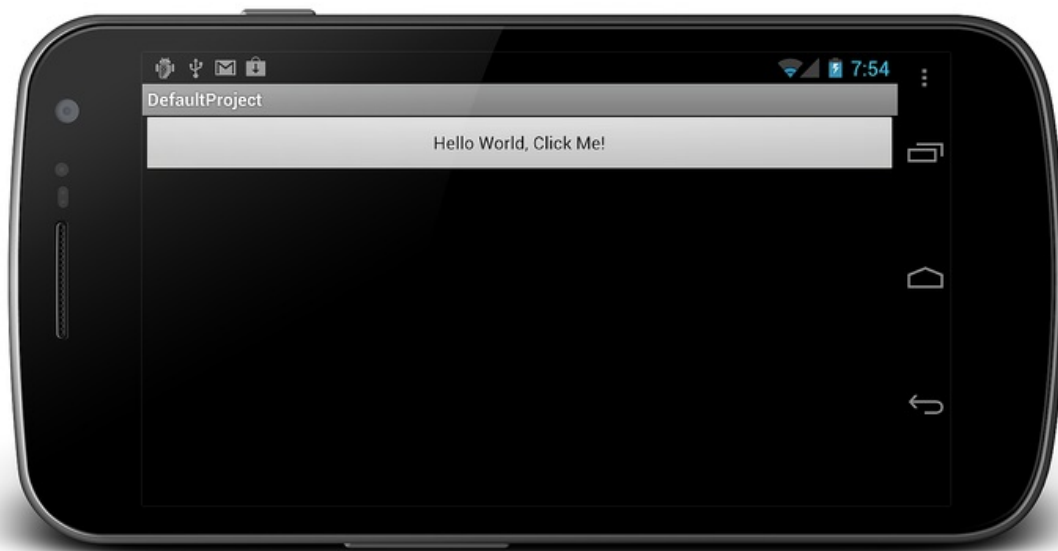
默认情况下，*Android XML (AXML)* 文件包含在资源/布局文件夹用于呈现视图的活动。如果没有其他布局的资源提供专为横向，所使用的纵向或横向方向该文件夹内的资源。默认项目模板创建的项目结构，请考虑：



此项目创建单个**Main.xml**中的文件资源/布局文件夹。当活动的 `onCreate` 方法调用时，它增加所空间中定义的视图**Main.xml**，其中声明一个按钮，如下面的 XML 中所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/myButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>
</LinearLayout>
```

如果将设备旋转为横向方向，该活动 `onCreate` 再次调用方法和相同**Main.xml**扩充文件，如下面的屏幕截图中所示：



特定于方向的布局

除了布局文件夹(其默认设置是纵向，还可以显式命名 **布局端口**通过包括名为的文件夹 `layout-land`)，应用程序可以定义它需要在无需任何代码的环境中的视图更改。

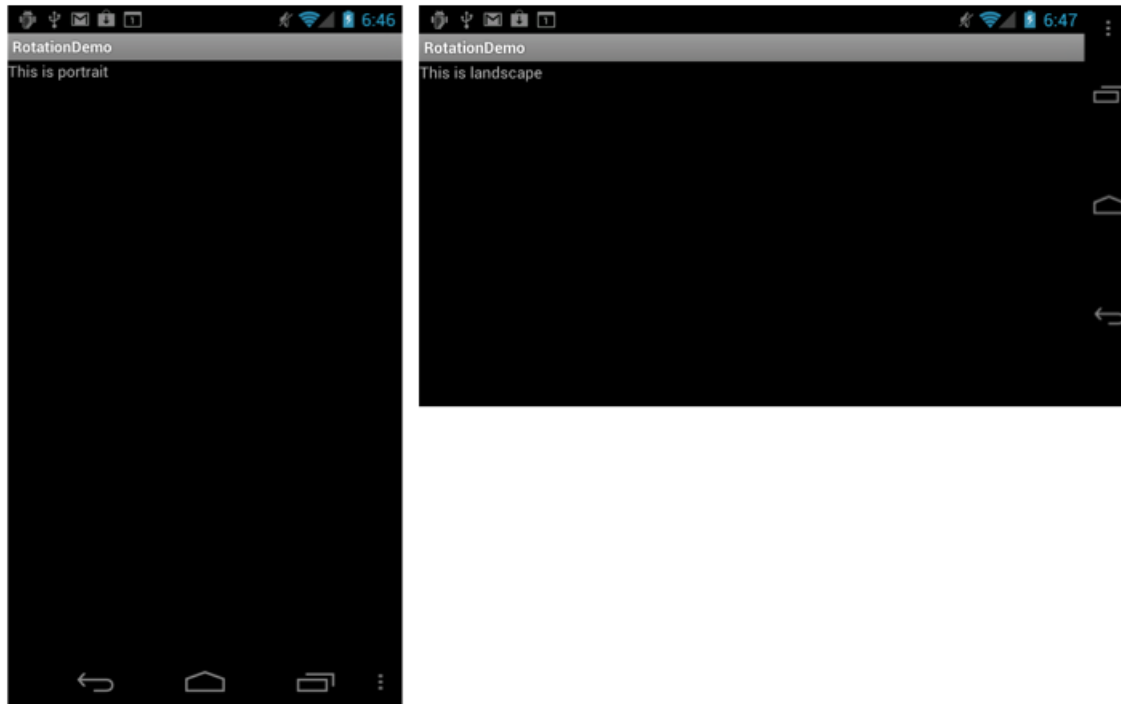
假设**Main.xml**文件包含以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:text="This is portrait"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
</RelativeLayout>
```

如果文件夹名为包含一个附加的布局 **land** **Main.xml**文件添加到项目中，放大布局在环境中时将立即导致加载新添加的 Android **Main.xml**。请考虑横向形式**Main.xml**文件，其中包含以下代码(为简单起见，此 XML 类似于默认纵向版本的代码，但使用中的不同字符串 `TextView`)：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:text="This is landscape"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
</RelativeLayout>
```

运行此代码并将设备从纵向向横向旋转演示新的 XML 加载，如下所示：



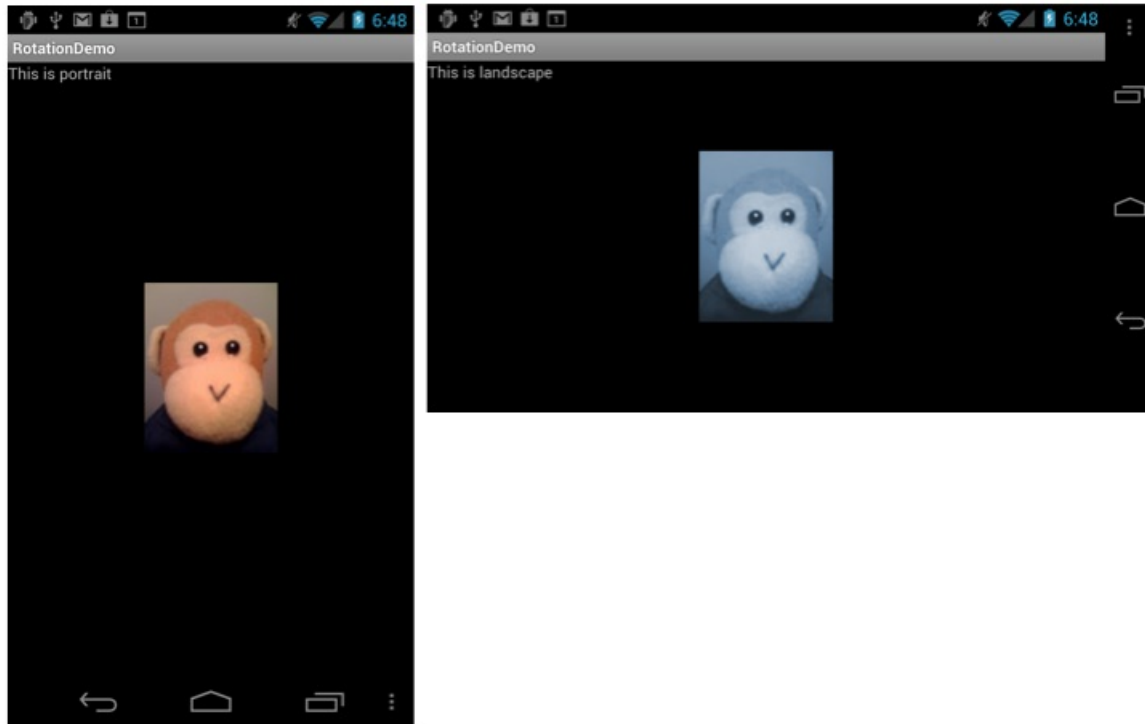
可绘制资源

在旋转，期间 Android 将视为可绘制资源同样到布局资源。在这种情况下，系统获取从绘图资源/**drawable**并资源/**drawable land**文件夹，分别。

例如，假设项目包含名为 Monkey.png 中的映像资源/**drawable**文件夹中，可绘制引用的位置从 `ImageView` 中 XML 如下：

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/monkey"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

让我们进一步假设的不同版本的**Monkey.png**包含下面资源/**drawable land**。就像布局文件，当设备时的旋转，针对给定方向，可绘制的更改，如下所示：



以编程方式处理旋转

有时，我们在代码中定义的布局。这可能由于各种原因，包括技术限制，限制、开发人员首选项等。当我们将以编程方式添加控件时，必须手动会考虑设备方向，当我们使用 XML 资源自动处理应用程序。

在代码中添加控件

若要以编程方式添加控件，应用程序需要执行以下步骤：

- 创建布局。
- 设置布局参数。
- 创建控件。
- 设置控件的布局参数。
- 将控件添加到布局。
- 设置为内容视图布局。

例如，考虑一个用户界面，包含单个 `TextView` 控件添加到 `RelativeLayout`，如下面的代码中所示。

```

protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // create a layout
    var rl = new RelativeLayout (this);

    // set layout parameters
    var layoutParams = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
    ViewGroup.LayoutParams.FillParent);
    rl.LayoutParameters = layoutParams;

    // create TextView control
    var tv = new TextView (this);

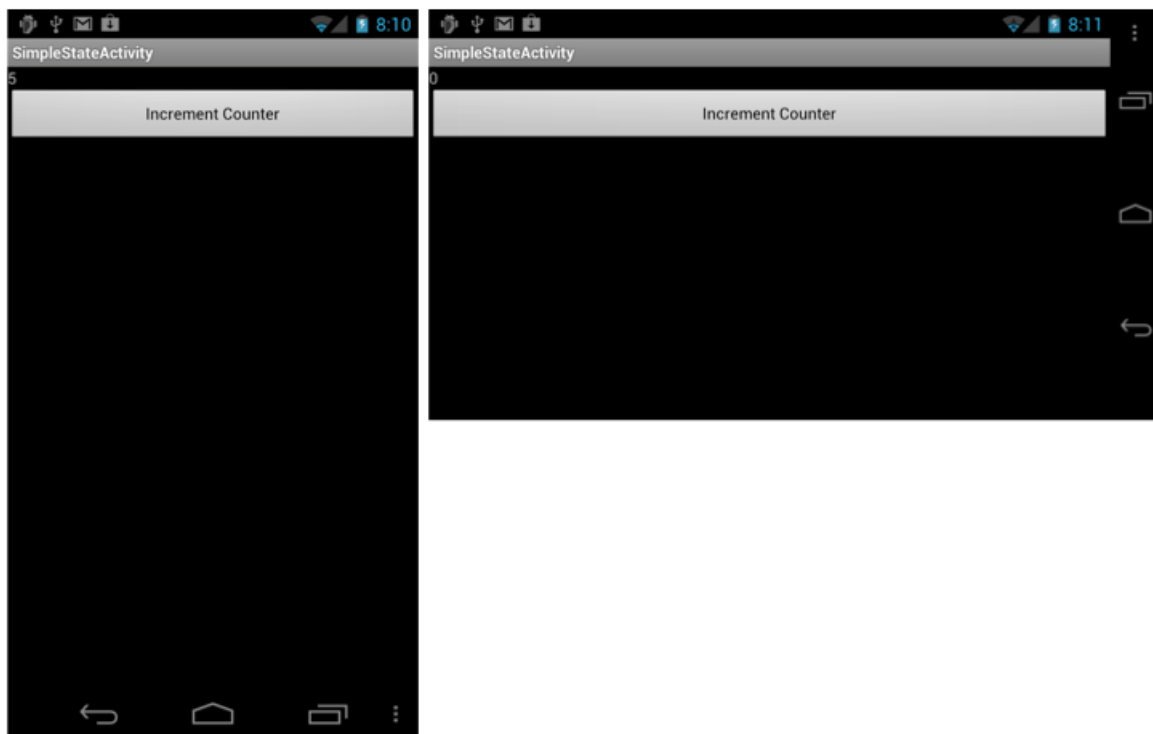
    // set TextView's LayoutParameters
    tv.LayoutParameters = layoutParams;
    tv.Text = "Programmatic layout";

    // add TextView to the layout
    rl.AddView (tv);

    // set the layout as the content view
    SetContentView (rl);
}

```

此代码创建的实例 `RelativeLayout` 类并设置其 `LayoutParameters` 属性。 `LayoutParams` 类是封装如何重用的方式放置控件的 Android 的方法。创建布局的实例后，可以创建并添加到该控件。控件还具有 `LayoutParameters`，如 `TextView` 在此示例中。之后 `TextView` 创建后，将其添加到 `RelativeLayout` 并设置 `RelativeLayout` 中显示的应用程序的内容视图结果作为 `TextView` 所示：



在代码中检测方向

如果应用程序尝试加载每个方向一个不同的用户界面时 `OnCreate` 称为（会出现这种每次旋转设备时），它必须检测方向，然后加载所需的用户界面代码。Android 有一个名为类 `WindowManager`，这可以用于确定通过当前的设备旋转 `WindowManager.DefaultDisplay.Rotation` 属性，如下所示：

```

protected override void onCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // create a layout
    var rl = new RelativeLayout (this);

    // set layout parameters
    var layoutParams = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
    ViewGroup.LayoutParams.FillParent);
    rl.LayoutParameters = layoutParams;

    // get the initial orientation
    var surfaceOrientation = WindowManager.DefaultDisplay.Rotation;
    // create layout based upon orientation
    RelativeLayout.LayoutParams tvLayoutParams;

    if (surfaceOrientation == SurfaceOrientation.Rotation0 || surfaceOrientation ==
    SurfaceOrientation.Rotation180) {
        tvLayoutParams = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
        ViewGroup.LayoutParams.WrapContent);
    } else {
        tvLayoutParams = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
        ViewGroup.LayoutParams.WrapContent);
        tvLayoutParams.LeftMargin = 100;
        tvLayoutParams.TopMargin = 100;
    }

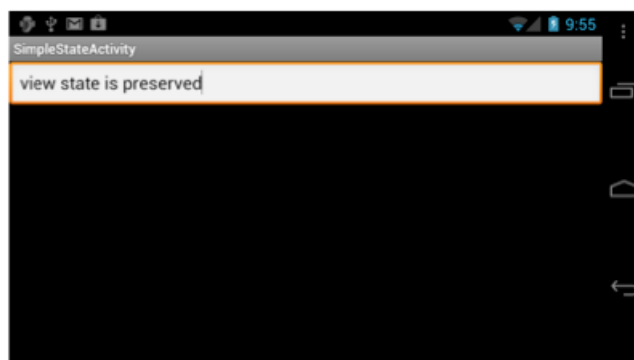
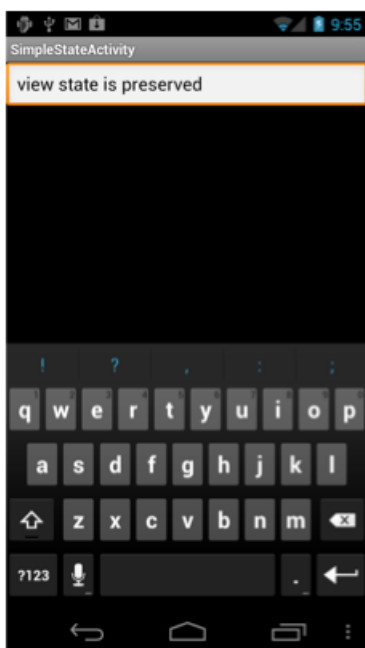
    // create TextView control
    var tv = new TextView (this);
    tv.LayoutParameters = tvLayoutParams;
    tv.Text = "Programmatic layout";

    // add TextView to the layout
    rl.AddView (tv);

    // set the layout as the content view
    SetContentView (rl);
}

```

此代码将设置 `TextView` 为定位的 100 像素, 角的屏幕上, 会自动进行动画处理到新的布局时旋转为横向, 如下所示:



阻止活动重新启动

除了处理中的所有内容 `OnCreate`，应用程序还可以防止某个活动正在重新启动时在方向更改通过设置 `ConfigurationChanges` 中 `ActivityAttribute`，如下所示：

```
[Activity (Label = "CodeLayoutActivity", ConfigurationChanges=Android.Content.PM.ConfigChanges.Orientation |
Android.Content.PM.ConfigChanges.ScreenSize)]
```

现在时将设备旋转，将不重新启动该活动。若要手动处理方向更改这种情况下，活动可以重写

`OnConfigurationChanged` 方法，并确定打印方向从 `Configuration` 对象传入，如下面的活动的新实现中所示：

```
[Activity (Label = "CodeLayoutActivity", ConfigurationChanges=Android.Content.PM.ConfigChanges.Orientation |
Android.Content.PM.ConfigChanges.ScreenSize)]
public class CodeLayoutActivity : Activity
{
    TextView _tv;
    RelativeLayout.LayoutParams _layoutParamsPortrait;
    RelativeLayout.LayoutParams _layoutParamsLandscape;

    protected override void OnCreate (Bundle bundle)
    {
        // create a layout
        // set layout parameters
        // get the initial orientation

        // create portrait and landscape layout for the TextView
        _layoutParamsPortrait = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
        ViewGroup.LayoutParams.WrapContent);

        _layoutParamsLandscape = new RelativeLayout.LayoutParams (ViewGroup.LayoutParams.FillParent,
        ViewGroup.LayoutParams.WrapContent);
        _layoutParamsLandscape.LeftMargin = 100;
        _layoutParamsLandscape.TopMargin = 100;

        _tv = new TextView (this);

        if (surfaceOrientation == SurfaceOrientation.Rotation0 || surfaceOrientation ==
        SurfaceOrientation.Rotation180) {
            _tv.LayoutParameters = _layoutParamsPortrait;
        } else {
            _tv.LayoutParameters = _layoutParamsLandscape;
        }

        _tv.Text = "Programmatic layout";
        rl.AddView (_tv);
        SetContentView (rl);
    }

    public override void OnConfigurationChanged (Android.Content.Res.Configuration newConfig)
    {
        base.OnConfigurationChanged (newConfig);

        if (newConfig.Orientation == Android.Content.Res.Orientation.Portrait) {
            _tv.LayoutParameters = _layoutParamsPortrait;
            _tv.Text = "Changed to portrait";
        } else if (newConfig.Orientation == Android.Content.Res.Orientation.Landscape) {
            _tv.LayoutParameters = _layoutParamsLandscape;
            _tv.Text = "Changed to landscape";
        }
    }
}
```

此处 `TextView's` 横向和纵向初始化布局参数。类变量保存参数，连同 `TextView` 本身，因为该活动不会重新创建方向更改时。该代码仍使用 `surfaceOrientartion` 中 `OnCreate` 若要设置的初始布局 `TextView`。在此之后，

`OnConfigurationChanged` 处理所有后续布局更改。

当我们运行该应用程序时，Android 将加载用户界面更改，如设备旋转发生，并且不会重启活动。

声明性布局的阻止活动重启

如果我们在 XML 中定义的布局，则还可以避免导致的设备旋转活动重启。例如，我们可以使用这种方法，如果我们想要防止活动重新启动 (出于性能原因，可能是) 和我们无需加载为不同的方向的新资源。

若要执行此操作，我们遵循以编程方式布局，我们使用的相同过程。只需将设置 `ConfigurationChanges` 中 `ActivityAttribute` 我们做的一样 `CodeLayoutActivity` 前面。确实需要运行方向更改可以重新实现中的任何代码 `OnConfigurationChanged` 方法。

在方向更改过程中维护状态

是否以声明方式或以编程方式处理旋转，所有的 Android 应用程序应实现的相同技术用于管理设备方向更改时的状态。管理状态非常重要，因为旋转 Android 设备时，系统重新启动正在运行的活动。Android 这样做是为了方便加载备用资源，例如布局和专为特定方向的绘图。它重新启动时，该活动将失去它可能已在本地类变量中存储任何暂时性状态。因此，如果某个活动是依赖状态，它必须保存在应用程序级别的状态。应用程序需要处理保存和还原，其目的是跨方向更改保留的任何应用程序状态。

有关在 Android 中持久保存状态的详细信息，请参阅[活动生命周期指南](#)。

总结

本文介绍如何使用 Android 的内置功能来使用旋转。首先，它介绍了如何使用 Android 资源系统可创建方向识别应用程序。然后，它显示如何在代码中添加控件，以及如何手动处理方向的更改。

相关链接

- [旋转演示 \(示例\)](#)
- [活动生命周期](#)
- [处理运行时更改](#)
- [快速屏幕方向更改](#)

Android 音频

2018/10/26 • [Edit Online](#)

Android OS 提供了广泛支持为多媒体, 其中包含音频和视频。本指南重点介绍在 Android 中的音频, 涵盖播放和录制音频, 使用内置的音频播放器和记录器类, 以及低级别的音频 API。它还介绍如何使用由其他应用程序, 广播音频事件, 以便开发人员可以构建良好的应用程序。

概述

新式移动设备已采用了功能, 以前需要的设备的专用的部件-照相机、音乐播放机和视频刻录机。正因为如此, 多媒体框架已成为移动 Api 中的第一类功能。

Android 提供了为多媒体的广泛支持。这篇文章将检查使用音频在 Android 中, 并涵盖以下主题

1. **播放音频使用 MediaPlayer**-使用内置 `MediaPlayer` 类来播放音频, 包括本地音频文件和使用的流式处理音频文件 `AudioTrack` 类。
2. **录制音频**-使用内置 `MediaRecorder` 录制音频的类。
3. **使用音频通知**-使用音频通知通过挂起或取消其音频输出创建良好的应用程序的正确响应事件 (如电话呼叫)。
4. **使用低级别音频**-播放音频使用 `AudioTrack` 通过直接写入内存缓冲区的类。录制音频使用 `AudioRecord` 类, 并直接从内存缓冲区中读取。

要求

本指南需要 Android 2.0 (API 级别 5) 或更高版本。请注意, 必须在设备上完成调试 Android 上的音频。

必须请求 `RECORD_AUDIO` 中的权限 **AndroidManifest.XML**:

| | |
|----------------------|---|
| Required permissions | <div><input type="checkbox"/> RECEIVE_SMS</div> <div><input type="checkbox"/> RECEIVE_WAP_PUSH</div> <div><input checked="" type="checkbox"/> RECORD_AUDIO</div> <div><input type="checkbox"/> REORDER_TASKS</div> <div><input type="checkbox"/> RESTART_PACKAGES</div> <div><input type="checkbox"/> SEND_SMS</div> <div><input type="checkbox"/> SET_ACTIVITY_WATCHER</div> <div><input type="checkbox"/> SET_ALARM</div> <div><input type="checkbox"/> SET_ALWAYS_FINISH</div> <div><input type="checkbox"/> SET_ANIMATION_SCALE</div> <div><input type="checkbox"/> SET_DEBUG_APP</div> <div><input type="checkbox"/> SET_ORIENTATION</div> <div><input type="checkbox"/> SET_POINTER_SPEED</div> <div><input type="checkbox"/> SET_PREFERRED_APPLICATIONS</div> <div><input type="checkbox"/> SET_PROCESS_LIMIT</div> <div><input type="checkbox"/> SET_TIME</div> <div><input type="checkbox"/> SET_TIME_ZONE</div> <div><input type="checkbox"/> SET_WALLPAPER</div> |
|----------------------|---|

播放音频与 MediaPlayer 类

播放音频在 Android 中的最简单方法是使用内置 [MediaPlayer](#) 类。`MediaPlayer` 可以通过传入的文件路径播放本地或远程文件。但是，`MediaPlayer` 是非常状态区分，并调用其方法之一在错误的状态将导致引发异常。务必要与之交互 `MediaPlayer` 以避免错误如下所述的顺序。

初始化和播放

播放音频与 `MediaPlayer` 需要按以下顺序：

1. 实例化一个新 [MediaPlayer](#) 对象。
2. 配置文件以通过播放 [SetDataSource](#) 方法。
3. 调用 [准备](#) 方法以初始化播放器。
4. 调用 [启动](#) 方法开始音频播放。

下面的代码示例说明了这种用法：

```
protected MediaPlayer player;
public void StartPlayer(String filePath)
{
    if (player == null) {
        player = new MediaPlayer();
    } else {
        player.Reset();
        player.SetDataSource(filePath);
        player.Prepare();
        player.Start();
    }
}
```

挂起和继续执行播放

可以通过调用挂起播放[暂停](#)方法：

```
player.Pause();
```

若要恢复已暂停的播放，请调用[启动](#)方法。这将恢复从中播放暂停的位置：

```
player.Start();
```

调用[停止](#)播放机上的方法可结束正在进行播放：

```
player.Stop();
```

当不再需要时播放机时，必须通过调用释放的资源[版本](#)方法：

```
player.Release();
```

使用录制音频 MediaRecorder 类

到推论 `MediaPlayer` 为在 Android 中的录制音频 `MediaRecorder` 类。如 `MediaPlayer`，其状态区分，并将经历若干种状态，若要访问它可以在那里开始录制的点。若要录制音频，`RECORD_AUDIO` 权限必须设置。有关说明如何将应用程序设置权限请参阅[使用 AndroidManifest.xml](#)。

初始化和记录

录制音频和 `MediaRecorder` 需要执行以下步骤：

1. 实例化一个新 `MediaRecorder` 对象。
2. 指定要用于捕获通过的音频输入哪些硬件设备 `SetAudioSource` 方法。
3. 设置输出文件的音频格式使用 `SetOutputFormat` 方法。有关受支持的音频类型的列表，请参阅[Android 支持媒体格式](#)。
4. 调用 `SetAudioEncoder` 方法以设置编码类型的音频。
5. 调用 `SetOutputFile` 方法，以指定将音频数据写入到输出文件的名称。
6. 调用 [准备](#) 方法来初始化记录器。
7. 调用 [启动](#) 方法开始录制。

下面的代码示例演示了此序列：


```
protected MediaRecorder recorder;
void RecordAudio (String filePath)
{
    try {
        if (File.Exists (filePath)) {
            File.Delete (filePath);
        }
        if (recorder == null) {
            recorder = new MediaRecorder (); // Initial state.
        } else {
            recorder.Reset ();
            recorder.SetAudioSource (AudioSource.Mic);
            recorder.SetOutputFormat (OutputFormat.ThreeGpp);
            recorder.SetAudioEncoder (AudioEncoder.AmrNb);
            // Initialized state.
            recorder.SetOutputFile (filePath);
            // DataSourceConfigured state.
            recorder.Prepare (); // Prepared state
            recorder.Start (); // Recording state.
        }
    } catch (Exception ex) {
        Console.Out.WriteLine( ex.StackTrace);
    }
}
```

正在停止录制

若要停止录制，请调用 `Stop` 方法 `MediaRecorder`：

```
recorder.Stop();
```

清理

一次 `MediaRecorder` 已停止，调用 [重置](#) 方法以将其放回其空闲状态：

```
recorder.Reset();
```

当 `MediaRecorder` 是不再需要其资源必须释放通过调用 [发行](#) 方法：

```
recorder.Release();
```

管理音频通知

AudioManager 类

[AudioManager](#) 类提供对音频通知，使应用程序知道音频事件发生时访问。此服务还提供其他音频功能，如卷和响铃模式控制访问。`AudioManager` 允许处理控制音频播放的音频通知应用程序。

管理音频焦点

设备（内置播放器和录制器）的音频资源可由所有正在运行的应用程序共享。

从概念上讲，它类似于其中只有一个应用程序具有键盘焦点的台式计算机上的应用程序：选择后一个正在运行的应用程序通过鼠标单击，键盘输入将仅对该应用程序。

音频的焦点是类似的原理，可防止从播放或在同一时间录制音频的多个应用程序。它是比键盘焦点要复杂一些，因为它是自愿-应用程序可以忽略这一事实，它不会当前不具有音频焦点且无论播放-，因为不同类型的可能的音频焦点请求。例如，如果请求程序应只播放音频很短时间，它可能会请求暂时性焦点。

音频的焦点可能会被立即授予或最初被拒绝并授予更高版本。例如，如果应用程序请求音频焦点在电话呼叫期间，

它将被拒绝，但电话呼叫完成后，也可能会授予焦点。在这种情况下，侦听器会注册以相应地响应，如果音频焦点移走。请求音频焦点用于确定它正常播放或录制音频。

有关音频焦点的详细信息，请参阅[管理音频焦点](#)。

注册回调的音频焦点

注册 `FocusChangeListener` 从回调 `IOOnAudioChangeListener` 是获取并释放音频焦点的一个重要部分。这是焦点的因为授予的音频可能会延迟到更高版本的时间。例如，应用程序可能会请求在进行电话呼叫时播放音乐。电话呼叫完成之前，将不授予音频焦点。

出于此原因，回调对象作为参数传递 `GetAudioFocus` 方法的 `AudioManager`，并且此注册回调的调用。应用程序如果音频的焦点是最初被拒绝，但更高版本授予，将得到通知，通过调用 `OnAudioFocusChange` 在回调上。同一方法用于告知应用程序，消失点音频焦点。

当应用程序完成后使用的音频资源时，它将调用 `AbandonFocus` 方法的 `AudioManager`，并再次将传递在回调中。此注销回调并释放的音频资源，以便其他应用程序可能会获取音频的焦点。

请求的音频焦点

若要请求设备的音频资源所需的步骤如下所示：

1. 获取的句柄 `AudioManager` 系统服务。
2. 创建回调类的实例。
3. 通过调用请求设备的音频资源 `RequestAudioFocus` 方法 `AudioManager`。参数是回调对象、流类型（音乐、语音呼叫、环等）和权限请求的访问权限的类型（音频资源可以请求暂时不可用或无限期，例如）。
4. 如果授予的请求，`playMusic` 立即调用方法并开始播放音频。
5. 如果请求遭拒绝，不执行任何进一步的操作。在这种情况下，如果请求批准在更高版本时，仅将播放音频。

下面的代码示例说明了这些步骤：

```
Boolean RequestAudioResources(INotificationReceiver parent)
{
    AudioManager audioMan = (AudioManager) GetSystemService(Context.AudioService);
    AudioManager.IOOnAudioFocusChangeListener listener = new MyAudioListener(this);
    var ret = audioMan.RequestAudioFocus (listener, Stream.Music, AudioFocus.Gain );
    if (ret == AudioFocusRequest.Granted) {
        playMusic();
        return (true);
    } else if (ret == AudioFocusRequest.Failed) {
        return (false);
    }
    return (false);
}
```

释放音频焦点

轨道的播放完毕后 `AbandonFocus` 方法 `AudioManager` 调用。这样，另一个应用程序以获取设备的音频资源。如果他们已注册其自己的侦听器，其他应用程序将收到此音频的焦点更改的通知。

低级别的音频 API

低级别的音频 Api 提供更好地控制音频播放和录制，因为它们直接交互而不是使用文件 Uri 的内存缓冲区。有一些情况下，这种方法更可取。这种情况下包括：

1. 当从播放加密的音频文件。
2. 当播放一系列简短的剪辑。
3. 音频流。

AudioTrack 类

AudioTrack类使用录制音频的低级别 Api, 而且相当低级别的 MediaPlayer 类。

初始化和播放

若要播放音频的新实例 AudioTrack 必须实例化。参数列表传递给构造函数指定如何播放音频缓冲区中包含的示例。参数是：

1. Stream 类型-语音、铃声、音乐、系统或警报。
2. 频率-以 Hz 的采样率。
3. 通道配置-Mono 或立体声。
4. 音频格式-8 位或 16 位编码。
5. 缓冲区大小-以字节为单位。
6. 缓冲区模式-流式处理或静态。

构造之后,播放方法的 AudioTrack 调用, 以将其设置为开始播放。写入到的音频缓冲区 AudioTrack 开始播放：

```
void PlayAudioTrack(byte[] audioBuffer)
{
    AudioTrack audioTrack = new AudioTrack(
        // Stream type
        Stream.Music,
        // Frequency
        11025,
        // Mono or stereo
        ChannelOut.Mono,
        // Audio encoding
        Android.Media.Encoding.Pcm16bit,
        // Length of the audio clip.
        audioBuffer.Length,
        // Mode. Stream or static.
        AudioTrackMode.Stream);

    audioTrack.Play();
    audioTrack.Write(audioBuffer, 0, audioBuffer.Length);
}
```

暂停和停止播放

调用暂停方法来暂停播放：

```
audioTrack.Pause();
```

调用停止方法将会永久终止播放：

```
audioTrack.Stop();
```

清理

当 AudioTrack 是不再需要其资源必须释放通过调用发行：

```
audioTrack.Release();
```

AudioRecord 类

AudioRecord类相当于 AudioTrack 录制端。如 AudioTrack, 它使用内存缓冲区直接, 来代替文件和 Uri。该配置要求 RECORD_AUDIO 权限设置在清单中。

初始化和记录

第一步是构造一个新`AudioRecord`对象。参数列表传递给构造函数提供用于记录所需的所有信息。不同于在`AudioTrack`，其中的参数是很大程度上枚举中的等效参数`AudioRecord`是整数。这些方法包括：

1. 硬件音频输入的源如麦克风。
2. Stream 类型-语音、铃声、音乐、系统或警报。
3. 频率-以 Hz 的采样率。
4. 通道配置-Mono 或立体声。
5. 音频格式-8 位或 16 位编码。
6. 缓冲区大小中字节

一次`AudioRecord`构造时，其`StartRecording`调用方法。现在已准备好开始录制。`AudioRecord`持续读取的音频缓冲区的输入，并编写出此输入到音频文件。

```
void RecordAudio()
{
    byte[] audioBuffer = new byte[100000];
    var audRecorder = new AudioRecord(
        // Hardware source of recording.
        AudioSource.Mic,
        // Frequency
        11025,
        // Mono or stereo
        ChannelIn.Mono,
        // Audio encoding
        Android.Media.Encoding.Pcm16bit,
        // Length of the audio clip.
        audioBuffer.Length
    );
    audRecorder.StartRecording();
    while (true) {
        try
        {
            // Keep reading the buffer while there is audio input.
            audRecorder.Read(audioBuffer, 0, audioBuffer.Length);
            // Write out the audio file.
        } catch (Exception ex) {
            Console.Out.WriteLine(ex.Message);
            break;
        }
    }
}
```

停止录制

调用[停止](#)方法终止该录制：

```
audRecorder.Stop();
```

清理

当`AudioRecord`不再需要对象，调用其[发行](#)方法释放与之关联的所有资源：

```
audRecorder.Release();
```

总结

Android OS 提供了一个功能强大的框架，用于播放、录制和管理音频。本文介绍了如何播放和录制音频，使用的高层次 `MediaPlayer` 和 `MediaRecorder` 类。接下来，它探讨了如何使用音频通知来共享音频设备之间不同的应用程序的资源。最后，它处理如何播放和录制音频，使用低级别 Api，该接口直接与内存缓冲区。

相关链接

- [使用与音频（示例）](#)
- [媒体播放器](#)
- [媒体录制器](#)
- [音频管理器](#)
- [音频曲目](#)
- [音频录制器](#)

在 Xamarin.Android 中的通知

2018/10/26 • [Edit Online](#)

概述

本部分介绍如何在 Xamarin.Android 中实现通知。其中介绍了 Android 通知的各种 UI 元素，并讨论了 API 的涉及创建并显示一条通知。

部分

在 Android 中的本地通知

本部分介绍如何在 Xamarin.Android 中实现本地通知。它介绍了 Android 通知的各种 UI 元素，并讨论了 API 的涉及创建并显示一条通知。

演练-在 Xamarin.Android 中使用本地通知

本演练介绍如何在 Xamarin.Android 应用程序中使用本地通知。它演示了创建和发布通知的基础知识。当用户单击通知抽屉中的通知时在启动第二个活动。

其他阅读材料

[Firebase Cloud Messaging](#) – Firebase Cloud Messaging (FCM) 是一种便于移动应用和服务器应用程序之间的消息传送的服务。Firebase Cloud Messaging 可用来实现远程通知（也称为推送通知）在 Xamarin.Android 应用程序中。

[通知](#)–此 Android 开发人员主题是 Android 通知的权威指南。它包括的设计注意事项部分，可帮助你设计您的通知，以便它们符合的 Android 用户界面的准则。启动活动，该活动时，它提供有关 preserving 导航的更多背景信息，并介绍如何在锁定屏幕上通知和控制媒体的播放中显示进度。

[NotificationListenerService](#) –此 Android 服务使您的应用程序以侦听（并与之交互）的所有通知都发布在 Android 设备上，而不仅仅是您的应用程序注册到通知接收。请注意，用户必须显式授予对您的应用程序，以便能够侦听的设备上的通知的权限。

相关链接

- [本地通知（示例）](#)
- [远程通知（示例）](#)

本地通知

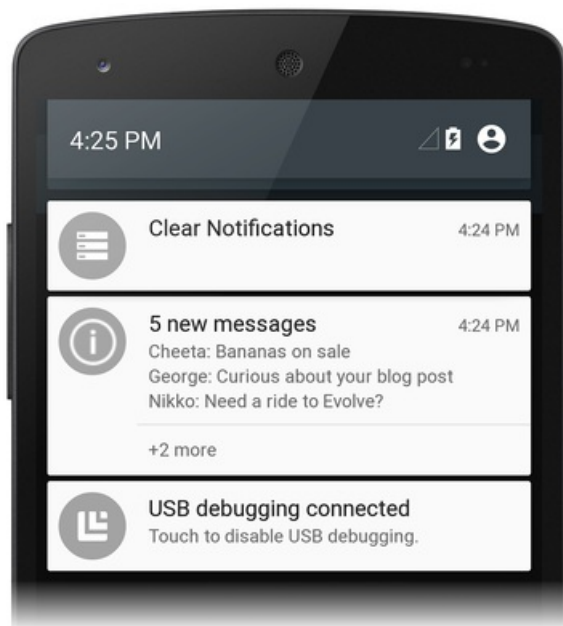
本部分演示如何在 *Xamarin.Android* 中实现本地通知。它介绍 Android 通知的各种 UI 元素, 并讨论了 API 的涉及创建并显示一条通知。

本地通知概述

Android 提供用于向用户显示通知图标和通知信息的两个系统控制的区域。在首次发布一个通知, 其图标显示在 *通知区域*, 如以下屏幕截图中所示:



若要获取有关通知的详细信息, 用户可以打开通知抽屉 (它可以扩展每个通知图标可显示通知内容), 并执行与通知关联的任何操作。以下屏幕截图显示 *通知抽屉*, 它对应于上面显示的通知区域:



Android 通知使用两种类型的布局:

- **基本布局** – compact、固定的显示格式。
- **展开的布局** – 可以扩展到更大的大小以显示详细信息的显示格式。

以下各节介绍了每个布局类型 (以及如何创建它们)。

NOTE

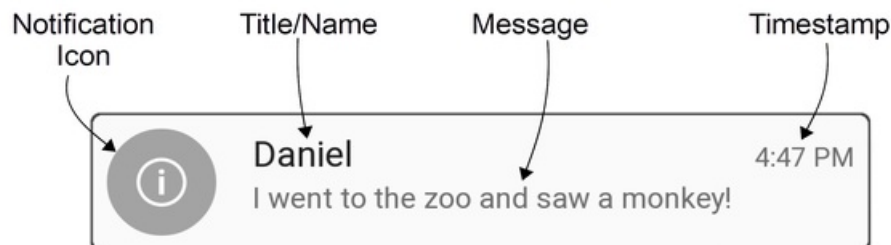
本指南重点 [NotificationCompat Api](#) 从 [Android 支持库](#)。这些 Api 将确保最大值向后兼容到 Android 4.0 (API 级别 14)。

基本布局

所有 Android 通知是基于基本布局格式，其中至少包括以下元素：

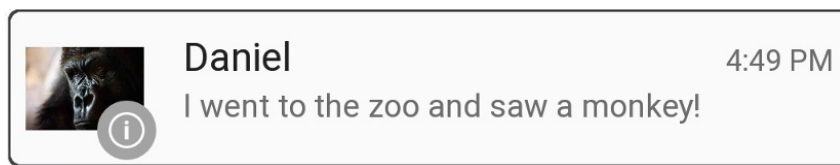
1. 一个 **通知图标**，如果应用支持不同类型的通知表示发起应用程序或通知类型。
2. 通知 **标题**，或如果通知是个人消息发件人的名称。
3. 通知消息。
4. 一个 **时间戳**。

这些元素将显示如以下关系图中所示：

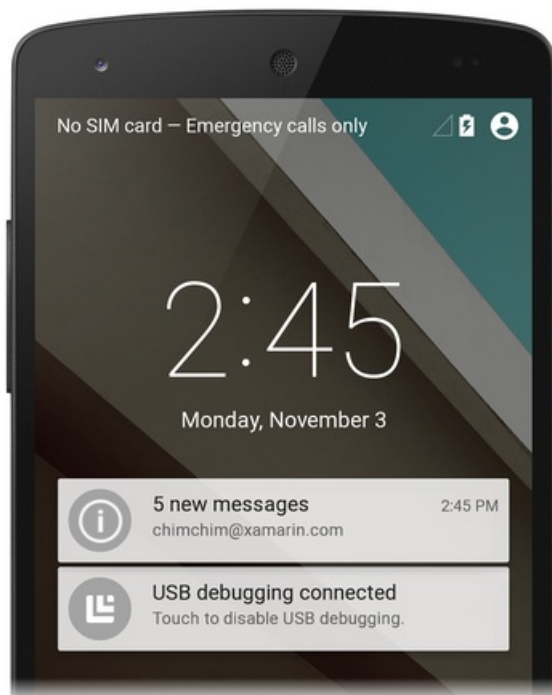


基本布局仅限于 64 密度无关的像素 (dp) 的高度。Android 默认情况下创建此基本通知样式。

(可选) 通知可以显示大图标，表示应用程序或发件人的照片。当通知 Android 5.0 及更高版本中使用大图标时，小的通知图标大图标上方显示的徽章：

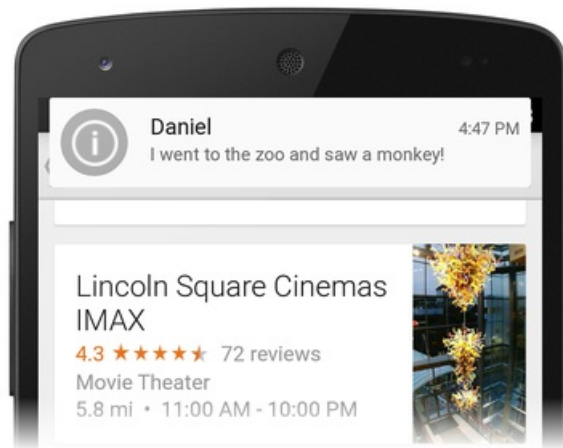


从 Android 5.0 开始，通知也会出现在锁定屏幕上：



用户可以双击锁定屏幕通知来解锁设备，并跳转到源自该通知的应用或轻扫以取消通知。应用程序可以设置通知，以控制锁定屏幕上显示的内容的可见性级别和用户可以选择是否允许敏感内容要在锁定屏幕通知中显示。

Android 5.0 引入了名为以高优先级通知显示格式 **平视**。危险警告通知从屏幕顶部几秒钟向下滑动，然后备份到通知区域参加：



危险警告通知使系统用户界面将呈现给用户的重要信息而不会中断当前正在运行的活动的状态。

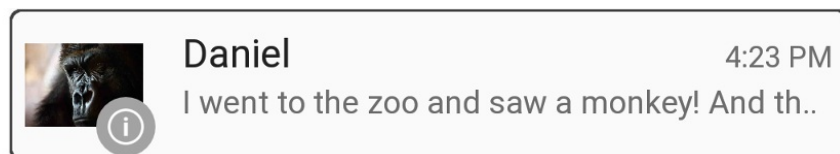
Android 包括对通知的元数据的支持，因此可以进行排序和以智能方式显示通知。通知的元数据还可以控制如何在锁定屏幕上和格式平视显示通知。应用程序可以设置以下类型的通知的元数据：

- **优先级**—优先级确定如何以及何时显示通知。例如，在 Android 5.0，高优先级的通知将显示为危险警告通知。
- **可见性**—指定多少通知内容为在锁定屏幕上会显示通知时要显示。
- **类别**—通知系统如何处理在各种情况下，例如设备时通知 *请勿打扰* 模式。

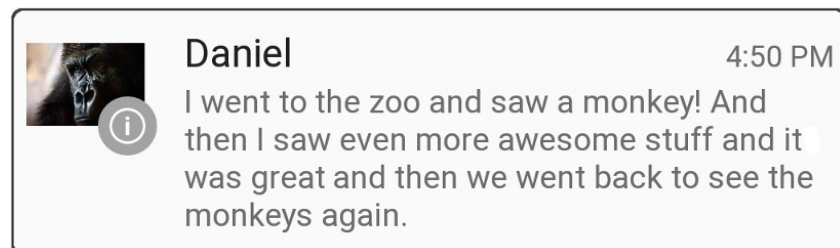
注意：可见性并类别引入在 Android 5.0 和早期版本的 Android 中不可用。从开始 Android 8.0 [通知通道](#) 用于控制如何向用户显示通知。

扩展的布局

从 Android 4.1 开始，可以使用允许用户以展开该通知查看更多的内容的高度扩展的布局样式配置通知。例如，下面的示例演示了在约定的模式下的展开的布局通知：



在展开此通知后，它将显示整个消息：



Android 支持单事件通知的三个扩展的布局样式：

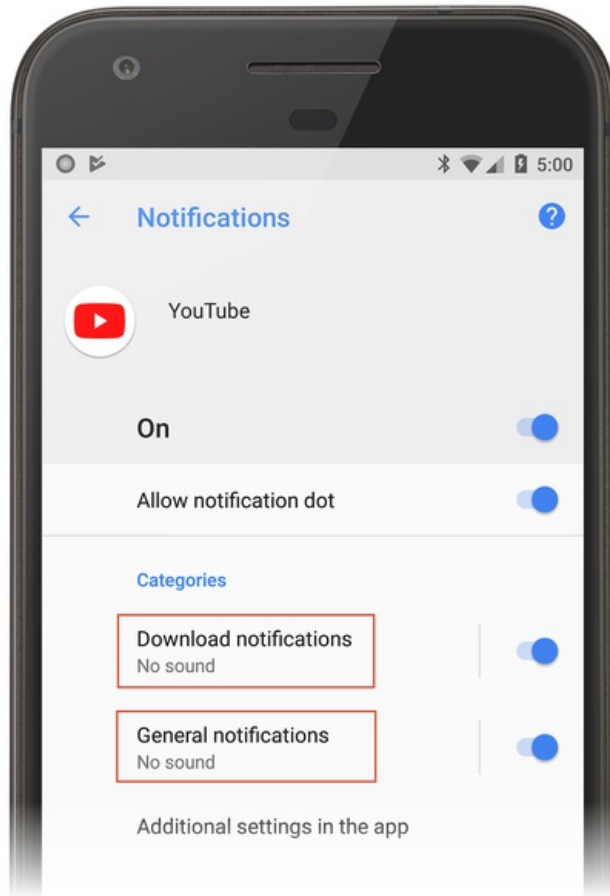
- **大文本**—在约定的模式下，将显示一段摘录的两个句点后，跟的消息的第一行。在展开模式下，显示整个消息（如在上面的示例所示）。
- **收件箱**—在约定的模式下，将显示新的消息数。在展开模式显示的第一个电子邮件消息或收件箱中的消息列表。
- **图像**—在约定的模式下，仅显示消息文本。在展开模式显示的文本和图像。

[基本通知超出](#)（在本文后面介绍）说明如何创建大文本，收件箱，和映像通知。

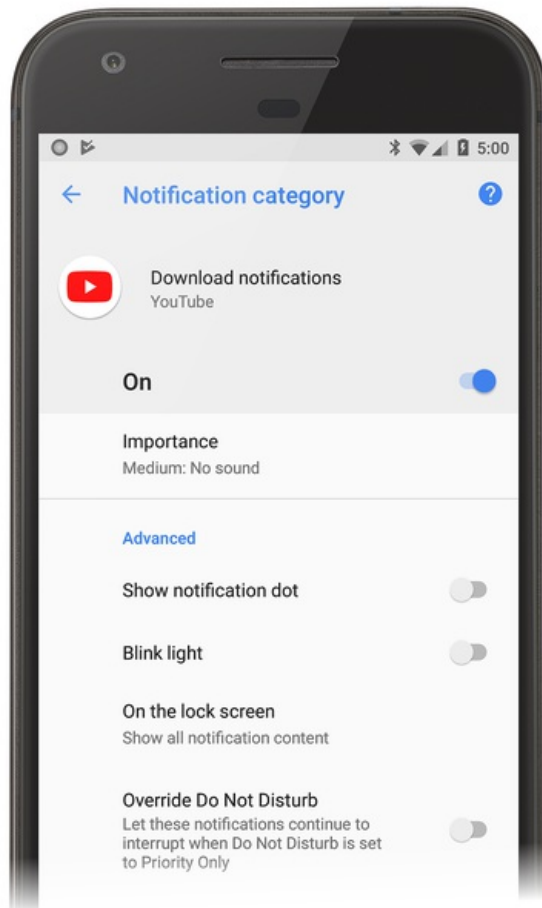
通知通道

从 Android 8.0 (Oreo) 开始, 你可以使用**通知通道**功能来创建用于每种类型的通知, 你想要显示的用户可自定义通道。通知通道使你组通知, 以便所有通知都发布到通道附录相同的行为。例如, 可能有适用于需要立即关注的通知的通知通道和单独的"更安静"通道用于信息性消息。

YouTube与 Android Oreo 一起安装的应用列出了两个通知类别:**下载通知**并**常规通知**:



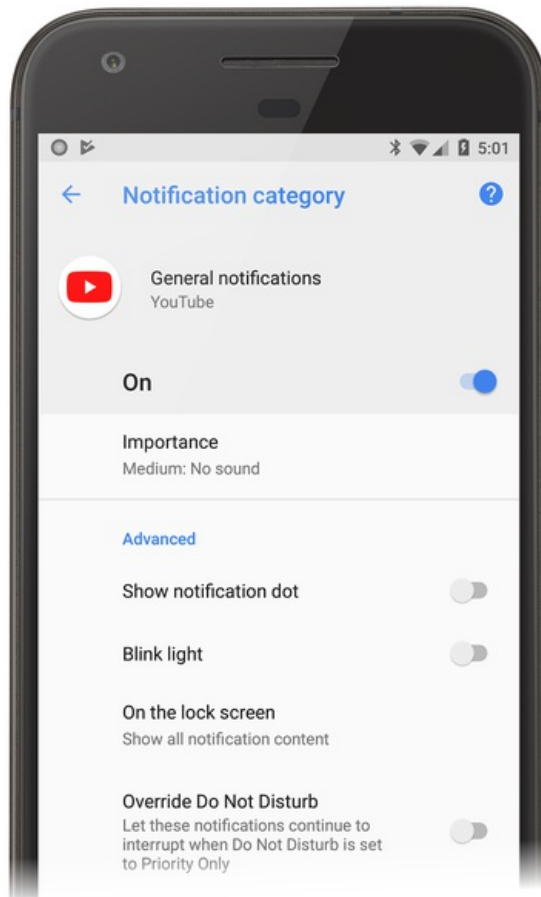
每个类别对应于通知通道。YouTube 应用实现**下载通知**通道和一个**常规通知**通道。用户可以点击**下载通知**, 这将显示设置屏幕的应用程序的**下载通知**通道:



在此屏幕中，用户可以修改的行为下载通知通道通过执行以下操作：

- 重要性级别设置为紧急，高，中等，或者低，配置级别的声音和视频的中断。
- 打开或关闭通知点。
- 打开或关闭指示灯闪烁。
- 显示或隐藏在锁定屏幕上的通知。
- 重写请勿打扰设置。

常规通知通道具有相似的设置：



请注意，没有绝对控制通知通道如何与用户交互-用户可以修改任何设备上的通知通道的设置，如上面的屏幕截图中所示。但是，可以配置默认值（如将如下所述）。如这些示例所示，新的通知通道功能使你能够向用户授予对不同类型的通知进行精细控制。

创建通知

若要在 Android 中创建一条通知，请使用 `NotificationCompat.Builder` 类派生 `Xamarin.Android.Support.v4` NuGet 包。此类使用可以创建并发布较旧版本的 Android 上的通知。有关使用详细信息

`NotificationCompat.Builder`，请参阅[兼容性](#)本主题中更高版本。

`NotificationCompat.Builder` 提供在通知中，如设置各种选项的方法：

- 内容，包括标题、消息文本和通知图标。
- 样式的通知，如大文本，收件箱，或图像样式。
- 通知的优先级：最小值、低，默认情况下，高，或最大值。在 Android 8.0 及更高版本，通过设置优先级[通知通道](#)。
- 在锁定屏幕上通知的可见性：公用、专用或机密。
- 类别的元数据，可帮助 Android 进行分类和筛选通知。
- 可选意向，该值指示用于启动点击通知时的活动。
- 通知通道，将 (Android 8.0 及更高版本) 上发布通知的 ID。

在生成器中设置这些选项后，您将生成一个包含设置的通知对象。若要发布通知，则此通知将对象传递给[通知管理器](#)。Android 提供 `NotificationManager` 类，该类负责发布通知并向用户显示它们。可以从任何上下文中，例如活动或服务获取对此类的引用。

创建通知通道

在 Android 8.0 运行的应用程序必须创建其通知的通知通道。通知通道需要以下三个信息片段：

- 仅适用于将标识信道的包 ID 字符串。
- 将向用户显示的通道的名称。该名称必须是介于 1 和 40 之间的字符。
- 通道的重要性。

应用将需要检查的正在运行的 Android 版本。运行版本早于 Android 8.0 设备不应创建通知通道。下面的方法是如何创建通知通道在活动中的一个示例：

```
void CreateNotificationChannel()
{
    if (Build.VERSION.SdkInt < BuildVersionCodes.O)
    {
        // Notification channels are new in API 26 (and not a part of the
        // support library). There is no need to create a notification
        // channel on older versions of Android.
        return;
    }

    var channelName = Resources.GetString(Resource.String.channel_name);
    var channelDescription = GetString(Resource.String.channel_description);
    var channel = new NotificationChannel(CHANNEL_ID, channelName, NotificationImportance.Default)
    {
        Description = channelDescription
    };

    var notificationManager = (NotificationManager) GetSystemService(NotificationService);
    notificationManager.CreateNotificationChannel(channel);
}
```

每次创建活动，应创建通知通道。有关 `CreateNotificationChannel` 方法，它应调用 `OnCreate` 活动的方法。

创建和发布通知

若要在 Android 中生成一条通知，请执行以下步骤：

1. 实例化 `NotificationCompat.Builder` 对象。
2. 在调用各种方法 `NotificationCompat.Builder` 对象，以设置通知选项。
3. 调用构建方法的 `NotificationCompat.Builder` 对象实例化通知对象。
4. 调用通知要发布通知的通知管理器的方法。

必须至少提供每个通知的以下信息：

- 小图标 (24x24 dp 的大小)
- 短标题
- 通知文本

下面的代码示例演示了如何使用 `NotificationCompat.Builder` 生成基本的通知。请注意，`NotificationCompat.Builder` 方法支持方法链接；也就是说，每个方法返回生成器对象，因此可以使用的最后一次方法调用结果调用下一个方法调用：

```
// Instantiate the builder and set notification elements:
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle ("Sample Notification")
    .setContentText ("Hello World! This is my first notification!")
    .setSmallIcon (Resource.Drawable.ic_notification);

// Build the notification:
Notification notification = builder.Build();

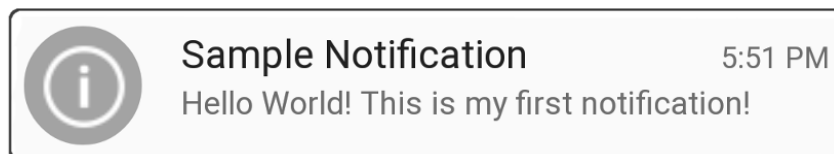
// Get the notification manager:
NotificationManager notificationManager =
    getSystemService (Context.NotificationService) as NotificationManager;

// Publish the notification:
const int notificationId = 0;
notificationManager.Notify (notificationId, notification);
```

在此示例中，一个新 `NotificationCompat.Builder` 对象调用 `builder` 实例化，以及要使用通知通道的 ID。设置的标题和通知的文本，并从加载的通知图标 `Resources/drawable/ic_notification.png`。通知生成器对 `Build` 方法使用这些设置创建的通知对象。下一步是调用 `Notify` 通知管理器的方法。若要查找通知管理器，请调用 `GetSystemService`，如上所示。

`Notify` 方法接受两个参数：通知标识符和通知对象。通知标识符是一个唯一的整数，它标识为你的应用程序的通知。在此示例中，通知标识符设置为零 (0);但是，在生产应用程序中，你将想要为每个通知提供唯一标识符。重复使用以前的标识符值对的调用中 `Notify` 导致最后一个通知被覆盖。

当在 Android 5.0 设备上运行此代码时，它会生成一条通知，如下示例所示：



通知图标显示通知左侧—此映像的带圆圈“i”具有 alpha 通道，以便 Android 可以绘制它后面灰色圆圈为背景。此外可以提供一个图标，而无需 alpha 通道。若要以图标形式显示照片的图像，请参阅[大型图标格式](#)本主题中更高版本。

时间戳设置自动进行，但可以替代此设置通过调用[集通知生成器](#)方法。例如，下面的代码示例将时间戳设置为当前时间：

```
builder.SetWhen (Java.Lang.JavaSystem.CurrentTimeMillis());
```

启用声音和振动

如果想要在通知还播放声音，则可以调用通知生成器[SetDefaults](#)方法并传入 `NotificationDefaults.Sound` 标志：

```
// Instantiate the notification builder and enable sound:
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle ("Sample Notification")
    .setContentText ("Hello World! This is my first notification!")
    .setDefaults (NotificationDefaults.Sound)
    .setSmallIcon (Resource.Drawable.ic_notification);
```

此调用 `SetDefaults` 将导致设备发布通知时播放声音。如果你想要振动而不播放声音的设备，可以将传递 `NotificationDefaults.Vibrate` 到 `SetDefaults`。如果你想要播放的声音和振动设备的设备，可以将传递到这两个标志 `SetDefaults`：

```
builder.SetDefaults (NotificationDefaults.Sound | NotificationDefaults.Vibrate);
```

如果您启用声音而无需指定要播放声音，Android 使用默认系统通知声音。但是，您可以通过调用通知生成器，都将播放的声音 `SetSound` 方法。例如，播放警报声音通知（而不是默认的通知声音），但您可以获取的 URI 警报中的声音 `RingtoneManager` 并将其传递给 `SetSound`：

```
builder.SetSound (RingtoneManager.GetDefaultUri(RingtoneType.Alarm));
```

或者，您可以使用系统默认铃声声音通知：

```
builder.SetSound (RingtoneManager.GetDefaultUri(RingtoneType.Ringtone));
```

创建通知对象后，就可以通知对象上设置通知属性（而不是将其配置为预先通过 `NotificationCompat.Builder` 方法）。例如，而不是调用 `SetDefaults` 方法，以使振动通知，可以直接修改该通知的位标志 `默认` 属性：

```
// Build the notification:
Notification notification = builder.Build();

// Turn on vibrate:
notification.Defaults |= NotificationDefaults.Vibrate;
```

此示例会导致设备振动时发布通知。

正在更新通知

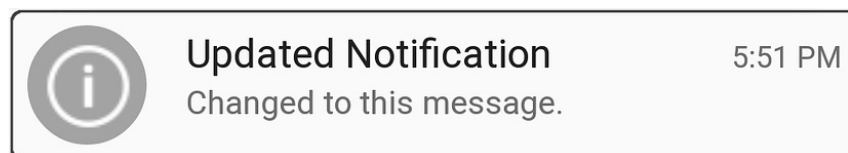
如果你想要更新通知的内容，已发布后，可以重复使用现有 `NotificationCompat.Builder` 对象来创建新的通知对象，并发布此具有标识符一次通知的通知。例如：

```
// Update the existing notification builder content:
builder.SetContentTitle ("Updated Notification");
builder.SetContentText ("Changed to this message.");

// Build a notification object with updated content:
notification = builder.Build();

// Publish the new notification with the existing ID:
notificationManager.Notify (notificationId, notification);
```

在此示例中，现有 `NotificationCompat.Builder` 对象用于创建新的通知对象具有不同的标题和消息。使用标识符前一次通知，发布新的通知对象，这会更新之前发布通知的内容：



重复使用前一次通知的正文—仅标题和通知更改时通知显示通知抽屉中的文本。标题文本从“示例通知”更改为“更新通知”，并从“Hello World 消息文本更改！这是我第一次通知！”为“已更改为此邮件”。

通知保持可见，直到发生下列三个操作之一：

- 在用户关闭通知（或点击 *全部清除*）。
- 应用程序会调用 `NotificationManager.Cancel`，传入通知已发布时分配的唯一通知 ID。
- 应用程序调用 `NotificationManager.CancelAll`。

有关更新 Android 通知的详细信息，请参阅[修改通知](#)。

从通知启动活动

在 Android 中，很常见的与之关联的通知操作-用户点击通知时启动的活动。此活动可以位于另一个应用程序或甚至在另一个任务中。若要将操作添加到一条通知，您创建[PendingIntent](#)对象，并将关联 [PendingIntent](#) 与通知。一个 [PendingIntent](#) 是一种特殊的允许接收方应用程序的发送应用程序的权限运行预定义的一段代码的意图。当用户点击该通知时，Android 启动指定的活动 [PendingIntent](#)。

以下代码片段演示如何创建一个通知，其中 [PendingIntent](#) 这将启动活动的发起应用程序， [MainActivity](#)：

```
// Set up an intent so that tapping the notifications returns to this app:
Intent intent = new Intent (this, typeof(MainActivity));

// Create a PendingIntent; we're only using one PendingIntent (ID = 0):
const int pendingIntentId = 0;
PendingIntent pendingIntent =
    PendingIntent.GetActivity (this, pendingIntentId, intent, PendingIntentFlags.OneShot);

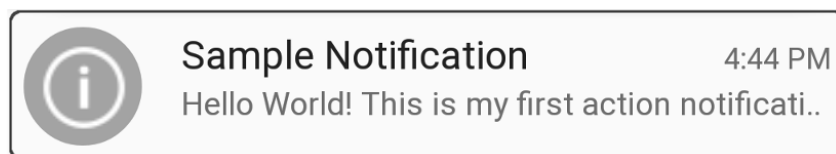
// Instantiate the builder and set notification elements, including pending intent:
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .SetContentIntent (pendingIntent)
    .SetContentTitle ("Sample Notification")
    .SetContentText ("Hello World! This is my first action notification!")
    .SetSmallIcon (Resource.Drawable.ic_notification);

// Build the notification:
Notification notification = builder.Build();

// Get the notification manager:
NotificationManager notificationManager =
    GetSystemService (Context.NotificationService) as NotificationManager;

// Publish the notification:
const int notificationId = 0;
notificationManager.Notify (notificationId, notification);
```

此代码是非常类似于在前面部分中，不同之处在于通知代码 [PendingIntent](#) 将添加到通知对象。在此示例中，[PendingIntent](#) 是与原始应用的活动关联之前将它传递到通知生成器[SetContentIntent](#)方法。[PendingIntentFlags.OneShot](#) 标志传递给 [PendingIntent.GetActivity](#) 方法，以便 [PendingIntent](#) 只能使用一次。此代码运行时，将显示以下通知：



点击此通知将返回到原始活动用户。

在生产应用中，您的应用程序必须处理back 堆栈当用户按回通知活动中的按钮（如果您不熟悉 Android 任务和 back 堆栈，请参阅[任务和后退堆栈](#)）。在大多数情况下，在向后导航出通知活动应返回用户退出应用程序并返回到主屏幕。若要管理 back 堆栈中，您的应用程序使用[TaskStackBuilder](#)类，以创建 [PendingIntent](#) back 堆栈使用。

现实世界的另一个注意事项是，原始活动可能需要将数据发送到通知活动。例如，通知可能表示文本消息已到达，并通知活动（消息查看屏幕），需要向用户显示消息的消息的 ID。创建的活动 [PendingIntent](#) 可以使用[Intent.PutExtra](#)方法将数据（例如，字符串）添加到意向，以便此数据将传递给通知活动。

下面的代码示例演示了如何使用 [TaskStackBuilder](#) 用于管理 back 堆栈中，和它包括如何将单个消息字符串发送到名为通知活动的一个示例 [SecondActivity](#)：


```
// Setup an intent for SecondActivity:
Intent secondIntent = new Intent (this, typeof(SecondActivity));

// Pass some information to SecondActivity:
secondIntent.PutExtra ("message", "Greetings from MainActivity!");

// Create a task stack builder to manage the back stack:
TaskStackBuilder stackBuilder = TaskStackBuilder.Create(this);

// Add all parents of SecondActivity to the stack:
stackBuilder.AddParentStack (Java.Lang.Class.FromType (typeof (SecondActivity)));

// Push the intent that starts SecondActivity onto the stack:
stackBuilder.AddNextIntent (secondIntent);

// Obtain the PendingIntent for launching the task constructed by
// stackbuilder. The pending intent can be used only once (one shot):
const int pendingIntentId = 0;
PendingIntent pendingIntent =
    stackBuilder.GetPendingIntent (pendingIntentId, PendingIntentFlags.OneShot);

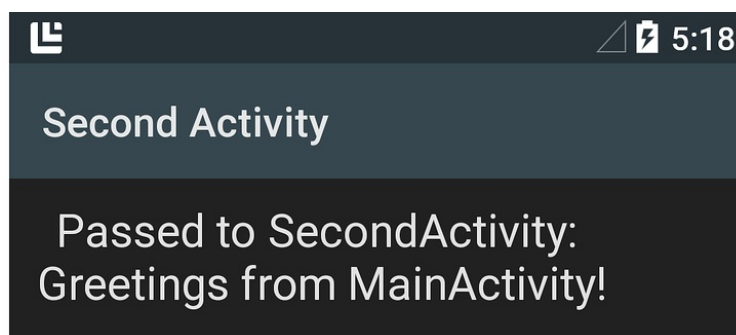
// Instantiate the builder and set notification elements, including
// the pending intent:
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .SetContentIntent (pendingIntent)
    .SetContentTitle ("Sample Notification")
    .SetContentText ("Hello World! This is my second action notification!")
    .SetSmallIcon (Resource.Drawable.ic_notification);

// Build the notification:
Notification notification = builder.Build();

// Get the notification manager:
NotificationManager notificationManager =
    GetSystemService (Context.NotificationService) as NotificationManager;

// Publish the notification:
const int notificationId = 0;
notificationManager.Notify (notificationId, notification);
```

在此代码示例中，应用程序包含两个活动：`MainActivity`（包含上面的通知代码）和 `SecondActivity`，用户点击该通知后将看到的屏幕。当运行此代码时，会看到简单的通知（类似于前面的示例）。点击该通知，用户会 `SecondActivity` 屏幕：



字符串消息 (传入的意图 `PutExtra` 方法) 中检索 `SecondActivity` 通过这行代码：

```
// Get the message from the intent:
string message = Intent.Extras.GetString ("message", "");
```

中会显示此检索到的消息，"Greetings 从 MainActivity ！、" `SecondActivity` 屏幕上，如上面的屏幕截图中所示。当用户按回按钮，同时在 `SecondActivity`，导航会导致退出应用程序并返回到前面的应用程序的启动屏幕。

有关创建挂起意向的详细信息，请参阅[PendingIntent](#)。

除了基本通知

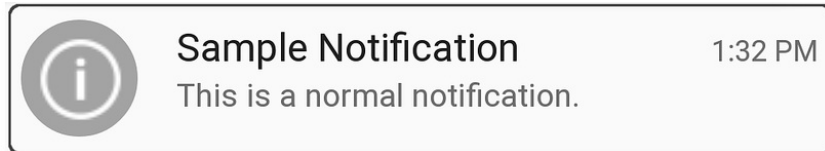
通知默认设置为简单的基本布局格式在 Android 中，但可以通过使用更多增强此基本格式

`NotificationCompat.Builder` 方法调用。在本部分中，您将学习如何将大型照片图标添加到你的通知，并将看到有关如何创建扩展的布局通知的示例。

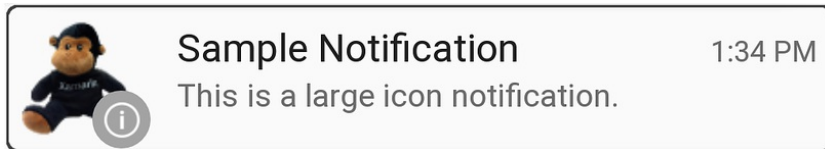
大图标格式

Android 通知通常显示发起应用程序的图标（在左侧和右侧的通知）。但是，通知可以显示图像或照片（大图标）而不是标准的小图标。例如，消息传送应用程序可以显示照片的发件人，而不是应用程序图标。

下面是基本的 Android 5.0 通知的示例—显示只有的小型应用图标：



下面是修改后的通知显示大图标的屏幕截图—它使用 Xamarin 代码 monkey 的映像中创建的图标：



请注意，如果以大图标格式显示一条通知，小型应用程序图标将显示大图标右下角上的徽章。

若要为大图标在通知中使用的映像，请调用通知生成器[SetLargeIcon](#)方法并传入的图像的位图。与不同

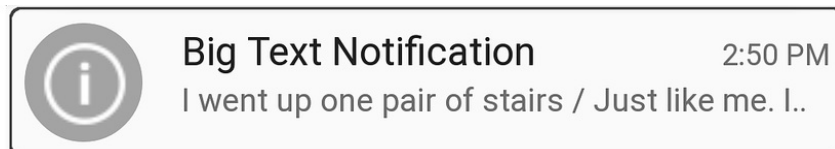
`SetSmallIcon`，`SetLargeIcon` 只接受一个位图。若要将转换位图的图像文件，请使用[BitmapFactory](#)类。例如：

```
builder.SetLargeIcon (BitmapFactory.DecodeResource (Resources, Resource.Drawable.monkey_icon));
```

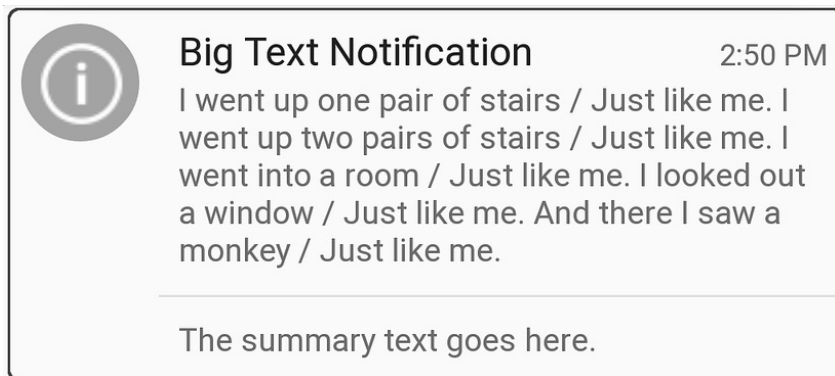
此示例代码随即会打开在映像文件 **Resources/drawable/monkey_icon.png**，将其转换为位图，并将传递到生成的位图 `NotificationCompat.Builder`。通常情况下，源图像分辨率大于的小图标—但不是大得多。太大的图像可能会导致不必要的调整大小操作可能会延迟通知的发布。

大文本样式

大文本样式是扩展的布局模板，用于在通知中显示长消息。所有扩展的布局与通知相似，大文本通知中的初始显示 compact 显示格式：



在这种格式，仅消息的一段摘录显示，两个句点来终止。当用户向下拖动的通知时，它将展开以显示整个通知消息：



此扩展的布局格式还包括摘要文本底部的通知。最大高度大文本通知是 256 个分发点。

若要创建大文本通知，您实例化 `NotificationCompat.Builder` 对象，与之前一样，然后实例化并添加 `BigTextStyle` 对象传递给 `NotificationCompat.Builder` 对象。下面是一个示例：

```
// Instantiate the Big Text style:
Notification.BigTextStyle textStyle = new Notification.BigTextStyle();

// Fill it with text:
string longTextMessage = "I went up one pair of stairs.";
longTextMessage += " / Just like me. ";
//...
textStyle.BigText (longTextMessage);

// Set the summary text:
textStyle.SetSummaryText ("The summary text goes here.");

// Plug this style into the builder:
builder.SetStyle (textStyle);

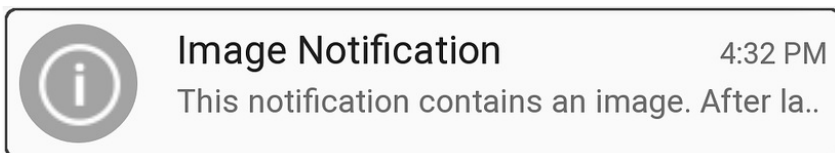
// Create the notification and publish it ...
```

在此示例中，消息文本和摘要文本存储在 `BigTextStyle` 对象 (`textStyle`) 传递给之前 `NotificationCompat.Builder`。

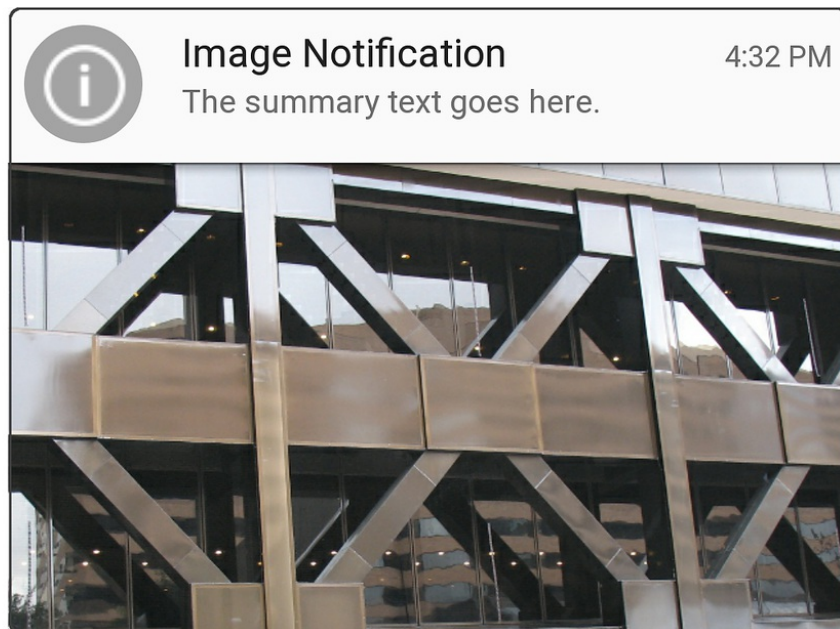
图像样式

图像样式 (也称为大局样式) 是一种扩展的通知格式，可用于通知的正文中显示的图像。例如，屏幕截图应用或照片应用程序可以使用图像捕获要向用户提供的最后一个通知样式图像的通知。请注意，最大高度图像通知是 256 个 dp – Android 将调整图像以适合此最大高度限制，可用内存的限制范围内的大小。

例如所有扩展布局通知图像通知第一次显示紧凑的形式显示随附的消息文本的一段摘录：



当用户拖动图像通知，它将展开以显示图像。例如，下面是前一次通知的扩展的版本：



请注意，当通知显示在紧凑的格式，它将显示通知文本（传递给通知生成器的文本 `setContentText` 方法，如前面所示）。但是，通知扩展以显示图像时，它将显示在图像上方的摘要文本。

若要创建 **图像通知**，您实例化 `NotificationCompat.Builder` 对象与之前一样，然后创建并插入 `BigPictureStyle` 对象插入 `NotificationCompat.Builder` 对象。例如：

```
// Instantiate the Image (Big Picture) style:
Notification.BigPictureStyle picStyle = new Notification.BigPictureStyle();

// Convert the image to a bitmap before passing it into the style:
picStyle.BigPicture (BitmapFactory.decodeResource (Resources, Resource.Drawable.x_bldg));

// Set the summary text that will appear with the image:
picStyle.SetSummaryText ("The summary text goes here.");

// Plug this style into the builder:
builder.SetStyle (picStyle);

// Create the notification and publish it ...
```

像 `setLargeIcon` 方法 `NotificationCompat.Builder`，则 `BigPicture` 方法 `BigPictureStyle` 需要你想要通知的正文中显示的图像的位图。在此示例中，`decodeResource` 方法 `BitmapFactory` 读取图像文件位于 `Resources/drawable/x_bldg.png` 并将其转换成位图。

此外可以作为资源显示不会打包到的映像。例如，下面的示例代码从本地的 SD 卡加载图像并将其显示 **图像通知**：

```
// Using the Image (Big Picture) style:
Notification.BigPictureStyle picStyle = new Notification.BigPictureStyle();

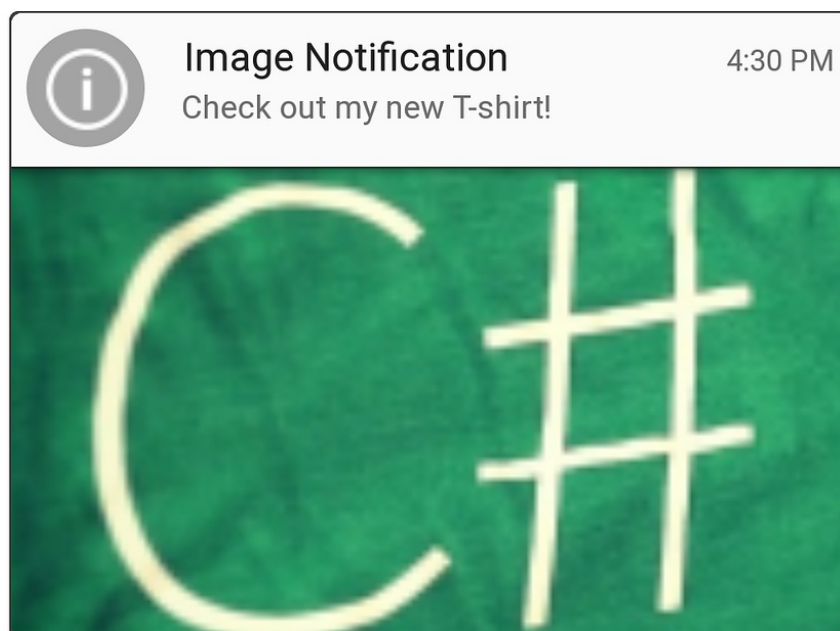
// Read an image from the SD card, subsample to half size:
BitmapFactory.Options options = new BitmapFactory.Options();
options.InSampleSize = 2;
string imagePath = "/sdcard/Pictures/my-tshirt.jpg";
picStyle.BigPicture (BitmapFactory.DecodeFile (imagePath, options));

// Set the summary text that will appear with the image:
picStyle.SetSummaryText ("Check out my new T-Shirt!");

// Plug this style into the builder:
builder.SetStyle (picStyle);

// Create notification and publish it ...
```

在此示例中，图像文件位于 `/sdcard/Pictures/my-tshirt.jpg` 是加载、大小调整回其原始大小的一半，然后转换为在通知中使用的位图：

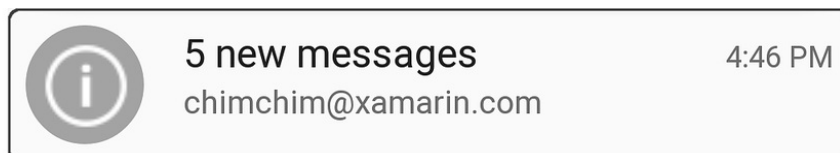


如果你在事先不知道图像文件的大小，则包装对调用一个好办法 `BitmapFactory.DecodeFile` 中的异常处理程序—`OutOfMemoryError` 如果图像太大，可能会引发异常若要调整大小的 android。

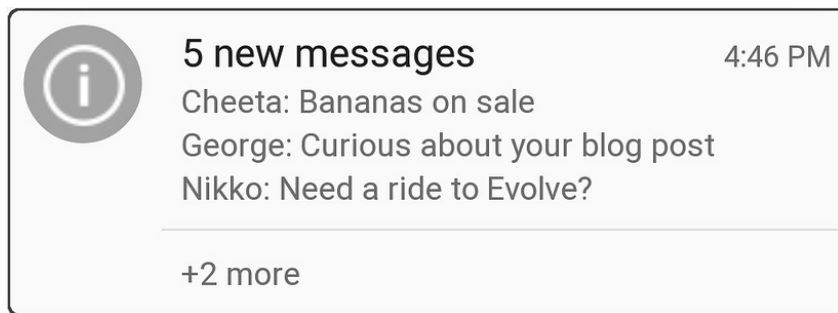
有关加载和解码大型位图图像的详细信息，请参阅 [高效加载大位图](#)。

收件箱样式

收件箱格式是一个扩展的布局模板，用于通知的正文中显示的文本（例如电子邮件收件箱摘要）单独的行。**收件箱格式通知**第一次紧凑的形式显示：



当用户向下拖动的通知时，它将展开以显示下面的屏幕截图中所示的电子邮件摘要：



若要创建收件箱通知，您实例化 `NotificationCompat.Builder` 对象，与之前一样，并添加 `InboxStyle` 对象传递给 `NotificationCompat.Builder`。下面是一个示例：

```
// Instantiate the Inbox style:
Notification.InboxStyle inboxStyle = new Notification.InboxStyle();

// Set the title and text of the notification:
builder.SetContentTitle ("5 new messages");
builder.SetContentText ("chimchim@xamarin.com");

// Generate a message summary for the body of the notification:
inboxStyle.AddLine ("Cheeta: Bananas on sale");
inboxStyle.AddLine ("George: Curious about your blog post");
inboxStyle.AddLine ("Nikko: Need a ride to Evolve?");
inboxStyle.SetSummaryText ("+2 more");

// Plug this style into the builder:
builder.SetStyle (inboxStyle);
```

若要将新的文本行添加到通知正文，请调用 `AddLine` 方法 `InboxStyle` 对象 (的最大高度收件箱通知是 256 个分发点)。请注意，与不同大文本样式收件箱样式支持通知正文中的各行文本。

此外可以使用收件箱样式扩展格式显示独立的文本行所需的任何通知。例如，收件箱通知样式可用于合并多个挂起通知到盖礁砵 – 可以更新单个收件箱样式与新的通知通知内容的行 (请参阅[更新通知](#)上方)，而不是不是生成一个连续的新，主要是类似的通知流。

配置元数据

`NotificationCompat.Builder` 包含可调用以设置你的通知，例如优先级、可见性和类别的元数据的方法。Android 使用此信息—以及用户首选项设置—来确定如何以及何时可以显示通知。

优先级设置

运行在 Android 7.1 和更低版本的应用需要直接在通知本身上设置的优先级。发布通知时，通知的优先级设置确定的两个结果：

- 其中会显示通知，相对于其他通知。例如，高优先级的通知时呈现上面较低优先级的通知在通知抽屉中，而不考虑每个通知已发布。
- 是否在 Heads-up 通知格式 (Android 5.0 及更高版本) 中显示通知。仅高并最大优先级的通知显示为危险警告通知。

Xamarin.Android 定义以下枚举来设置通知优先级：

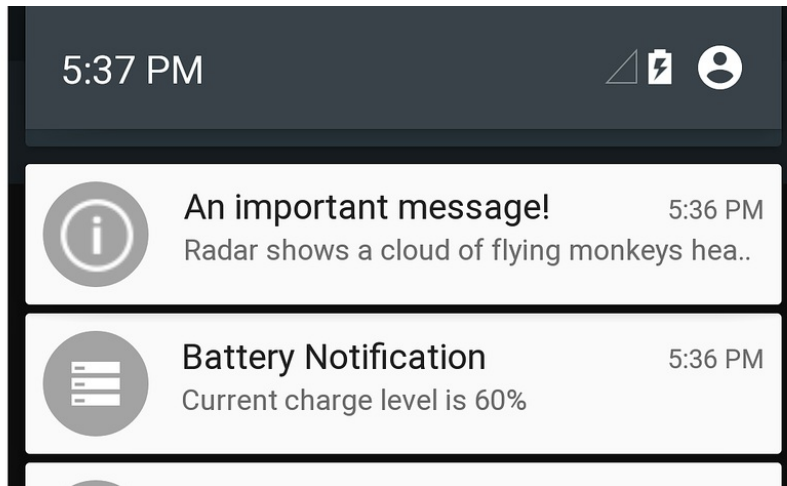
- `NotificationPriority.Max` – 提醒用户的紧急或关键条件 (例如，传入呼叫、打开的打开方向或紧急警报)。在 Android 5.0 和更高版本设备，最高优先级别通知平视格式显示。
- `NotificationPriority.High` – 将通知用户的重要事件 (如重要电子邮件或实时聊天消息抵达)。在 Android 5.0 和更高版本设备，危险警告格式显示高优先级的通知。

- `NotificationPriority.Default` – 通知用户具有的重要性中等级别的条件。
- `NotificationPriority.Low` – 有关非紧急，用户必须是明智的（例如，软件更新提醒或社交网络更新）的信息。
- `NotificationPriority.Min` – 用户通知时，才的背景信息查看通知（例如，位置或天气信息）。

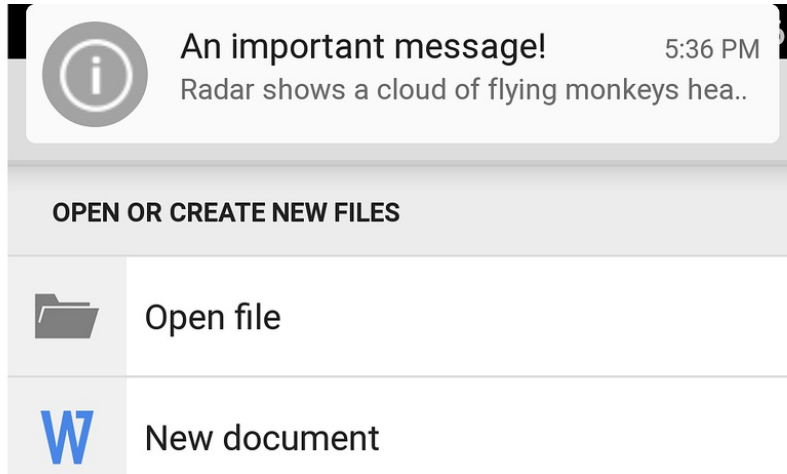
若要设置通知的优先级，请调用 `SetPriority` 方法的 `NotificationCompat.Builder` 对象，传入的优先级别。例如：

```
builder.SetPriority (NotificationPriority.High);
```

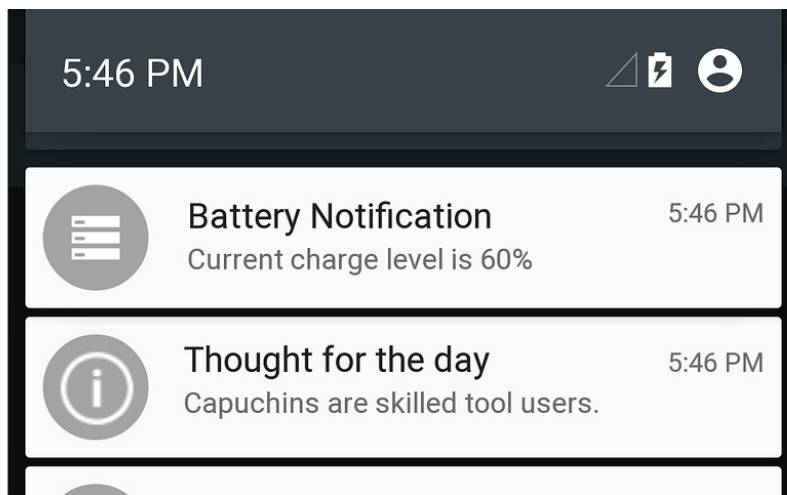
在下面的示例、高优先级通知、“重要邮件！”显示在通知抽屉的顶部：



由于这是优先级较高的通知，它还显示为 Android 5.0 中用户的当前活动屏幕上方的危险警告通知：



在下一步的示例中，低优先级“的那一天认为”通知显示在优先级较高的电池电量级别通知：



因为“一天的思想”通知是一个低优先级的通知，Android 不会显示它 Heads-up 格式。

NOTE

在 Android 8.0 及更高版本，则通知通道和用户设置的优先级将确定通知的优先级。

可见性设置

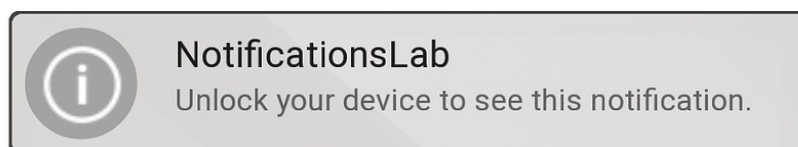
从开始 Android 5.0 可见性设置才可用来控制安全锁定屏幕上显示的内容量通知。Xamarin.Android 定义以下枚举来设置通知可见性：

- `NotificationVisibility.Public` – 安全锁定屏幕上显示通知的完整内容。
- `NotificationVisibility.Private` – 仅基本显示的信息（例如通知图标和把它发布的应用的名称），在安全锁定屏幕上但通知的详细信息的其余部分隐藏。所有通知都默认为 `NotificationVisibility.Private`。
- `NotificationVisibility.Secret` – 不会显示在安全锁定屏幕上，甚至不通知图标。仅在用户解锁设备后才能提供通知内容。

若要设置通知，应用程序调用的可见性 `SetVisibility` 方法的 `NotificationCompat.Builder` 对象，传入的可见性设置。例如，对此调用 `SetVisibility` 使通知 `Private`：

```
builder.SetVisibility (NotificationVisibility.Private);
```

当 `Private` 发布通知，安全锁定屏幕上显示的名称和应用程序的图标。而不是通知消息，用户将看到“解锁你的设备以查看此通知”：



在此示例中，**NotificationsLab** 发起应用程序的名称。此版本经过修订的通知时才出现在锁定屏幕安全（即，通过 PIN、图案或密码保护）– 锁定屏幕不安全，通知的完整内容是否可用在锁定屏幕上。

类别设置

从 Android 5.0 开始，预定义的类别是可用于进行排名和筛选通知。Xamarin.Android 提供了以下枚举这些类别：

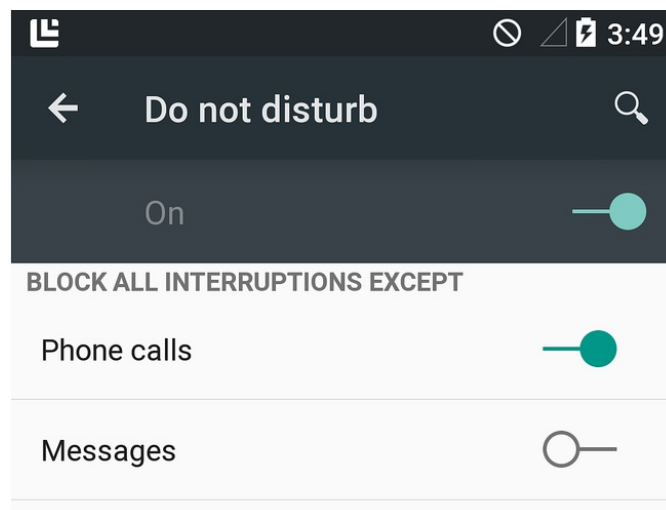
- `Notification.CategoryCall` – 传入的电话呼叫。
- `Notification.CategoryMessage` – 传入的文本消息。

- `Notification.CategoryAlarm` – 警报条件或计时器过期。
- `Notification.CategoryEmail` – 传入的电子邮件消息。
- `Notification.CategoryEvent` – 日历事件。
- `Notification.CategoryPromo` – 促销邮件或广告。
- `Notification.CategoryProgress` – 后台操作的进度。
- `Notification.CategorySocial` – 社交网络的更新。
- `Notification.CategoryError` – 失败的后台操作或身份验证进程。
- `Notification.CategoryTransport` – 媒体播放更新。
- `Notification.CategorySystem` – 保留供系统使用（系统或设备状态）。
- `Notification.CategoryService` – 指示后台服务正在运行。
- `Notification.CategoryRecommendation` – 建议消息，与当前正在运行的应用。
- `Notification.CategoryStatus` – 有关设备的信息。

当通知进行排序时，通知的优先级将优先于其类别设置。例如，高优先级通知将显示为平视即使其所属 `Promo` 类别。若要设置通知的类别，请调用 `SetCategory` 方法的 `NotificationCompat.Builder` 对象，传入的类别设置。例如：

```
builder.SetCategory (Notification.CategoryCall);
```

请勿打扰 功能（Android 5.0 中的新增功能）筛选器基于类别的通知。例如，不会妨碍屏幕中设置免除通知用户进行电话呼叫和消息：



当用户配置 *请勿打扰* 若要阻止所有中断除了电话呼叫（如上面的屏幕截图中所示），Android 允许的类别设置通知 `Notification.CategoryCall` 设备时显示处于 *请勿打扰* 模式。请注意，`Notification.CategoryAlarm` 中，永远不会被阻止通知 *请勿打扰* 模式。

[LocalNotifications](#) 示例演示如何使用 `NotificationCompat.Builder` 以后动从通知第二个活动。此代码示例所述在 [Xamarin.Android 中使用本地通知](#) 演练。

通知样式

若要创建大文本，映像，或收件箱样式与通知 `NotificationCompat.Builder`，您的应用程序必须使用这些样式的兼容性版本。例如，若要使用大文本样式、实例化 `NotificationCompat.BigTextStyle`：

```
NotificationCompat.BigTextStyle textStyle = new NotificationCompat.BigTextStyle();

// Plug this style into the builder:
builder.SetStyle (textStyle);
```

同样，可以使用您的应用程序 `NotificationCompat.InboxStyle` 和 `NotificationCompat.BigPictureStyle` 有关收件箱和图像分别设计样式。

通知的优先级和类别

`NotificationCompat.Builder` 支持 `SetPriority` 方法（可从 Android 4.1）。但是，`SetCategory` 方法是不受 `NotificationCompat.Builder` 由于类别是 Android 5.0 中引入了新通知的元数据系统的一部分。

若要支持较旧版本的 Android，where `SetCategory` 是不可用，你的代码可以检查在运行时有条件地调用的 API 级别 `SetCategory` API 级别时为等于或大于 Android 5.0（API 级别 21）：

```
if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop) {
    builder.SetCategory (Notification.CategoryEmail);
}
```

在此示例中，应用程序的目标框架设置为 Android 5.0 和最低 **Android 版本** 设置为 **Android 4.1 (API 级别为 16)**。因为 `SetCategory` 是在 API 级别 21 和更高版本中可用，此示例代码将调用 `SetCategory` 仅当有可用—不会调用 `SetCategory` API 级别时是小于 21。

锁定屏幕可见性

因为 Android 不支持 Android 5.0（API 级别 21）之前，的锁定屏幕通知 `NotificationCompat.Builder` 不支持 `SetVisibility` 方法。如上文所述的 `SetCategory`，你的代码可以检查在运行时和调用的 API 级别 `SetVisibility` 它时才可用：

```
if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop) {
    builder.SetVisibility (Notification.Public);
}
```

总结

本文介绍了如何在 Android 中创建本地通知。所述的通知剖析，它介绍了如何使用 `NotificationCompat.Builder` 若要创建通知，如何在大图标样式通知大文本，图像和收件箱格式、如何设置通知的元数据设置，如优先级、可见性和类别，以及如何启动通知中的活动。本文还介绍了这些通知设置如何使用新的危险警告，锁定屏幕和请勿打扰 Android 5.0 中引入的功能。最后，您学习了如何使用 `NotificationCompat.Builder` 以保持与早期版本的 Android 通知兼容。

设计适用于 Android 的通知的指南，请参阅[通知](#)。

相关链接

- [NotificationsLab（示例）](#)
- [LocalNotifications（示例）](#)
- [在 Android 演练中的本地通知](#)
- [通知用户](#)
- [通知](#)
- [NotificationManager](#)
- [NotificationCompat.Builder](#)
- [PendingIntent](#)

演练-在 Xamarin.Android 中使用本地通知

2018/10/26 • [Edit Online](#)

本演练演示如何在 Xamarin.Android 应用程序中使用本地通知。它演示了创建和发布本地通知的基础知识。当用户单击通知区域中的通知时，在启动第二个活动。

概述

在本演练中，我们将创建 Android 应用程序，当用户单击按钮在活动中的引发的通知。当用户单击通知时，它将启动的第二个活动，显示用户已单击的第一个活动中的按钮的次数。

下面的屏幕截图演示了此应用程序的一些示例：



NOTE

本指南重点 [NotificationCompat Api](#) 从 [Android 支持库](#)。这些 Api 将确保最大值向后兼容到 Android 4.0 (API 级别 14)。

创建项目

若要开始，让我们使用以下工具创建新的 Android 项目 **Android** 应用模板。让我们称此项目 **LocalNotifications**。(如果不熟悉创建 Xamarin.Android 项目，请参阅 [Hello, Android](#)。)

编辑资源文件 **values/Strings.xml**，使其包含两个额外的字符串资源时就可以创建通知通道时使用：

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
  <string name="Hello">Hello World, Click Me!</string>
  <string name="ApplicationName">Notifications</string>

  <string name="channel_name">Local Notifications</string>
  <string name="channel_description">The count from MainActivity.</string>
</resources>
```

添加 Android.Support.V4 NuGet 包

在本演练中，我们将使用 `NotificationCompat.Builder` 来构建我们本地通知。中所述 [本地通知](#)，我们必须包

括 **Android 支持库 v4** 我们的项目中的 NuGet, 若要使用 `NotificationCompat.Builder` 。

接下来, 让我们编辑 **MainActivity.cs** 并添加以下 `using` 语句, 以便在类型 `Android.Support.V4.App` 可供我们的代码:

```
using Android.Support.V4.App;
```

此外, 我们必须使我们将使用编译器清楚地 `Android.Support.V4.App` 新版 `TaskStackBuilder` 而不是 `Android.App` 版本。以下代码添加到 `using` 语句, 若要解决不明确性:

```
using TaskStackBuilder = Android.Support.V4.App.TaskStackBuilder;
```

创建通知通道

接下来, 将方法添加到 `MainActivity`, 将创建通知通道 (如有必要):

```
void CreateNotificationChannel()
{
    if (Build.VERSION.SdkInt < BuildVersionCodes.O)
    {
        // Notification channels are new in API 26 (and not a part of the
        // support library). There is no need to create a notification
        // channel on older versions of Android.
        return;
    }

    var name = Resources.GetString(Resource.String.channel_name);
    var description = GetString(Resource.String.channel_description);
    var channel = new NotificationChannel(CHANNEL_ID, name, NotificationImportance.Default)
    {
        Description = description
    };

    var notificationManager = (NotificationManager) GetSystemService(NotificationService);
    notificationManager.CreateNotificationChannel(channel);
}
```

更新 `OnCreate` 方法来调用这种新方法:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    CreateNotificationChannel();
}
```

定义通知 ID

我们将需要我们的通知和通知通道的唯一 ID。让我们编辑 **MainActivity.cs** 并添加以下静态实例变量 `MainActivity` 类:

```
static readonly int NOTIFICATION_ID = 1000;
static readonly string CHANNEL_ID = "location_notification";
internal static readonly string COUNT_KEY = "count";
```

添加代码以生成通知

接下来, 我们需要创建一个新的事件处理程序, 为按钮 `Click` 事件。添加以下方法 `MainActivity`:

```

void ButtonOnClick(object sender, EventArgs eventArgs)
{
    // Pass the current button press count value to the next activity:
    var valuesForActivity = new Bundle();
    valuesForActivity.PutInt(COUNT_KEY, count);

    // When the user clicks the notification, SecondActivity will start up.
    var resultIntent = new Intent(this, typeof(SecondActivity));

    // Pass some values to SecondActivity:
    resultIntent.PutExtras(valuesForActivity);

    // Construct a back stack for cross-task navigation:
    var stackBuilder = TaskStackBuilder.Create(this);
    stackBuilder.AddParentStack(Class.FromType(typeof(SecondActivity)));
    stackBuilder.AddNextIntent(resultIntent);

    // Create the PendingIntent with the back stack:
    var resultPendingIntent = stackBuilder.GetPendingIntent(0, (int) PendingIntentFlags.UpdateCurrent);

    // Build the notification:
    var builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .SetAutoCancel(true) // Dismiss the notification from the notification area when the user
        clicks on it
        .SetContentIntent(resultPendingIntent) // Start up this activity when the user clicks the
        intent.
        .SetContentTitle("Button Clicked") // Set the title
        .SetNumber(count) // Display the count in the Content Info
        .SetSmallIcon(Resource.Drawable.ic_stat_button_click) // This is the icon to display
        .SetContentText($"The button has been clicked {count} times."); // the message to display.

    // Finally, publish the notification:
    var notificationManager = NotificationManagerCompat.From(this);
    notificationManager.Notify(NOTIFICATION_ID, builder.Build());

    // Increment the button press count:
    count++;
}

```

`OnCreate` `MainActivity` 方法必须进行调用以创建通知通道并分配 `ButtonOnClick` 方法 `Click` 按钮（替换该模板提供的委托事件处理程序）的事件：

```

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    CreateNotificationChannel();

    // Display the "Hello World, Click Me!" button and register its event handler:
    var button = FindViewById<Button>(Resource.Id.MyButton);
    button.Click += ButtonOnClick;
}

```

创建第二个活动

现在，我们需要创建 Android 将显示在用户单击通知时的另一个活动。将另一个 Android 活动添加到你的项目称为 **SecondActivity**。打开 **SecondActivity.cs** 并将其内容替换此代码：

```

using System;
using Android.App;
using Android.OS;
using Android.Widget;

namespace LocalNotifications
{
    [Activity(Label = "Second Activity")]
    public class SecondActivity : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Get the count value passed to us from MainActivity:
            var count = Intent.Extras.GetInt(MainActivity.COUNT_KEY, -1);

            // No count was passed? Then just return.
            if (count <= 0)
            {
                return;
            }

            // Display the count sent from the first activity:
            SetContentView(Resource.Layout.Second);
            var textView = FindViewById<TextView>(Resource.Id.textView1);
            textView.Text = $"You clicked the button {count} times in the previous activity.";
        }
    }
}

```

我们还必须创建适用于的资源布局**SecondActivity**。添加一个新**Android 布局**到名为你的项目文件**Second.axml**。编辑**Second.axml**并粘贴以下布局代码：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:minWidth="25px"
    android:minHeight="25px">
    <TextView
        android:text=""
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView1" />
</LinearLayout>

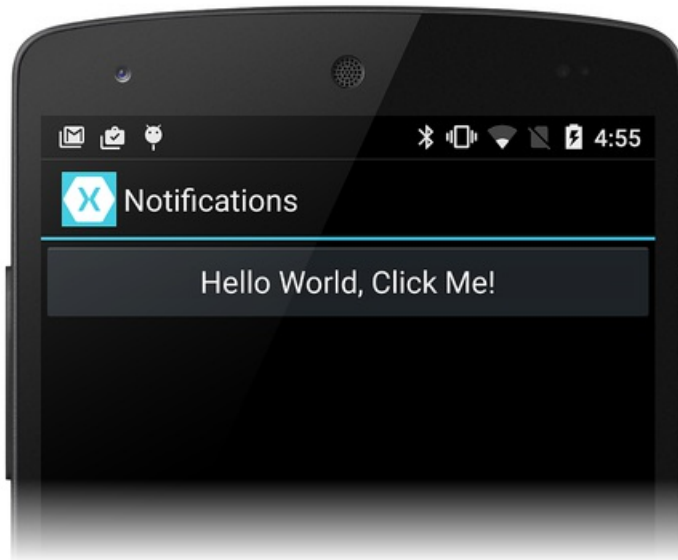
```

添加通知图标

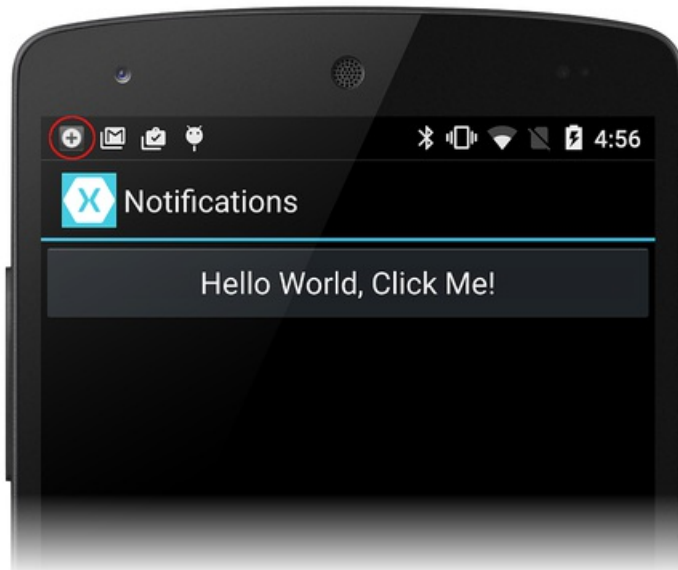
最后，添加启动通知时将出现在通知区域中的小图标。可以将复制[此图标](#)到你的项目或创建你自己的自定义图标。图标文件命名**ic_stat_按钮_click.png**并将其复制到资源/**drawable**文件夹。请记住使用**添加 > 现有项...** 此图标文件包含在项目中。

运行此应用程序

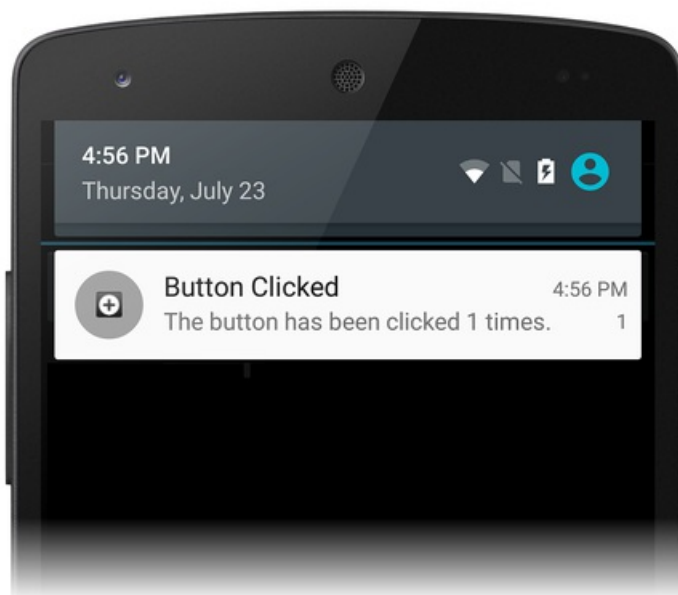
生成并运行应用程序。您应看到类似于以下屏幕截图的第一个活动：



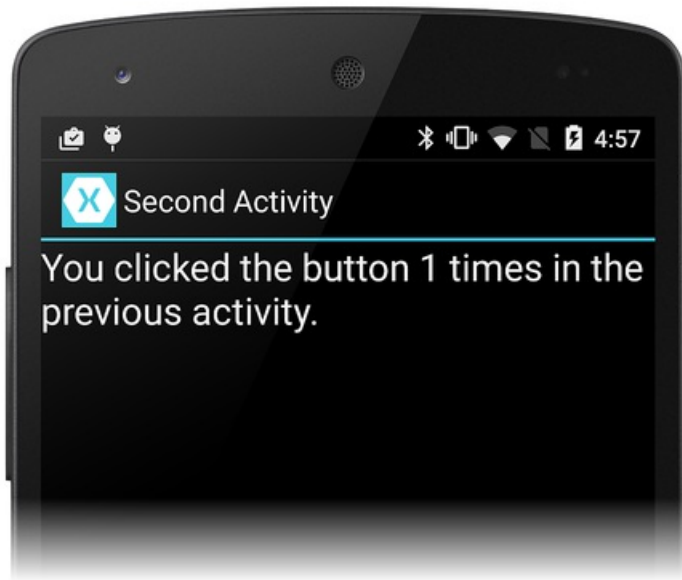
单击按钮时，您会注意到该通知的小图标显示在通知区域中：



如果向下轻扫并公开通知抽屉，你应该会看到通知：



如果您单击该通知，它应会消失，并且我们其他的活动应启动-看起来有点像下面的屏幕截图：



祝贺你！现在您已经完成了 Android 本地通知演练，您必须可以引用的工作示例。有很多通知的详细信息比我们已经演示了在这里，因此如果要了解详细信息，看一看[通知上的 Google 文档](#)。

总结

使用本演练 `NotificationCompat.Builder` 来创建和显示通知。它介绍了如何启动作为一种方法来响应用户交互并发出通知，第二个活动的基本示例，它从第一个活动中演示的数据传输到第二个活动。

相关链接

- [LocalNotifications（示例）](#)
- [Android Oreo 通知通道](#)
- [通知](#)
- [NotificationManager](#)
- [NotificationCompat.Builder](#)
- [PendingIntent](#)

触控和手势在 Xamarin.Android 中

2018/10/26 • [Edit Online](#)

在许多现有的设备上的触摸屏允许用户快速高效地与设备交互自然和直观的方式。这种交互不只限于简单触摸检测-可以使用手势以及。例如，捏合缩放手势是一个非常常见的示例通过收缩手势用户可以放大或缩小的两根手指与屏幕的一部分。本指南检查触摸屏输入，并在 Android 中的笔势。

触控概述

iOS 和 Android 是在它们处理触控的方式类似。都可以支持多点触控的很多的联系点在屏幕上的和复杂的笔势。本指南介绍一些概念中的相似之处以及实现触摸的 particularities，并在这两个平台上笔势。

Android 使用 `MotionEvent` 对象封装触控数据和要侦听的收尾工作了视图对象上的方法。

除了捕获触摸屏输入数据，iOS 和 Android 提供种解释模式的到手势收尾工作了。这些笔势识别器又用于解释特定于应用程序的命令，例如图像的旋转或页面的一个轮次。Android 提供了少量的受支持的手势，以及资源，以使添加简单的复杂自定义笔势。

无论您在 Android 或 iOS 上使用，触摸和手势识别器之间的选择，可以是一个令人困惑。本指南建议，一般情况下，首选项应授予给手势识别器。笔势识别器都实现为离散类，它们提供更大关注点分离和更好的封装。这样可以方便地共享尽量减少编写的代码量的不同视图之间的逻辑。

本指南遵循类似的格式为每个操作系统：首先，该平台的触控 Api 会引入并解释了，因为它们是哪一些触摸交互的基础。然后，我们深入探讨的手势识别器 – 世界上首次通过浏览一些常见的手势，并创建自定义笔势中的进行应用程序正在完成。最后，您将了解如何跟踪各手指使用低级别触控跟踪创建 finger-paint 程序。

部分

- [Android 中的触控](#)
- [演练：在 Android 中使用触控](#)
- [多点触控跟踪](#)

总结

在本指南中，我们将探讨在 Android 中的触控。对于这两个操作系统，我们了解了如何启用触摸以及如何响应触摸事件。接下来，我们了解到有关手势和一些手势识别器，Android 和 iOS 提供能够处理一些较为常见的情景。介绍了如何创建自定义笔势和应用程序中实现它们。演练演示的概念和 Api，用于在操作中，每个操作系统，您还了解到如何跟踪各手指。

相关链接

- [Android 接触启动（示例）](#)
- [Android 接触最终（示例）](#)
- [FingerPaint（示例）](#)

Android 中的触控

2018/11/13 • [Edit Online](#)

得如 iOS, Android 创建一个对象, 包含有关与屏幕的用户的物理交互数据—`Android.View.MotionEvent` 对象。此对象保存数据, 例如执行哪些操作, 在触摸屏输入花费了将置于, 多少压力已应用, 等等。一个 `MotionEvent` 对象分解为以下值移入:

- 描述类型的动作, 如在最初触摸, 触摸屏输入跨屏幕上或触摸结束移动操作代码。
- 一组描述的位置的轴值 `MotionEvent` 和其他移动属性, 如触摸屏输入正在进行、触摸屏输入时已发生, 和使用多大的压力。轴的值可能不同, 具体取决于该设备, 因此前面的列表不介绍所有轴值。

`MotionEvent` 对象将传递给应用程序中适当的方法。有三种方法为 Xamarin.Android 应用程序以响应触摸事件:

- 将分配到一个事件处理程序 `View.Touch` - `Android.Views.View` 类具有 `EventHandler<View.TouchEventArgs>` 哪些应用程序可以将分配到一个处理程序。这是典型.NET 行为。
- 实现 `View.IOnTouchListener` -此接口的实例可能会分配给使用视图的视图对象。 `SetOnTouchListener` 方法。这是功能上等效于分配到一个事件处理程序 `View.Touch` 事件。如果有许多不同的视图可能需要在接触时一些常见或共享逻辑, 它将创建一个类并实现此方法比若要分配每个视图自己的事件处理程序更有效。
- 重写 `View.OnTouchEvent` -Android 子类中的所有视图 `Android.Views.View`。当访问视图时, 将调用 `Android.OnTouchEvent` 并将其传递 `MotionEvent` 对象作为参数。

NOTE

并非所有的 Android 设备支持触摸屏。

将以下标记添加到清单文件导致 Google Play 为仅显示你的应用在启用了触摸这些设备:

```
<uses-configuration android:reqTouchScreen="finger" />
```

笔势

笔势, 是手绘形状上的触摸屏。笔势可以有一个或多个笔画, 每个笔画包含的点创建联系人与屏幕的不同点的序列。Android 可以支持的手势, 从涉及多点触控的复杂手势简单 fling 在屏幕上的许多不同的类型。

Android 提供 `Android.Gestures` 专门针对管理和响应手势的命名空间。在所有手势的核心是一个名为的特殊类 `Android.Gestures.GestureDetector`。顾名思义, 此类将侦听的手势和基于事件 `MotionEvents` 由操作系统提供。

若要实现手势检测程序, 活动必须实例化 `GestureDetector` 类, 并提供的一个实例 `IOnGestureListener`, 如下面的代码段所示:

```
GestureOverlayView.IOnGestureListener myListener = new MyGestureListener();  
_gestureDetector = new GestureDetector(this, myListener);
```

活动还必须实现 `OnTouchEvent` 并且将 `MotionEvent` 传递给手势检测程序。下面的代码段显示了此示例:

```
public override bool OnTouchEvent(MotionEvent e)
{
    // This method is in an Activity
    return _gestureDetector.OnTouchEvent(e);
}
```

实例时 `GestureDetector` 标识一个手势的感兴趣，它会通知的活动或应用程序通过引发事件或通过提供的回调 `GestureDetector.OnGestureListener`。此接口提供了各种手势的六个方法：

- *OnDown* -当点击发生，但不是会被释放时调用。
- *OnFling* -fling 发生并触发事件的开始和结束触摸设备上提供的数据时调用。
- *OnLongPress* -长按发生时调用。
- *OnScroll*的滚动事件发生时调用。
- *OnShowPress* -OnDown 发生之后并移动一个调用或事件未执行。
- *OnSingleTapUp* -点击一下发生时调用。

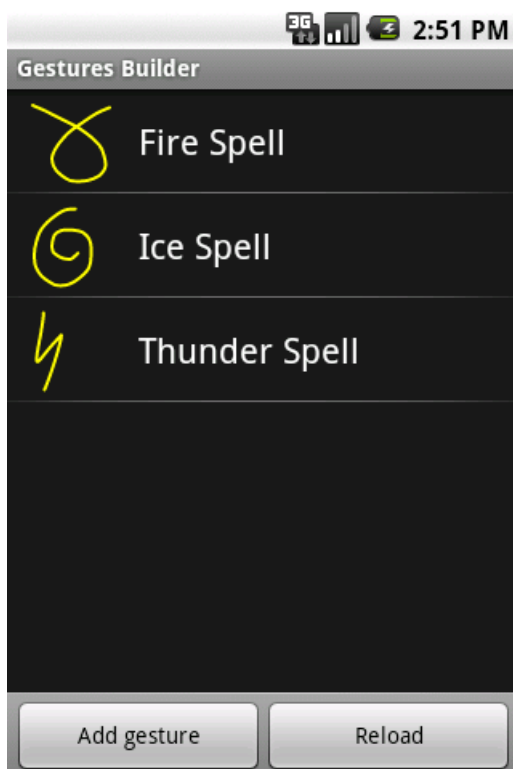
在许多情况下应用程序仅可能感兴趣的手势子集。在这种情况下，应用程序应扩展类 `GestureDetector.SimpleOnGestureListener` 和重写到他们感兴趣的事件对应的方法。

自定义笔势

手势是用户与应用程序交互的极佳方法。我们迄今为止看到的 Api 的简单手势，即可满足要求，但有可能证实有点麻烦更复杂的笔势。为了帮助实现更复杂的手势，Android 提供了 API 的另一组简化的一些负担与自定义笔势关联的 `Android.Gestures` 命名空间中。

创建自定义笔势

自 Android 1.6, Android SDK 附带了名为手势生成器在仿真器上预安装的应用程序。此应用程序允许开发人员创建可以嵌入到应用程序的预定义的笔势。下面的屏幕截图显示了手势生成器的示例：



Google Play, 可以找到名为手势工具此应用程序的改进的版本。手势工具非常类似于笔势生成器，只是它允许您测试笔势后创建它们。此下一步的屏幕截图显示了手势生成器：

-7° 



20:13

Gesture Tool



checkmark1



erase1



erase2

Add gesture

Reload

Test



手势工具是用于创建自定义的手势，因为通过它，因为它们正在创建要测试的笔势更加有用，轻松地可通过 Google Play。

手势工具允许您通过在屏幕上绘制并指定一个名称创建一个手势。创建笔势后它们保存在你的设备的 SD 卡上的二进制文件。此文件需要从设备中检索，然后与文件夹 /Resources/raw 中的应用程序一起打包。此文件可以检索从仿真程序使用 Android 调试桥。下面的示例演示从 Galaxy Nexus 的文件复制到应用程序的资源目录：

```
$ adb pull /storage/sdcard0/gestures <projectdirectory>/Resources/raw
```

一旦已检索到它必须与你在目录 /Resources 内应用程序/原始一起打包的文件。若要使用此手势文件的最简单方法是将文件加载到 GestureLibrary，如以下代码片段中所示：

```
GestureLibrary myGestures = GestureLibraries.FromRawResources(this, Resource.Raw.gestures);
if (!myGestures.Load())
{
    // The library didn't load, so close the activity.
    Finish();
}
```

使用自定义笔势

若要识别在活动中的自定义操作，它必须具有 Android.Gesture.GestureOverlay 对象添加到其布局。下面的代码段演示如何以编程方式向活动中添加 GestureOverlayView：

```
GestureOverlayView gestureOverlayView = new GestureOverlayView(this);
gestureOverlayView.AddOnGesturePerformedListener(this);
SetContentView(gestureOverlayView);
```

以下 XML 代码段演示如何以声明方式添加 GestureOverlayView：

```
<android.gesture.GestureOverlayView
    android:id="@+id/gestures"
    android:layout_width="match_parent "
    android:layout_height="match_parent" />
```

`GestureOverlayView` 具有多个将在绘制笔势的过程中引发的事件。最值得关注的事件是 `GesturePerformed`。当用户完成绘制其手势时，引发此事件。

当引发此事件时，该活动要求 `GestureLibrary` 尝试进行匹配手势工具具有一个手势的用户创建的笔势。

`GestureLibrary` 将返回的预测对象的列表。

每个预测对象保留的分数和一个在手势的名称 `GestureLibrary`。更高级分数、更有可能在预测中名为的笔势匹配用户绘制的笔势。通常情况下，分数低于 1.0 被视为不佳的匹配项。

下面的代码演示匹配笔势的示例：

```
private void GestureOverlayViewOnGesturePerformed(object sender, GestureOverlayView.GesturePerformedEventArgs gesturePerformedEventArgs)
{
    // In this example _gestureLibrary was instantiated in OnCreate
    IEnumerable<Prediction> predictions = from p in
    _gestureLibrary.Recognize(gesturePerformedEventArgs.Gesture)
    orderby p.Score descending
    where p.Score > 1.0
    select p;
    Prediction prediction = predictions.FirstOrDefault();

    if (prediction == null)
    {
        Log.Debug(GetType().FullName, "Nothing matched the user's gesture.");
        return;
    }

    Toast.MakeText(this, prediction.Name, ToastLength.Short).Show();
}
```

此操作完成后，应了解如何在 Xamarin.Android 应用程序中使用触控和手势。现在让我们转到演练并查看所有工作示例应用程序中的概念。

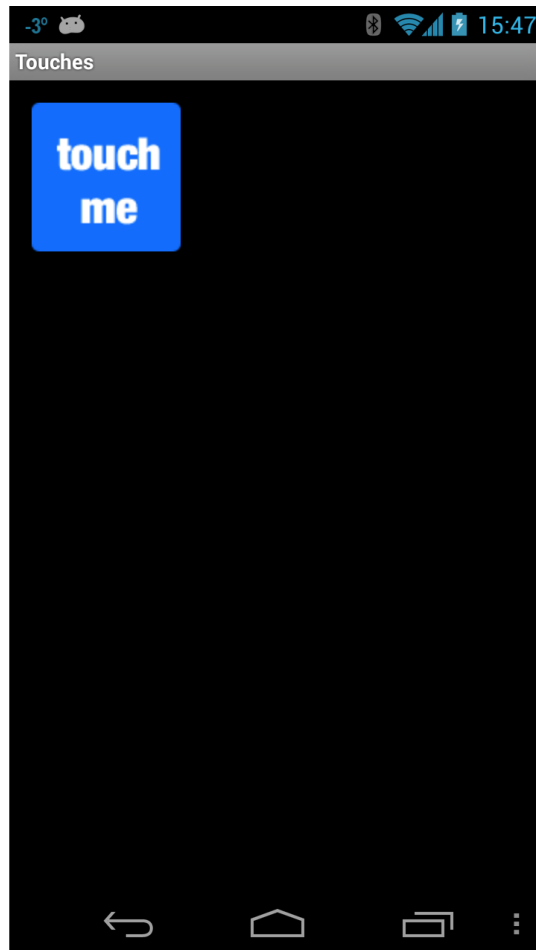
相关链接

- [Android 接触启动（示例）](#)
- [Android 接触最终（示例）](#)

演练-在 Android 中使用触控

2018/10/26 • [Edit Online](#)

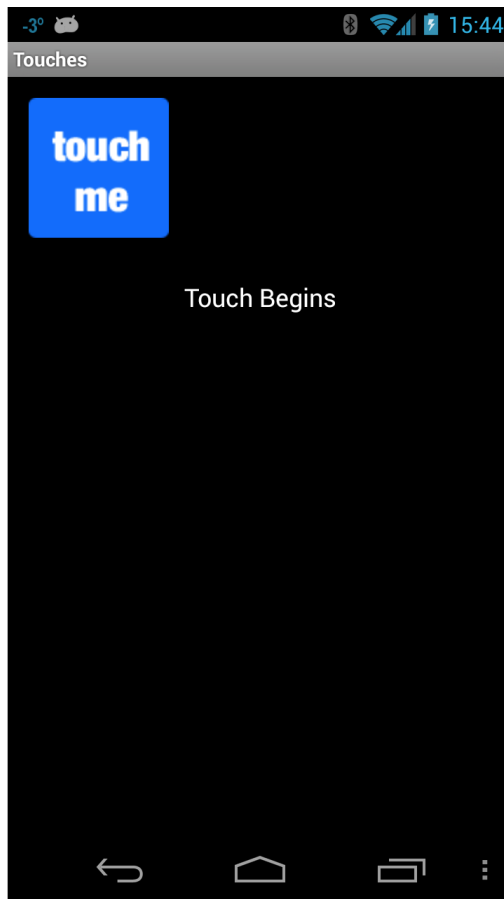
让我们了解如何从上一节中运行的应用程序使用的概念。我们将使用四个活动创建一个应用。第一个活动将一个菜单或将启动其他活动，以演示各种 Api 切换面板。下面的屏幕截图显示了主活动：



第一个活动，触控示例中，将显示如何使用事件处理程序，更改视图。笔势识别器活动将演示如何创建子类 `Android.View.Views` 和处理事件，以及演示如何处理捏合手势。在第三个和最后一个活动自定义笔势，将展示了如何使用自定义笔势。若要使操作更轻松地遵循和 absorb，我们将分解为本演练部分中，将重点放在一个活动的每个部分。

触控示例活动

- 打开项目 **TouchWalkthrough_启动**。**MainActivity**是所有组转—都负责将活动中实现的触摸行为。如果运行该应用程序并单击**触控示例**，下面的活动应启动：



- 现在, 我们确认活动启动之后, 打开文件**TouchActivity.cs**并添加一个处理程序 **Touch** 事件的 **ImageView** :

```
_touchMeImageView.Touch += TouchMeImageViewOnTouch;
```

- 接下来, 添加以下方法**TouchActivity.cs**:

```
private void TouchMeImageViewOnTouch(object sender, View.TouchEventArgs touchEventArgs)
{
    string message;
    switch (touchEventArgs.Event.Action & MotionEventActions.Mask)
    {
        case MotionEventActions.Down:
        case MotionEventActions.Move:
            message = "Touch Begins";
            break;

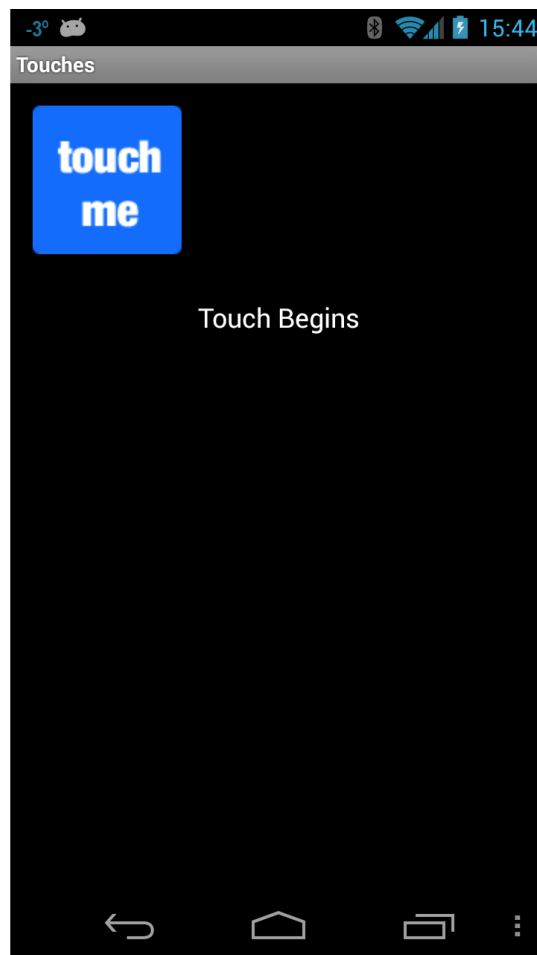
        case MotionEventActions.Up:
            message = "Touch Ends";
            break;

        default:
            message = string.Empty;
            break;
    }

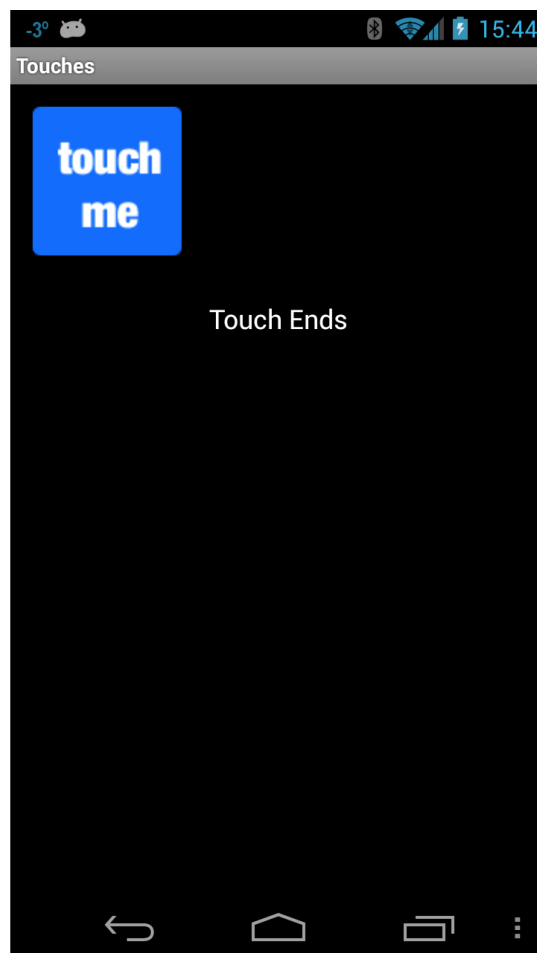
    _touchInfoTextView.Text = message;
}
```

请注意, 在上面的代码, 我们将视为 **Move** 和 **Down** 的相同的操作。这是因为用户可能不提升他们的手指的即使 **ImageView**、可能四处移动, 或由用户所施加的压力可能会更改。这些类型的更改将生成 **Move** 操作。

每次在用户触摸 **ImageView**, 则 **Touch** 将引发事件和我们的处理程序将显示消息**接触开始**在屏幕上, 如以下屏幕截图中所示:



只要用户触摸 `ImageView`，接触开始将显示在 `TextView`。当用户不再触摸 `ImageView`，该消息接触结束将显示在 `TextView`，如以下屏幕截图中所示：



笔势识别器活动

现在让我们实现笔势识别器活动。此活动将演示如何将视图在屏幕上四处拖动，并说明一种可实现捏合缩放。

- 将新活动添加到应用程序调用 `GestureRecognizer`。编辑此活动的代码，使其类似于以下代码：

```
public class GestureRecognizerActivity : Activity
{
    protected override void onCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        View v = new GestureRecognizerView(this);
        SetContentView(v);
    }
}
```

- 添加一个新的 Android 查看到项目中，并将其命名 `GestureRecognizerView`。将以下变量添加到此类：

```
private static readonly int InvalidPointerId = -1;

private readonly Drawable _icon;
private readonly ScaleGestureDetector _scaleDetector;

private int _activePointerId = InvalidPointerId;
private float _lastTouchX;
private float _lastTouchY;
private float _posX;
private float _posY;
private float _scaleFactor = 1.0f;
```

- 添加以下构造函数为 `GestureRecognizerView`。此构造函数将添加 `ImageView` 到我们的活动。此时代码仍将不会编译-我们需要创建该类 `MyScaleListener` 以帮助尺寸调整 `ImageView` 时用户 pinches 它：

```
public GestureRecognizerView(Context context): base(context, null, 0)
{
    _icon = context.Resources.GetDrawable(Resource.Drawable.Icon);
    _icon.SetBounds(0, 0, _icon.IntrinsicWidth, _icon.IntrinsicHeight);
    _scaleDetector = new ScaleGestureDetector(context, new MyScaleListener(this));
}
```

- 若要在我们的活动上绘制图像，我们需要重写 `OnDraw` 视图类，如以下代码片段中所示的方法。此代码将移动 `ImageView` 到指定的位置 `_posX` 和 `_posY` 也为调整大小的缩放因子根据映像：

```
protected override void OnDraw(Canvas canvas)
{
    base.OnDraw(canvas);
    canvas.Save();
    canvas.Translate(_posX, _posY);
    canvas.Scale(_scaleFactor, _scaleFactor);
    _icon.Draw(canvas);
    canvas.Restore();
}
```

- 接下来我们需要更新实例变量 `_scaleFactor` 作为用户 pinches `ImageView`。我们将添加一个名为类 `MyScaleListener`。此类将侦听时用户 pinches 将由 Android 引发缩放事件 `ImageView`。添加到以下的内部类 `GestureRecognizerView`。此类是 `ScaleGesture.SimpleOnScaleGestureListener`。此类是一个方便的类侦听器可以子类化，当你感兴趣的手势子集：

```

private class MyScaleListener : ScaleGestureDetector.SimpleOnScaleGestureListener
{
    private readonly GestureRecognizerView _view;

    public MyScaleListener(GestureRecognizerView view)
    {
        _view = view;
    }

    public override bool OnScale(ScaleGestureDetector detector)
    {
        _view._scaleFactor *= detector.ScaleFactor;

        // put a limit on how small or big the image can get.
        if (_view._scaleFactor > 5.0f)
        {
            _view._scaleFactor = 5.0f;
        }
        if (_view._scaleFactor < 0.1f)
        {
            _view._scaleFactor = 0.1f;
        }

        _view.Invalidate();
        return true;
    }
}

```

- 我们需要重写中的下一步方法 `GestureRecognizerView` 是 `OnTouchEvent`。下面的代码列出了此方法的完整实现。有很多代码，因此允许花点时间看这里，这怎么回。此方法执行的第一个操作是缩放图标，如有必要-这通过调用处理 `_scaleDetector.OnTouchEvent`。接下来，我们尝试找出哪些操作调用此方法：
 - 如果用户触摸屏幕，其中，我们记录的 X 和 Y 位置和接触屏幕的第一个指针的 ID。
 - 如果用户在屏幕上移动其触摸屏输入，然后我们确定用户移动鼠标指针的距离。
 - 如果用户具有提升了手指在屏幕上的，我们将停止跟踪手势。

```

public override bool OnTouchEvent(MotionEvent ev)
{
    _scaleDetector.OnTouchEvent(ev);

    MotionEventActions action = ev.Action & MotionEventActions.Mask;
    int pointerIndex;

    switch (action)
    {
        case MotionEventActions.Down:
            _lastTouchX = ev.GetX();
            _lastTouchY = ev.GetY();
            _activePointerId = ev.GetPointerId(0);
            break;

        case MotionEventActions.Move:
            pointerIndex = ev.FindPointerIndex(_activePointerId);
            float x = ev.GetX(pointerIndex);
            float y = ev.GetY(pointerIndex);
            if (!_scaleDetector.IsInProgress)
            {
                // Only move the ScaleGestureDetector isn't already processing a gesture.
                float deltaX = x - _lastTouchX;
                float deltaY = y - _lastTouchY;
                _posX += deltaX;
                _posY += deltaY;
                Invalidate();
            }

            _lastTouchX = x;
            _lastTouchY = y;
            break;

        case MotionEventActions.Up:
        case MotionEventActions.Cancel:
            // We no longer need to keep track of the active pointer.
            _activePointerId = InvalidPointerId;
            break;

        case MotionEventActions.PointerUp:
            // check to make sure that the pointer that went up is for the gesture we're tracking.
            pointerIndex = (int) (ev.Action & MotionEventActions.PointerIndexMask) >> (int)
MotionEventActions.PointerIndexShift;
            int pointerId = ev.GetPointerId(pointerIndex);
            if (pointerId == _activePointerId)
            {
                // This was our active pointer going up. Choose a new
                // action pointer and adjust accordingly
                int newPointerIndex = pointerIndex == 0 ? 1 : 0;
                _lastTouchX = ev.GetX(newPointerIndex);
                _lastTouchY = ev.GetY(newPointerIndex);
                _activePointerId = ev.GetPointerId(newPointerIndex);
            }
            break;
    }
    return true;
}

```

- 现在运行应用程序，并启动笔势识别器活动。在启动时屏幕应如下所示的屏幕截图：



- 现在触摸图标，并在屏幕上拖动它。请尝试捏合缩放手势。在某一时刻您的屏幕可能看起来类似于以下屏幕截图的内容：



此时您应该给自己加 pat 背面：只是已在 Android 应用程序中实现捏合缩放！需要有短暂的休息，并允许将上移至在本演练中的第三个和最后一个活动-使用自定义笔势。

自定义笔势活动

在本演练中的最后一个屏幕将使用自定义笔势。

出于本演练的目的, 手势库已使用手势工具创建并添加到项目文件中资源/原始/手势。使用的独立地完成此位, 允许继续本演练中的最后一个活动。

- 添加名为的布局文件自定义_手势_layout.xml到包含以下内容项目。该项目已经包含所有映像资源文件夹:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />
    <ImageView
        android:src="@drawable/check_me"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:id="@+id/imageView1"
        android:layout_gravity="center_vertical" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />
</LinearLayout>
```

- 接下来将新活动添加到项目并将其命名 CustomGestureRecognizerActivity.cs。将两个实例变量作为显示在以下两行代码添加到类:

```
private GestureLibrary _gestureLibrary;
private ImageView _imageView;
```

- 编辑 OnCreate 方法的活动中, 以便其类似于下面的代码。让我们看一分钟来说明这怎么回事在此代码中。第一件事是实例化 GestureOverlayView 并将其设为活动的根视图。我们还将分配到一个事件处理程序 GesturePerformed 事件的 GestureOverlayView。接下来我们放大量更早版本, 已创建的布局文件, 并将它添加为子视图的 GestureOverlayView。最后一步是将变量初始化 _gestureLibrary 并从应用程序资源加载手势文件。如果出于某种原因无法加载手势文件, 没有太多可以执行此活动, 因此关闭也一样:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    GestureOverlayView gestureOverlayView = new GestureOverlayView(this);
    SetContentView(gestureOverlayView);
    gestureOverlayView.GesturePerformed += GestureOverlayViewOnGesturePerformed;

    View view = LayoutInflater.Inflate(Resource.Layout.custom_gesture_layout, null);
    _imageView = view.FindViewById<ImageView>(Resource.Id.imageView1);
    gestureOverlayView.AddView(view);

    _gestureLibrary = GestureLibraries.FromRawResource(this, Resource.Raw.gestures);
    if (!_gestureLibrary.Load())
    {
        Log.Wtf(GetType().FullName, "There was a problem loading the gesture library.");
        Finish();
    }
}
```

- 我们需要确实实现了该方法的最后一件事 `GestureOverlayViewOnGesturePerformed` 如下面的代码段中所示。当 `GestureOverlayView` 检测到笔势，它返回调用此方法。尝试获取第一件事 `IList<Prediction>` 对象通过调用匹配手势 `_gestureLibrary.Recognize()`。我们使用 LINQ 的位来获取 `Prediction` 具有最高分数的笔势。

如果不存在匹配手势具有足够的分数，则事件处理程序退出时不执行任何操作。否则为，我们检查该预测的名称并更改所显示的图像基于手势的名称：

```
private void GestureOverlayViewOnGesturePerformed(object sender,
    GestureOverlayView.GesturePerformedEventArgs gesturePerformedEventArgs)
{
    IEnumerable<Prediction> predictions = from p in
        _gestureLibrary.Recognize(gesturePerformedEventArgs.Gesture)
        orderby p.Score descending
        where p.Score > 1.0
        select p;
    Prediction prediction = predictions.FirstOrDefault();

    if (prediction == null)
    {
        Log.Debug(GetType().FullName, "Nothing seemed to match the user's gesture, so don't do anything.");
        return;
    }

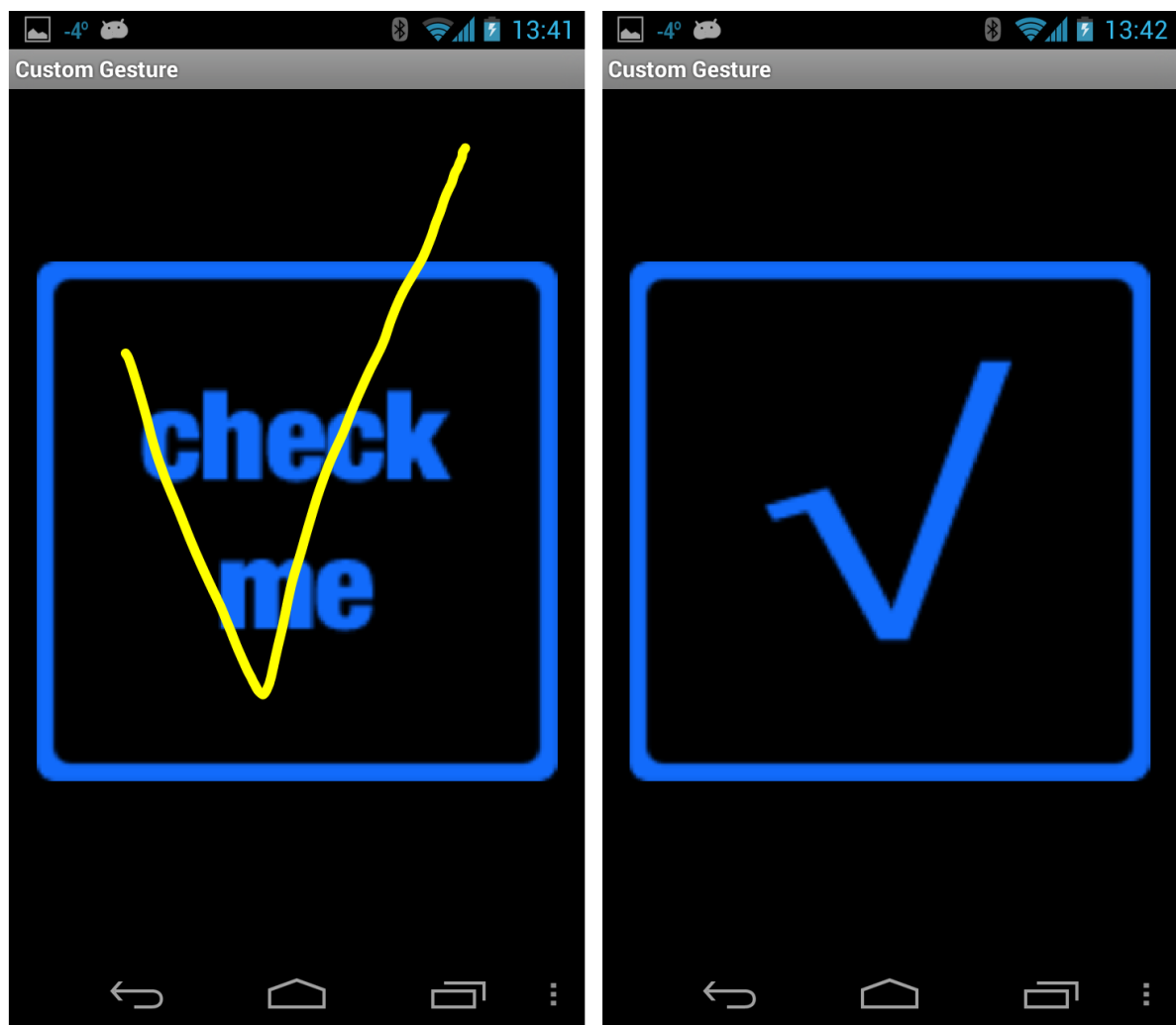
    Log.Debug(GetType().FullName, "Using the prediction named {0} with a score of {1}.",
        prediction.Name, prediction.Score);

    if (prediction.Name.StartsWith("checkmark"))
    {
        _imageView.SetImageResource(Resource.Drawable.checked_me);
    }
    else if (prediction.Name.StartsWith("erase", StringComparison.OrdinalIgnoreCase))
    {
        // Match one of our "erase" gestures
        _imageView.SetImageResource(Resource.Drawable.check_me);
    }
}
```

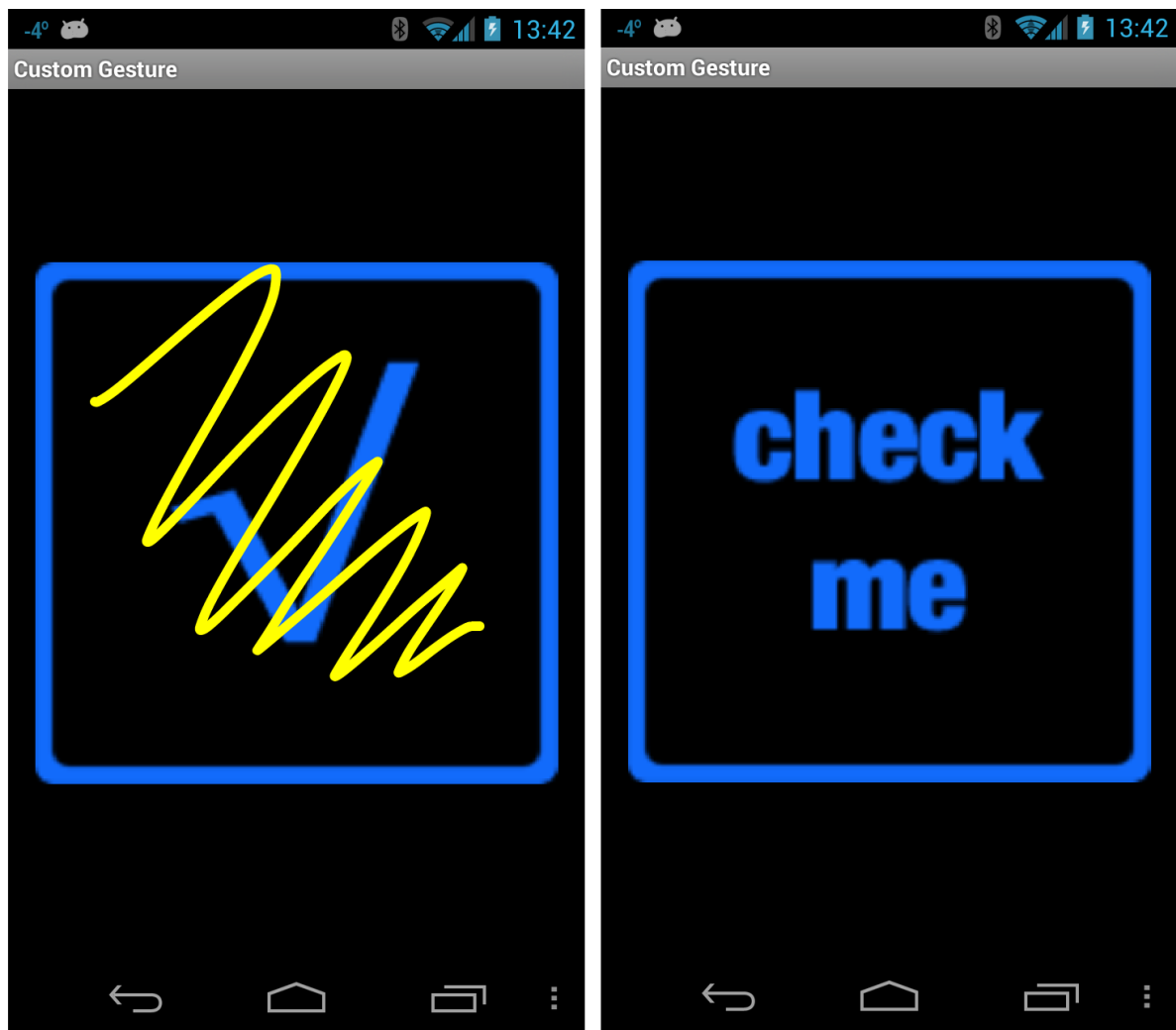
- 运行应用程序和自定义笔势识别器活动启动。其外观应类似于以下屏幕截图的内容：



现在, 在屏幕上, 绘制一个复选标记和显示位图应如下所示的下一个屏幕截图所示:



最后, 在屏幕上绘制涂鸦。复选框应改回其原始映像, 如以下屏幕截图中所示:



你现在已了解了如何将触控和手势使用 Xamarin.Android 的 Android 应用程序中进行。

相关链接

- [Android 接触启动（示例）](#)
- [Android 接触最终（示例）](#)

多点触控手指跟踪

2018/10/26 • [Edit Online](#)

本主题演示如何跟踪从多个手指触摸事件

有些的时候多点触控应用程序需要跟踪各手指，因为它们在屏幕上同时移动。一个典型的应用程序是一个 finger-paint 程序。你希望用户能够绘制的一根手指，而且还可以同时使用多个手指绘制。在您的程序处理多个触控事件，它需要区分哪些事件对应于每个手指。Android 提供 ID 代码实现此目的，但获取和处理该代码可能有点棘手。

对于特定手指与关联的所有事件，ID 代码保持不变。当手指首先触摸屏幕，并从屏幕上抬起手指后变为无效时分配的 ID 代码。这些 ID 代码是通常非常小的整数，并且 Android 将它们重新用于更高版本的触控事件。

几乎总是跟踪各手指的程序维护触控体验跟踪的字典。字典的键是标识特定手指的 ID 代码。字典的值取决于应用程序。在中; [FingerPaint](#) 程序中，每个手指笔划（从触摸以释放）是与包含呈现与该手指绘制的线条所需的所有信息的对象相关联。程序定义了一个较小 `FingerPaintPolyline` 类实现此目的：

```
class FingerPaintPolyline
{
    public FingerPaintPolyline()
    {
        Path = new Path();
    }

    public Color Color { set; get; }

    public float StrokeWidth { set; get; }

    public Path Path { private set; get; }
}
```

每个 polyline 具有一种颜色，笔划宽度和 Android 图形 `Path` 地累积并呈现在行的多个点，因为所绘制的对象。

如下所示的代码的其余部分包含在 `View` 派生类名为 `FingerPaintCanvasView`。类维护类型的对象的字典 `FingerPaintPolyline` 他们主动正在由一个或多个手指绘制期间：

```
Dictionary<int, FingerPaintPolyline> inProgressPolylines = new Dictionary<int, FingerPaintPolyline>();
```

此字典可让视图快速获取 `FingerPaintPolyline` 特定手指与关联的信息。

`FingerPaintCanvasView` 类还维护 `List` 折线已完成的对象：

```
List<FingerPaintPolyline> completedPolylines = new List<FingerPaintPolyline>();
```

在此对象 `List` 中已绘制它们的顺序相同。

`FingerPaintCanvasView` 重写两个方法定义的 `View`：`OnDraw` 并 `OnTouchEvent`。在其 `OnDraw` 重写时，该视图绘制已完成的折线，然后绘制正在折线。

重写 `OnTouchEvent` 方法首先获取 `pointerIndex` 值从 `ActionIndex` 属性。这 `ActionIndex` 值区分多个手指，但它不一致跨多个事件。为此，您使用 `pointerIndex` 若要获取指针 `id` 值从 `GetPointerId` 方法。此 ID 是一致跨多个事件：

```

public override bool OnTouchEvent(MotionEvent args)
{
    // Get the pointer index
    int pointerIndex = args.ActionIndex;

    // Get the id to identify a finger over the course of its progress
    int id = args.GetPointerId(pointerIndex);

    // Use ActionMasked here rather than Action to reduce the number of possibilities
    switch (args.ActionMasked)
    {
        // ...
    }

    // Invalidate to update the view
    Invalidate();

    // Request continued touch input
    return true;
}

```

请注意，使用重写 `ActionMasked` 中的属性 `switch` 语句而不是 `Action` 属性。原因是：

时就需要使用多点触控 `Action` 属性的值为 `MotionEventAction.Down` 的第一个手指触摸屏幕，和的值 `Pointer2Down` 和 `Pointer3Down` 因为第二个和第三个手指同时触摸屏幕。第四个和第五个手指进行联系人 `Action` 属性具有甚至不对应于的成员的数值 `MotionEventAction` 枚举！您需要检查中的值来解释它们的含义的位标志的值。

同样，如在手指离开屏幕，请联系 `Action` 属性具有值 `Pointer2Up` 并 `Pointer3Up` 第二个和第三个手指，和 `Up` 的第一根手指。

`ActionMasked` 属性采用上较少因为它旨在与结合使用的值数目 `ActionIndex` 属性来区分多个手指。当手指触摸屏幕时，可以仅为属性 `MotionEventActions.Down` 的第一根手指和 `PointerDown` 的后续手指。在手指离开屏幕，如 `ActionMasked` 具有值 `Pointer1Up` 后续手指和 `Up` 的第一根手指。

使用时 `ActionMasked`，则 `ActionIndex` 后续手指触摸和保留的屏幕，但您通常不需要使用该以外的值作为中的其他方法的参数用于区分 `MotionEvent` 对象。多点触控，最重要的一种方法是 `GetPointerId` 在上面的代码中调用。方法返回一个值可用于字典键可以将手指的特定事件关联。

`OnTouchEvent` 中重写; [FingerPaint](#) 程序进程 `MotionEventActions.Down` 并 `PointerDown` 相同通过创建一个新的事件 `FingerPaintPolyline` 对象并将其添加到字典：

```

public override bool OnTouchEvent(MotionEvent args)
{
    // Get the pointer index
    int pointerIndex = args.ActionIndex;

    // Get the id to identify a finger over the course of its progress
    int id = args.GetPointerId(pointerIndex);

    // Use ActionMasked here rather than Action to reduce the number of possibilities
    switch (args.ActionMasked)
    {
        case MotionEventActions.Down:
        case MotionEventActions.PointerDown:

            // Create a Polyline, set the initial point, and store it
            FingerPaintPolyline polyline = new FingerPaintPolyline
            {
                Color = StrokeColor,
                StrokeWidth = StrokeWidth
            };

            polyline.Path.MoveTo(args.GetX(pointerIndex),
                                args.GetY(pointerIndex));

            inProgressPolylines.Add(id, polyline);
            break;
        // ...
    }
    // ...
}

```

请注意，`pointerIndex` 还用于获取视图中的手指的位置。与之关联的所有触摸信息 `pointerIndex` 值。`id` 唯一标识多条消息，手指，以便创建字典项的使用。

同样，`OnTouchEvent` 还重写句柄 `MotionEventActions.Up` 并 `Pointer1Up` 相同通过转移到已完成的折线 `completedPolylines` 集合，因此它们可以在绘制 `OnDraw` 重写。该代码还会删除 `id` 字典中的条目：

```

public override bool OnTouchEvent(MotionEvent args)
{
    // ...
    switch (args.ActionMasked)
    {
        // ...
        case MotionEventActions.Up:
        case MotionEventActions.Pointer1Up:

            inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                  args.GetY(pointerIndex));

            // Transfer the in-progress polyline to a completed polyline
            completedPolylines.Add(inProgressPolylines[id]);
            inProgressPolylines.Remove(id);
            break;

        case MotionEventActions.Cancel:
            inProgressPolylines.Remove(id);
            break;
    }
    // ...
}

```

现在的最为棘手的部分。

之间的向下和向上事件，通常有很多 `MotionEventActions.Move` 事件。对单个调用中这些捆绑在一起 `OnTouchEvent`，

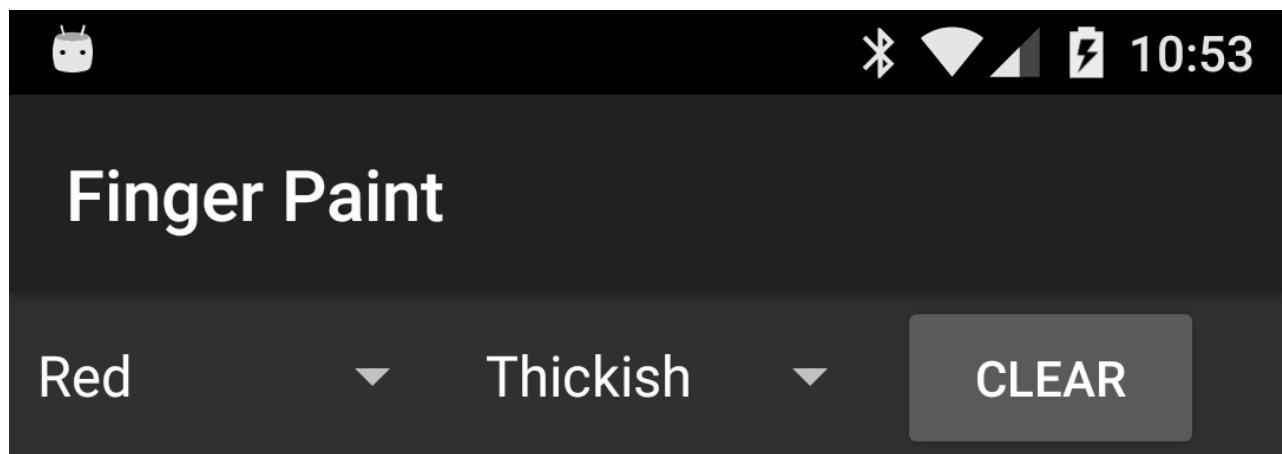
并且必须从以不同方式处理它们 `Down` 和 `Up` 事件。 `pointerIndex` 从前面获取的值 `ActionIndex` 必须忽略属性。相反, 该方法必须获得多个 `pointerIndex` 循环之间 0 的值和 `PointerCount` 属性, 然后获取 `id` 其中每个 `pointerIndex` 值:

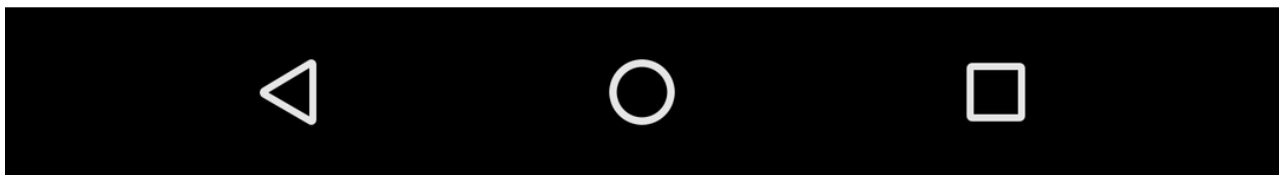
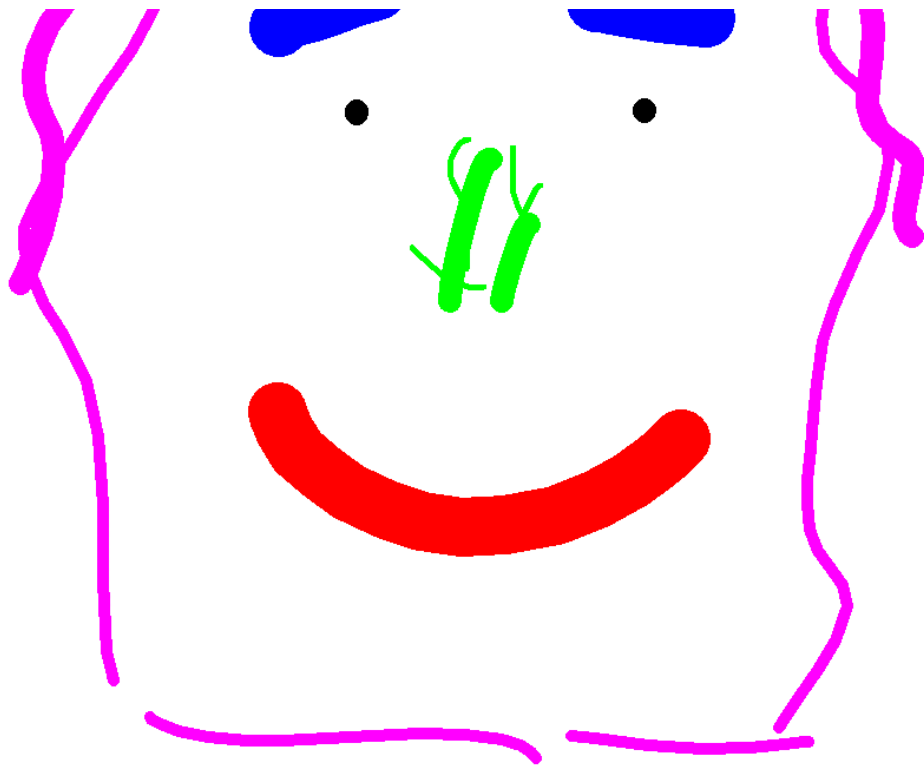
```
public override bool OnTouchEvent(MotionEvent args)
{
    // ...
    switch (args.ActionMasked)
    {
        // ...
        case MotionEventActions.Move:

            // Multiple Move events are bundled, so handle them differently
            for (pointerIndex = 0; pointerIndex < args.PointerCount; pointerIndex++)
            {
                id = args.GetPointerId(pointerIndex);

                inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                    args.GetY(pointerIndex));
            }
            break;
        // ...
    }
    // ...
}
```

处理此类允许; `FingerPaint` 程序来跟踪各手指, 并在屏幕上绘制结果:





现在已了解如何跟踪各手指在屏幕上并区分它们。

相关链接

- [等效的 Xamarin.iOS 》指南](#)
- [FingerPaint \(示例\)](#)

HttpClient 堆栈和适用于 Android 的 SSL/TLS 实现选择器

2018/11/14 • [Edit Online](#)

HttpClient 堆栈和 SSL/TLS 实现选择器确定将由您的 Xamarin.Android 应用程序的 HttpClient 和 SSL/TLS 实现。项目必须引用 **System.Net.Http** 程序集。

WARNING

2018 年 4 月，– 由于增强的安全性要求，包括 PCI 合规性主要云提供程序和 web 服务器应停止对 TLS 版本低于 1.2 的支持。在以前版本的 Visual Studio 默认使用 TLS 的较旧版本创建的 Xamarin 项目。

为了确保您的应用程序继续使用这些服务器和服务，应更新与 **Xamarin 项目** **Android HttpClient** 并 **Native TLS 1.2** 设置如下所示，然后重新生成并重新部署您的应用程序到你用户。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

Xamarin.Android HttpClient 配置处于项目选项 > **Android** 选项，然后单击高级选项按钮。

以下是建议的设置以支持 TLS 1.2:

HttpClient implementation

Android

SSL/TLS implementation

Native TLS 1.2+

备用配置选项

AndroidClientHandler

AndroidClientHandler 是委托给本机 Java/OS 代码而不是在托管代码中实现的所有内容的新处理程序。这是建议的选项。

专业人员

- 更好的性能和较小的可执行文件大小对使用本机 API。
- 例如支持的最新标准。TLS 1.2。

缺点

- 需要 Android 5.0 或更高版本。
- 某些 HttpClient 功能/选项不可用。

托管 (HttpClientHandler)

托管处理程序是完全托管的 HttpClient 处理程序已附带先前的 Xamarin.Android 版本。

专业人员

- 它是最兼容（功能）与 MS.NET 和较旧的 Xamarin 版本。

缺点

- 它不完全集成与操作系统（例如。仅限 TLS 1.0）。

- 它是通常要慢得多（例如。加密）比本机 API。
- 它需要更多托管的代码中，创建更大的应用程序。

选择一个处理程序

之间的选择 `AndroidClientHandler` 和 `HttpClientHandler` 取决于应用程序的需求。`AndroidClientHandler` 建议的最新的安全支持，例如。

- 需要使用 TLS 1.2 + 支持。
- 您的应用程序定目标到 Android 5.0 (API 21) 或更高版本。
- 需要 TLS 1.2 + 支持 `HttpClient`。
- 不需要 TLS 1.2 + 支持 `WebClient`。

`HttpClientHandler` 是一个不错的选择，如果您需要 TLS 1.2 + 支持，但必须支持早于 Android 5.0 的 Android 版本。它也是一个不错的选择，如果您需要 TLS 1.2 + 支持 `WebClient`。

从开始 Xamarin.Android 8.3 `HttpClientHandler` 默认为令人乏味的 SSL (`btls`) 为基础的 TLS 提供程序。令人乏味的 SSL TLS 提供程序提供以下优势：

- 它支持 TLS 1.2 +。
- 它支持所有的 Android 版本。
- 它提供了两个的 TLS 1.2 + 支持 `HttpClient` 和 `WebClient`。

将令人乏味的 SSL 用作基础 TLS 提供程序的缺点是它可以增加对生成的 APK（它将添加其他每个受支持的 ABI 的 APK 大小的大约 1 MB）的大小。

从 Xamarin.Android 8.3 开始，默认 TLS 提供程序是令人乏味的 SSL (`btls`)。如果您不想要使用令人乏味的 SSL，你可以通过设置还原到历史托管 SSL 实施 `$(AndroidTlsProvider)` 属性设置为 `legacy`（有关设置生成属性的详细信息，请参阅[Build 过程](#)）。

以编程方式使用 `AndroidClientHandler`

`Xamarin.Android.Net.AndroidClientHandler` 是 `HttpMessageHandler` 专门适用于 Xamarin.Android 的实现。此类的实例将使用本机 `java.net.URLConnection` 实现适用于所有 HTTP 连接。从理论上讲，这将提供 HTTP 性能和较小的 APK 大小的增加。

此代码片段示范了如何为的单个实例显式 `HttpClient` 类：

```
// Android 5.0 or higher, Xamarin.Android 6.1 or higher
HttpClient client = new HttpClient(new Xamarin.Android.Net.AndroidClientHandler ());
```

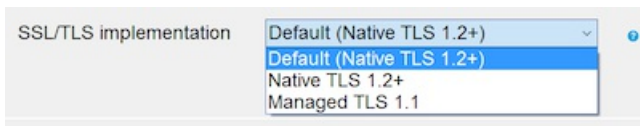
NOTE

基础 Android 设备必须支持 TLS 1.2 (ie. Android 5.0 及更高版本)

SSL/TLS 实现生成选项

此项目选项，可以控制哪些基础 TLS 库将由所有 web 请求，同时 `HttpClient` 和 `WebRequest`。默认情况下，选择 TLS 1.2:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



例如：

```
var client = new HttpClient();
```

如果 HttpClient 实现已设置为托管 TLS 实现设置为本机 TLS 1.2 +，然后 `client` 对象将自动使用托管 `HttpClientHandler` 和 TLS 1.2（由 BoringSSL 库提供）用于其 HTTP 请求。

但是，如果 **HttpClient** 实现设置为 `AndroidHttpClient`，然后所有 `HttpClient` 对象将使用基础 Java 类 `java.net.URLConnection` 并将不会影响实现 TLS/SSL 值。`WebRequest` 对象将使用 BoringSSL 库。

其他方法来控制 SSL/TLS 配置

有三种方法的 Xamarin.Android 应用程序可以控制的 TLS 设置：

1. 项目选项中选择 HttpClient 实现和默认 TLS 库。
2. 以编程方式使用 `Xamarin.Android.Net.AndroidClientHandler`。
3. 环境变量声明（可选）。

建议的方法是使用 Xamarin.Android 项目选项来声明默认值的三个选项，`HttpMessageHandler` 和 TLS 对整个应用程序。然后，如有必要，以编程方式实例化 `Xamarin.Android.Net.AndroidClientHandler` 对象。这些选项是上面所述。

第三个选项-使用环境变量-如下所述。

声明的环境变量

有两个与在 Xamarin.Android 中的 TLS 的使用相关的环境变量：

- `XA_HTTP_CLIENT_HANDLER_TYPE` – 此环境变量声明的默认 `HttpMessageHandler` 将使用该应用程序。例如：

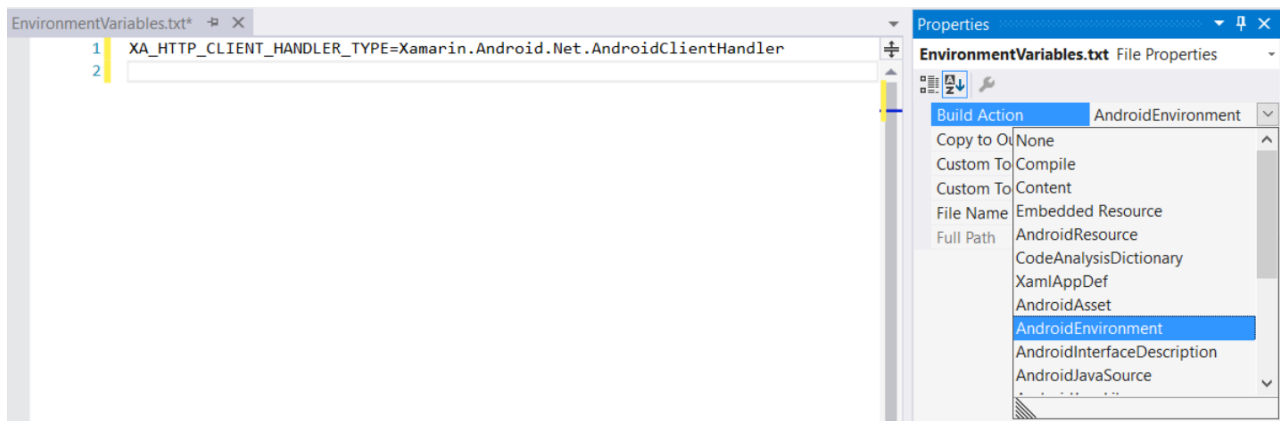
```
XA_HTTP_CLIENT_HANDLER_TYPE=Xamarin.Android.Net.AndroidClientHandler
```

- `XA_TLS_PROVIDER` – 此环境变量将声明的 TLS 库将使用，或者 `btls`，`legacy`，或 `default`（这是与省略此变量相同）：

```
XA_TLS_PROVIDER=btls
```

通过添加设置此环境变量_环境文件_到项目。环境文件是 Unix 格式的纯文本格式的文件生成操作 **AndroidEnvironment**：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



请参阅[Xamarin.Android 环境](#)有关环境变量和 Xamarin.Android 的更多详细信息的指南。

相关链接

- [传输层安全性 \(TLS\)](#)

编写响应式应用程序

2018/10/26 • [Edit Online](#)

维护响应式 GUI 的关键之一是为 GUI 不会阻止此后台线程上的长时间运行任务。假设我们想要计算的值以显示给用户，但此值需要 5 秒钟来计算：

```
public class ThreadDemo : Activity
{
    TextView textview;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Create a new TextView and set it as our view
        textview = new TextView (this);
        textview.Text = "Working..";

        SetContentView (textview);

        SlowMethod ();
    }

    private void SlowMethod ()
    {
        Thread.Sleep (5000);
        textview.Text = "Method Complete";
    }
}
```

这将起作用，但应用程序将“挂起”5 秒时计算的值。在此期间，应用程序将不响应任何用户交互。若要解决此问题，我们要做我们在后台线程上的计算：

```
public class ThreadDemo : Activity
{
    TextView textview;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Create a new TextView and set it as our view
        textview = new TextView (this);
        textview.Text = "Working..";

        SetContentView (textview);

        ThreadPool.QueueUserWorkItem (o => SlowMethod ());
    }

    private void SlowMethod ()
    {
        Thread.Sleep (5000);
        textview.Text = "Method Complete";
    }
}
```

现在我们计算在后台线程上的值，因此我们 GUI 在计算期间保持响应状态。但是，完成计算后，我们的应用程序崩

溃时，请在日志中保留此：

```
E/mono (11207): EXCEPTION handling: Android.Util.AndroidRuntimeException: Exception of type
'Android.Util.AndroidRuntimeException' was thrown.
E/mono (11207):
E/mono (11207): Unhandled Exception: Android.Util.AndroidRuntimeException: Exception of type
'Android.Util.AndroidRuntimeException' was thrown.
E/mono (11207): at Android.Runtime.JNIEnv.CallVoidMethod (IntPtr jobject, IntPtr jmethod,
Android.Runtime.JValue[] parms)
E/mono (11207): at Android.Widget.TextView.set_Text (IEnumerable`1 value)
E/mono (11207): at MonoDroidDebugging.Activity1.SlowMethod ()
```

这是因为必须更新从 GUI 线程 GUI。我们的代码将更新从线程池线程，导致应用崩溃的 GUI。我们需要计算我们，在后台线程上的值，但然后完成我们的更新，与处理在 GUI 线程上[Activity.RunOnUiThread](#):

```
public class ThreadDemo : Activity
{
    TextView textview;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Create a new TextView and set it as our view
        textview = new TextView (this);
        textview.Text = "Working..";

        SetContentView (textview);

        ThreadPool.QueueUserWorkItem (o => SlowMethod ());
    }

    private void SlowMethod ()
    {
        Thread.Sleep (5000);
        RunOnUiThread (() => textview.Text = "Method Complete");
    }
}
```

此代码按预期运行。此 GUI 仍然能够做出响应，并完成计算后获取正确更新。

请注意这种技术不只是用于计算成本高的值。它可以用于任何长时间运行的任务可以在后台，如 web 服务调用或 internet 数据下载完成的。

用户界面

2018/10/26 • [Edit Online](#)

以下部分介绍各种工具和用于构成 Xamarin.Android 应用中的用户界面的构建基块。

Android Designer

本部分介绍如何使用 Android 设计器以可视方式布置控件和编辑属性。它还说明了如何使用设计器使用跨各种配置，如主题、语言和设备配置的用户界面和资源，以及如何横向和纵向等的替代视图的设计。

材料主题

*材料主题*是确定视图和活动在 Android 中的外观的用户界面样式。材料主题是内置到 Android，因此使用由系统用户界面以及应用程序。本指南介绍材料设计原则，并介绍了如何使用内置的材料主题或自定义主题的应用的主题。

用户配置文件

本指南介绍如何访问个人配置文件的所有者的设备，包括设备所有者的姓名和电话号码等的联系人数据。

初始屏幕

Android 应用程序需要一些时间才能启动，尤其是在设备上首次启动应用程序。初始屏幕可能会显示开始向上向用户的进度。本指南介绍如何创建您的应用程序的初始屏幕。

布局

布局用于定义用户界面的可视结构。如布局 `ListView` 和 `RecyclerView` 是 Android 应用程序的最基本的构建基块。通常情况下，将使用一种布局 `Adapter` 充当从布局到基础数据用于填充布局中的数据项的桥梁。本部分介绍如何使用如下所示的布局 `LinearLayout`，`RelativeLayout`，`TableLayout`，`RecyclerView`，和 `GridView`。

控件

Android 控件 (也称为 *小组件*) 是用于生成用户界面的 UI 元素。本部分介绍如何使用如按钮、工具栏、日期/时间选取器、日历、微调控件、切换、弹出菜单、视图寻呼程序和 web 视图的控件。

Xamarin.Android 设计器

2018/10/26 • • [Edit Online](#)

本文介绍 Xamarin.Android 设计器的功能。它还说明了设计器基础知识，演示如何使用设计器以可视方式布置小组件和编辑属性。它还阐释了如何使用设计器使用跨各种配置，如主题、语言和设备配置的用户界面和资源，以及如何替代如横向和纵向视图的设计。

概述

Xamarin.Android 支持基于 XML 文件，以及在代码中创建的程式用户界面中的用户界面设计的这两个以声明性方式。使用声明性方法时，XML 文件可以是手动编辑或修改直观地使用 Xamarin.Android 设计器。使用设计器的 UI 创建期间允许即时的反馈，以加速开发，并使 UI 创建的过程不太费力。

本文调查了 Xamarin.Android 设计器的许多功能。它说明了：

1. 使用设计器的基础知识。
2. 使设计器中的各个部分。
3. 如何将 Android 布局加载到设计器。
4. 如何添加小组件。
5. 如何编辑属性。
6. 如何使用各种资源和设备配置。
7. 如何修改横向与纵向如替代视图的用户界面。
8. 如何处理可能时使用替代视图导致的冲突。
9. 如何使用材料设计工具来构建材料设计-兼容的应用程序。

部分

[使用 Android Designer](#)

[设计器基础知识](#)

[资源限定符和可视化效果选项](#)

[备选布局视图](#)

[材料设计功能](#)

总结

本文讨论了 Xamarin.Android 设计器的功能集。它介绍了如何开始使用设计器中，并说明其各部分。它描述如何加载布局，以及如何添加和修改小组件，通过使用两设计器图面并将源视图。它还介绍了如何使用各种资源和设备配置。最后，它会检查如何使用设计器开发专为替代视图，如横向与纵向，生成的用户界面，以及如何解决可能会出现此类视图之间的冲突。

相关链接

- [设计器的演练](#)
- [Android 资源](#)

使用 Xamarin.Android 设计器

2018/10/26 • [Edit Online](#)

本文是 Xamarin.Android 设计器的演练。它演示了如何创建小型彩色的浏览器应用中，一个用户界面完全在设计器中创建此用户界面。

概述

使用 XML 文件或以编程方式编写代码，可以以声明方式创建 android 用户界面。Xamarin.Android 设计器允许开发人员能够创建和修改声明性布局以可视方式，而无需手动编辑的 XML 文件。在设计器还提供允许开发人员评估 UI 更改而无需重新部署到设备或仿真程序的应用程序的实时反馈。这些设计器功能可以极大地加快 Android UI 开发。本文演示如何使用 Xamarin.Android 设计器直观地创建用户界面。

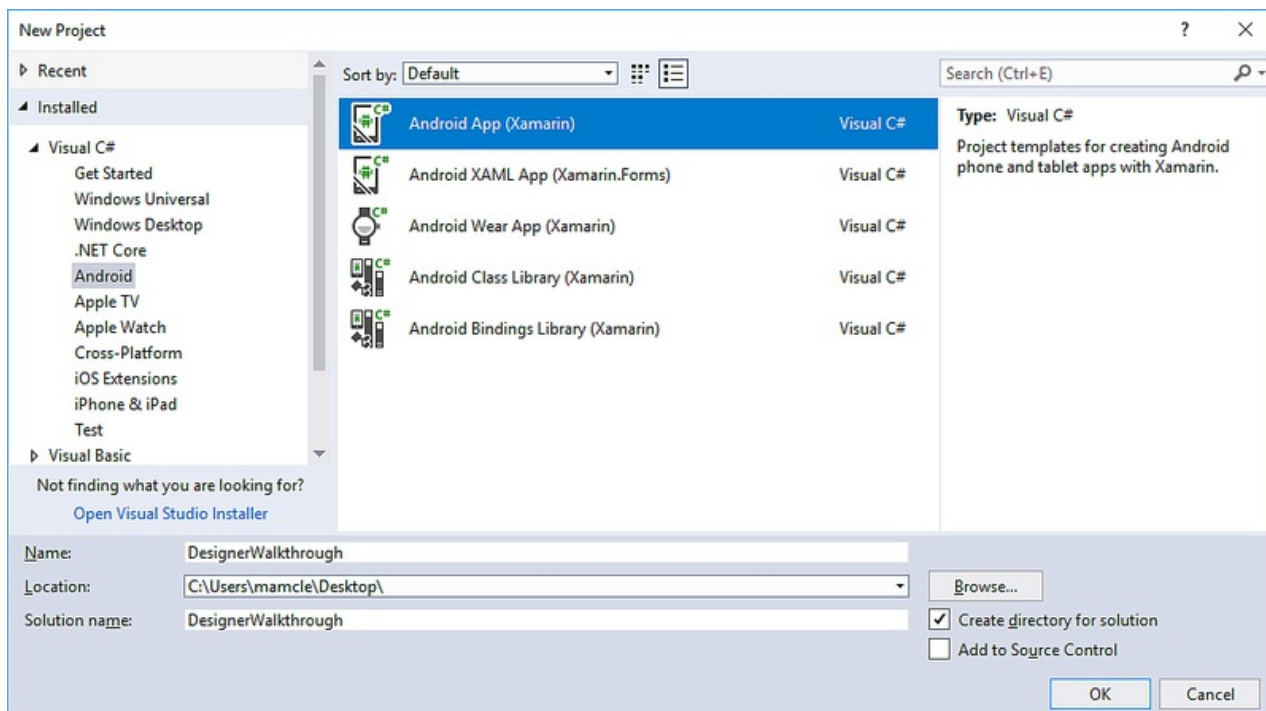
演练

本演练的目的是使用 Android 设计器创建一个颜色浏览器应用示例的用户界面。颜色浏览器应用会提供颜色、它们的名称和其 RGB 值的列表。您将学习如何将添加到小组件设计图面以及如何以可视方式布置这些小组件。之后，您将学习如何修改小组件上以交互方式设计图面或使用设计器的属性窗格。最后，您将了解设计时在设备或仿真器上运行应用的外观。

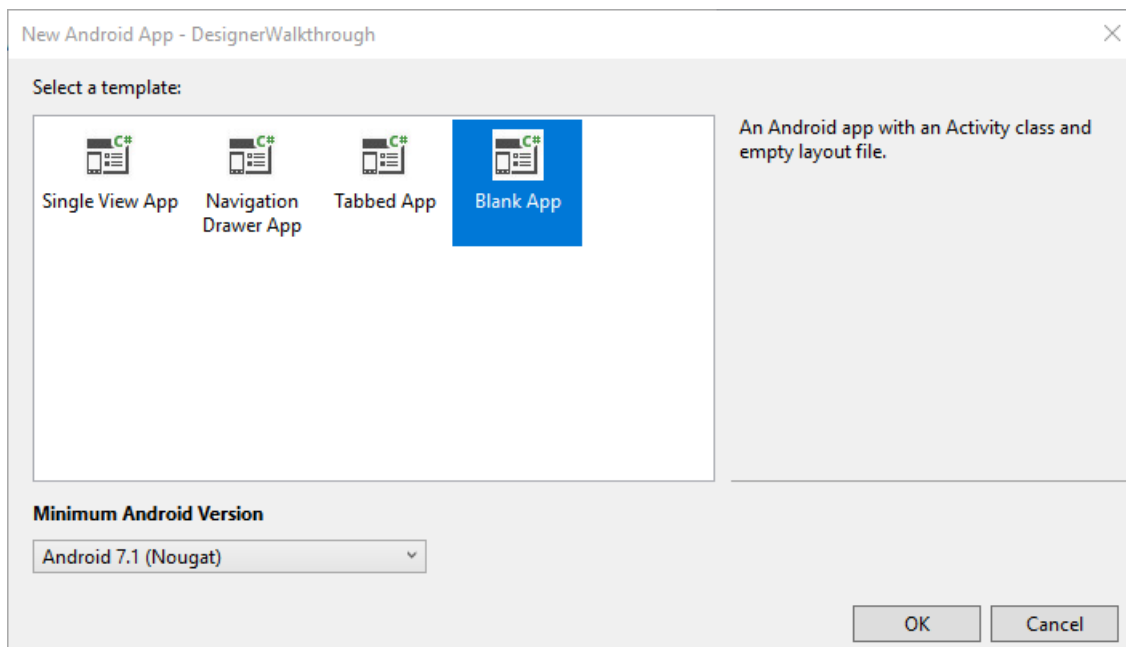
- [Visual Studio](#)
- [Visual Studio for Mac](#)

创建新的项目

第一步是创建新的 Xamarin.Android 项目。启动 Visual Studio 中，单击**新建项目...**，然后选择**Visual C# > Android > Android 应用 (Xamarin)** 模板。新应用命名**DesignerWalkthrough**然后单击**确定**。

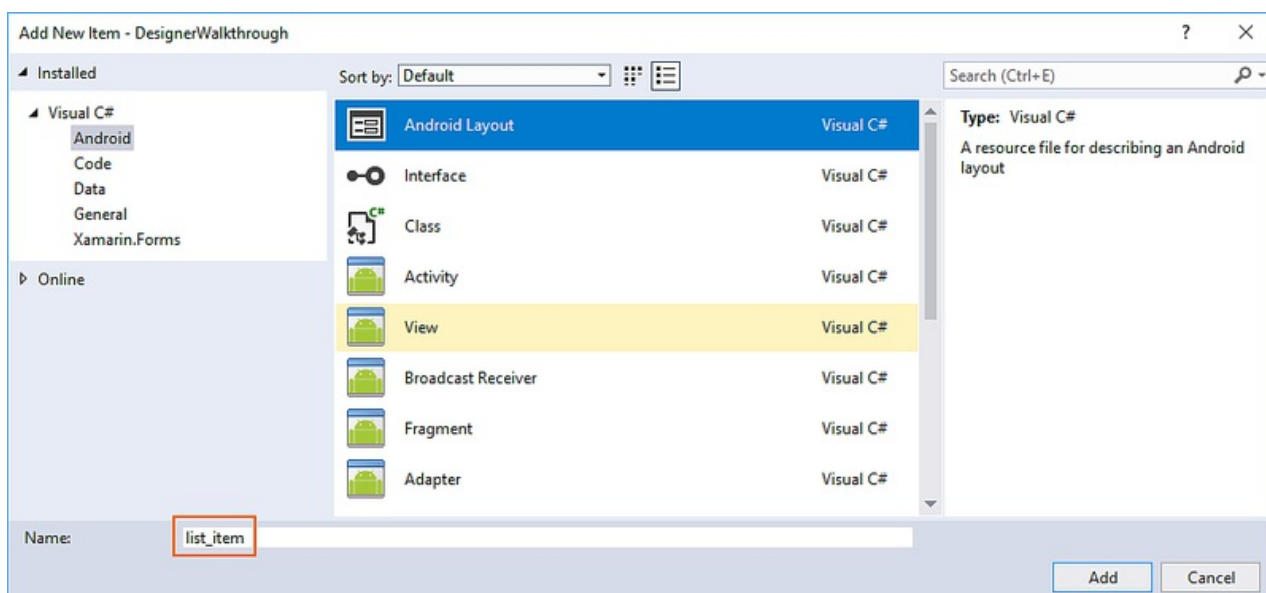


在中新的 **Android 应用** 对话框中，选择**空白应用**然后单击**确定**：

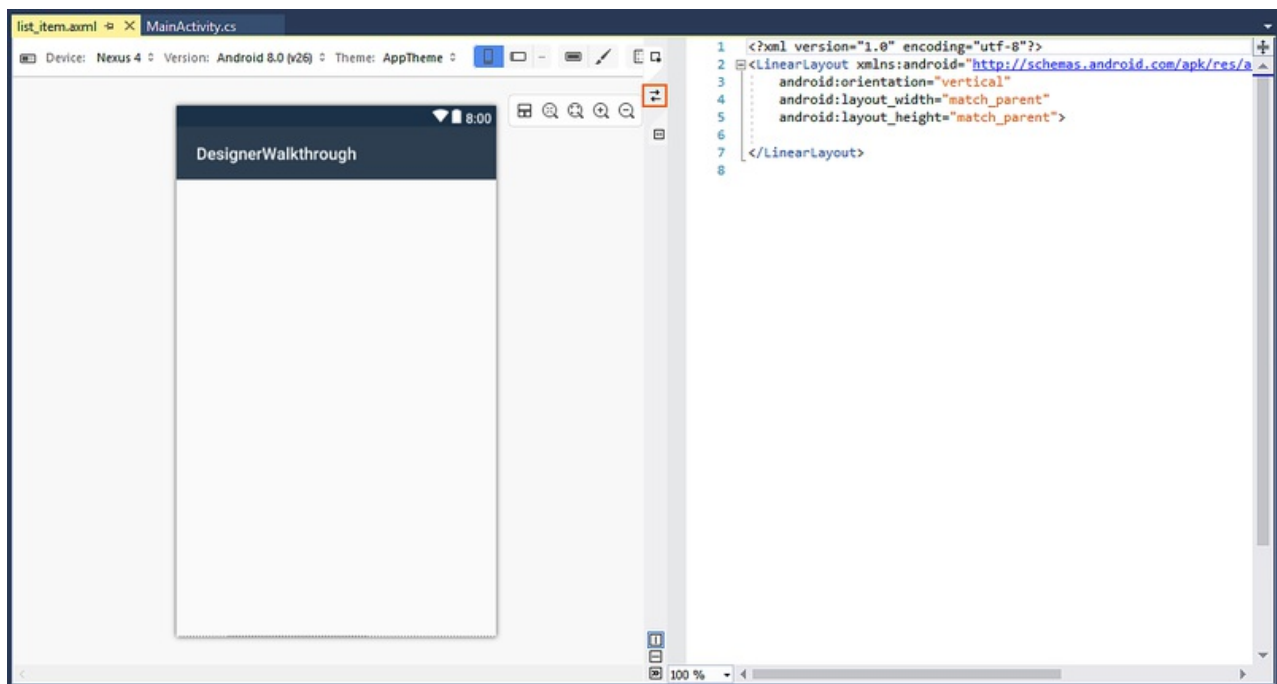


添加布局

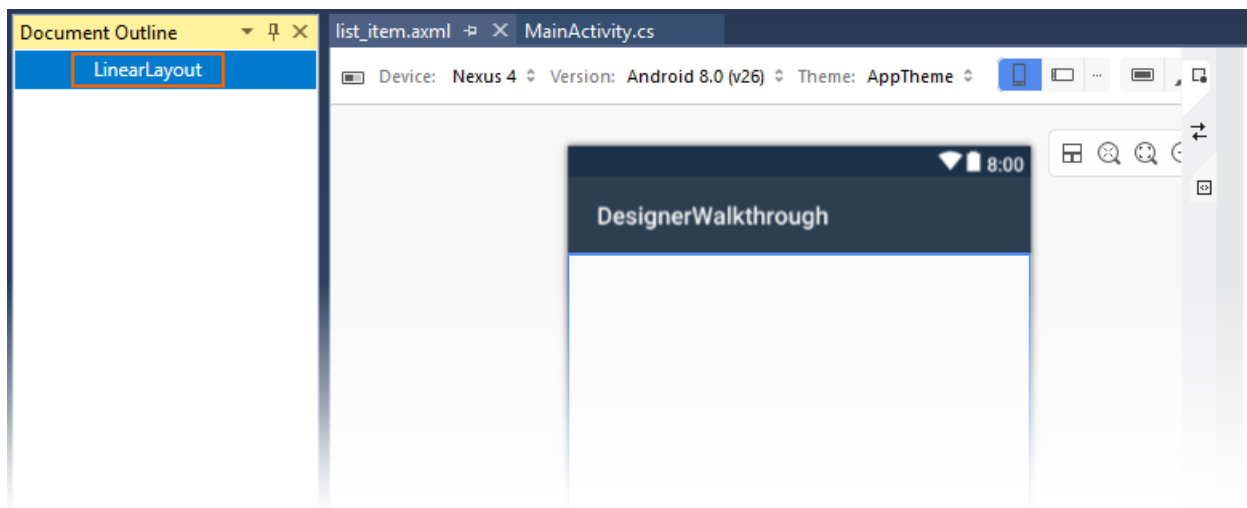
下一步是创建**LinearLayout**，将存放用户界面元素。右键单击资源/布局中解决方案资源管理器，然后选择**添加 > 新建项...**在**添加新项**对话框中，选择**Android 布局**。将文件命名**list_item**然后单击**添加**：



新**list_item**布局显示在设计器中。请注意两个窗格将显示-设计图面有关**list_item**是可见的左窗格中，而其 XML 源显示在右窗格。你可以交换的位置设计图面和源通过单击窗格交换窗格图标位于之间的两个窗格：



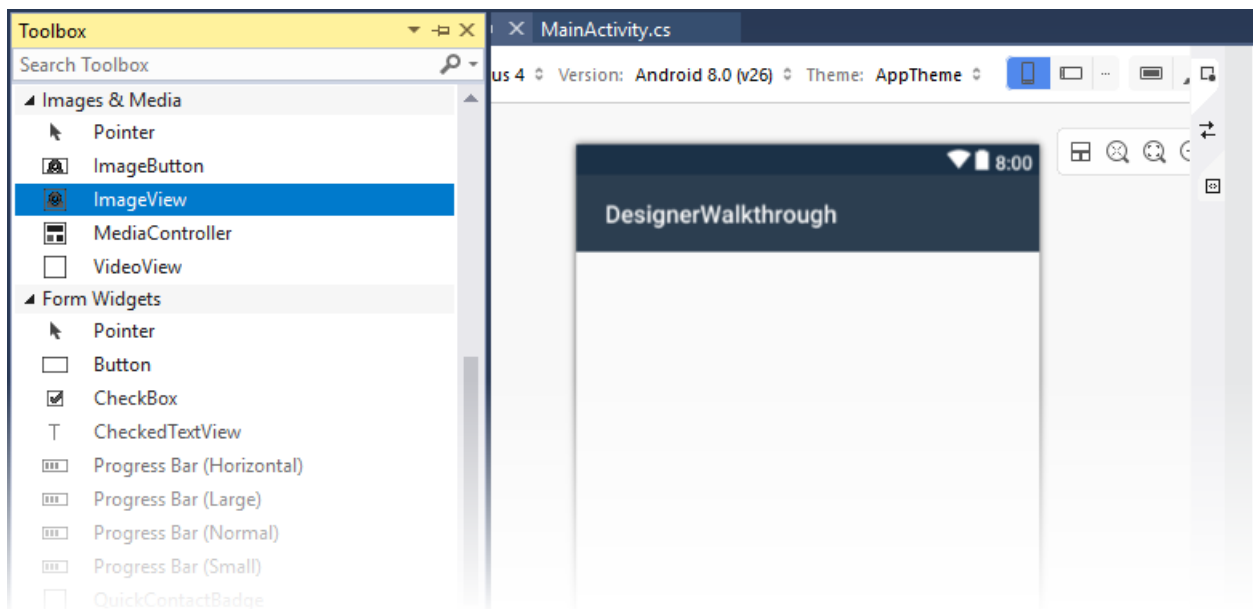
从视图菜单上, 单击**其他 Windows > 文档大纲**以打开文档大纲。文档大纲显示布局当前包含单个**LinearLayout**小组件:



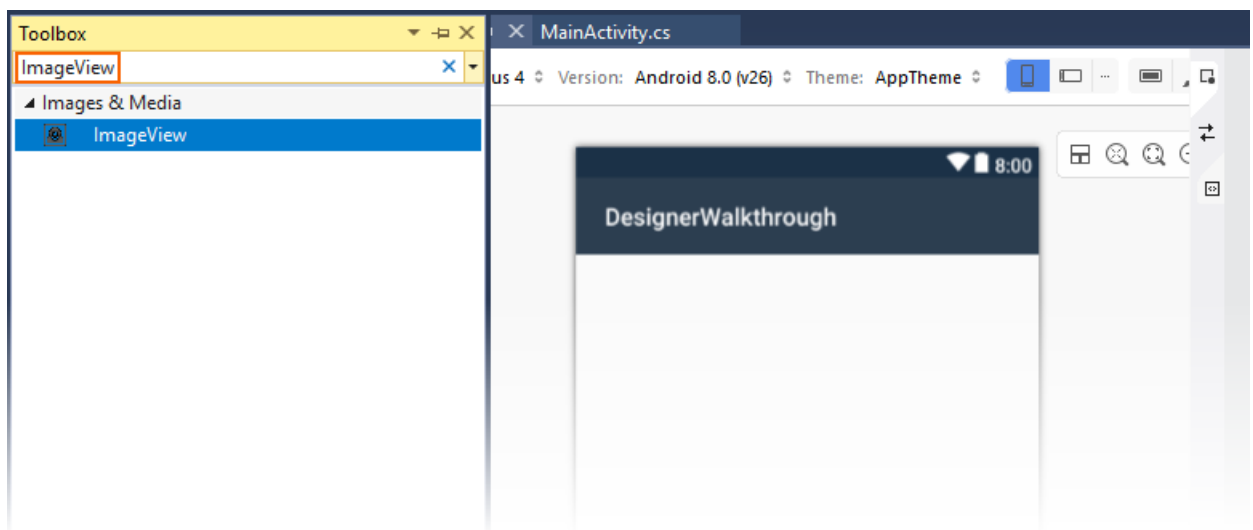
下一步是创建在此颜色浏览器应用的用户界面 **LinearLayout**。

创建列表项用户界面

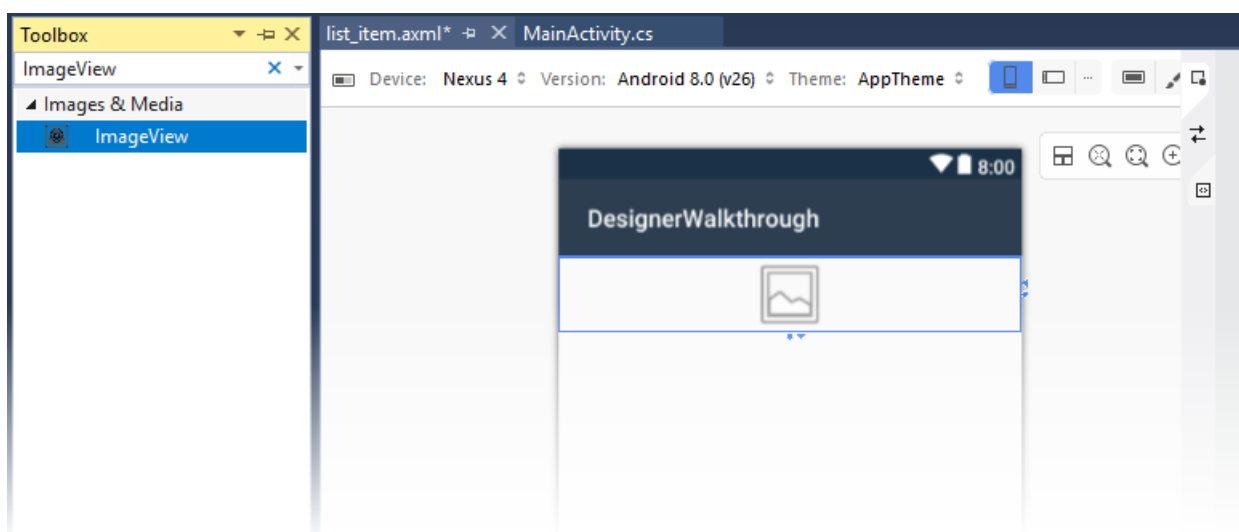
如果工具箱未显示窗格中, 单击**工具箱**在左侧选项卡。在中**工具箱**, 向下滚动到**图像和媒体**部分, 并继续向下滚动直到找到 **ImageView**:



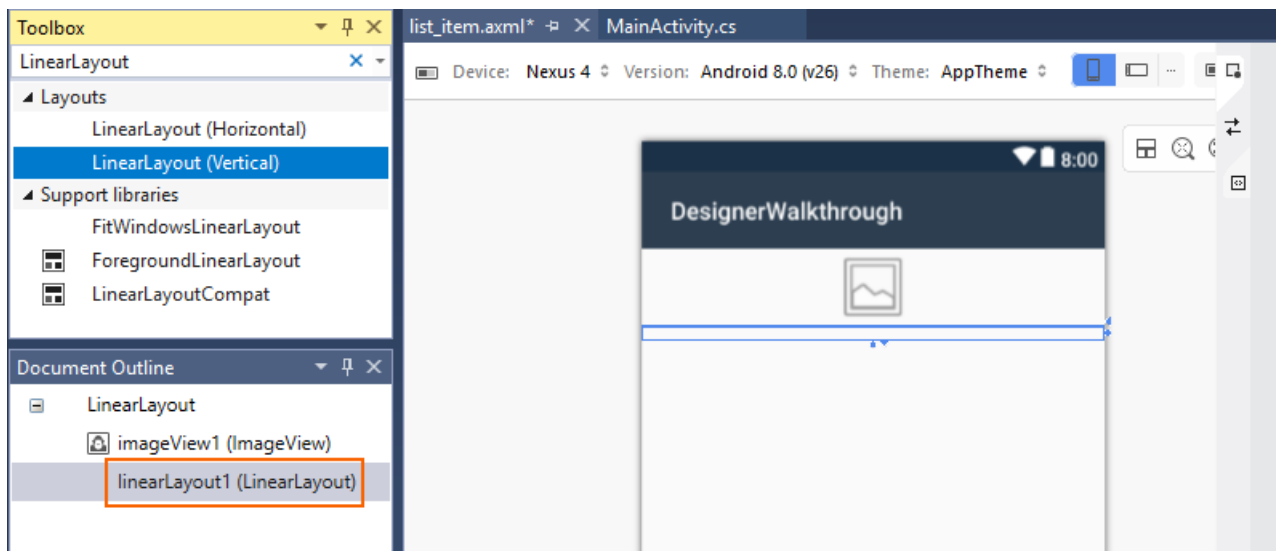
或者，可以输入`ImageView`在搜索栏查找 `ImageView`：



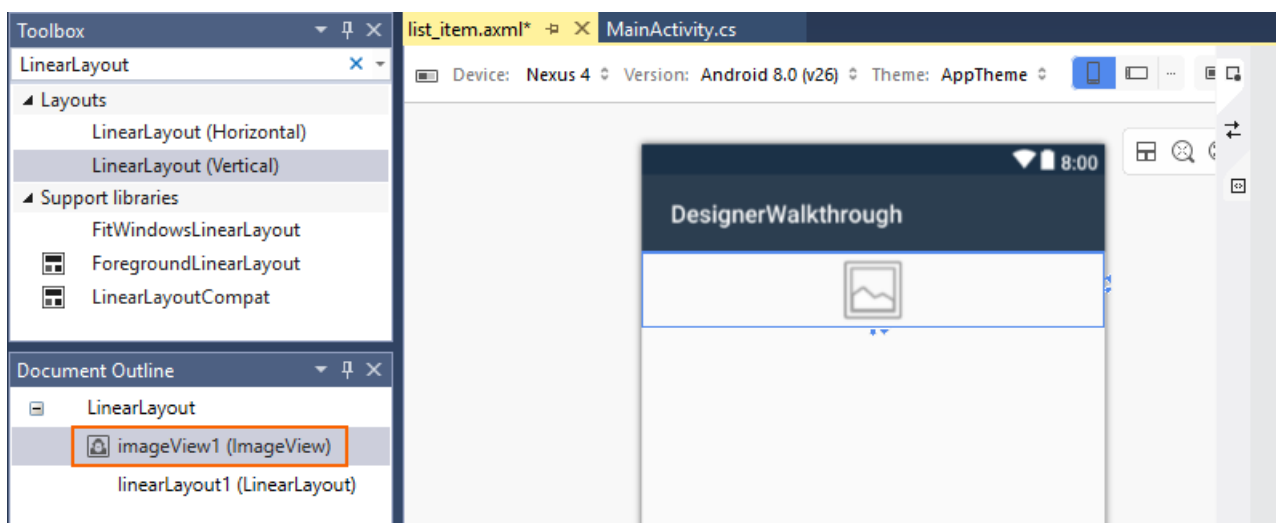
将该 `ImageView` 拖动到设计图面上 (这 `ImageView` 用于颜色浏览器应用中显示颜色样本)：



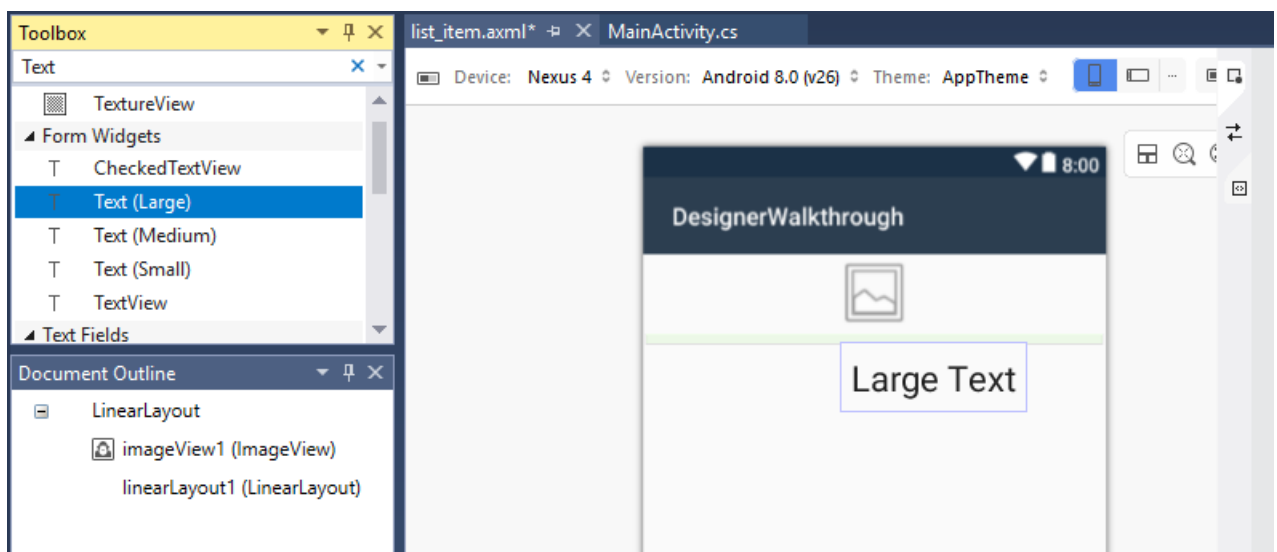
接下来，拖 `LinearLayout (Vertical)` 小组件从工具箱到设计器。请注意，一个蓝色的轮廓指示添加的边界 `LinearLayout`。文档大纲显示它是的子级 `LinearLayout` 下的 `imageView1 (ImageView)`：



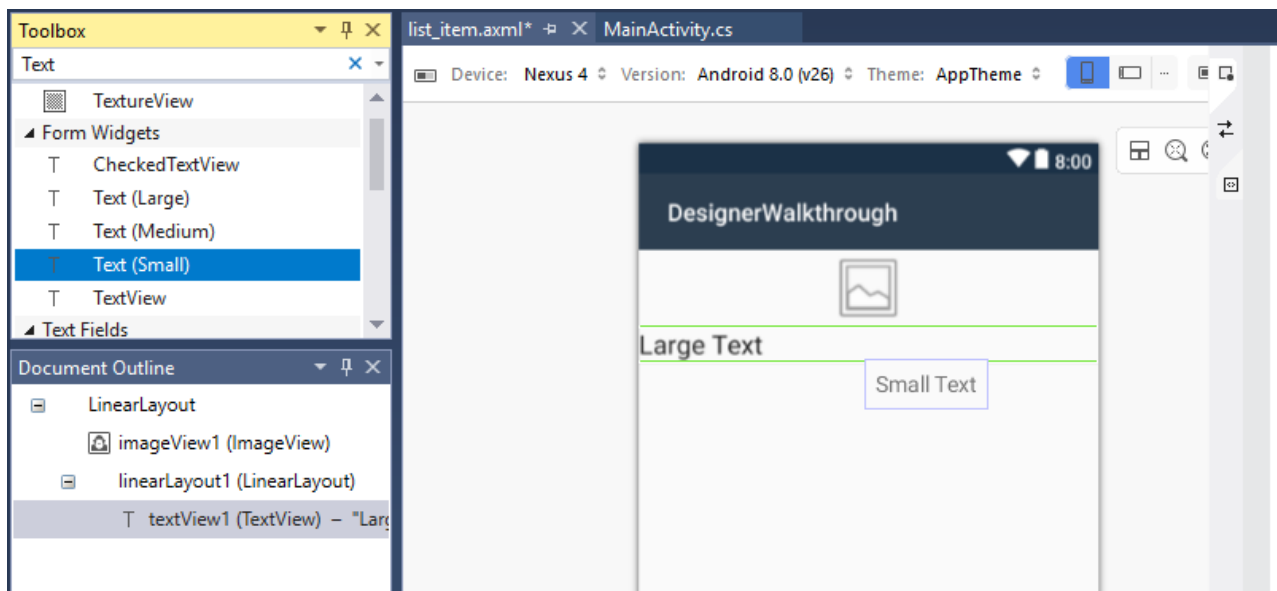
当选择 `ImageView` 设计器中，在蓝色边框移动来包围 `ImageView`。此外，将所选内容移到 `imageView1 (ImageView)` 中文档大纲：



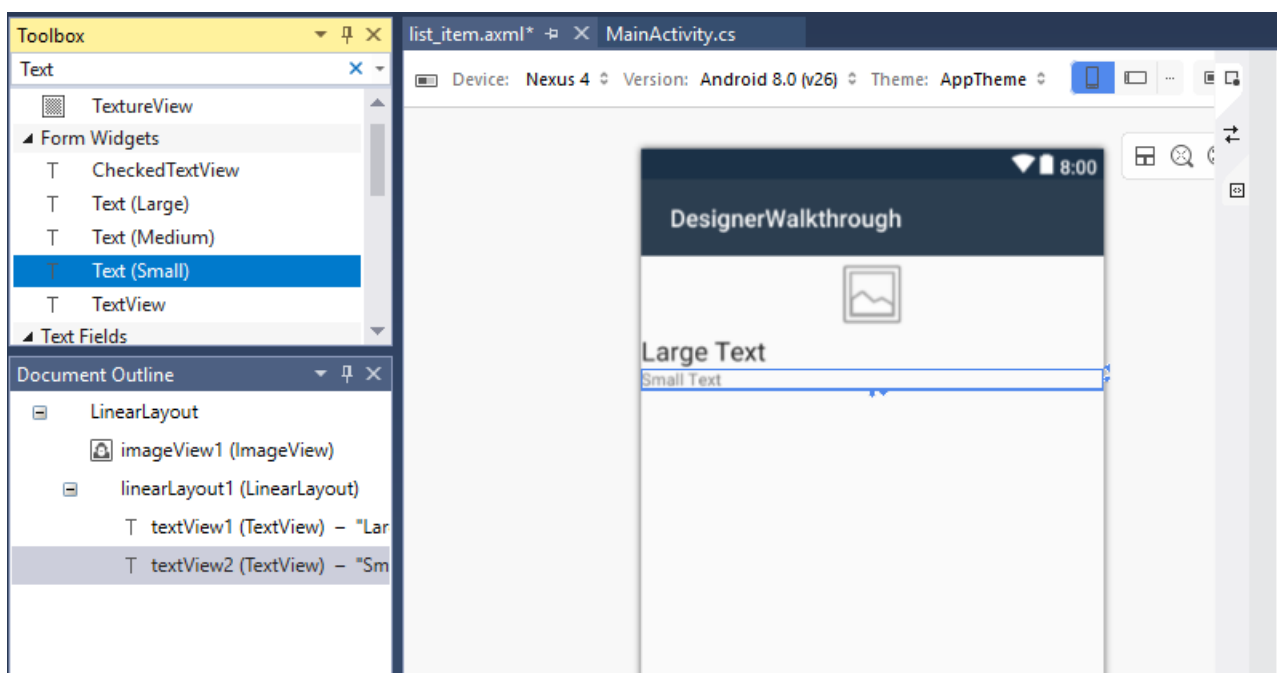
接下来，拖 `Text (Large)` 小组件从工具箱到新添加 `LinearLayout`。请注意，在设计器使用绿色突出显示来指示将插入新的小组件的位置：



接下来，添加 `Text (Small)` 小组件下 `Text (Large)` 小组件：



在此情况下，设计器图面应类似于以下屏幕截图：

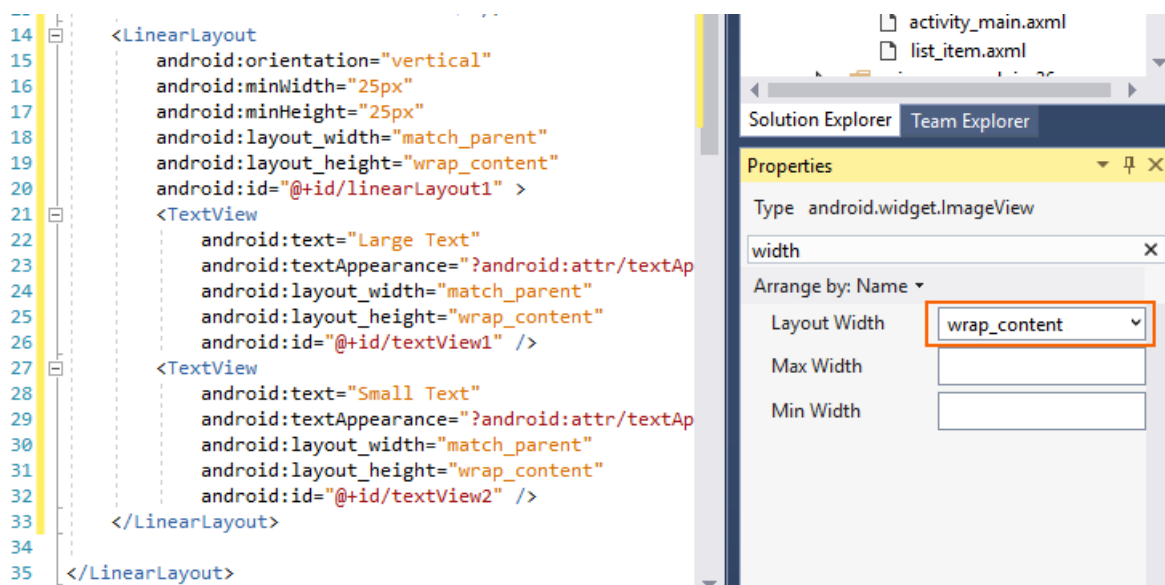


如果两个 `textView` 小组件不是全面 `linearLayout1`，你可以将其拖至 `linearLayout1` 中文档大纲并将它们置于，因此它们出现在上面的屏幕截图中所示（下缩进 `linearLayout1`）。

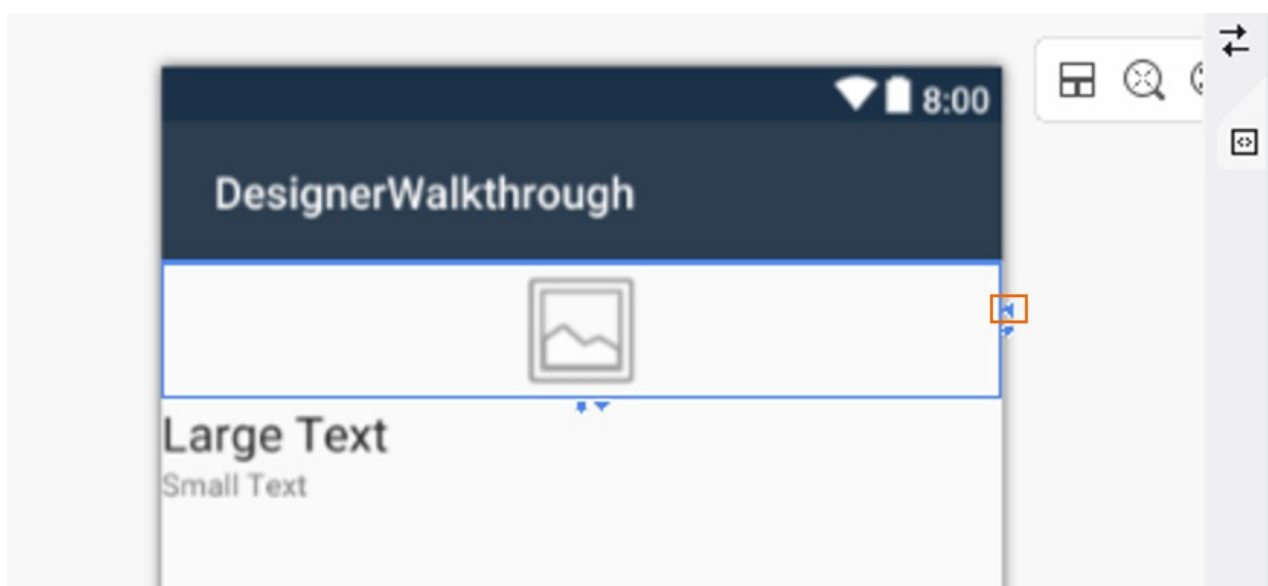
排列的用户界面

下一步是修改 UI 以显示 `ImageView` 在左侧，两个 `TextView` 右侧的小组件堆积 `ImageView`。

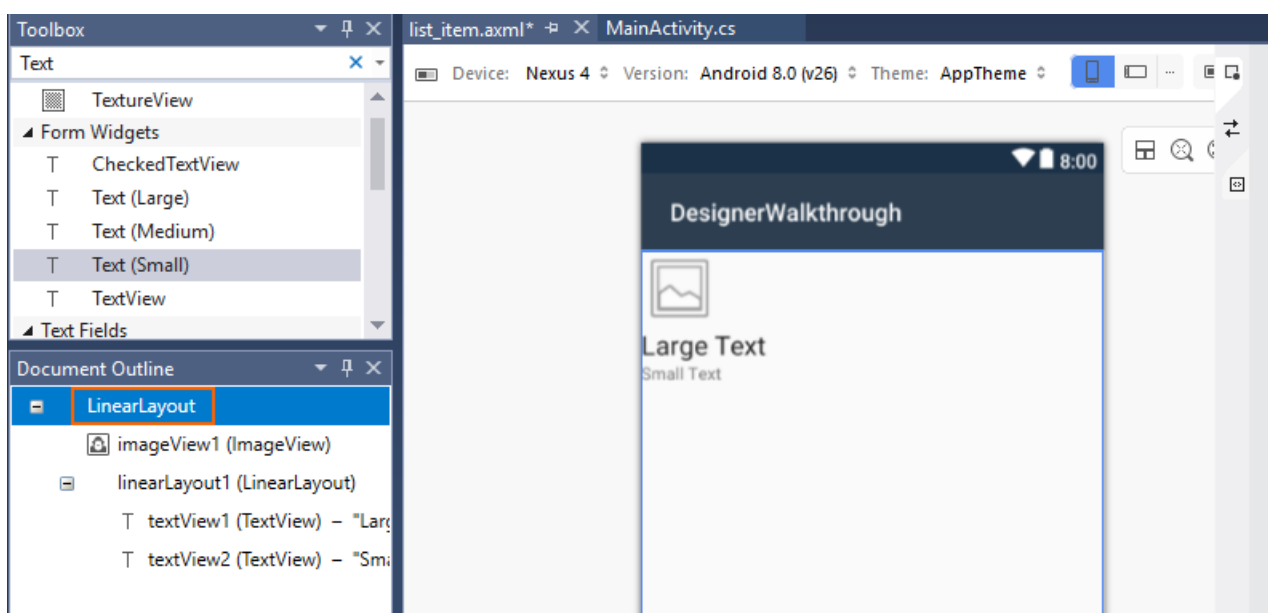
1. 选择 `ImageView`。
2. 在属性窗口，输入 `宽度` 在搜索框中，找到布局宽度。
3. 更改布局宽度将设置为 `wrap_content`：



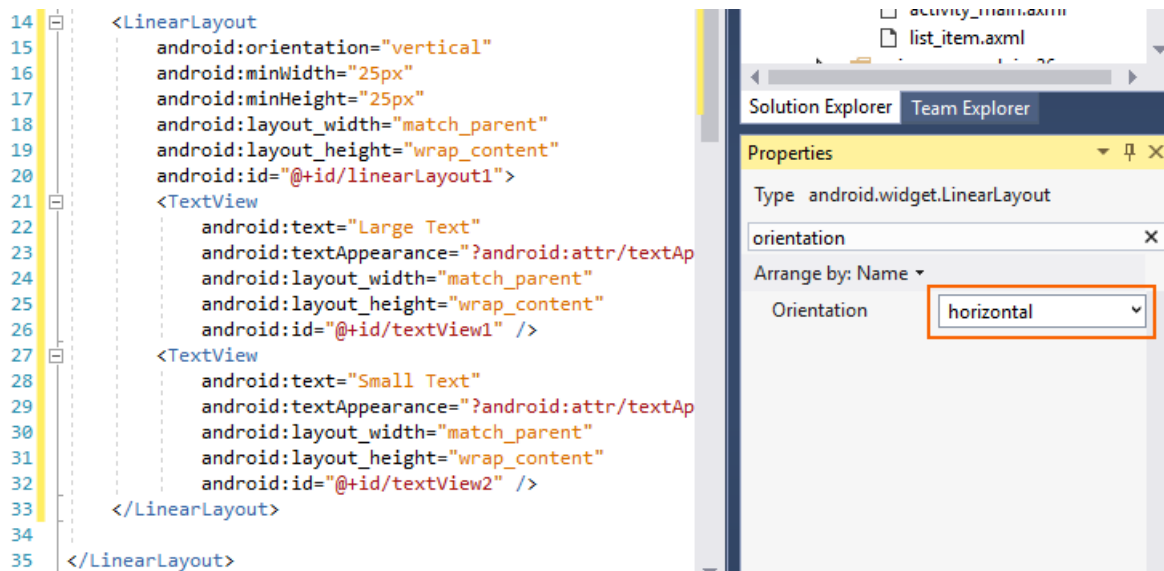
若要更改的另一种方法 `width` 设置为单击小组件以其宽度设置切换到右侧的三角形 `wrap_content` :



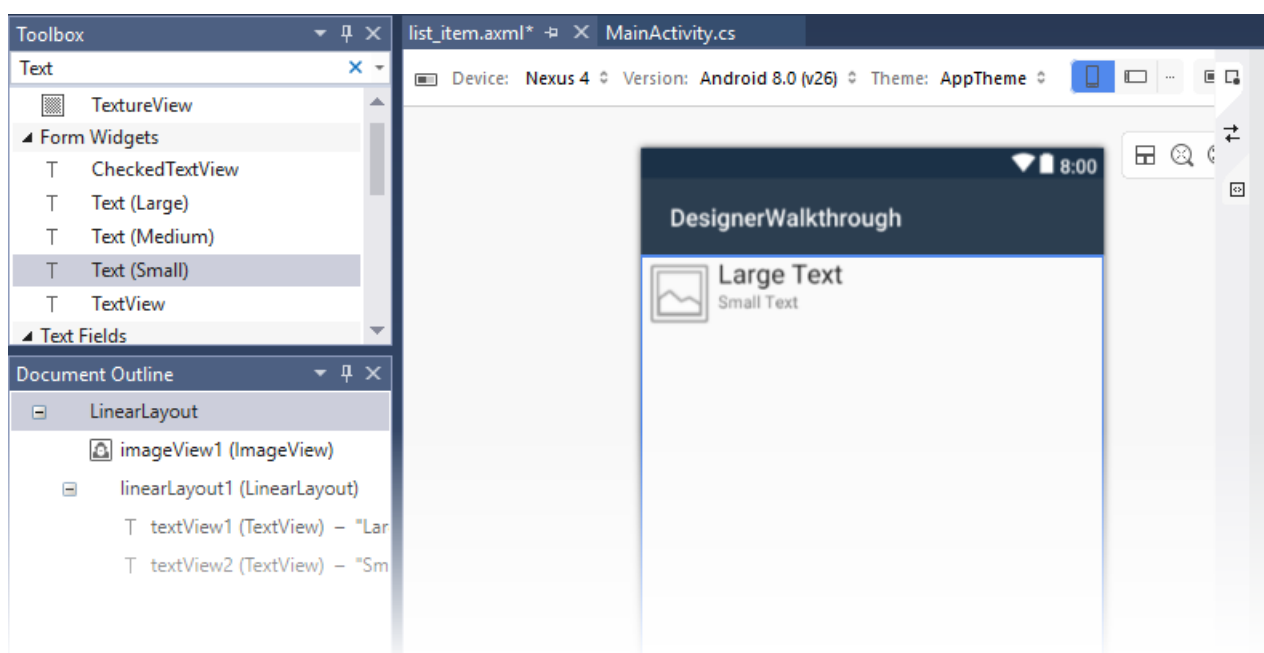
再次单击三角形将返回 `width` 设置为 `match_parent` 。接下来, 请转到文档大纲窗格, 然后选择根 `LinearLayout` :



与根 `LinearLayout` 选择, 返回到属性窗格中, 输入 `方向` 到搜索框中, 找到 `方向` 设置。更改方向到 `horizontal` :

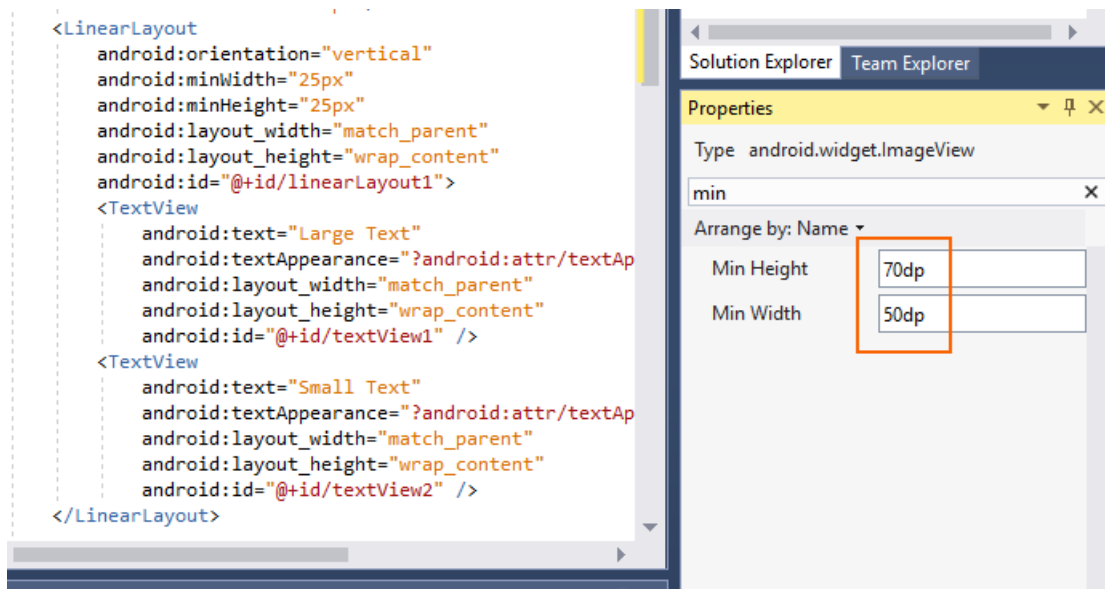


在此情况下，设计器图面应类似于以下屏幕截图。请注意，`TextView` 小组件已移至右侧 `ImageView`：

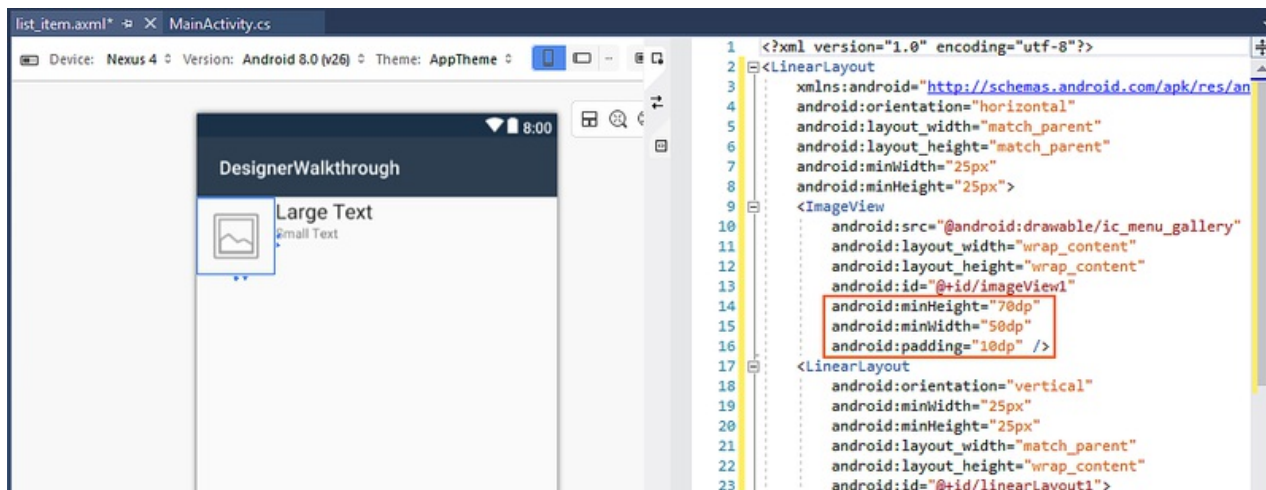


修改间距

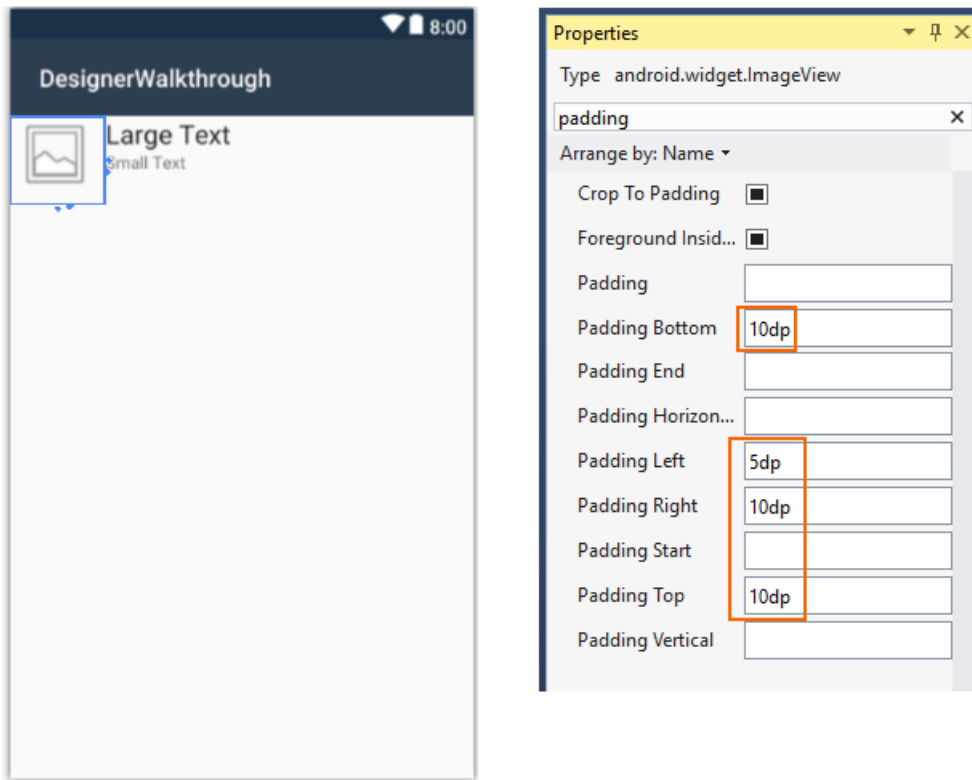
下一步是修改 UI，以提供更多空间之间的小组件中的填充和边距设置。选择 `ImageView` 设计图面上。在中属性窗格中，输入 `min` 在搜索框中。输入 `70dp` 有关最小高度并 `50dp` 有关最小宽度：



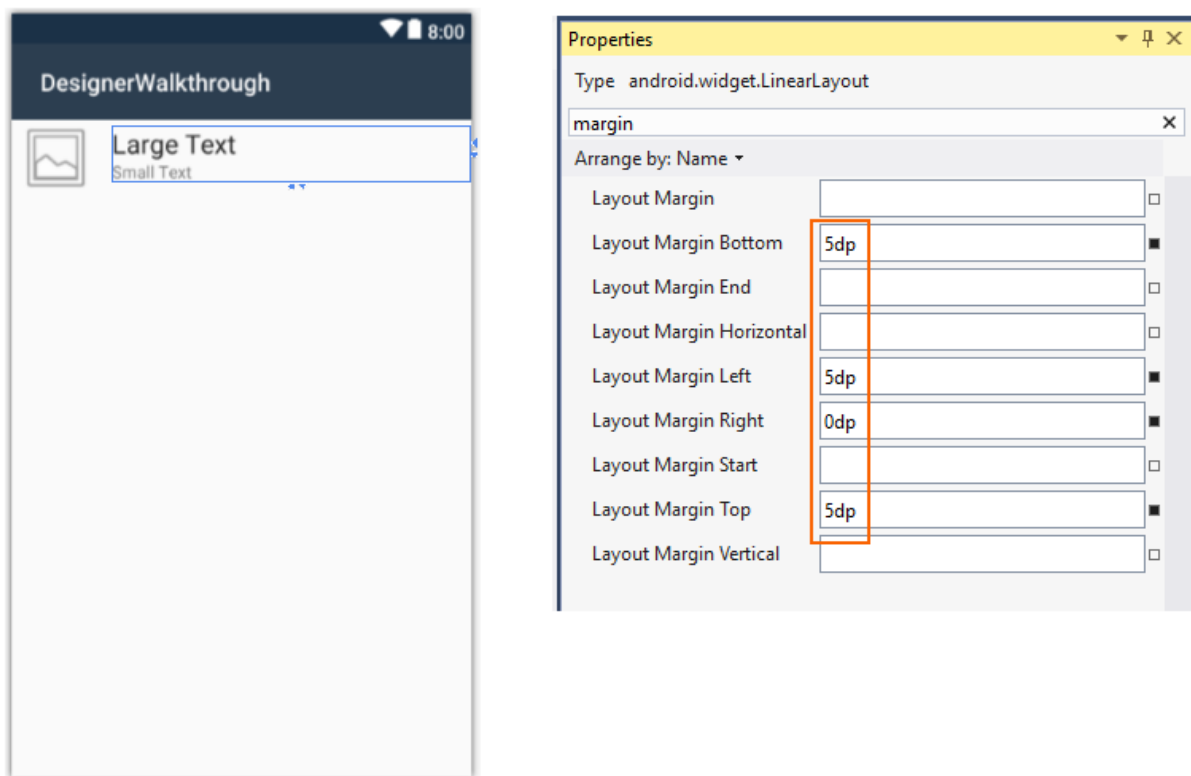
在中属性窗格中，输入 `padding` 在搜索框，并输入 `10dp` 有关填充。这些 `minHeight`，`minWidth` 并 `padding` 设置周围添加衬距的所有边 `ImageView` 和垂直拉长。请注意，布局 XML 更改为输入以下值：



下、左、右、和可以通过输入值到独立设置上边距设置填充底部，填充左侧，填充右侧，和填充顶部字段，分别。例如，设置填充左侧字段 `5dp` 和填充底部，填充右侧，并填充顶部字段到 `10dp`：



接下来, 调整的位置 `LinearLayout` 小工具, 它包含两个 `TextView` 小组件。在中文档大纲, 选择 `linearLayout1`。在中属性窗口中, 输入 `margin` 在搜索框中。设置布局边距底部, 布局距左侧, 并布局上边距到 `5dp`。设置布局边距右到 `0dp` :

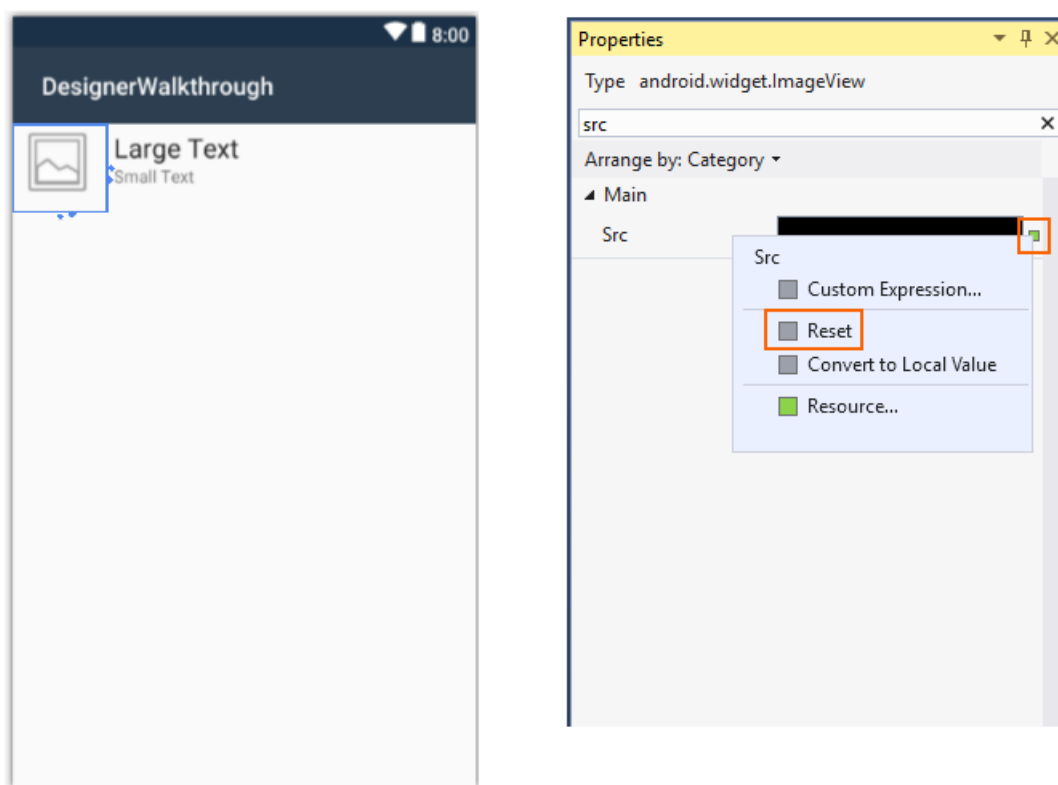


删除默认图像

因为 `ImageView` 用于显示的颜色 (而不是映像) 下, 一步是添加模板的默认图像源中删除。

1. 选择 `ImageView` 上设计器图面。

2. 在中属性, 输入src在搜索框中。
3. 单击右侧的小正方形Src属性设置并选择重置:

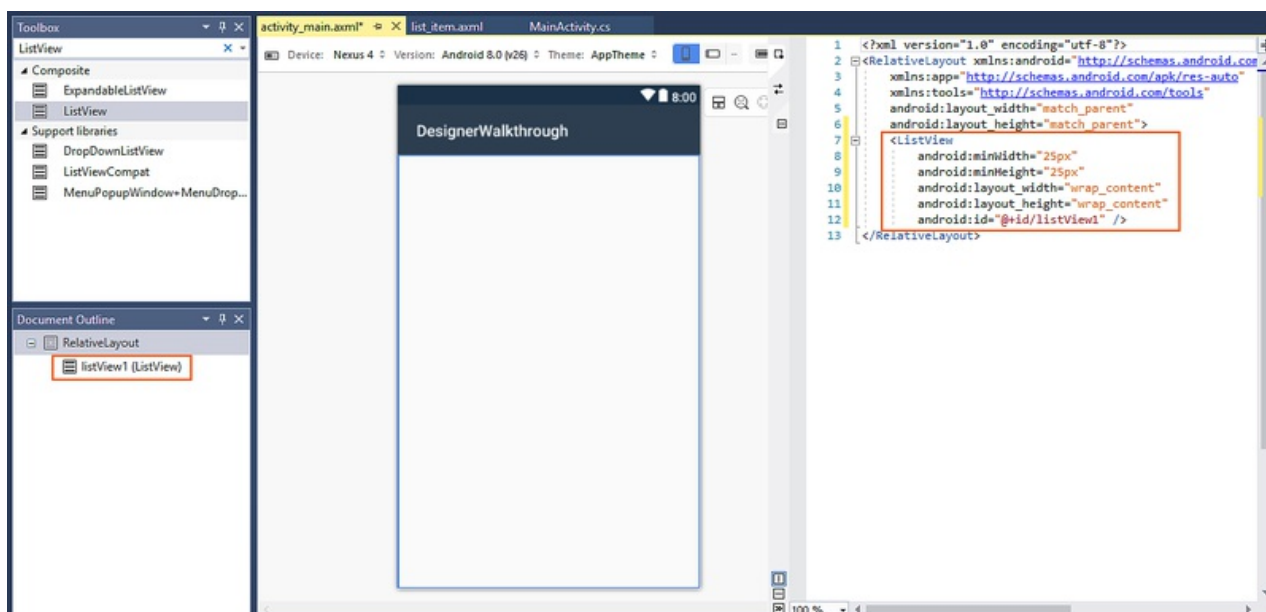


此操作将删除 `android:src="@android:drawable/ic_menu_gallery"` 源 XML 中为此, `ImageView` 。

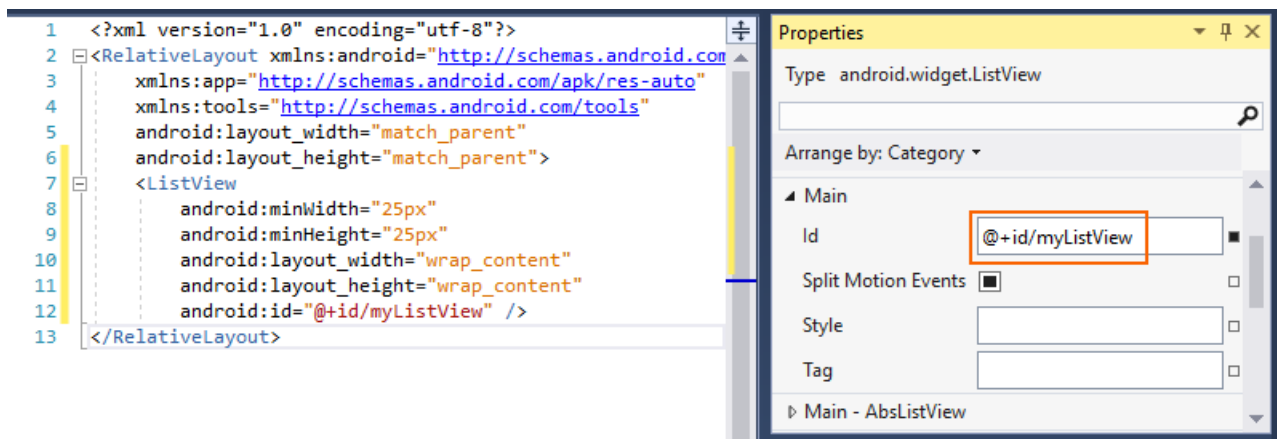
将 ListView 容器添加

既然list_item定义布局下, 一步是添加 `ListView` 到主布局。这 `ListView` 将包含一系列list_item。

在中解决方案资源管理器, 打开 `Resources/layout/activity_main.xml`。在中工具箱, 找到 `ListView` 小组件并将其拖动到设计图面。 `ListView` 在设计器中将为空白概述选中时其边框的蓝线除外。您可以查看文档大纲来确认 `ListView` 已正确添加:



默认情况下 `ListView` 给定 `Id` 的值 `@+id/listView1`。虽然 `listView1` 中仍处于选中状态文档大纲, 打开属性窗格中, 单击排列方式, 然后选择类别。打开 `Main`, 找到 `Id` 属性, 将其值更改为 `@+id/myListView`:



此时, 已准备好使用用户界面。

运行应用程序

打开**MainActivity.cs**和其代码替换为以下代码:

```
using Android.App;
using Android.Widget;
using Android.Views;
using Android.OS;
using Android.Support.V7.App;
using System.Collections.Generic;

namespace DesignerWalkthrough
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        List<ColorItem> colorItems = new List<ColorItem>();
        ListView listView;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);
            listView = FindViewById<ListView>(Resource.Id.myListView);

            colorItems.Add(new ColorItem()
            {
                Color = Android.Graphics.Color.DarkRed,
                ColorName = "Dark Red",
                Code = "8B0000"
            });
            colorItems.Add(new ColorItem()
            {
                Color = Android.Graphics.Color.SlateBlue,
                ColorName = "Slate Blue",
                Code = "6A5ACD"
            });
            colorItems.Add(new ColorItem()
            {
                Color = Android.Graphics.Color.ForestGreen,
                ColorName = "Forest Green",
                Code = "228B22"
            });

            listView.Adapter = new ColorAdapter(this, colorItems);
        }
    }

    public class ColorAdapter : BaseAdapter<ColorItem>
```

```

{
    List<ColorItem> items;
    Activity context;
    public ColorAdapter(Activity context, List<ColorItem> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }
    public override long GetItemId(int position)
    {
        return position;
    }
    public override ColorItem this[int position]
    {
        get { return items[position]; }
    }
    public override int Count
    {
        get { return items.Count; }
    }
    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];

        View view = convertView;
        if (view == null) // no view to re-use, create new
            view = context.LayoutInflater.Inflate(Resource.Layout.list_item, null);
        view.FindViewById<TextView>(Resource.Id.textView1).Text = item.ColorName;
        view.FindViewById<TextView>(Resource.Id.textView2).Text = item.Code;
        view.FindViewById<ImageView>(Resource.Id.imageView1).SetBackgroundColor(item.Color);

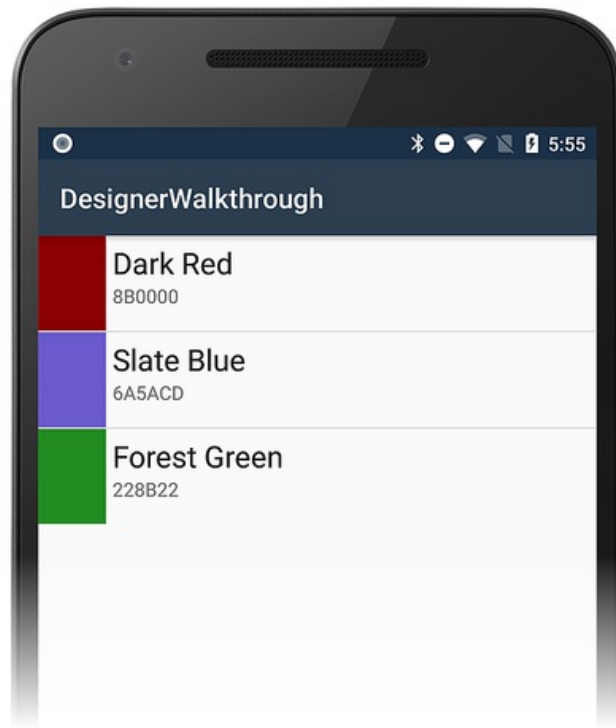
        return view;
    }
}

public class ColorItem
{
    public string ColorName { get; set; }
    public string Code { get; set; }
    public Android.Graphics.Color Color { get; set; }
}
}

```

此代码使用自定义 `ListView` 适配器加载颜色信息并在刚创建的 UI 中显示此数据。若要使此简短的示例，该颜色信息是硬编码在列表中，但无法修改该适配器，以便从数据源中提取颜色信息或动态计算。有关详细信息 `ListView` 适配器，请参阅 [ListView](#)。

生成并运行应用程序。下面的屏幕截图是在设备上运行时应用的显示方式的示例：



总结

本文介绍使用 Visual Studio 中 Xamarin.Android 设计器创建一个基本的应用的用户界面的过程。它演示了如何在列表中, 创建单个项的界面, 它说明如何添加小组件和直观地对其进行安排。它还介绍了如何将资源分配, 然后对这些小组件中设置各种属性。

Xamarin.Android 设计器基础知识

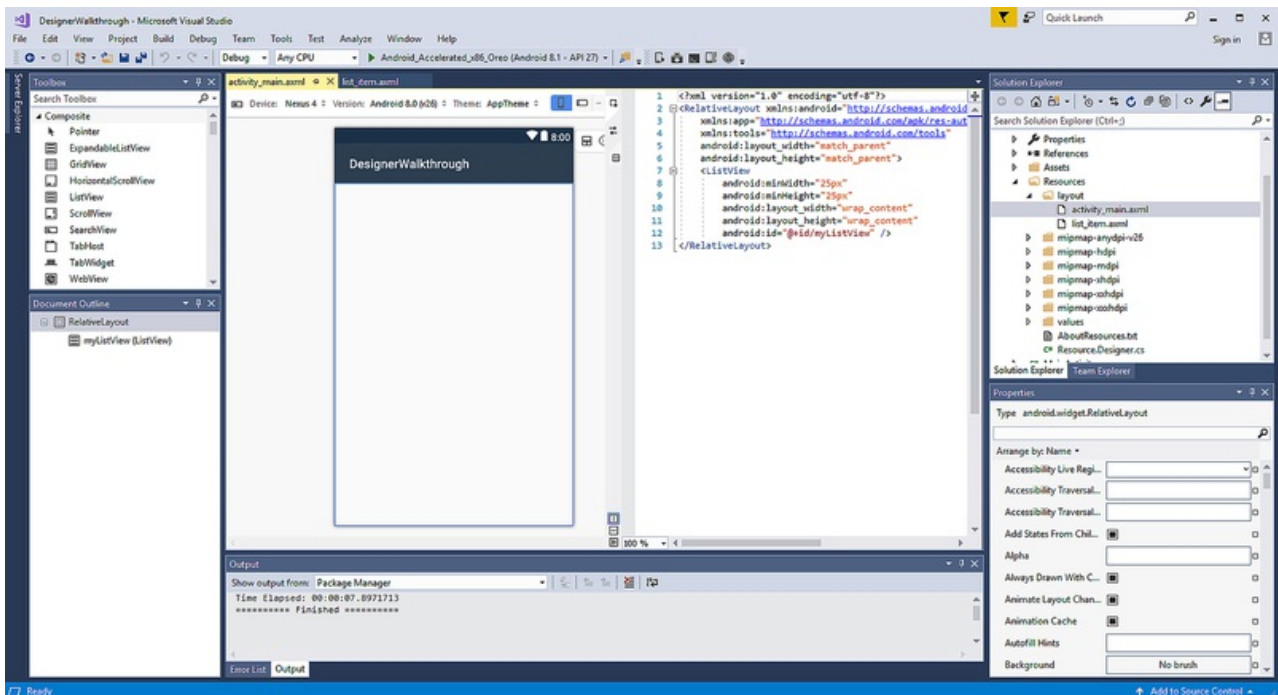
2018/10/26 • [Edit Online](#)

本主题介绍 Xamarin.Android 设计器功能, 说明如何启动设计器中, 介绍设计图面上, 并详细介绍了如何使用属性窗格编辑小组件属性。

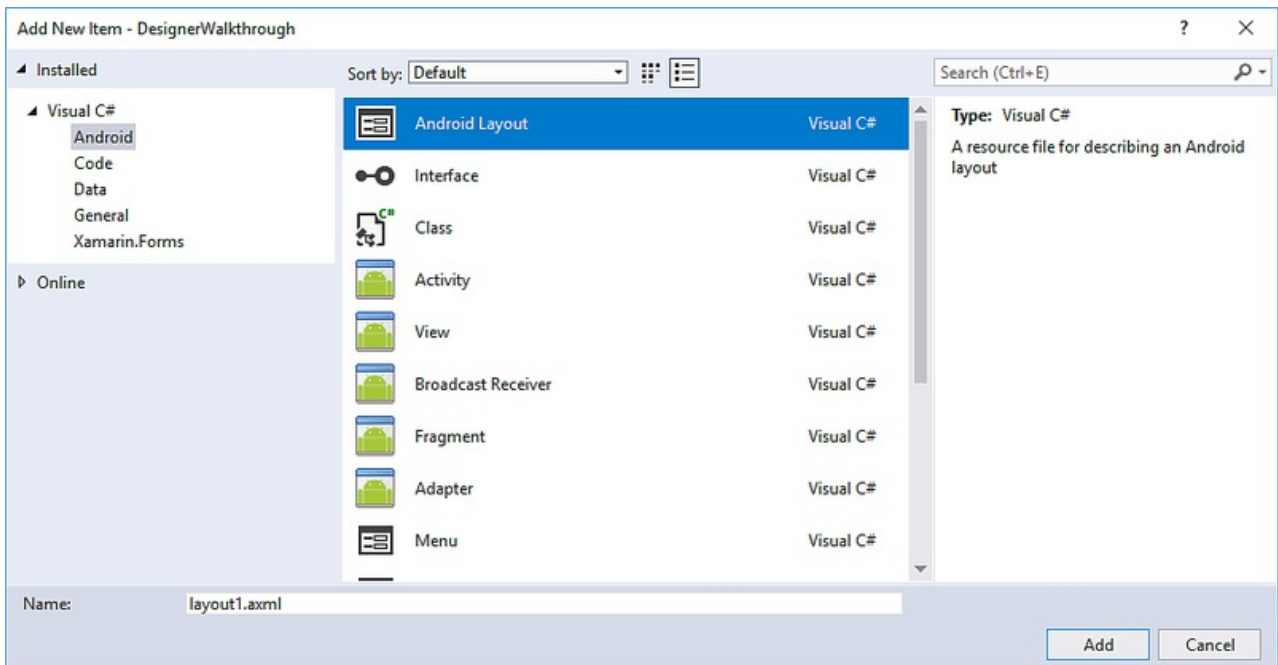
- [Visual Studio](#)
- [Visual Studio for Mac](#)

启动设计器

创建布局, 或通过双击现有的布局文件, 它可以启动时, 在设计器将自动启动。例如, 双击 **activity_main.xml** 中资源 > 布局文件夹将加载在设计器, 如以下屏幕截图中所示:



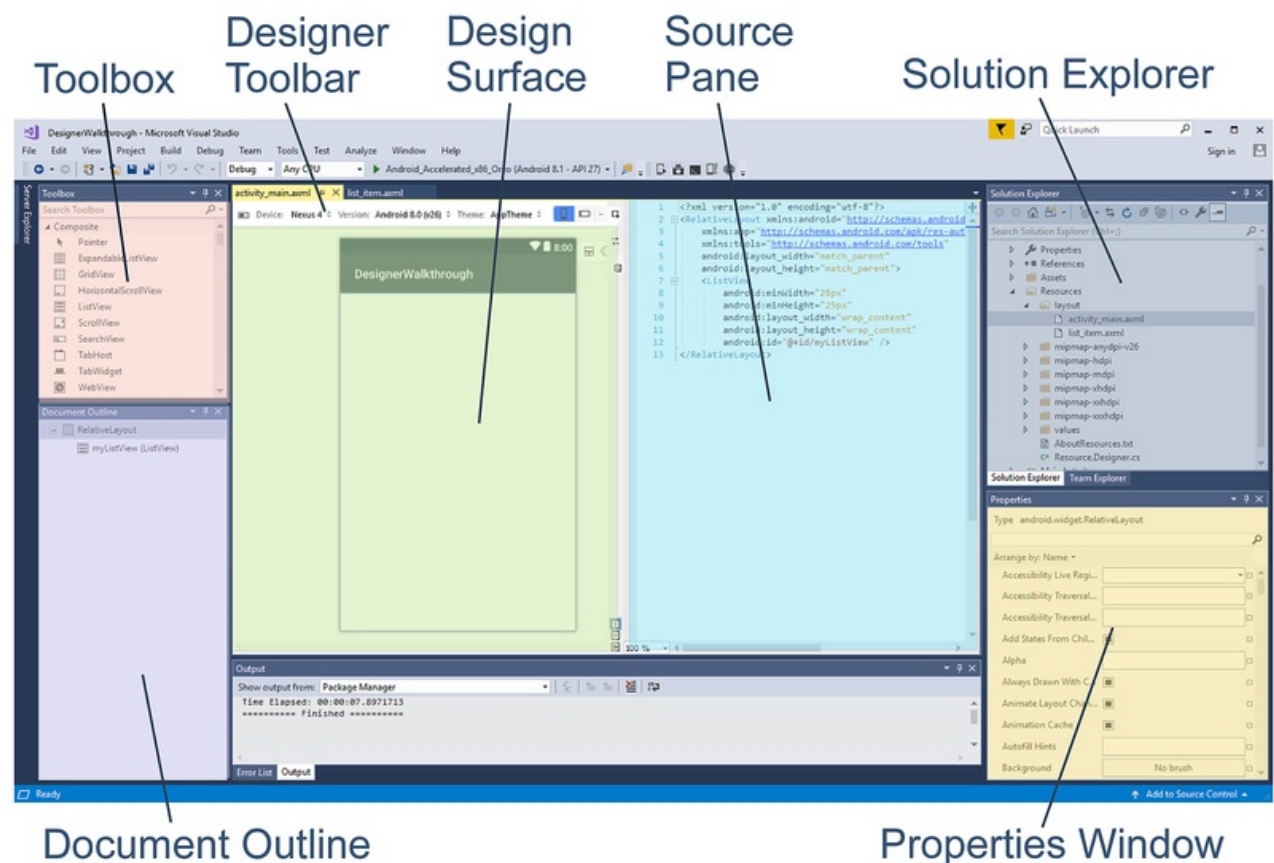
同样, 右键单击可以添加新的布局布局中的文件夹解决方案资源管理器, 然后选择添加 > 新建项...> **Android 布局**:



这将创建一个新 .axml 布局文件并将其加载到设计器。

设计器功能

在设计器由几个部分，支持它的各种功能，如以下屏幕截图中所示：



当您编辑设计器中的布局时，使用以下功能来创建和调整你的设计：

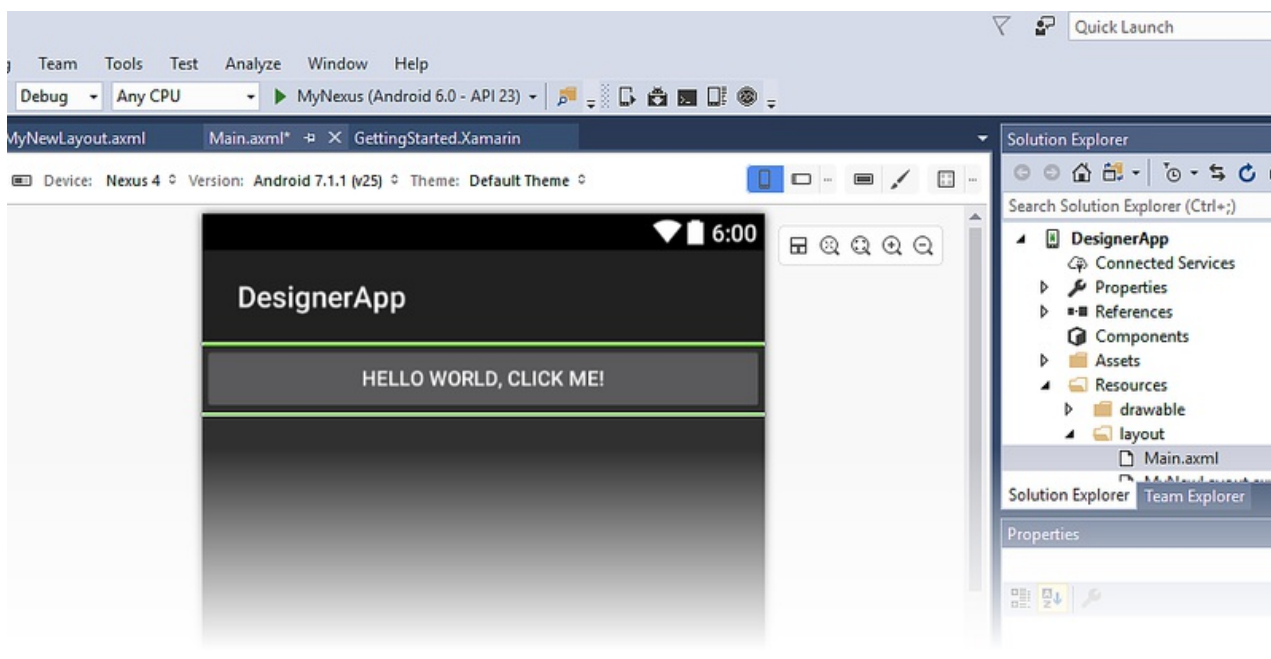
- **设计图面**—可简化用户界面可视方式构建过程，为您提供的可编辑表示形式如何布局将显示在设备上。设计图面内显示设计窗格(使用设计器工具栏位于其上方)。
- **源窗格**—提供的基础上显示的设计与相对应的 XML 源视图设计图面。
- **设计器工具栏**—显示的选择器列表：设备，版本，主题，布局配置和操作栏设置。设计器工具栏还包括图标

用于启动主题编辑器和用于启用材料设计网格。

- **工具箱**–提供了一系列小组件和布局，您可以拖放到设计图面。
- **属性窗口**–列出了用于查看和编辑所选小组件的属性。
- **文档大纲**–显示构成布局小组件的树。可以单击以使其上所选的树中的项设计图面。此外，单击树中的项目加载项的属性存储到**属性窗口**。

设计图面

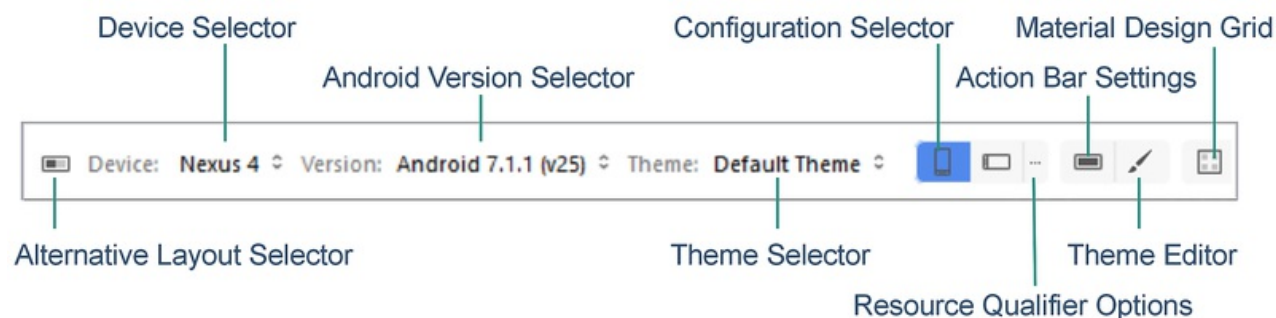
在设计器使您可以从工具箱拖到拖放小组件设计图面。当与设计器中的小组件交互（通过添加新的小组件或重新定位现有的）时，将显示垂直和水平线以标记可插入点。在以下示例中，一个新 `Button` 要小组件拖动到设计图面：



此外，可以将小组件复制：您可以使用复制和粘贴将复制小组件，也可以拖放时按现有小组件CTRL密钥。

设计器工具栏

设计器工具栏(之上设计图面) 提供配置选择器和工具菜单：



设计器工具栏可以访问以下功能：

- **备用布局选择器**–允许你选择从不同的布局版本。
- **设备选择器**–定义的限定符（如屏幕尺寸、分辨率和键盘可用性）的一组与特定设备关联。此外可以添加和删除新的设备。
- **Android 版本选择器**–布局所面向的 Android 版本。在设计器将呈现根据所选的 Android 版本的布局。
- **主题选择器**–选择布局的用户界面主题。
- **配置选择器**–选择该设备的配置，例如**纵向**或**横向**。

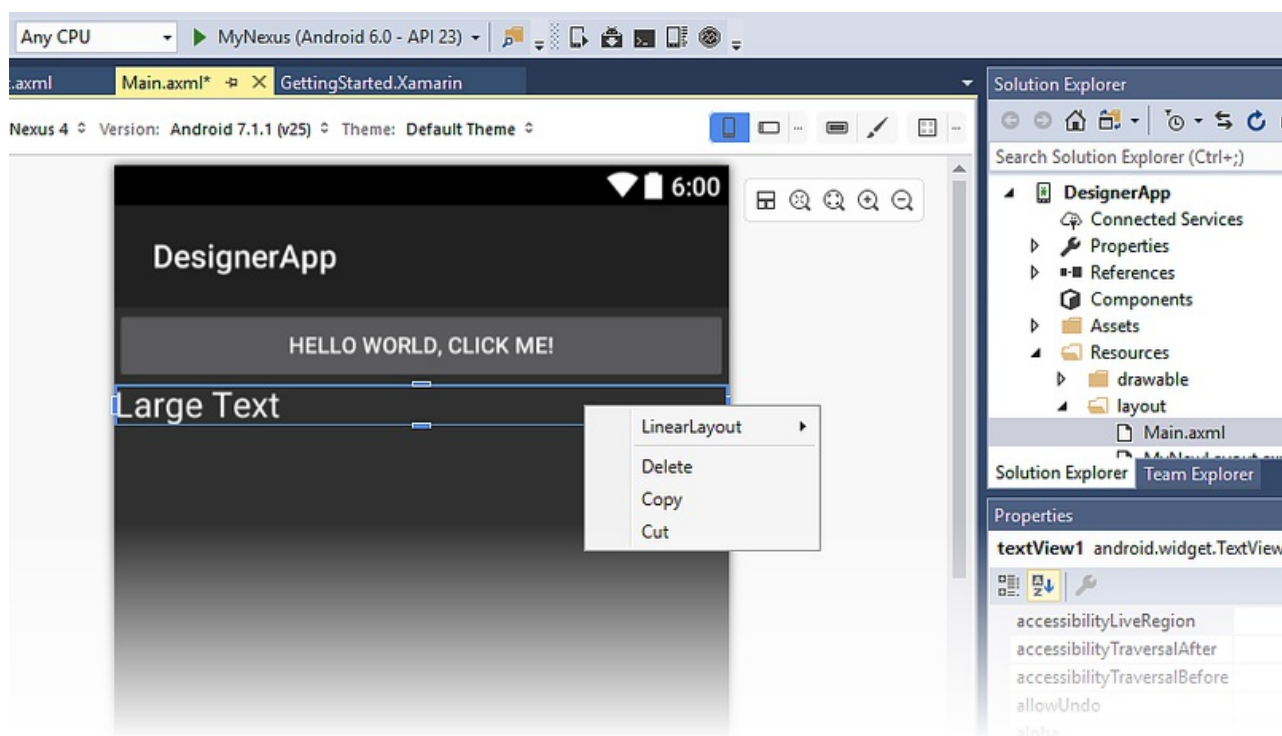
- **资源限定符选项**–将打开一个对话框，其中提供了用于选择下拉列表菜单 *语言*，*UI 模式*，*夜间模式*，和 *圆角屏幕* 选项。
- **操作栏设置**–配置布局的操作栏设置。
- **主题编辑器**–会打开 *Theme Editor*，从而有可能用于自定义的所选主题元素。
- **材质设计网格**–启用或禁用 *材料设计网格*。材料设计网格与相邻的下拉列表菜单项打开一个对话框，允许你自定义网格。

这些主题中的更详细地解释每个功能：

- **资源限定符和可视化效果选项**提供有关的信息设备选择器，**Android 版本选择器**，主题选择器，配置选择器，资源限定选项，并操作栏设置。
- **备选布局视图**介绍了如何使用备用布局选择器。
- **Xamarin.Android 设计器材料设计功能**提供的全面概述**Theme Editor**并且材料设计网格。

上下文菜单命令

上下文菜单中同时设计图面并在文档大纲。此菜单会显示可用于所选小组件和其容器，使你更轻松地对容器执行操作的命令（这并不总是容易上选择设计图面）。下面是在上下文菜单的示例：

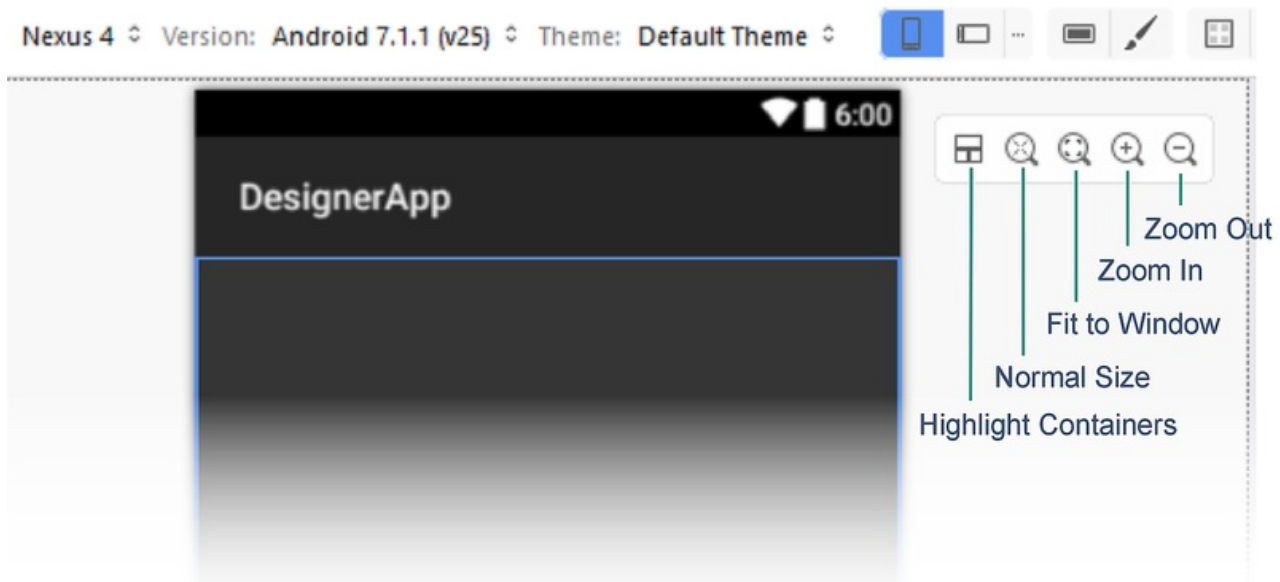


在此示例中，右键单击 `TextView` 打开上下文菜单，提供多个选项：

- **LinearLayout** –将打开用于编辑的子菜单 `LinearLayout` 的父级 `TextView`。
- **删除，副本，和剪切**–适用于在右击操作 `TextView`。

缩放控件

设计图面支持缩放通过多个控件，如所示：



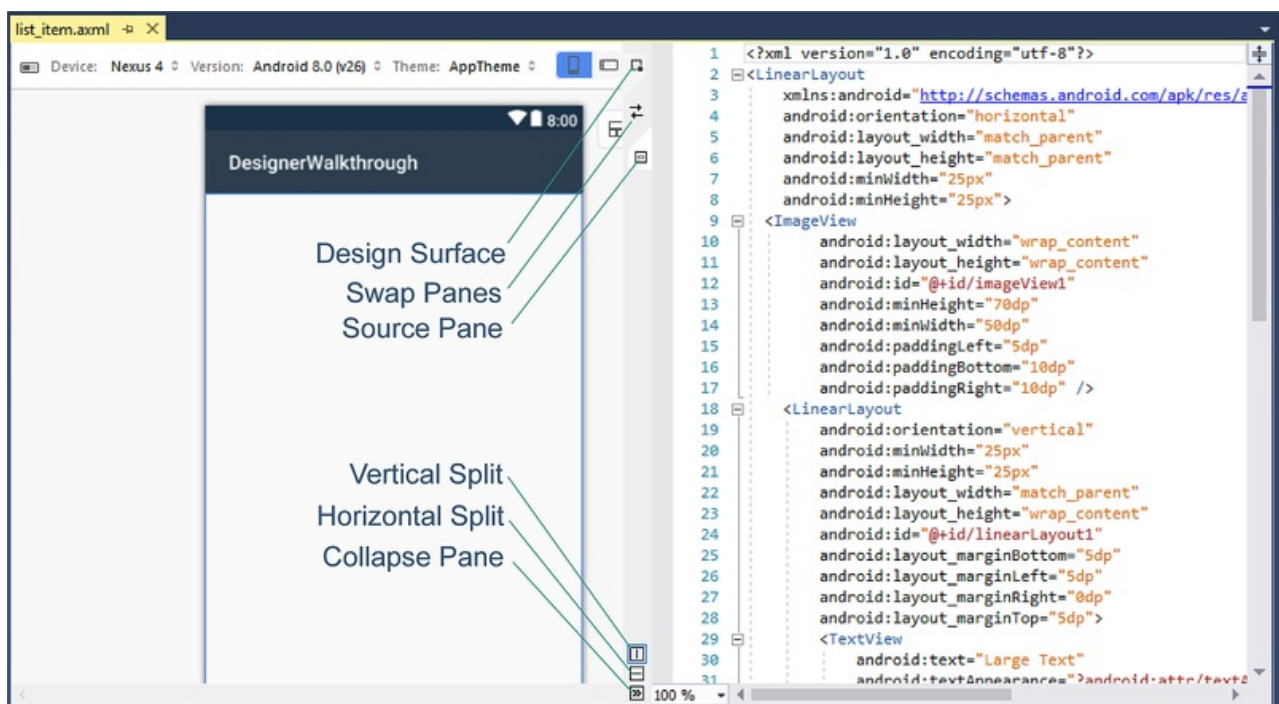
这些控件使其更轻松地查看特定区域的设计器中的用户界面：

- **突出显示容器**—上将突出显示容器设计图面，以便可以更轻松地找到时放大和缩小。
- **常规尺寸**—将布局的像素呈现，以便您可以查看所选设备分辨率布局外观。
- **适合窗口**—设置缩放级别，使整个布局位于设计图面上可见。
- **放大**—放大布局每次单击将以增量方式放大。
- **缩小**—以增量方式缩小每次单击，使显示在设计图面上较小的布局。

请注意，所选的缩放设置不影响在运行时应用程序的用户界面。

设计和源窗格之间切换

之间的中心条带中设计并源窗格中，有几个按钮用于修改如何设计和源窗格将显示：



这些按钮执行以下步骤：

- **设计**—此最顶部的按钮设计，选择设计窗格。单击此按钮时，工具箱并属性启用窗格和文本编辑器工具

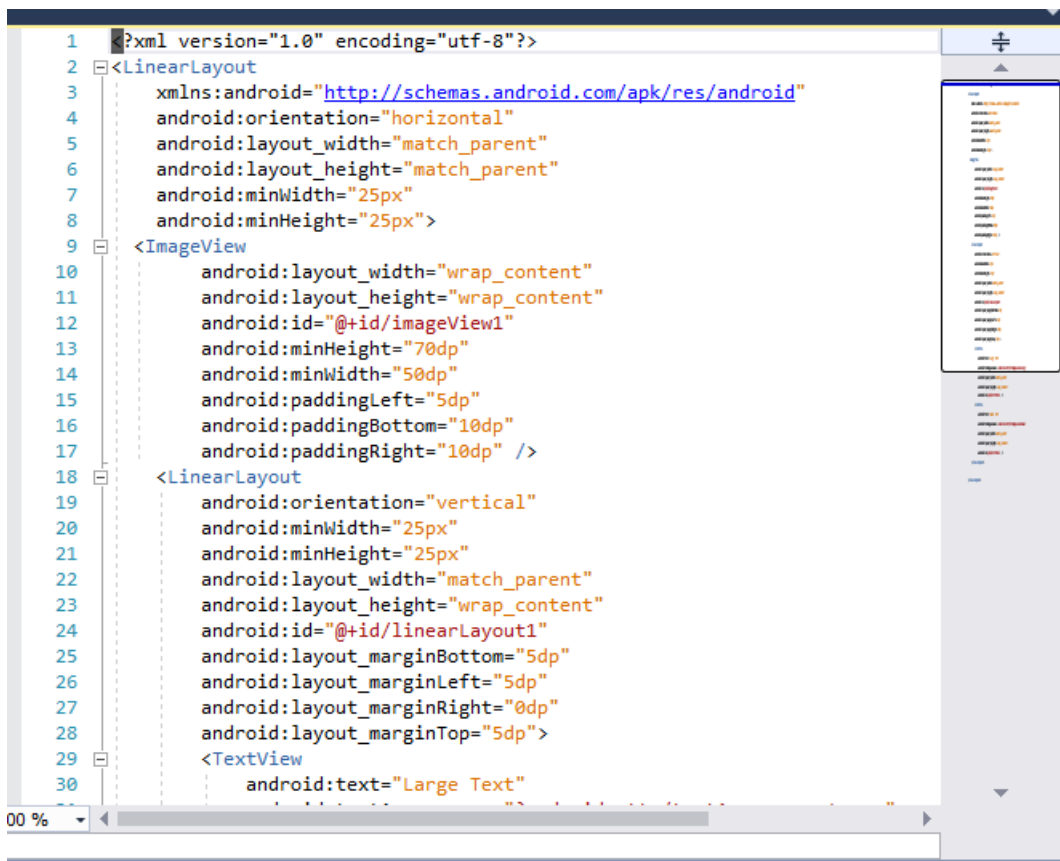
栏不会显示。当**折叠**(见下文), 在单击按钮设计而无需单独显示窗格源窗格。

- **交换窗格**-此按钮 (类似于两个对立的箭头) 交换设计并源窗格, 以便源窗格位于左侧并设计窗格位于右侧。然后再次单击该交换两个窗格重新添加到其原始位置。
- **源**-此按钮 (类似于两个对立的尖括号) 选择源窗格。单击此按钮时, 工具箱和属性窗格处于禁用状态并文本编辑器工具栏由 Visual Studio 顶部显示。当**折叠**单击按钮后 (请参阅下文), 单击**源**按钮将显示源窗格中, 而不是设计窗格。
- **垂直拆分** - (这类似于垂直条) 此按钮, 将显示设计并源窗格并行。这是默认排列方式。
- **水平拆分**-此按钮 (这类似于水平栏), 将显示设计上面源窗格。交换窗格可通过单击放置源上面设计窗格。
- **折叠窗格**-此按钮 (类似于两个右尖括号)"折叠"的双窗格显示设计并源到之一的一个视图两个窗格。此按钮将变为**展开窗格**按钮 (类似于两个左尖括号), 可通过单击该视图将返回到双窗格 (设计和源) 显示模式。

当**折叠窗格**单击时, 仅设计显示窗格。但是, 您可以单击**源**按钮, 而不是视图仅源窗格。单击设计按钮将再次返回到设计窗格。

源窗格

源窗格中显示基础上显示的设计的 XML 源设计图面。因为这两个视图在同一时间不可用, 则可以通过该设计的可视表示形式和设计的基础 XML 源之间来回创建 UI 设计:



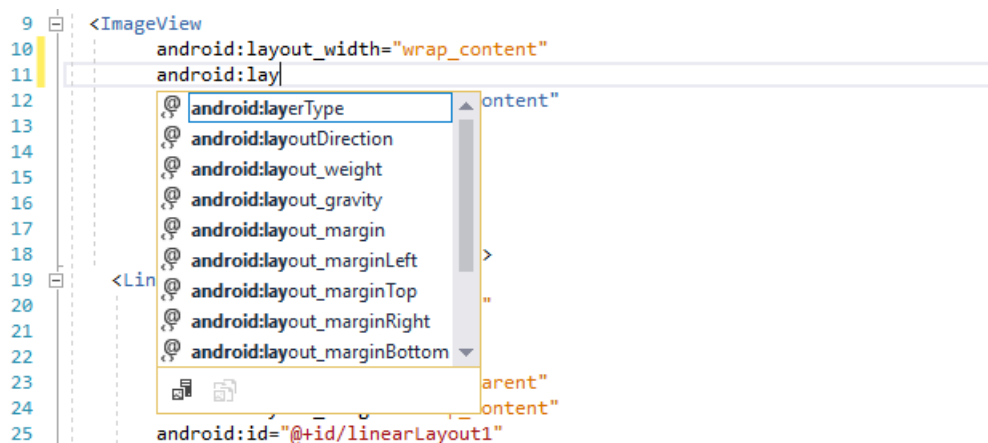
XML 源所做的更改立即呈现上设计图面; 上所做的更改设计图面会导致 XML 源中显示源窗格相应地更新。当对中的 XML 进行更改源窗格、自动完成和 IntelliSense 功能可供下一步所述的基于 XML 的 UI 开发速度。

对于更高版本中方便地导航时使用的长 XML 文件, 源窗格支持 Visual Studio 滚动条 (如上直接上面的屏幕截图中所示)。有关滚动条的详细信息, 请参阅[如何通过自定义滚动条来跟踪代码](#)。

自动完成功能

当您开始键入某个小组件属性的名称时, 您可以按 CTRL + 空格键若要查看可能的完成列表。例如, 输入后

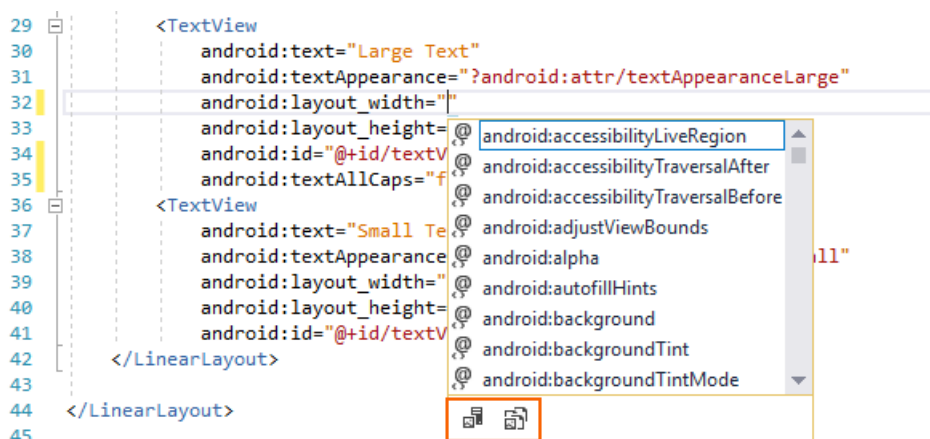
android:lay 在下面的示例 (键入后跟 CTRL + 空格键), 显示以下列表:



按ENTER接受第一个列出的完成时，或使用箭头键来滚动到所需的完成，并按ENTER。或者，可以使用鼠标滚动到并单击所需的完成。

IntelliSense

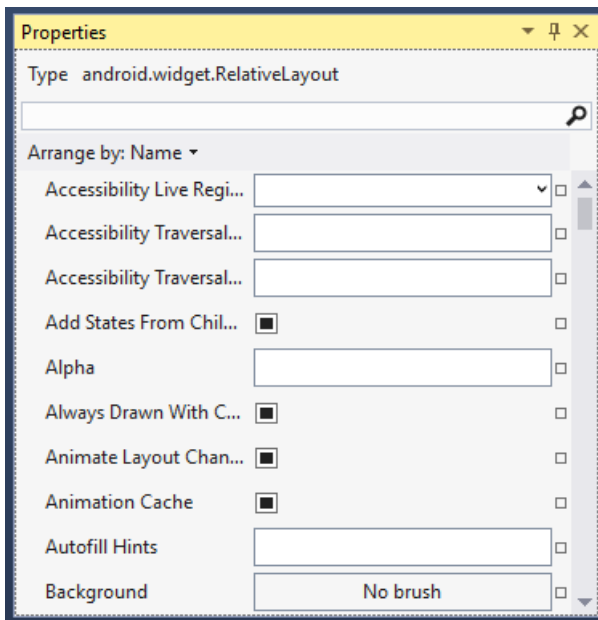
小组件的输入的新属性并开始将其分配一个值后，IntelliSense 会弹出后触发器字符类型化，并提供要用于该属性的有效值的列表。例如，在第一个双引号输入 `android:layout_width` 在以下示例中，自动完成功能选择器会弹出来提供针对此宽度的有效选项的列表：



在此弹出窗口的底部两个按钮（如上面的屏幕截图中红色中所述）。单击 **Project** 资源左侧的按钮将列表限制对资源的同时单击应用程序项目的一部分框架资源右侧按钮限制到列表显示框架中的可用资源。这些按钮来切换打开或关闭：你可以单击它们，以便禁用筛选操作，每个都提供。

属性窗格

设计器支持通过小组件属性的编辑属性窗格：



中列出的属性属性根据该小组件选择窗格中更改设计图面。

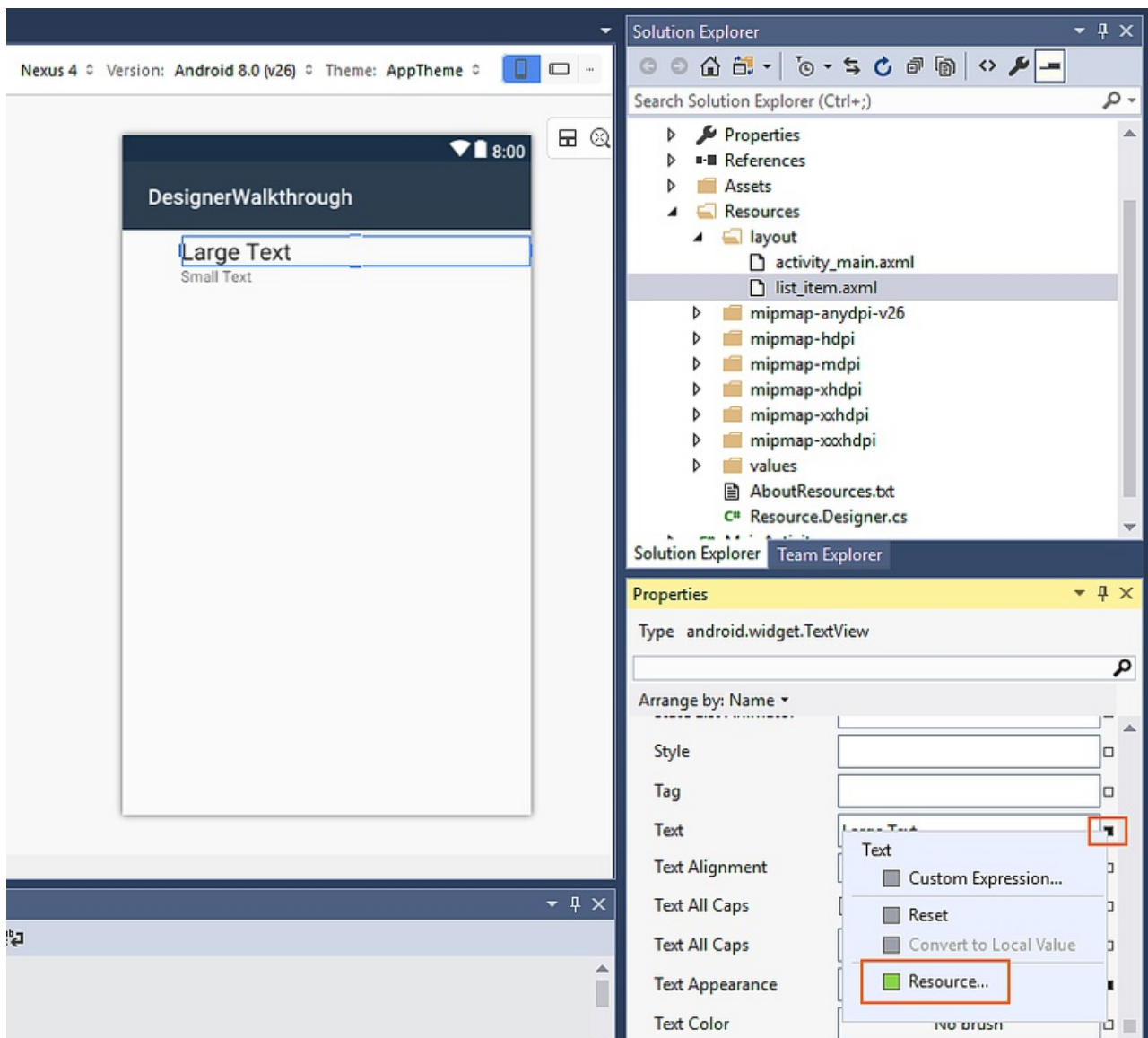
默认值

大多数小组件的属性中将为空白属性窗口由于它们的值继承自 Android 所选主题。属性窗口将仅显示显式设置为所选小组件的值; 它不会从主题继承的值。

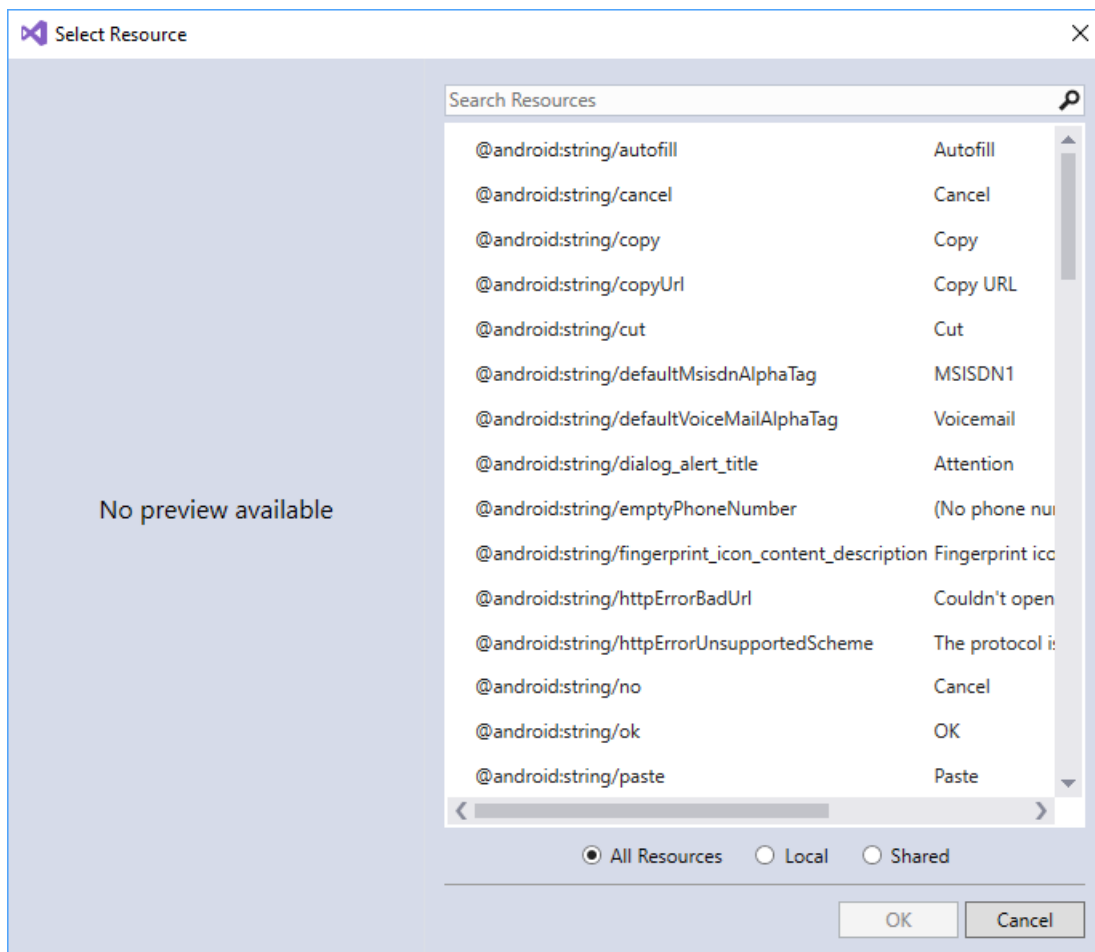
引用资源

某些属性可以引用布局以外的文件中定义的资源 .axml 文件。此类型的最常见的用例都 string 和 drawable 资源。但是, 引用可还用于其他资源, 如 Boolean 值和维度。如果某个属性支持资源引用, 属性的文本项旁边显示浏览图标 (方形)。此按钮将打开一个资源选择器, 单击时。

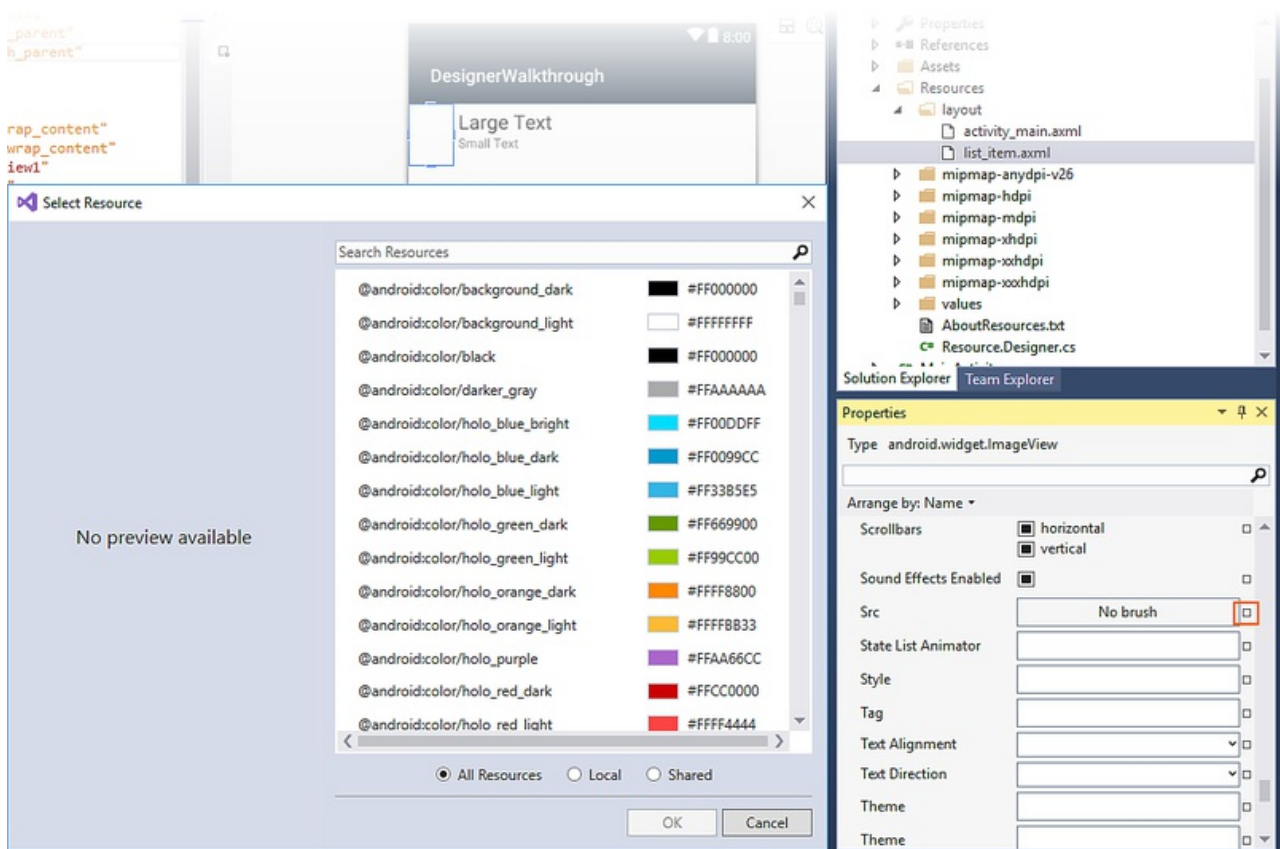
例如, 下面的屏幕截图显示了可用的选项时单击右侧的文本字段的变暗的正方形 Text 中的小组件属性窗口:



当资源... 单击时, 选择资源的对话框:



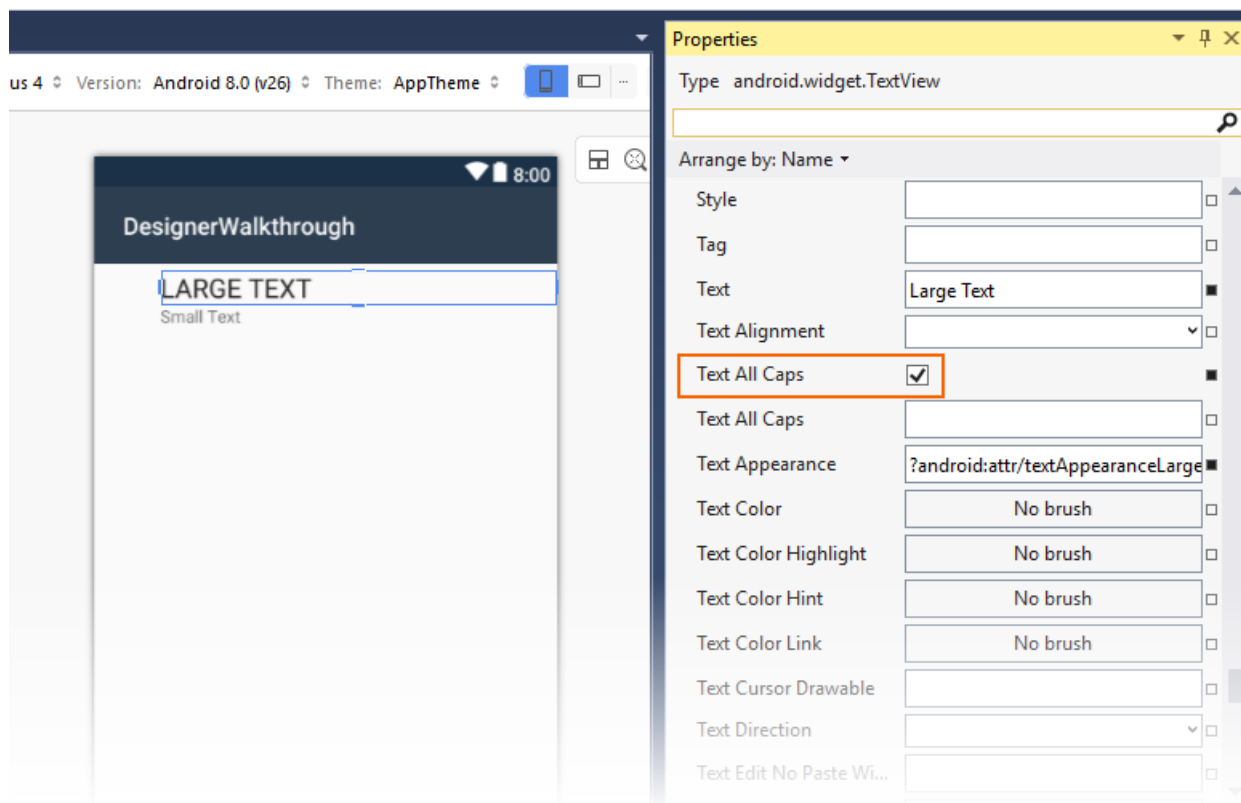
可以从此列表中，选择要用于该小组件，而不是硬编码中的文本的文本资源属性窗格。下一步的示例所示的资源选择器 `Src` 属性的 `ImageView`：



单击右侧的空方块 `Src` 属性将打开选择资源对话框，其中包含从颜色（如上所示）到绘图资源的列表。

布尔值属性的引用

布尔属性通常选择作为属性窗口中的属性旁边的复选标记。可以指定 `true` 或 `false` 值通过选中或取消选中此复选框，也可以通过单击属性右侧的填充深的方块选择的属性引用。在以下示例中，文本更改为全部大写通过单击文本的所有大写关联与所选的布尔属性引用 `TextView`：

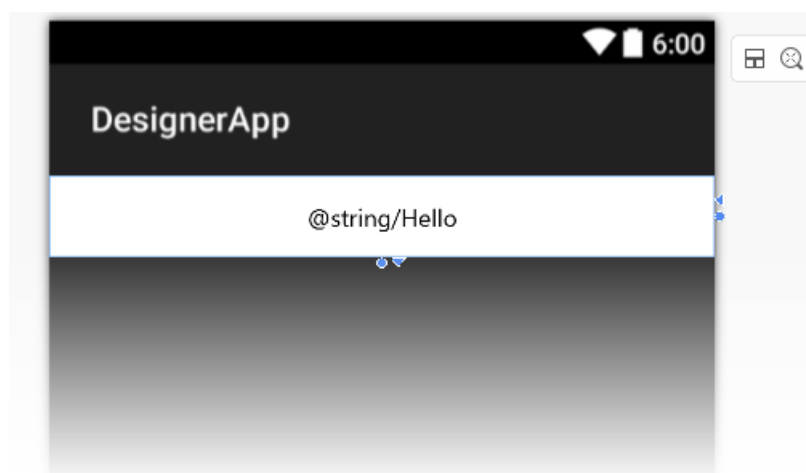


编辑属性内联

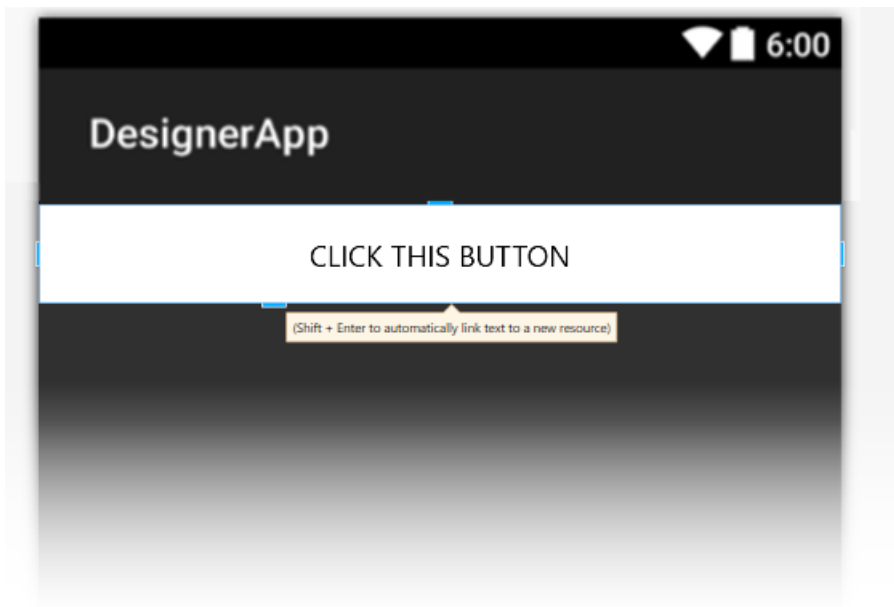
Android 设计器支持上直接编辑某些属性设计图面（因此您无需搜索属性列表中的这些属性）。可以直接编辑的属性包括文本、边距和大小。

Text

一些小组件的文本属性（如 `Button` 并 `TextView`），可以直接上编辑设计图面。双击某个小组件将其置于编辑模式下，如下所示：



可以输入新的文本值，也可以输入新的资源字符串。在以下示例中，`@string/hello` 资源将被替换为文本，`CLICK THIS BUTTON`：

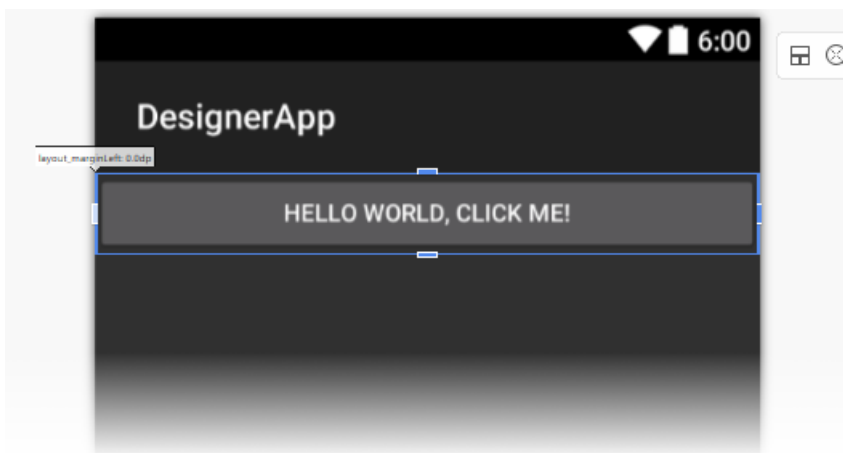


此更改存储在该小组件 `text` 属性; 它不会修改分配给的值 `@string/hello` 资源。当你键入新的文本字符串中时, 可以按 `Shift + Enter` 自动链接到新的资源的输入的文本。

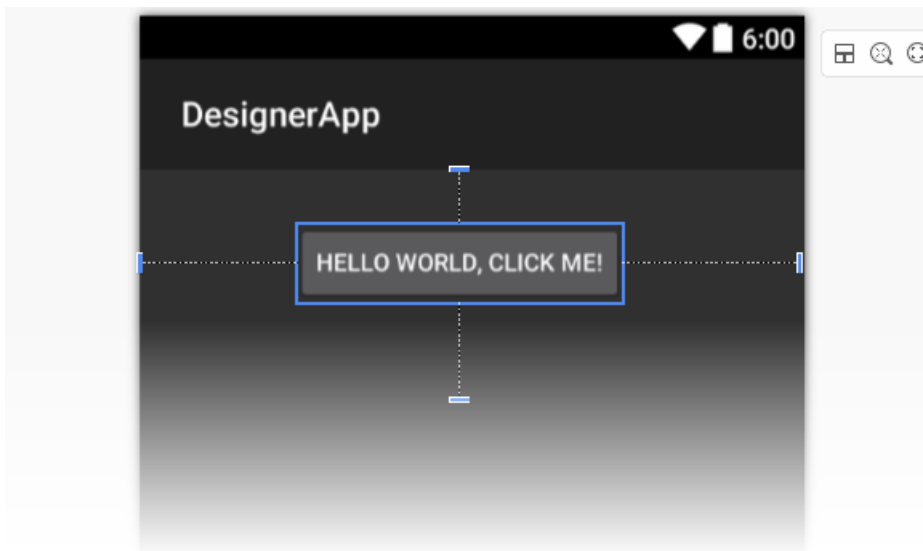
边距

当选择小组件时, 设计器会显示句柄, 可用于以交互方式更改的大小或边距的小组件。处于选定状态时单击小组件边距编辑模式和大小编辑模式之间切换。

当您第一次单击小组件时, 显示边距句柄。如果将鼠标移动到一个句柄, 设计器会显示该句柄将更改的属性 (如下所示的 `layout_marginLeft` 属性):

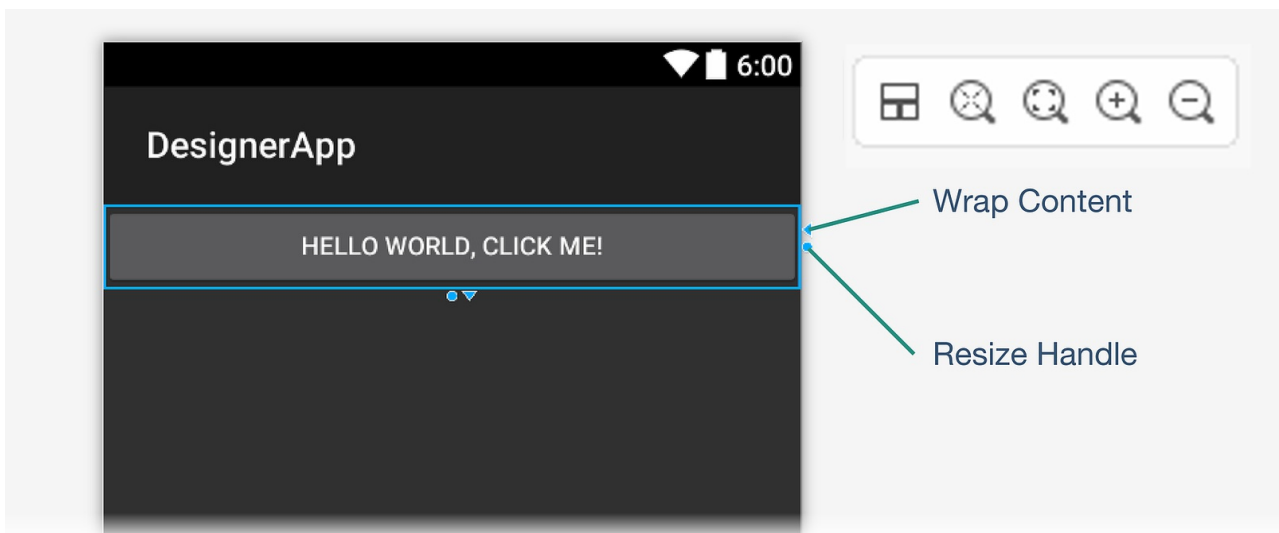


如果已设置了边距, 会显示虚线, 指示边距占用的空间:



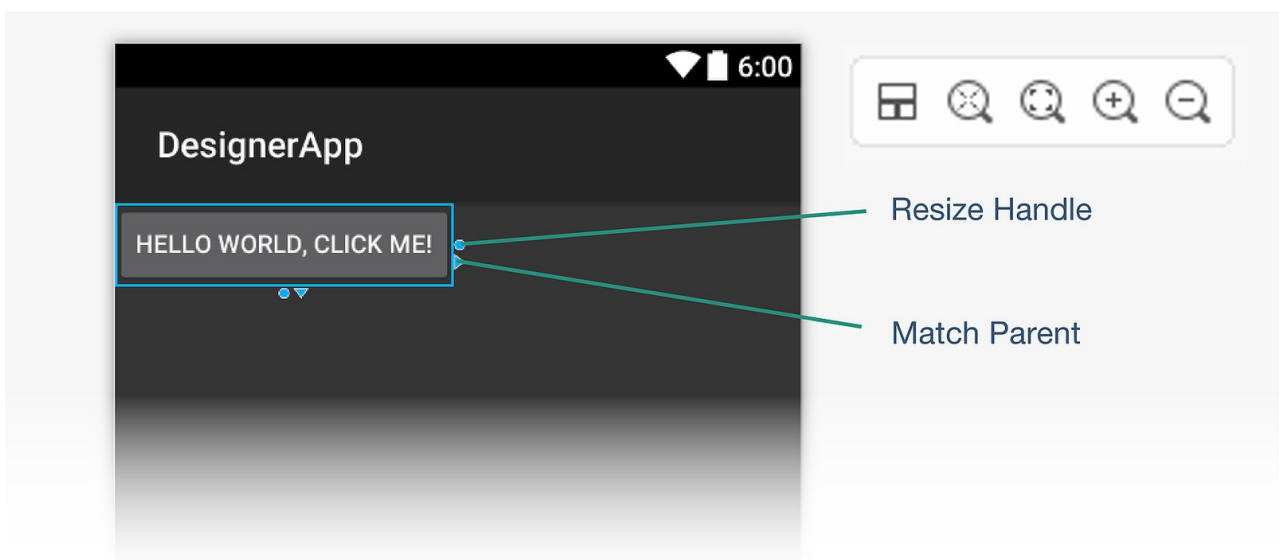
大小

前面曾提到, 可以通过单击小组件, 它已处于选定状态时切换到大小编辑模式。单击要设置到的指示维度的大小的三角形句柄 `wrap_content` :



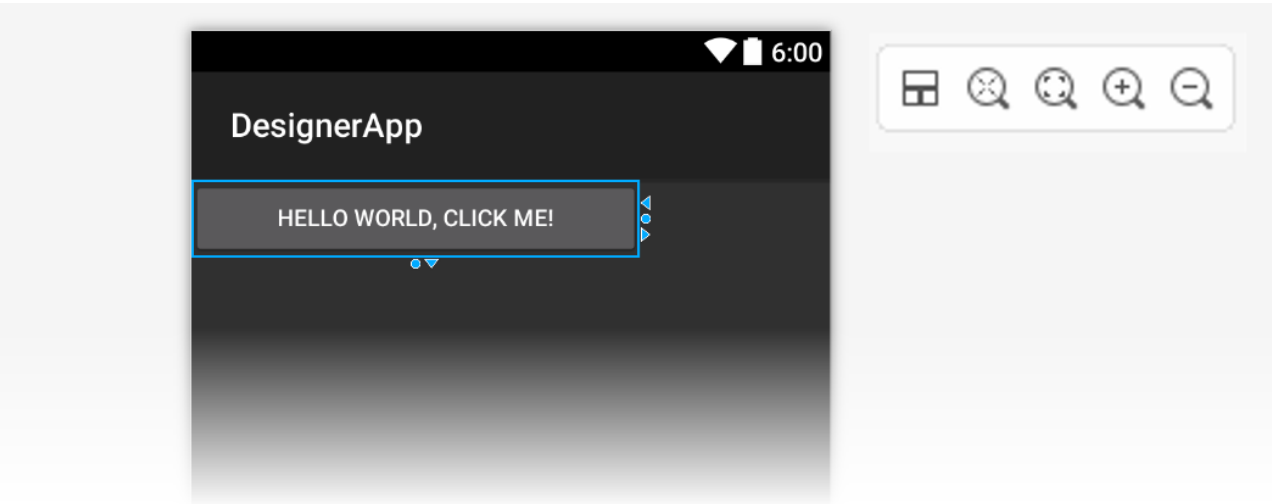
单击**包装内容**句柄收缩该维度中的小组件, 这样就不大于所需包装包含的内容。在此示例中, 按钮文本中的下一步的屏幕截图所示的水平缩小。

如果大小值设置为**内容包装**, 在设计器将显示指向中更改到大小的相反方向的三角形句柄 `match_parent` :

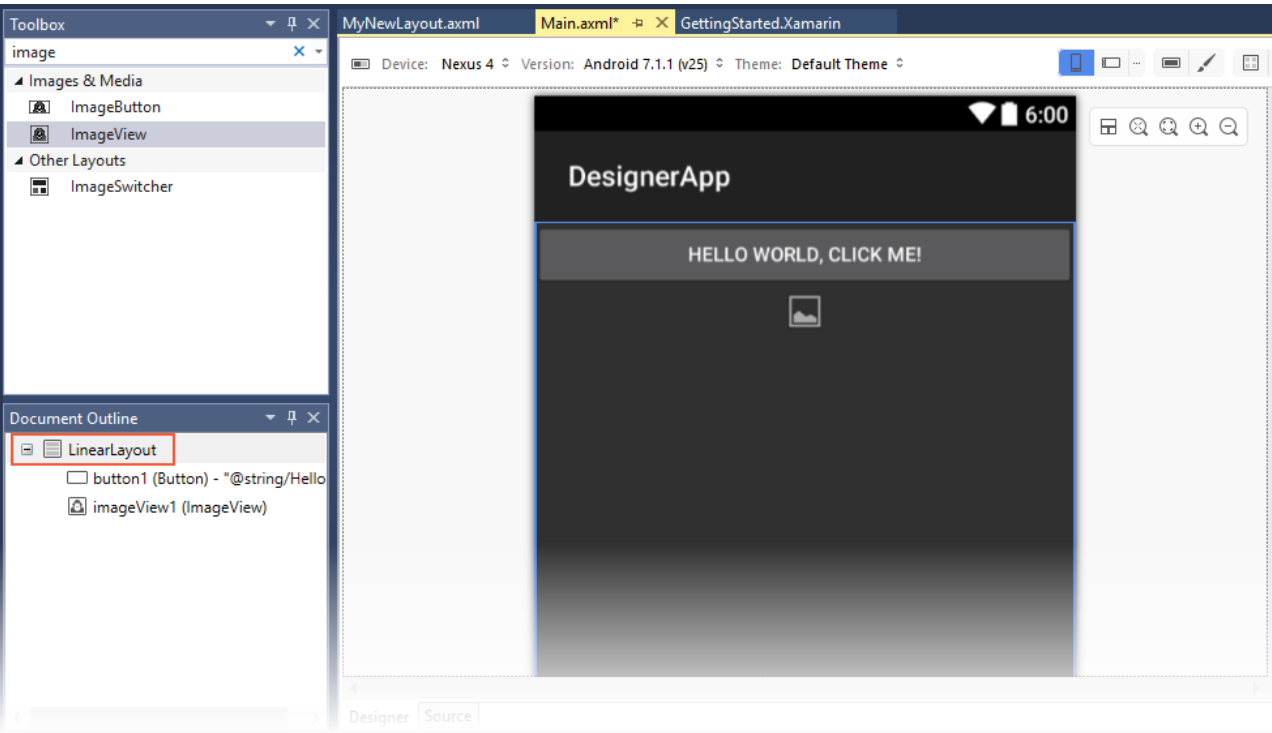


单击**匹配父句柄**将还原该维度中的大小，以便它与父级小组件相同。

此外，您可以拖动的循环重设大小句柄（如上面的屏幕截图中所示）以调整大小小组件设为任意 `dp` 值。当你执行此操作，同时**内容包装并匹配父句柄**显示为该维度：

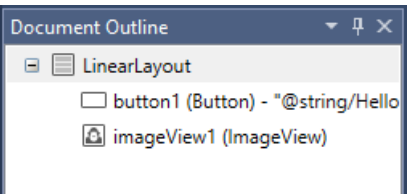


并非所有容器都允许编辑 `Size` 的小组件。例如，请注意，在下面的屏幕截图与 `LinearLayout` 选择，调整大小图柄不会显示：



文档大纲

文档大纲显示布局的小组件层次结构。在下面的示例，其中包含 `LinearLayout` 选择小组件：



轮廓的所选小组件 (在这种情况下， `LinearLayout`) 上也会突出显示设计图面。在文档大纲中的所选小组件保持在与同步设计图面。这可用于选择查看组，并不总是容易上选择设计图面。

文档大纲支持复制和粘贴，或您可以使用拖放。拖放功能支持从文档大纲到设计图面以及从设计图面到文档大

纲. 此外, 右键单击中的项**文档大纲**显示该项的上下文菜单 (右键单击该同一个小组件时, 将显示相同的上下文菜单设计图面)。

资源限定符和可视化效果选项

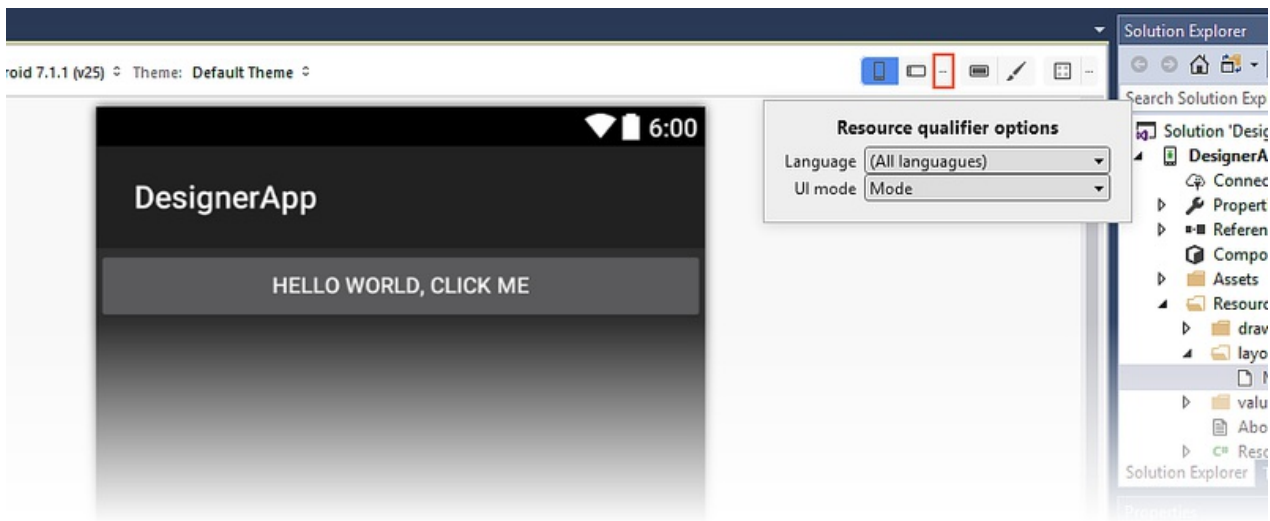
2018/10/26 • [Edit Online](#)

本主题说明如何定义仅某些限定符值匹配时，将使用的资源。一个简单的示例是一个语言限定字符串资源。与其他定义要用于其他语言的替代资源，可以为默认值，定义字符串资源。可限定所有资源类型，包括自身的布局。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

资源限定符选项

资源限定符选项可以通过单击右侧的省略号图标可访问横向模式按钮：



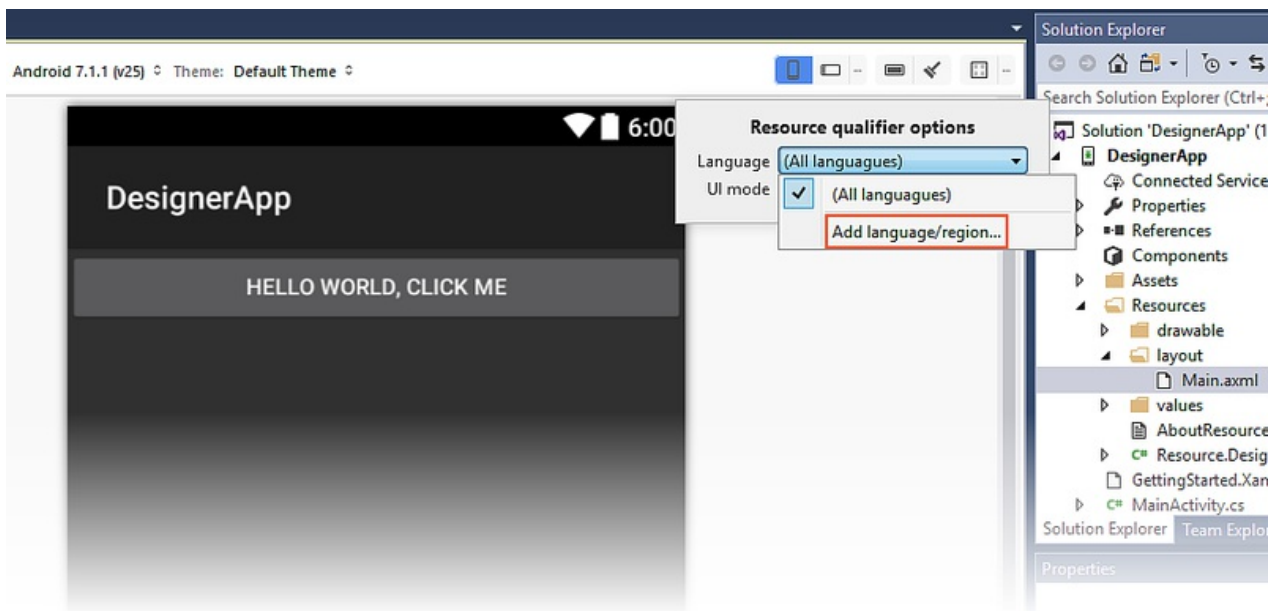
此对话框将显示下面的资源限定符的下拉式菜单：

- **语言**—显示可用的语言资源，并提供了一个选项以添加新语言/区域资源。
- **用户界面模式**—列表显示模式（如汽车底座并桌面底座）以及布局方向。

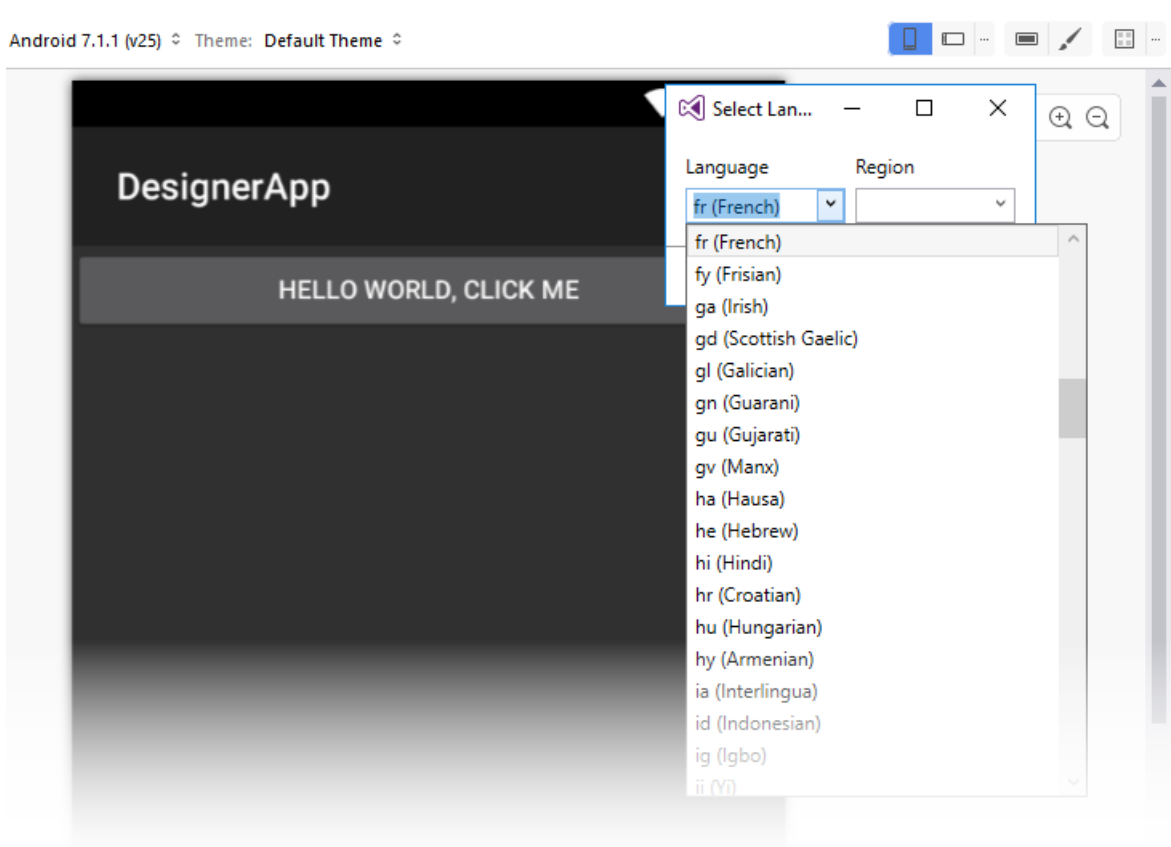
每个这些下拉列表菜单将打开新对话框可以在其中选择并配置的资源限定符（如下所述）。

语言

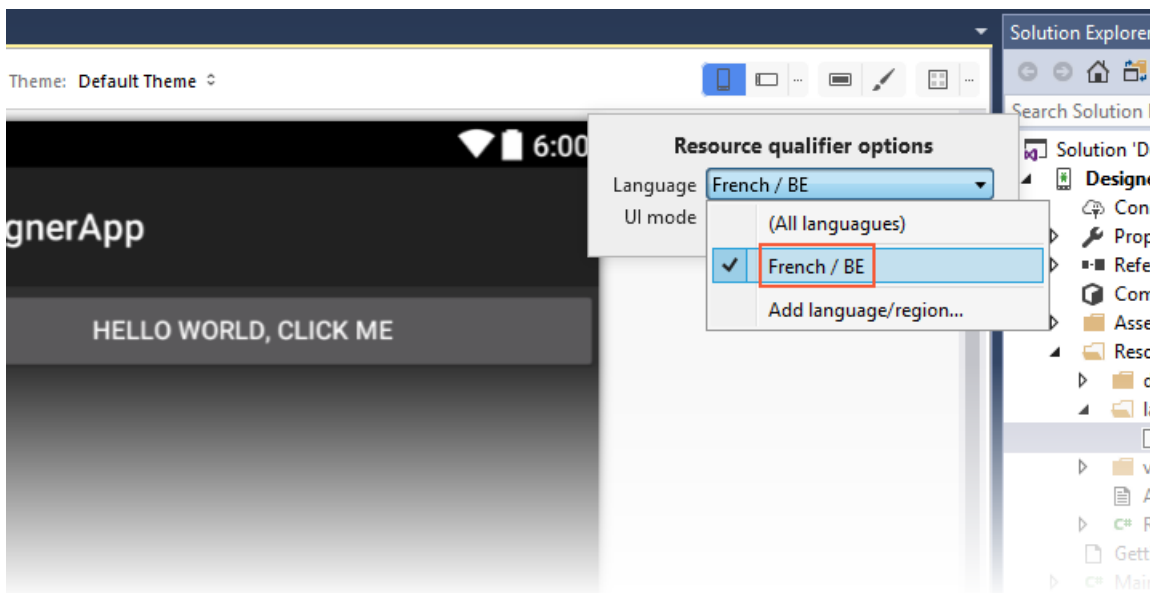
语言下拉菜单列出了只有那些具有定义的资源语言（或所有语言，这是默认设置）。但是，还有**添加语言/区域...**选项，可用于将一种新语言添加到列表：



当您单击**添加语言/区域...**，则选择语言对话框会打开，显示的可用语言和区域的下拉列表：



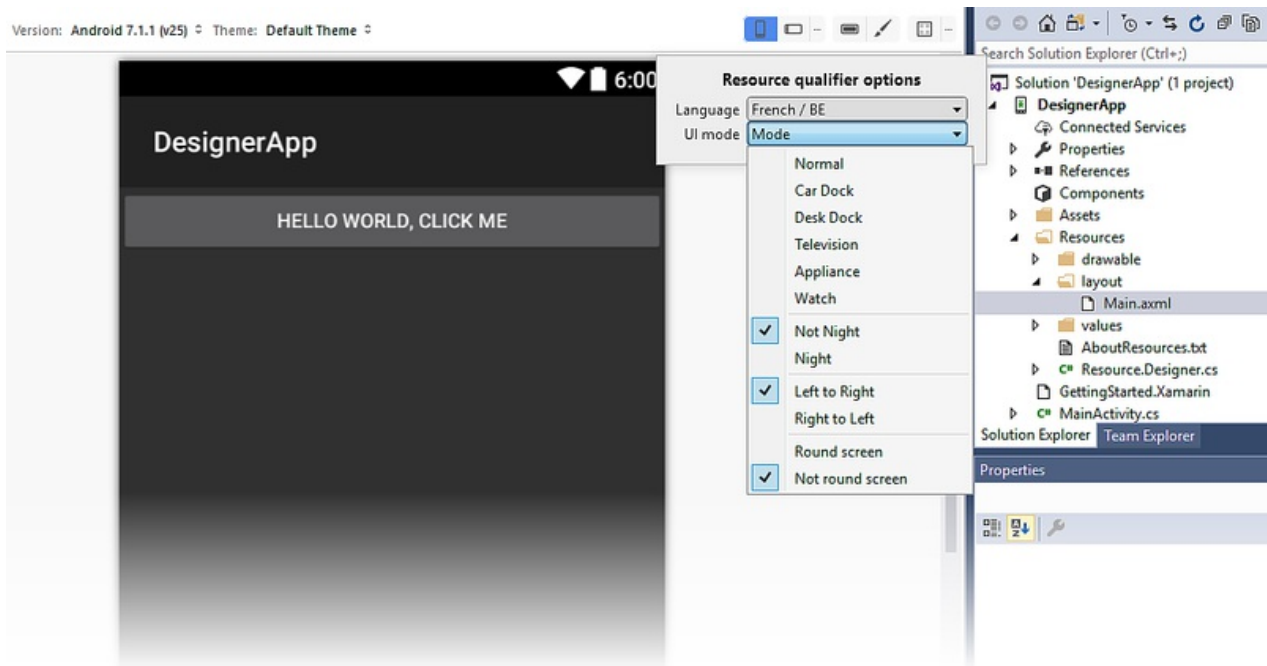
在此示例中，我们已选择**fr（法语）**的语言和**BE（比利时）**为法语的区域语言。请注意，**区域**字段是可选的因为许多语言可以指定而不考虑特定的区域。当语言再次打开的下拉菜单，则会显示新添加的语言/区域资源：



请注意，是否添加新语言，但在不创建新资源，为它添加的语言将不再显示在下次您打开该项目。

用户界面模式

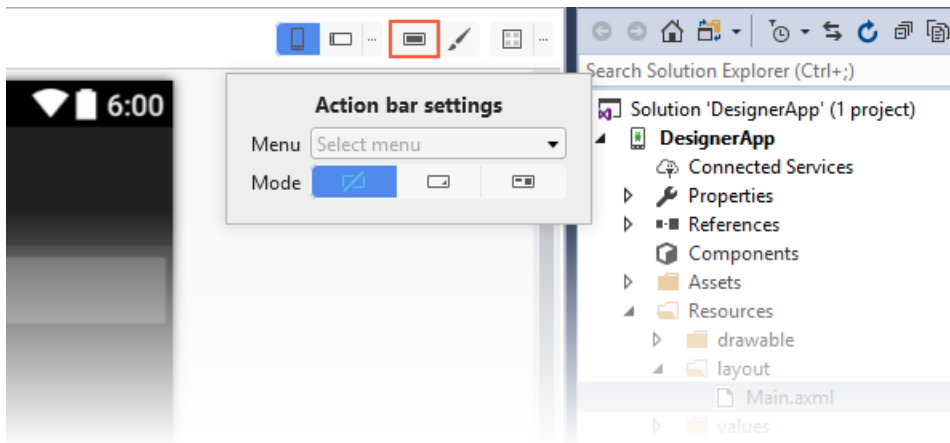
当您单击**用户界面模式**显示的下拉菜单，模式的列表，如正常，汽车底座，桌面底座，电视，装置，并监视：



此列表下方是晚上模式不晚上和晚上后，跟布局方向从左到右并**从右到左**（适用于有关的信息**从左到右并**从右到左****选项，请参见[LayoutDirection](#)）。中的最后一个项**资源限定符**选项对话框均**舍入屏幕**（适用于与 Android Wear 一起使用）或**非圆角屏幕**。Round 和非圆角屏幕有关的信息，请参阅[布局](#)。有关 Android UI 模式的详细信息，请参阅[UiModeManager](#)。

操作栏设置

操作栏设置图标位于左侧的画笔（主题编辑器）图标：

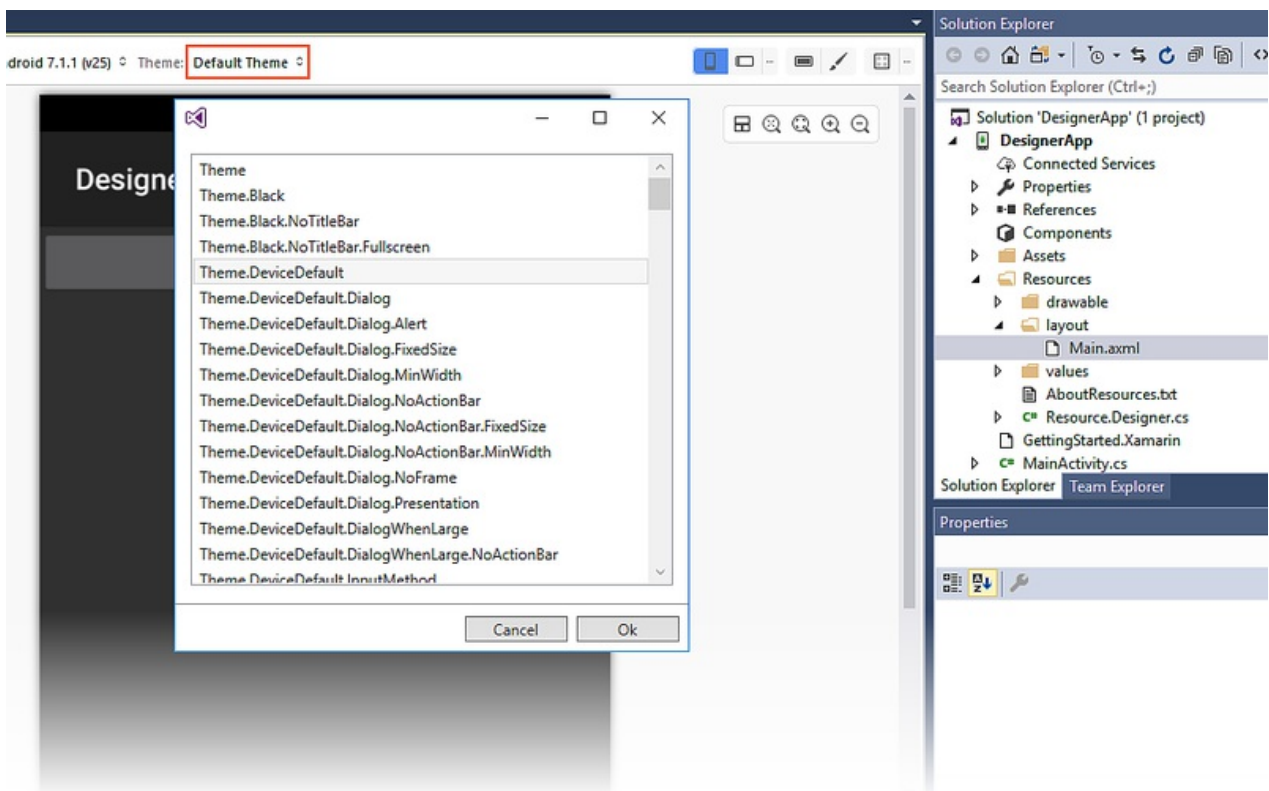


此图标将打开对话框弹出框，使您能够选择三种操作栏模式之一：

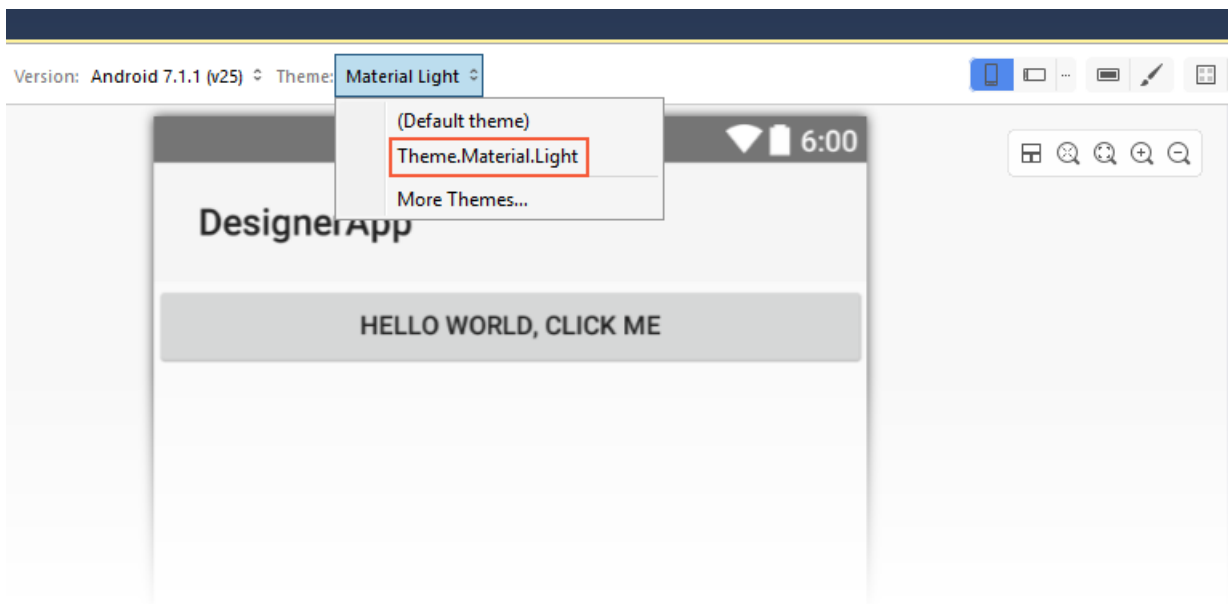
- **标准**—组成一个徽标或使用可选副标题一个图标和标题的文本。
- **列表**—列表导航模式。此模式而不是静态的标题文本显示在活动中的导航列表菜单（即，它可以向用户显示与下拉列表）。
- **选项卡**—tab 键导航模式。而不是静态的标题文本，此模式下提供了一系列活动中的导航选项卡。

主题

主题下拉列表菜单将显示所有项目中定义的主题。选择**更多主题**打开一个对话框，可以从已安装的 Android SDK 的所有主题 of 列表，如下所示：



选择一个主题后，设计图面上将更新以显示新的主题的效果。请注意此更改是永久才确定中单击按钮主题对话框。一旦选择了一个主题，它将会包含在主题下拉列表菜单操作，如图所示如下：



Android 版本

Android 版本选择器设置用于呈现设计器中的布局的 Android 版本。选择器将显示所有与该项目的目标框架版本兼容的版本：



可以在项目的设置中设置目标框架版本属性 > 应用程序 > 使用 **Android 版本** 编译。有关目标框架版本的详细信息，请参阅[了解 Android API 级别](#)。

通过项目的目标框架版本确定的可用工具箱中的小部件集。这也适用于中的可用属性属性窗口。可用列表的小部件是不中选择的值由决定版本工具栏的选择器。例如，如果您设置项目的目标版本为 Android 4.4，仍可以选择 Android 6.0 中工具栏版本选择器来查看该项目在 Android 6.0 中的显示效果，但无法将特定于 Android 6.0 的小部件添加- 仍将限制为可在 Android 4.4 的小部件。

有关资源类型的详细信息，请参阅[Android 资源](#)。

备选布局视图

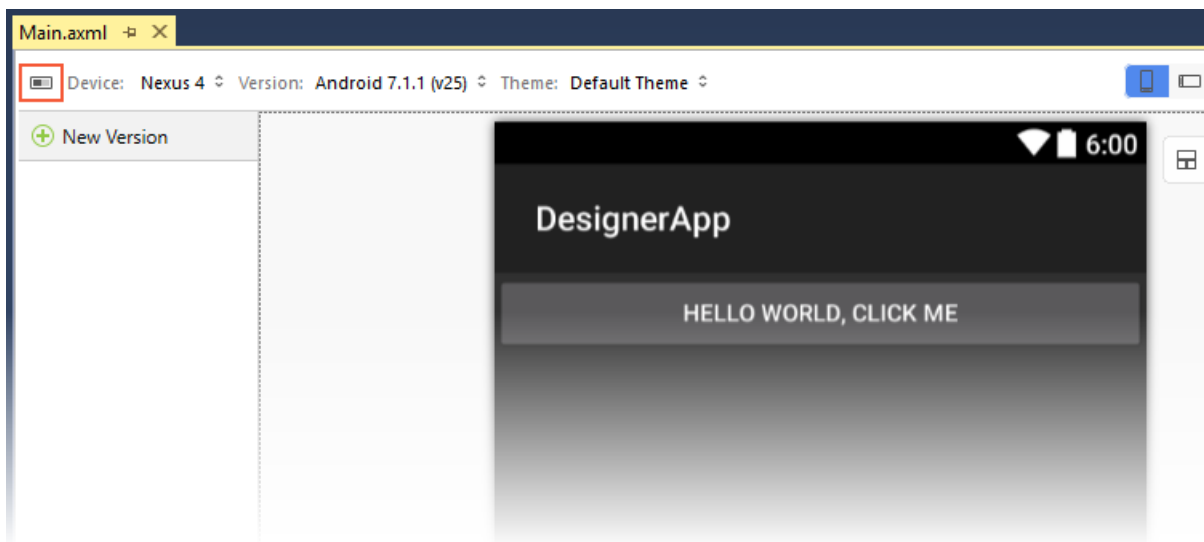
2018/10/26 • [Edit Online](#)

本主题说明如何通过使用的资源限定符可以版本布局。例如，创建在设备处于横向模式时，才使用的布局的版本和仅适用于纵向模式下的布局版本。

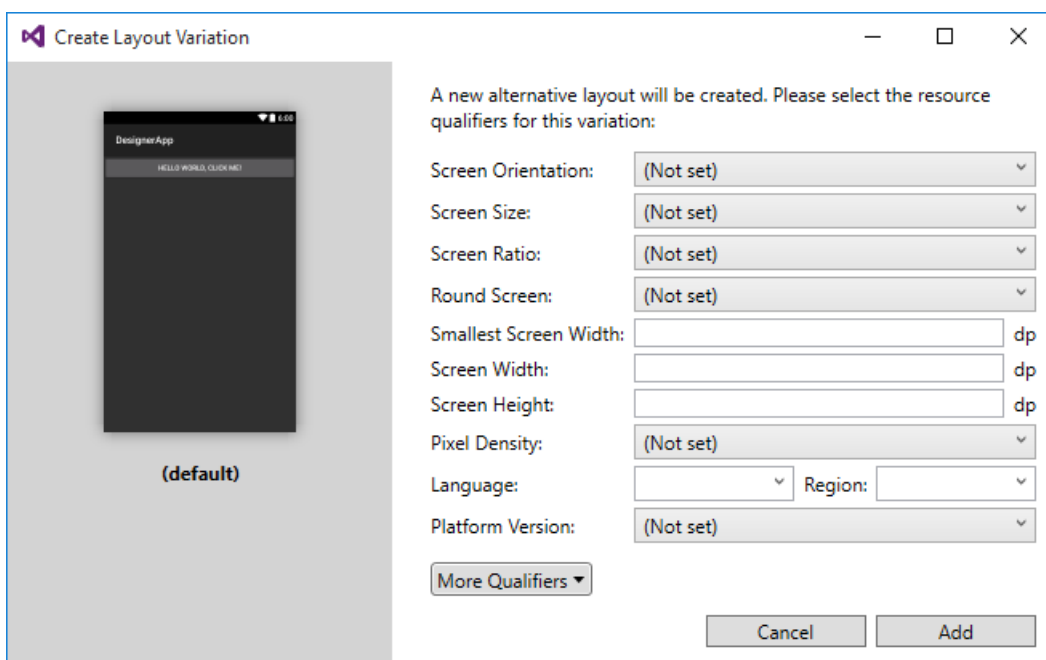
- [Visual Studio](#)
- [Visual Studio for Mac](#)

创建备用布局

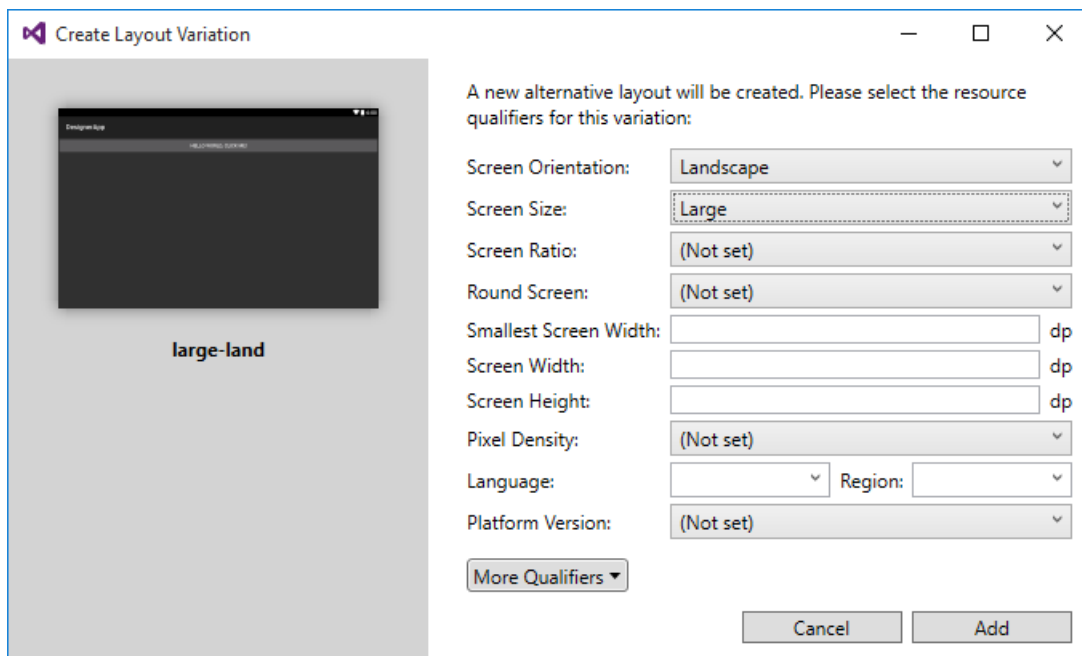
当您单击备选布局视图图标 (左侧设备)，预览窗格将打开，并列项目中的可用的备用布局。如果没有任何其他布局默认显示视图：



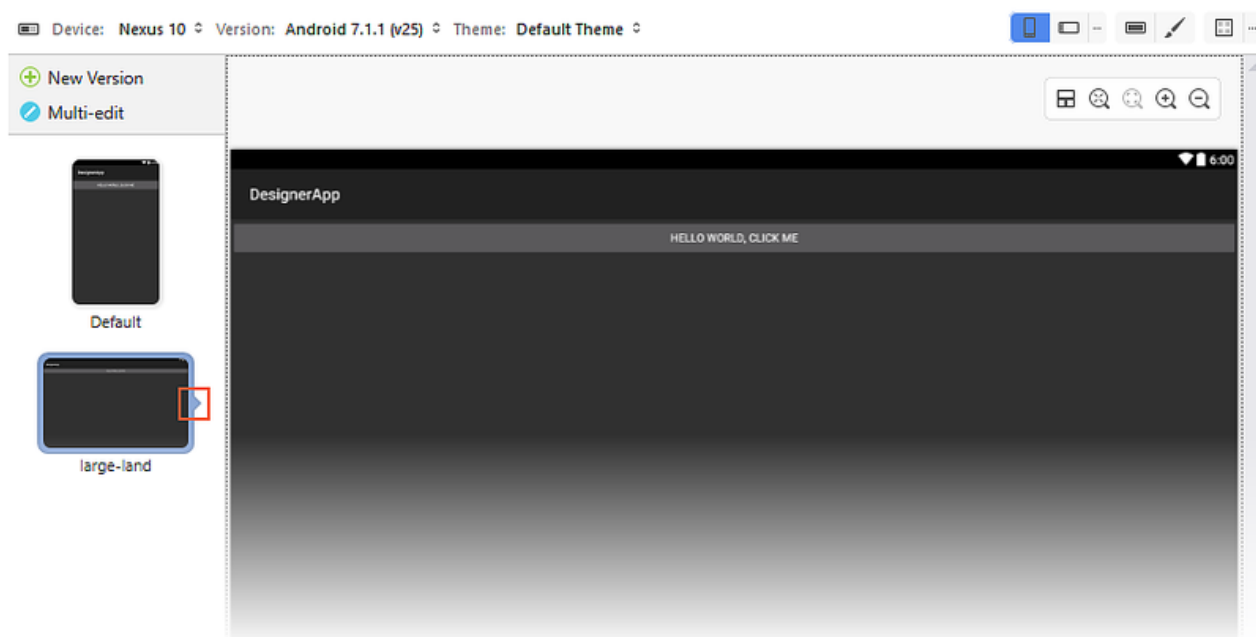
当单击绿色加号旁边新版，则创建布局变体对话框随即打开，以便你可以选择用于此布局变体的资源限定符：



在下面的示例的资源限定符屏幕方向 设置为横向，并屏幕大小更改为大. 这将创建名为的新布局版本大型 **land**:



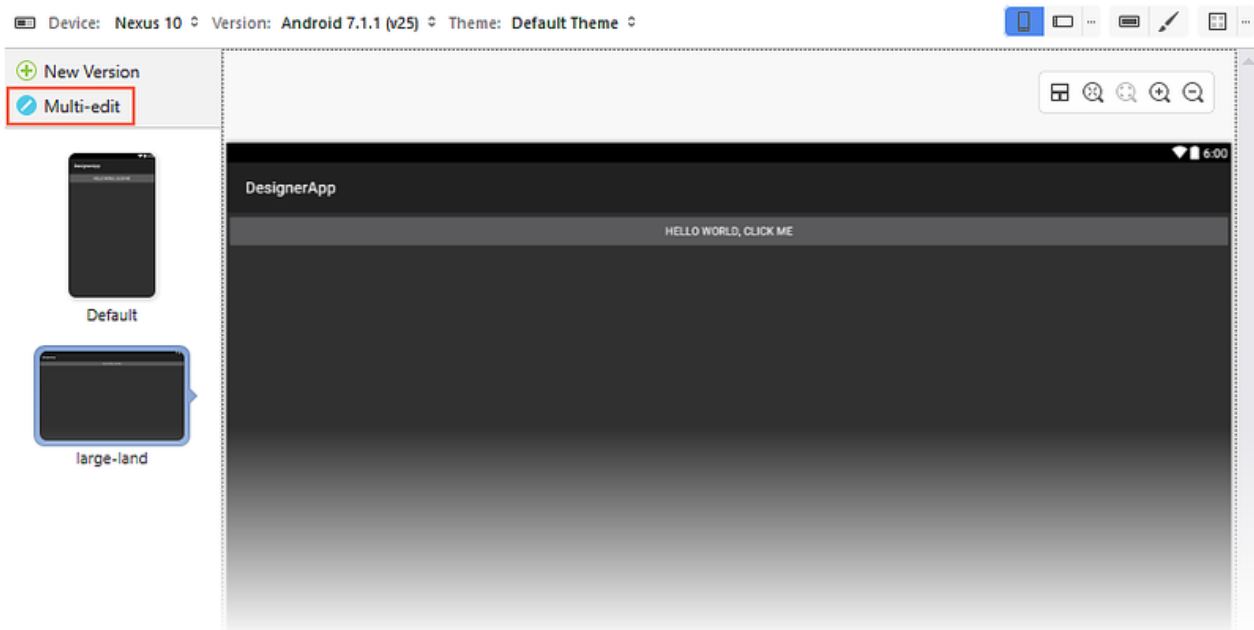
请注意，在左侧预览窗格中显示的资源限定符选项的效果。单击**添加**创建备选布局，并切换到该布局的设计器。**替代项布局**视图预览窗格指示哪种布局已加载到小型右指针通过设计器，如下屏幕截图中所示：



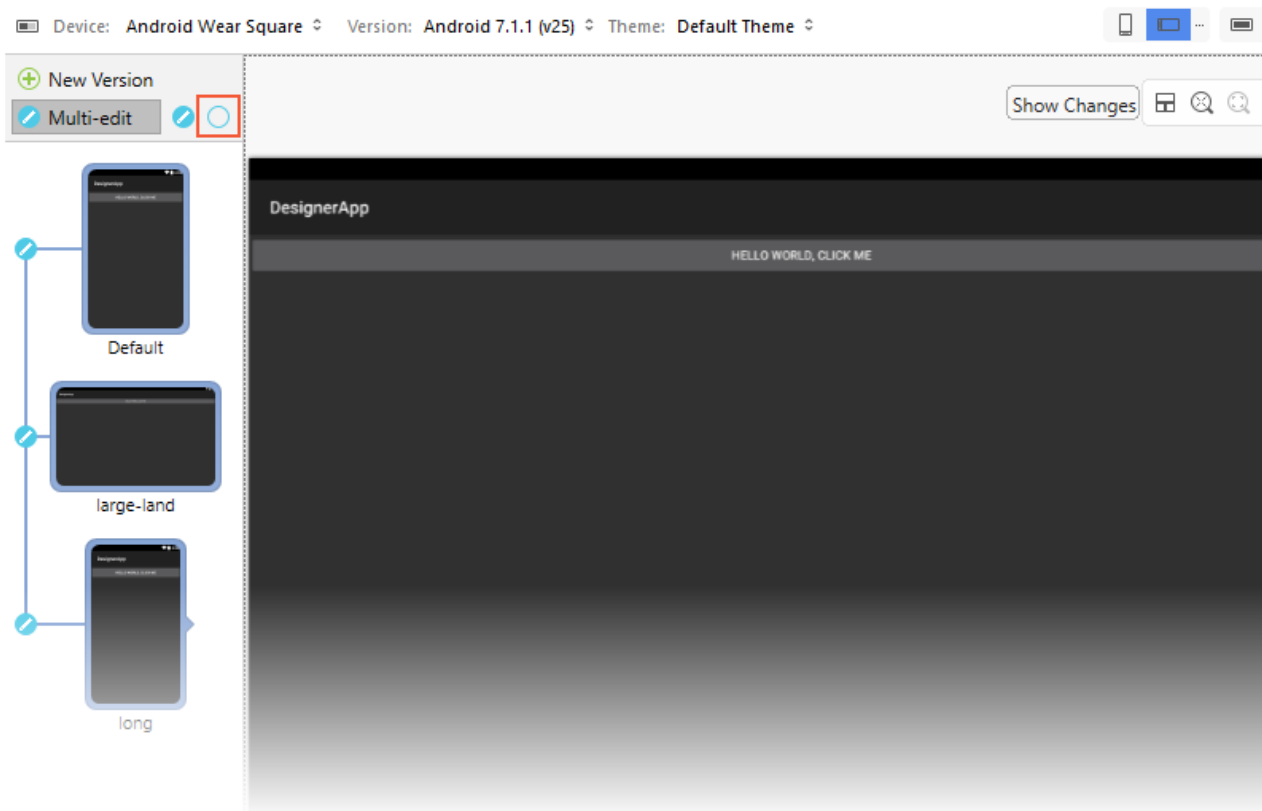
编辑其他布局

通常，在创建其他布局时，很才能使适用于布局的所有分支版本的单个更改。例如，你可能想要将按钮文本更改为在所有布局中的黄色。如果有大量的布局，并且您需要将传播到所有版本的单个更改，维护可快速变得繁琐且容易出错。

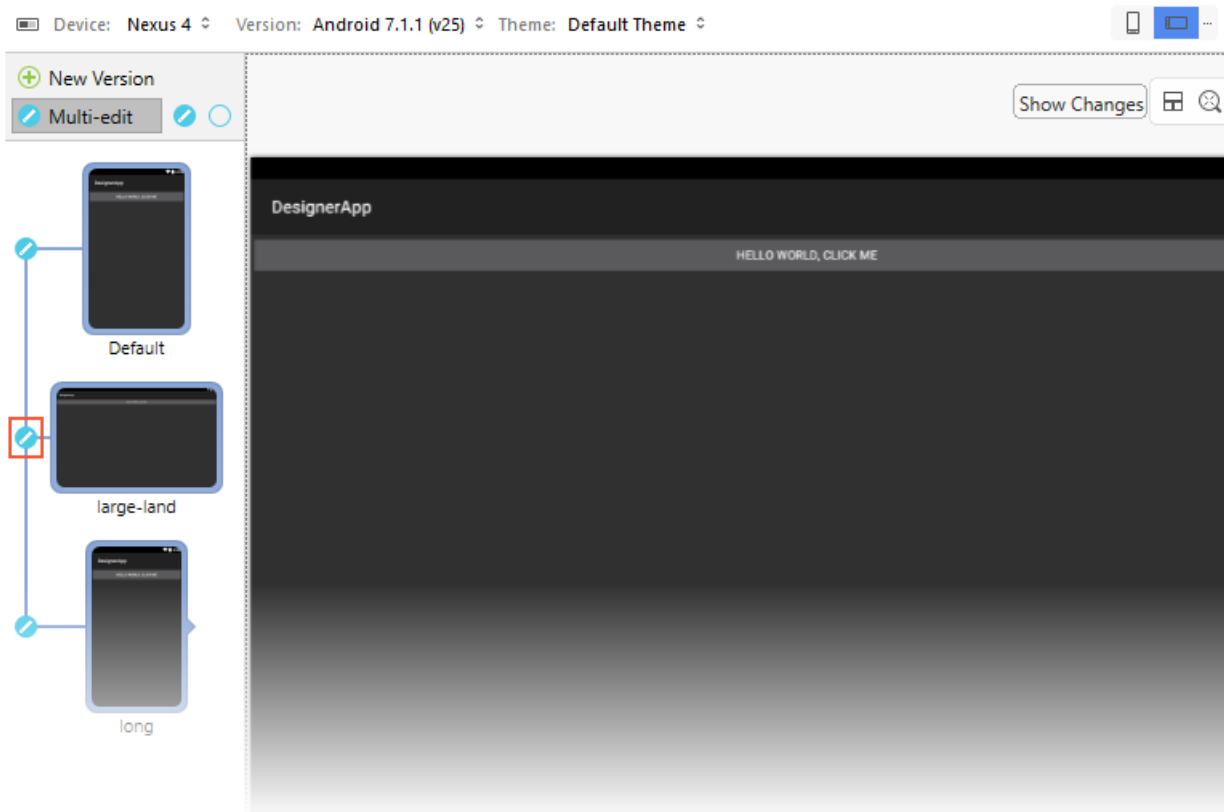
若要简化维护的多个布局版本，在设计器提供了多编辑跨一个或多个布局传播所做的更改的模式。如果存在，多个布局**多编辑**显示图标：



当您单击多编辑图标，显示线指示（如下所示）链接布局；即，当更改一个布局，则所做的更改传播到任何链接的布局。您可以通过单击下面的屏幕截图所示的带圆圈的图标中取消链接所有布局：



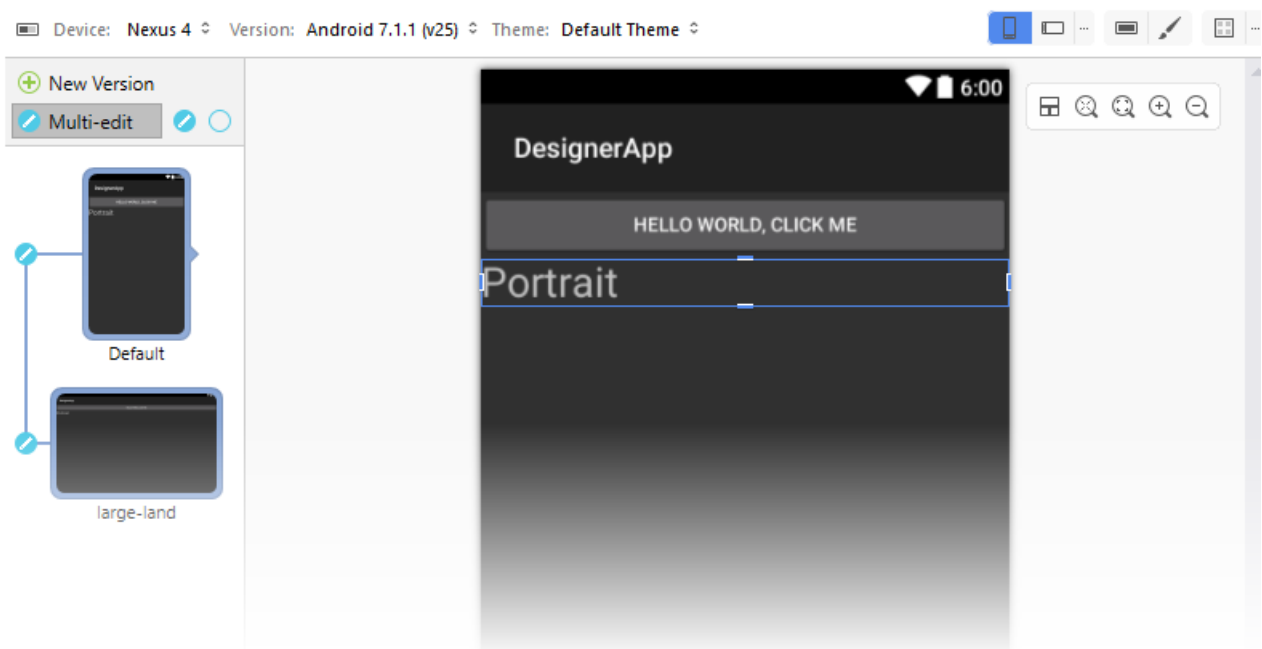
如果您有两个以上的布局，可以有选择地切换到左侧的每个布局预览版来确定哪些布局链接在一起的编辑按钮。例如，如果你想要进行传播的单个更改为第一个和最后的三种布局，您将首先取消链接的中间布局如下所示：



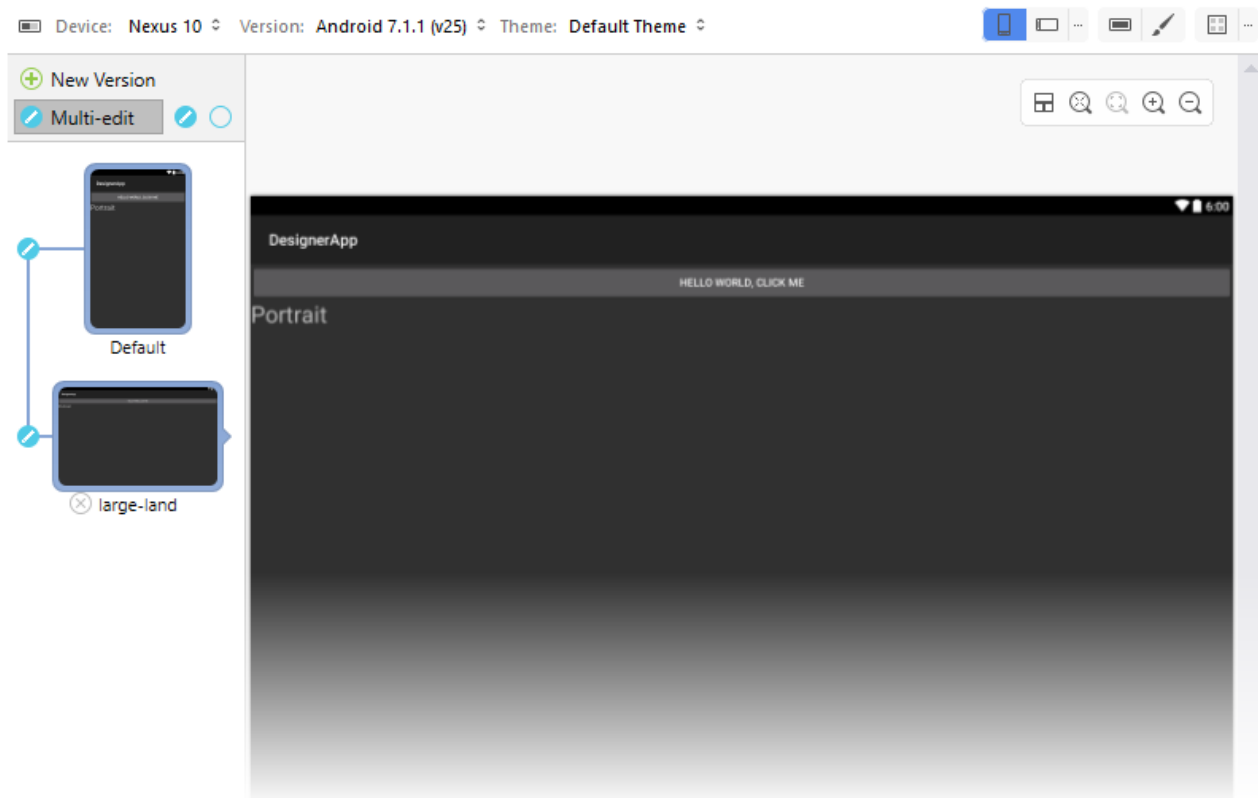
在此示例中，更改为默认或长布局将传播到其他布局而不适用于大型 **land** 布局。

多项编辑示例

一般情况下，当更改一个布局，该相同的更改传播到所有其他链接的布局。例如，添加一个新 `TextView` 小组件设默认布局，并更改它的文本字符串到 `Portrait` 将导致相同的更改不会对所有链接的布局。下面是其外观默认布局：



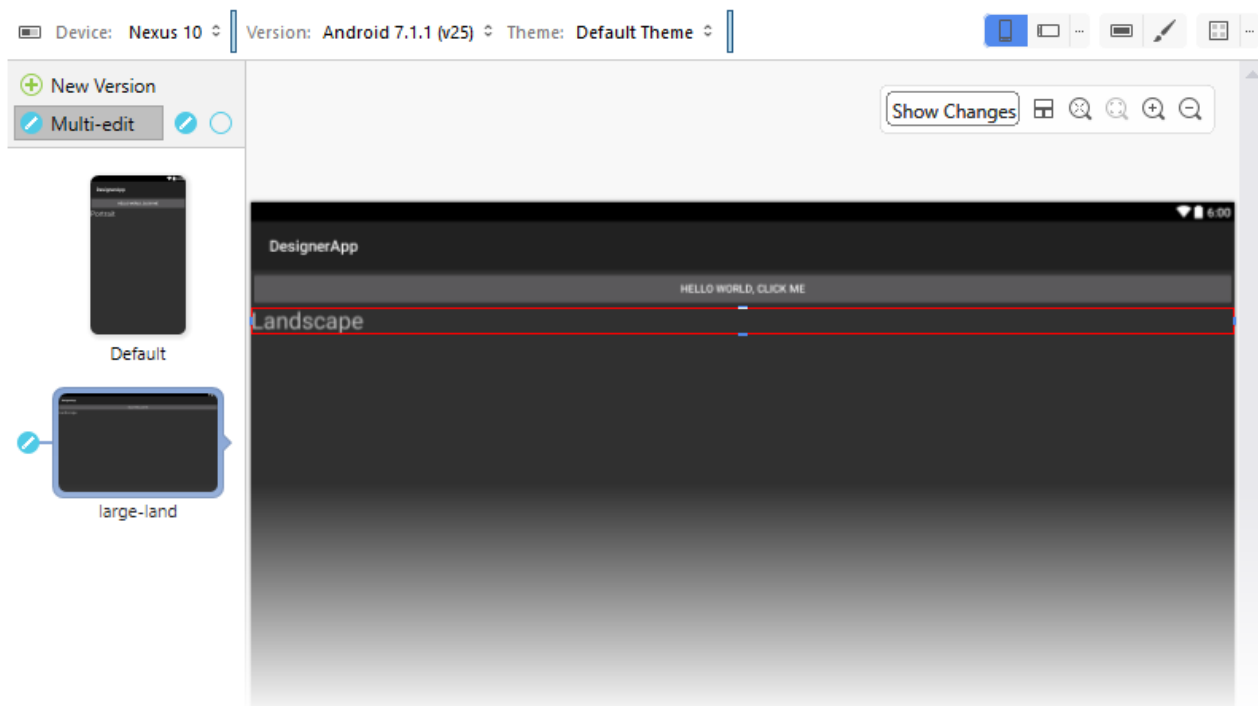
`TextView` 还将添加到大 **land** 布局查看，因为它链接到默认布局：



但如果你想要进行更改的本地只有一种布局（也就是说，您不希望更改传播到任何其他布局）？若要执行此操作，必须取消链接你想要更改在修改之前，如下所述的布局。

进行本地更改

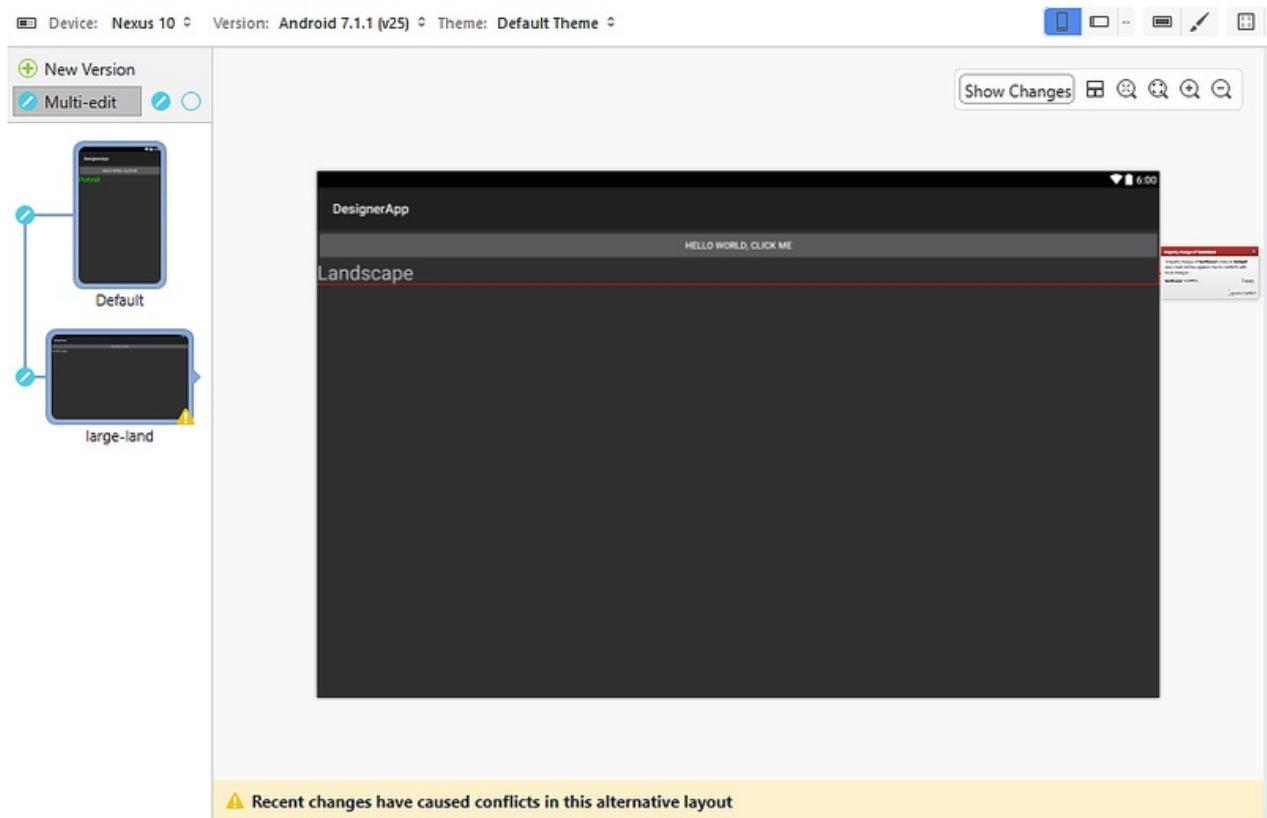
假设我们希望能够在添加这两种布局 `TextView`，但我们还想要更改中的文本字符串大型 `land` 到布局 `Landscape` 而非 `Portrait`。如果我们进行到此更改大型 `land` 虽然链接这两种布局，更改将传播回默认布局。因此，我们必须首先取消链接两个布局之前做出更改。当我们修改中的文本大型 `land` 到 `Landscape`，在设计器将标记的红框指示的更改是本地的与此更改大型 `land` 布局是不传播回默认布局：



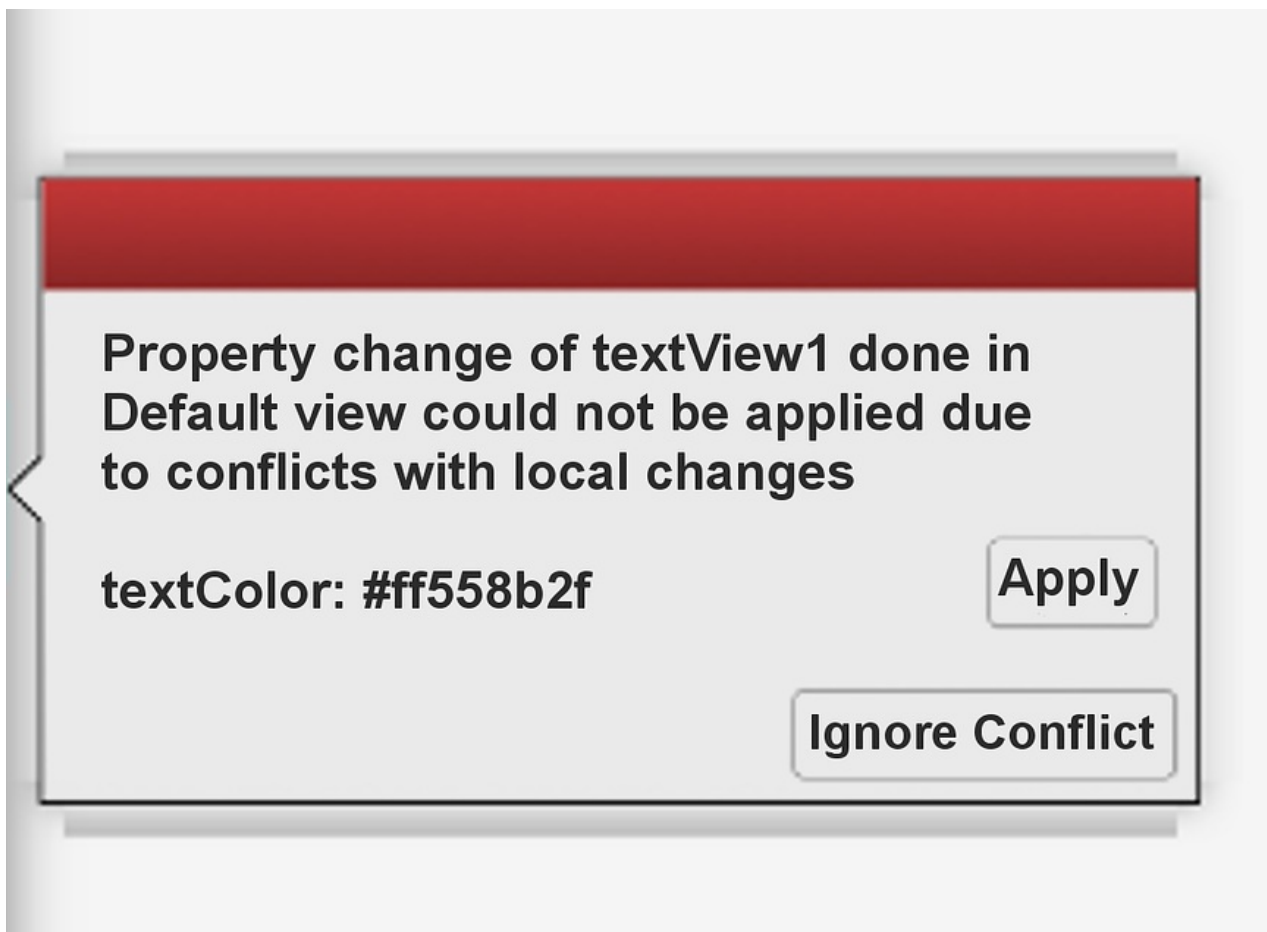
当您单击默认布局进行查看时，`TextView` 文本字符串仍设置为 `Portrait`。

处理冲突

如果您决定更改中的文本的颜色默认为绿色的布局，您将看到链接布局上出现一个警告图标。单击该布局将打开以显示冲突的布局。红框突出显示导致冲突的小组件，并显示以下消息：**最近的更改冲突导致此备选布局中。**



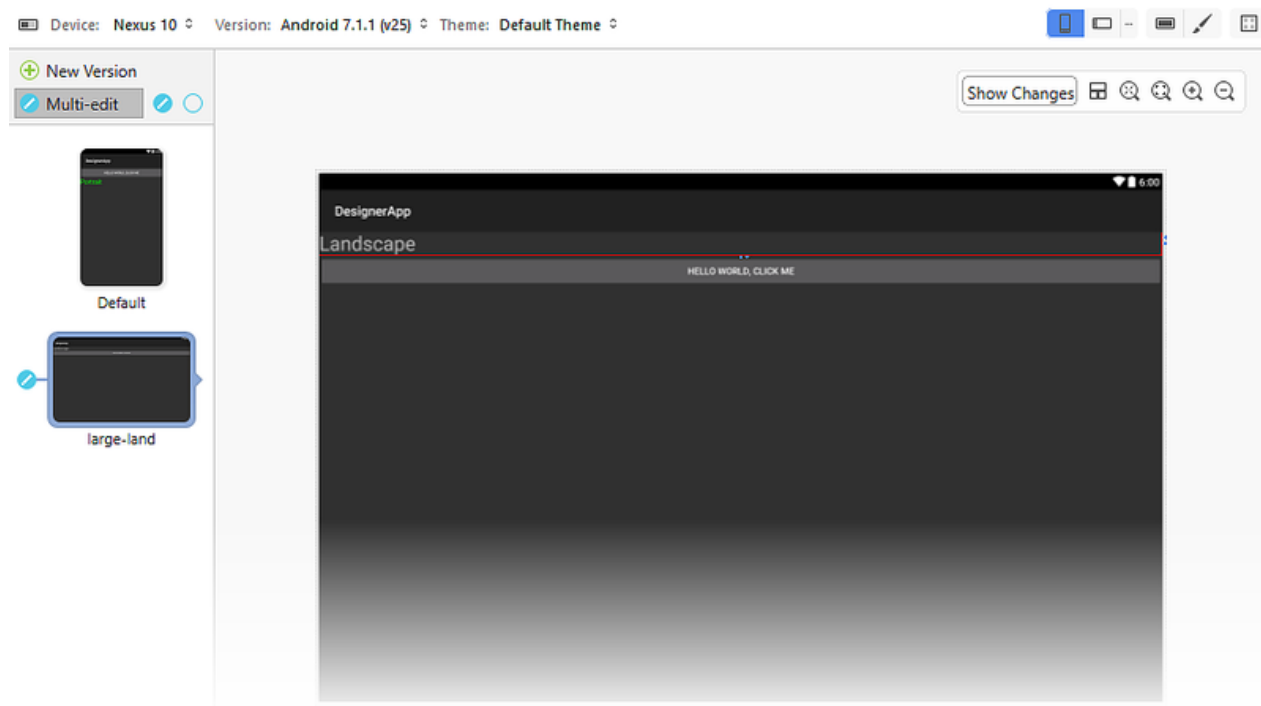
一个冲突框显示在小组件的右侧，以解释该冲突：



冲突框中显示的已更改的属性列表并列出了它们的值。单击**忽略冲突**属性更改仅适用于此小组件。单击**Apply**属性更改应用于此小组件以及关于中链接的与小组件默认布局。如果应用了所有属性更改，将自动丢弃该冲突。

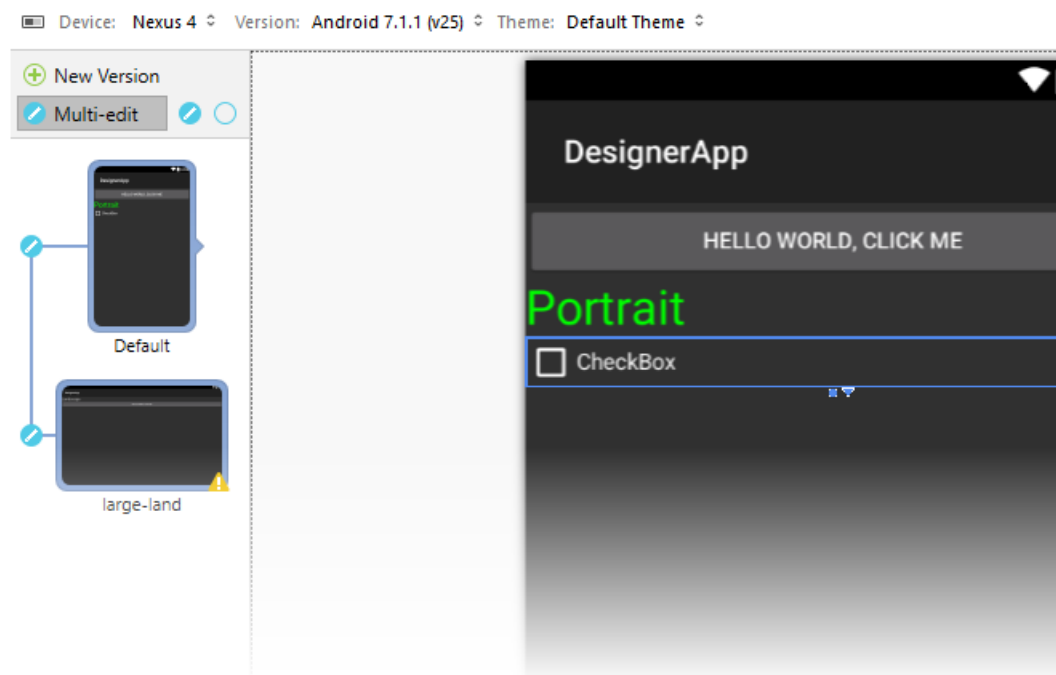
查看组冲突

属性更改不会冲突的唯一来源。插入或删除小组件时，可以检测到冲突。例如，当大型 **land** 布局从已取消链接默认布局，并 `TextView` 中大型 **land** 布局拖放上面 `Button`，设计器将标记移动小组件中，以指示冲突：

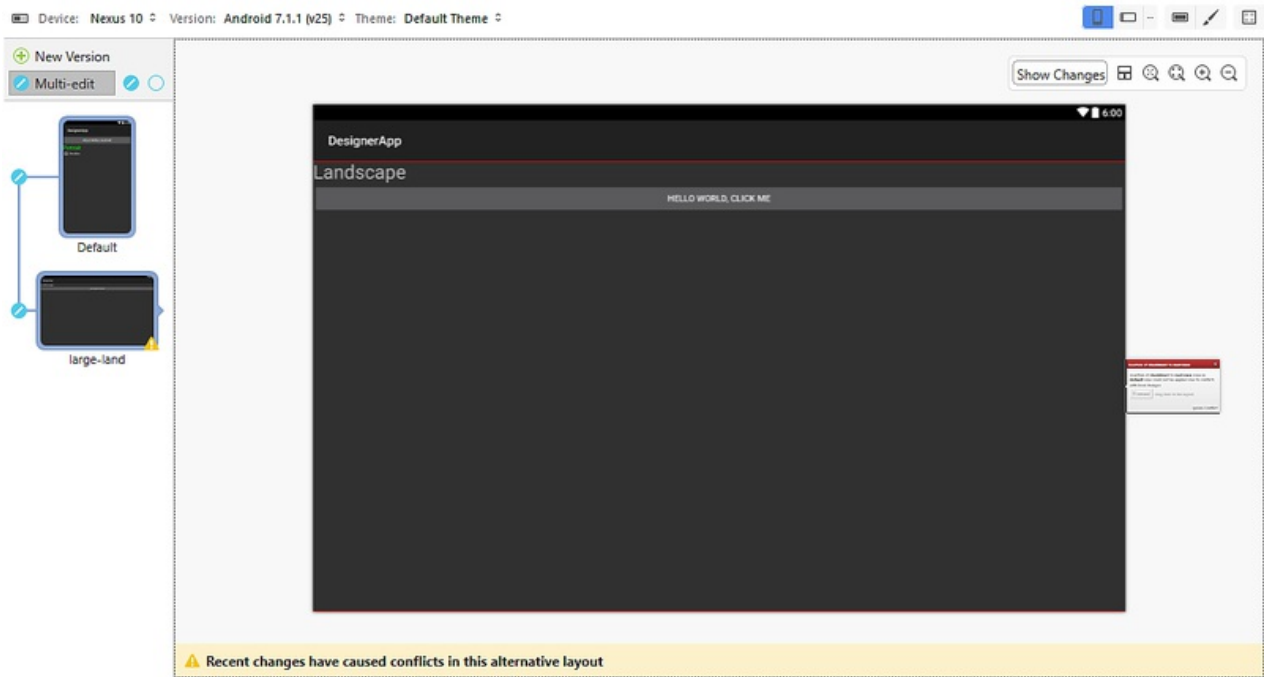


但是，没有任何标记上 `Button`。尽管的位置 `Button` 已更改 `Button` 显示了特定于任何应用的更改大型 **land** 配置。

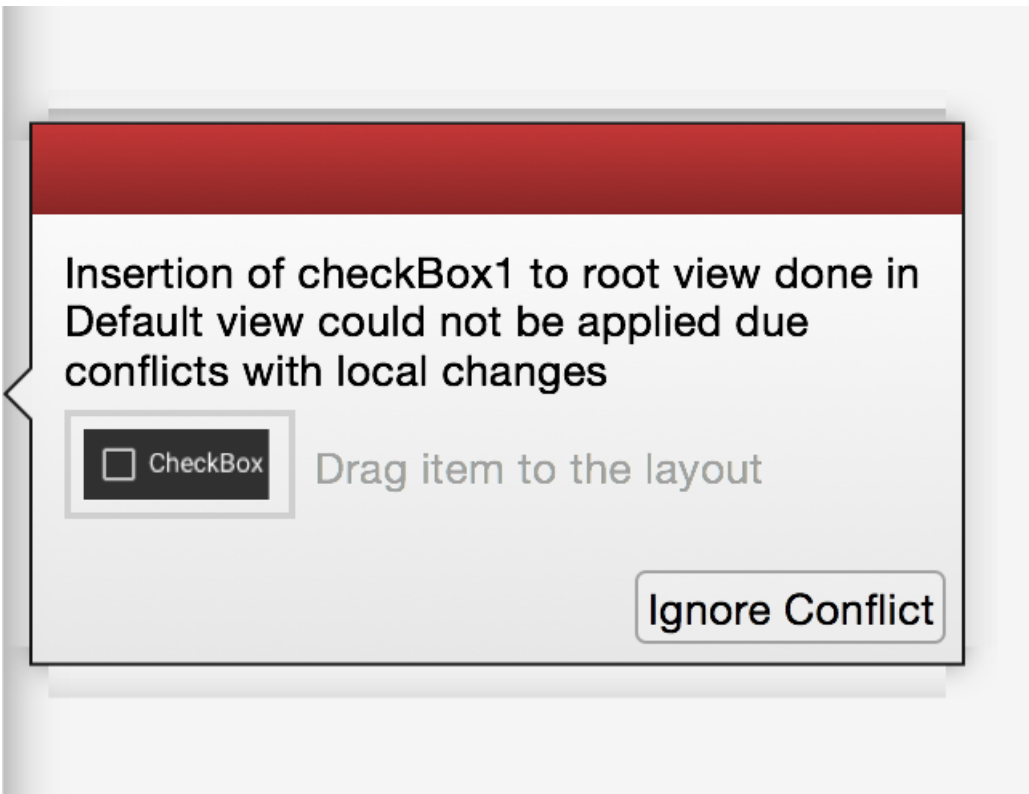
如果 `CheckBox` 添加到默认布局，将生成另一个冲突，并且通过显示一个警告图标大型 **land** 布局：



单击大型 **land** 布局会显示冲突。显示以下消息：最近的更改冲突导致此备选布局中：

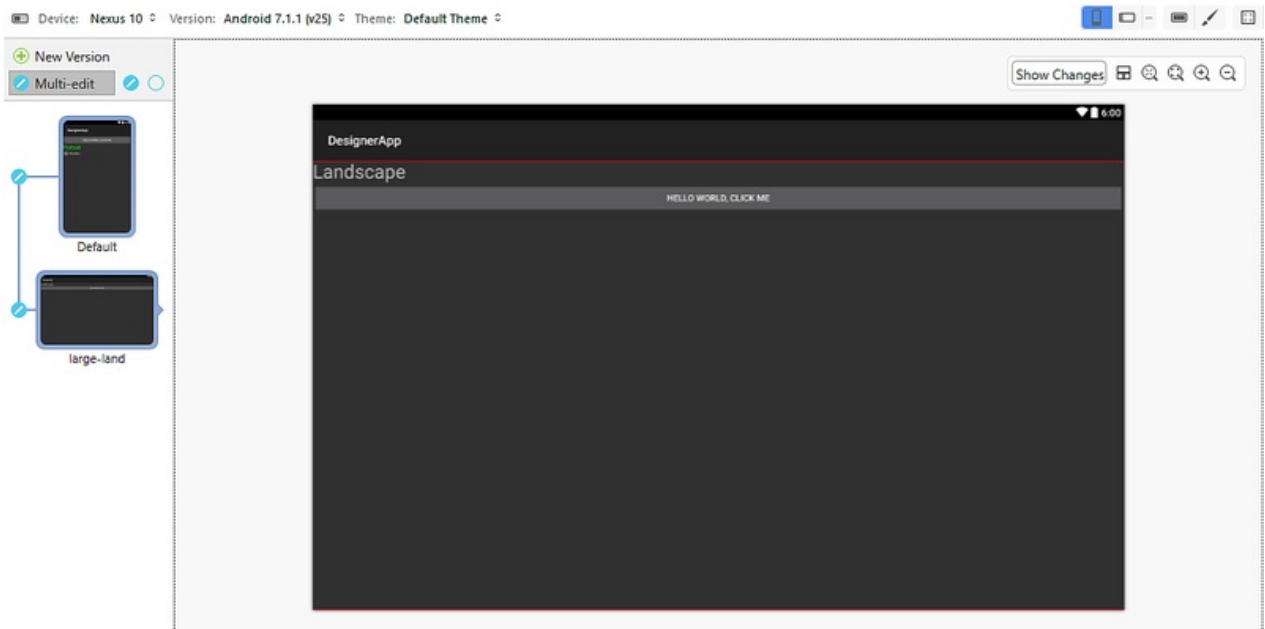


此外，冲突框中显示以下消息：



添加 `CheckBox` 导致冲突，因为大型 **land** 布局中发生更改 `LinearLayout` 包含它。但是，在这种情况下冲突框中显示刚刚插入的小组件默认布局（`CheckBox`）。

如果单击**忽略冲突**，在设计器将解决冲突，允许的框中显示冲突拖放到布局小组件缺少小组件（在这种情况下，大型 **land** 布局）：



上一示例中所示 `Button`，则 `CheckBox` 不具有红色更改标记，因为只有 `LinearLayout` 已应用中的更改大型 **land** 布局。

冲突暂留

冲突将保留在布局文件中为 XML 注释，如下所示：

```
<!-- Widget Inserted Conflict | id:__root__ | @+id/checkBox1 -->
```

因此，当关闭并重新打开项目，所有冲突仍在那里—甚至是那些已忽略。

Xamarin.Android 设计器材料设计功能

2018/10/26 • [Edit Online](#)

本主题介绍了设计器功能，可简化开发人员创建材料设计符合的布局。本部分介绍，并说明如何使用材料网格、材料颜色调色板、版式小数位数和主题编辑器。

Evolve 2016: 每个人都可以创建美观的应用与材料设计

概述

Xamarin.Android 设计器包括功能，使你更轻松创建材料设计符合布局。如果您不熟悉 Material Design，请参阅[材料设计简介](#)。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

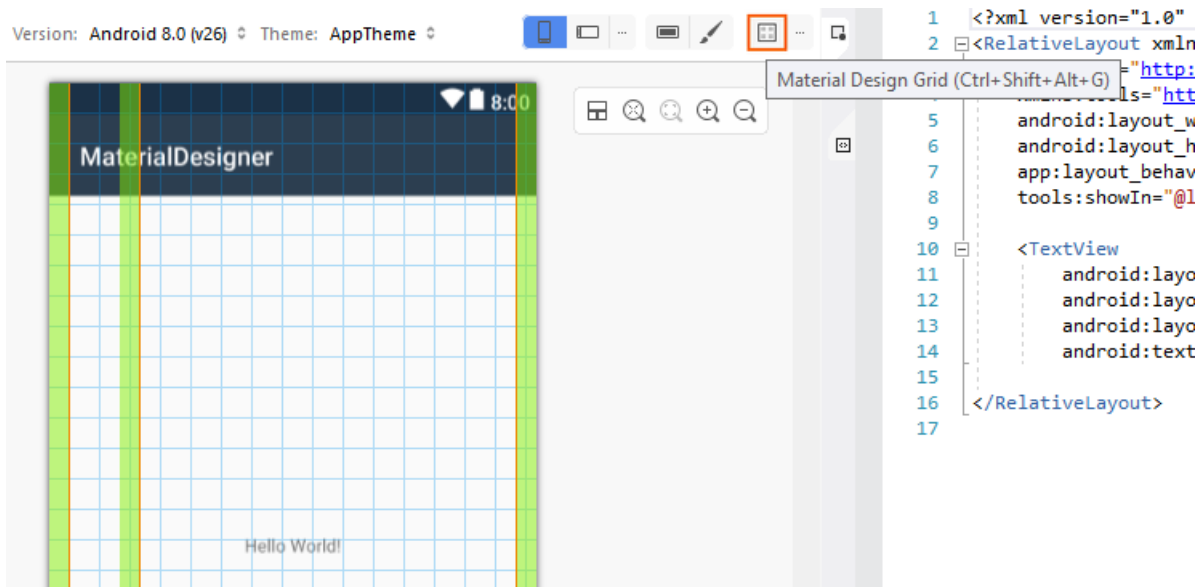
在本指南中，我们将来看一下以下的设计器功能：

- **材料网格**—显示网格、间距和线条来帮助你将根据 Material Design 准则、布局小部件放在设计图面上的覆盖。
- **主题编辑器**—小颜色资源编辑器，您可以设置主题的子集的颜色信息。例如，您可以预览，并修改材料的颜色，如 `colorPrimary`，`colorPrimaryDark`，和 `colorAccent`。

我们将看上述每项功能，并举例说明如何使用它们。

材质设计网格

材料设计网格菜单会显示在设计器的顶部工具栏中：

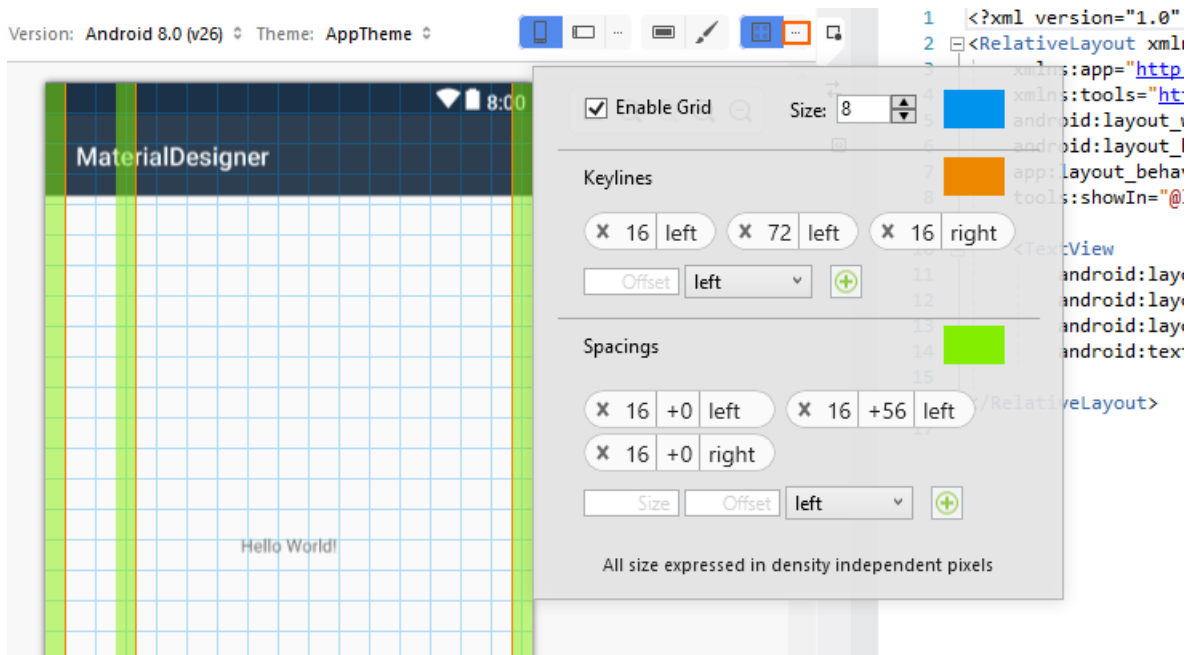


单击材料设计网格图标时，在设计器包括以下元素在设计图面上显示一个覆盖区：

- 线条（橙色行）
- 间距（绿色区域）

- 网格（蓝色线）

可以在上面的屏幕截图中看到这些元素。其中每个覆盖项目进行配置。当您单击材料设计网格菜单旁边的省略号时，将打开对话框弹出框，您可以禁用/启用网格、配置的线条，位置和设置间距。请注意，所有值都以都表示 `dp`（密度无关像素为单位）：

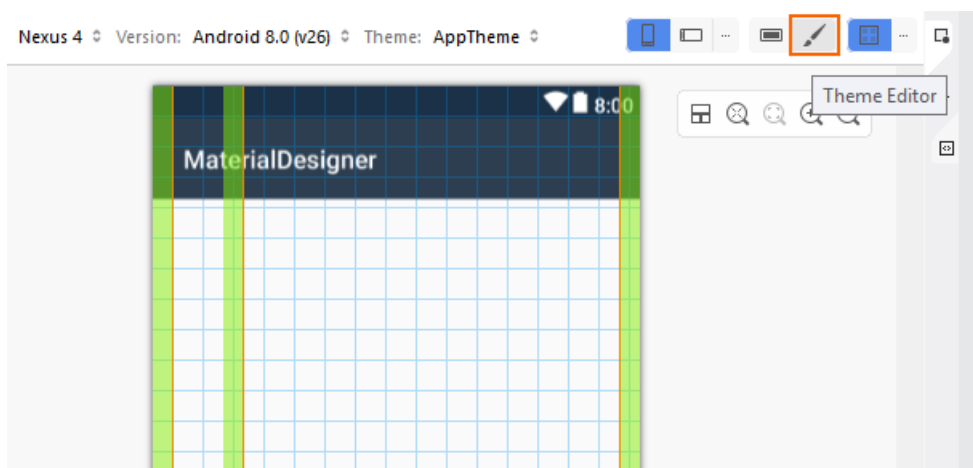


若要添加新的准线，请输入中的新偏移量的值偏移量框中，选择一个位置（左，顶部，右，或底部）并单击 + 图标以添加新的准线。同样，若要添加新的间距，输入的大小和偏移量（以 dp）到大小并偏移量框，分别。选择一个位置（左，顶部，右，或者底部）单击 + 图标以添加新的间距。

更改这些配置值时，它们保存在布局 XML 文件中并重复使用时重新打开该布局。

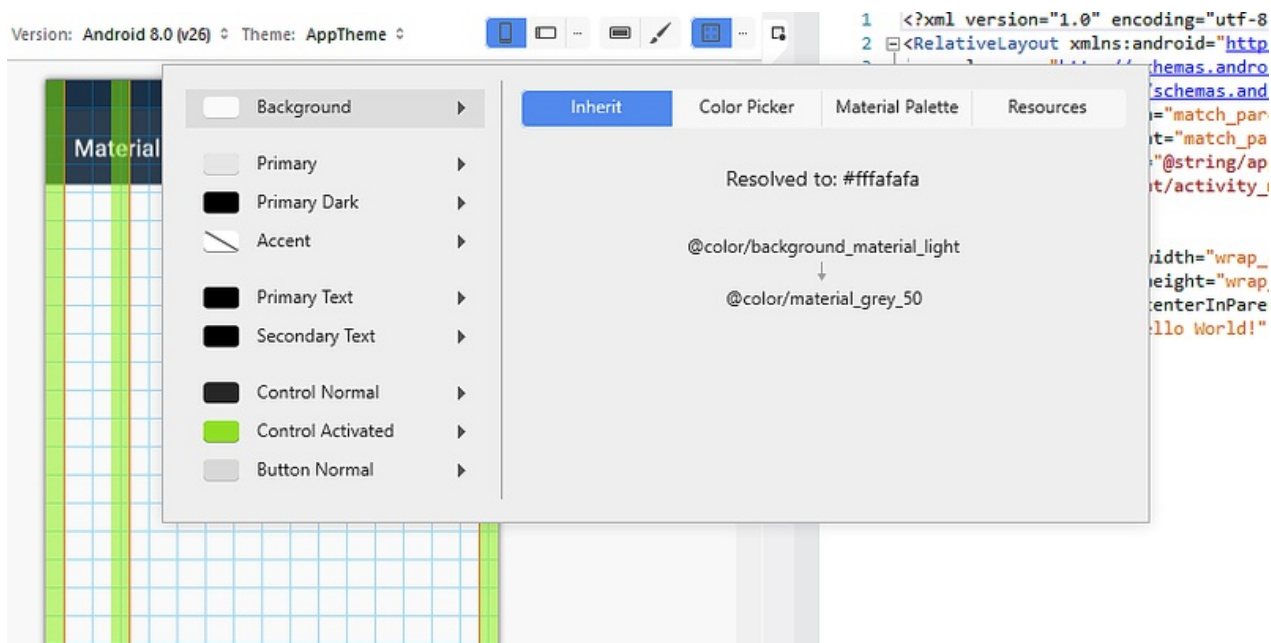
主题编辑器

Theme Editor，可以自定义主题属性的子集的颜色信息。若要打开**Theme Editor**，单击工具栏上的画笔图标：



尽管**Theme Editor**只可以使用如下所述的功能的子集如果目标 API 级别低于 API 21 版 (Android 5.0 是可从工具栏访问的所有目标 Android 版本和 API 级别棒棒糖形)。

左侧的面板**Theme Editor**显示构成了当前所选主题颜色的列表（在此示例中，我们将使用 `Default Theme`）：



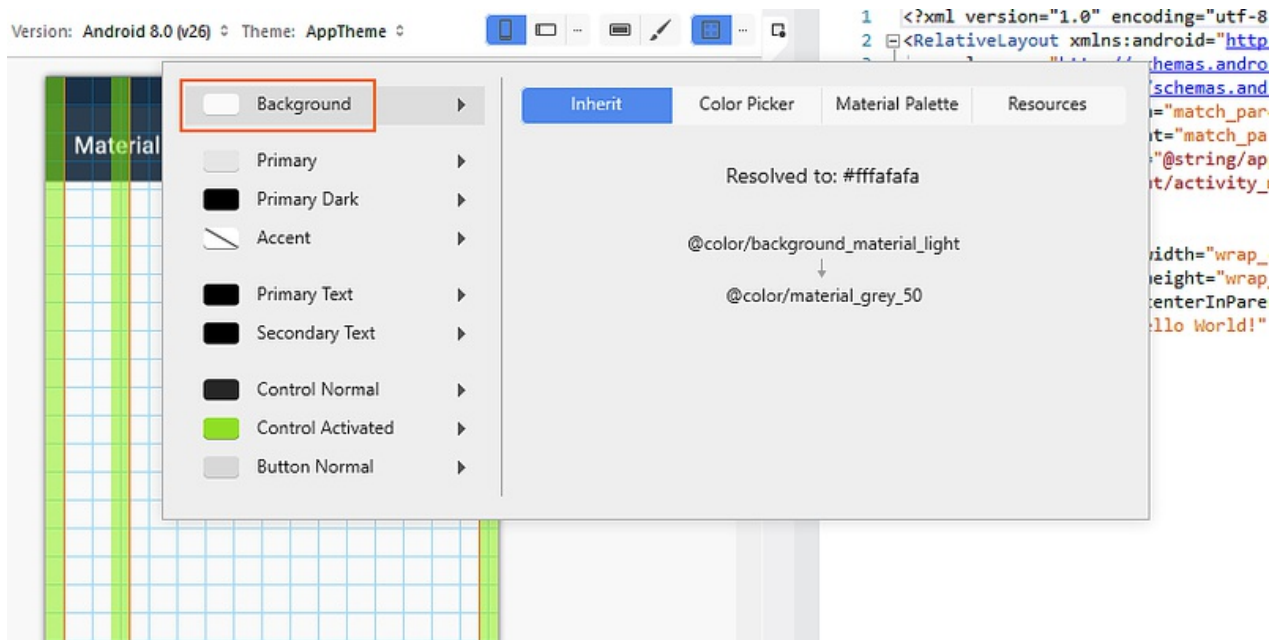
如果选择颜色位于左侧，右侧面板中提供了以下选项卡中，可以帮助您编辑该颜色：

- **继承**—显示所选颜色的样式继承关系图，并列出已解析的颜色和颜色代码分配给该主题颜色。
- **颜色选取器**—，可以更改所选的颜色为任意值。
- **材质调色板**—允许将所选的颜色更改为符合到 Material Design 的值。
- **资源**—，可以更改所选的颜色为一个主题中的其他现有的颜色资源。

让我们看看每个详细信息中的这些选项卡。

继承选项卡

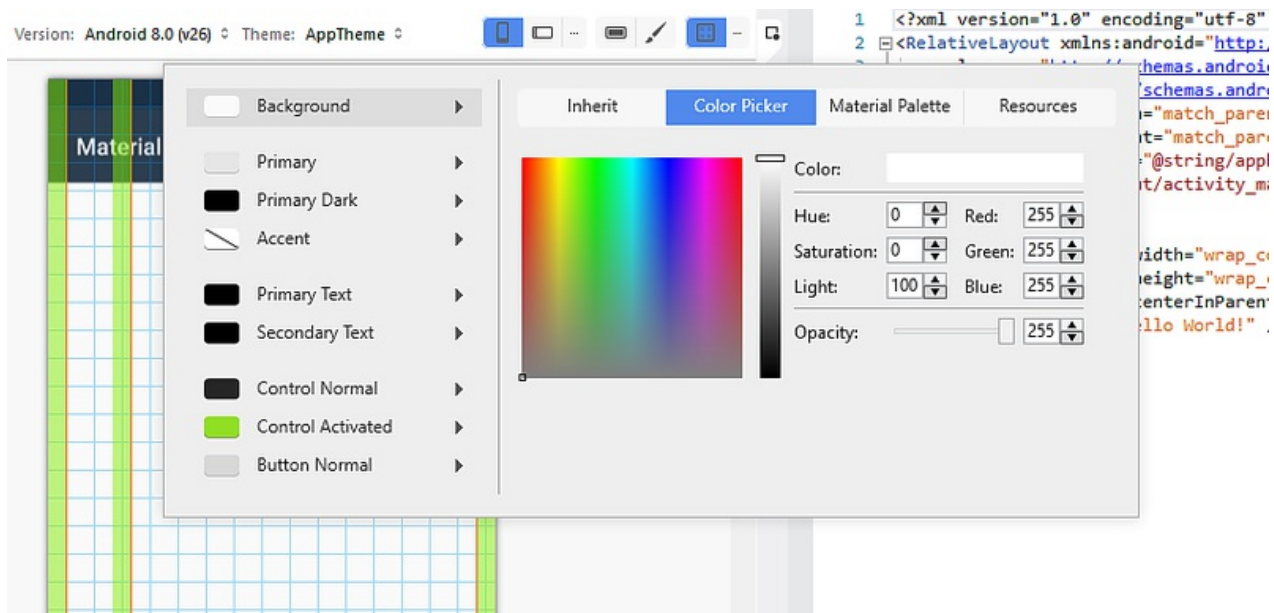
下面的示例中所示**Inherit**选项卡上列出的样式继承背景的颜色默认主题：



在此示例中，默认主题从一种使用样式继承 `@color/background_material_light` 但将重写它与 `color/material_grey_50`，其中包含的颜色代码值 `#ffffafafa`。有关样式继承的详细信息，请参阅[样式和主题](#)。

颜色选取器

以下屏幕截图演示了颜色选取器：



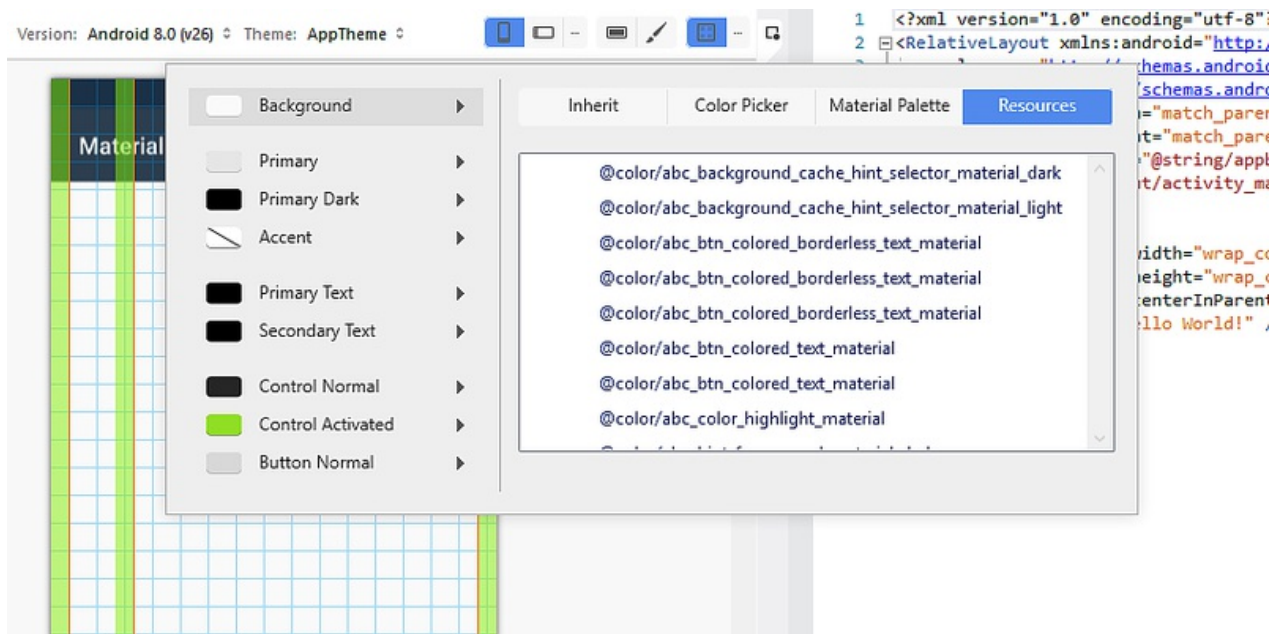
在此示例中，背景颜色可以更改通过不同的方式的任何值为：

- 直接单击一种颜色。
- 输入色调、饱和度和亮度值。
- 输入以十进制表示的 RGB（红色、绿色、蓝色）值。
- 设置所选颜色的 alpha（透明度）。
- 直接输入十六进制颜色代码。

在颜色选取器中选择的颜色是不限制到 Material Design 准则、或的可用颜色资源集。

资源

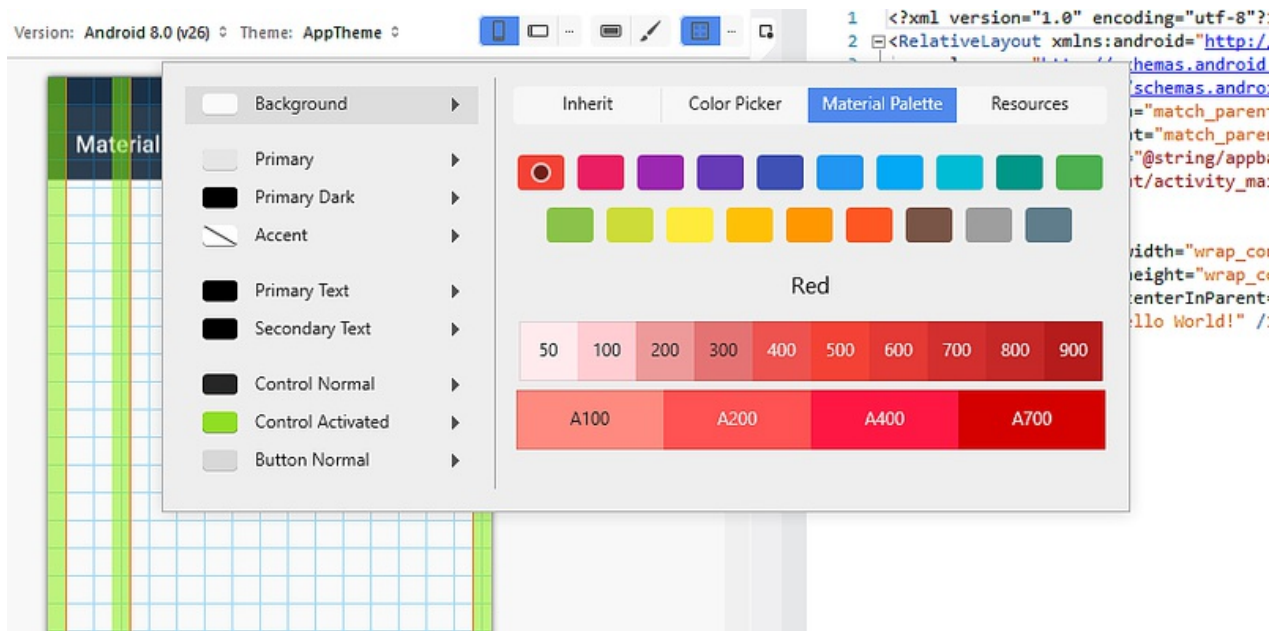
资源选项卡提供了一系列主题中已存在的颜色资源：



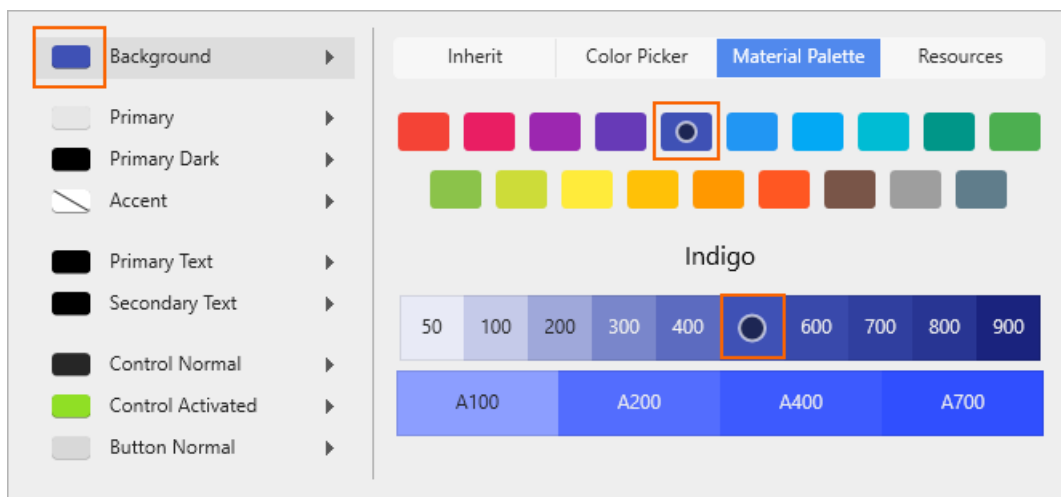
使用资源选项卡可限制您的选择为此颜色列表。请记住，如果您选择颜色资源已分配给另一个部分的主题，UI 的两个相邻元素可能“一起运行”（因为它们具有相同的颜色）且变得困难的用户来区分。

材质调色板

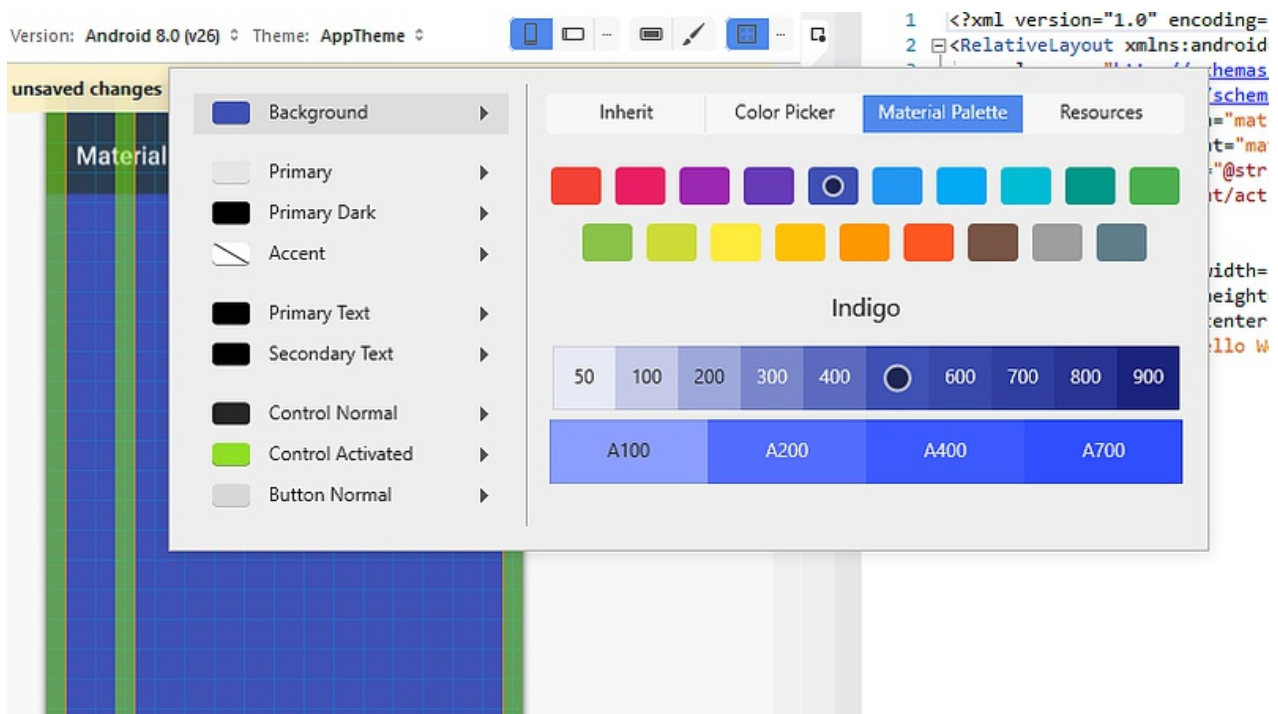
材料调色板选项卡随即打开材料设计调色板。从调色板选择颜色值限制您的颜色选择，以便与材料设计指南一致：



调色板的顶部显示主 Material Design 颜色的调色板的底部显示一系列所选的主要颜色的色调。例如，选择 **Indigo**，一系列 **Indigo** 色调显示在对话框的底部。时选择了 hue，该属性的颜色更改为所选的色调。在以下示例中，`Background Tint` 的按钮将变为 *Indigo 500*：



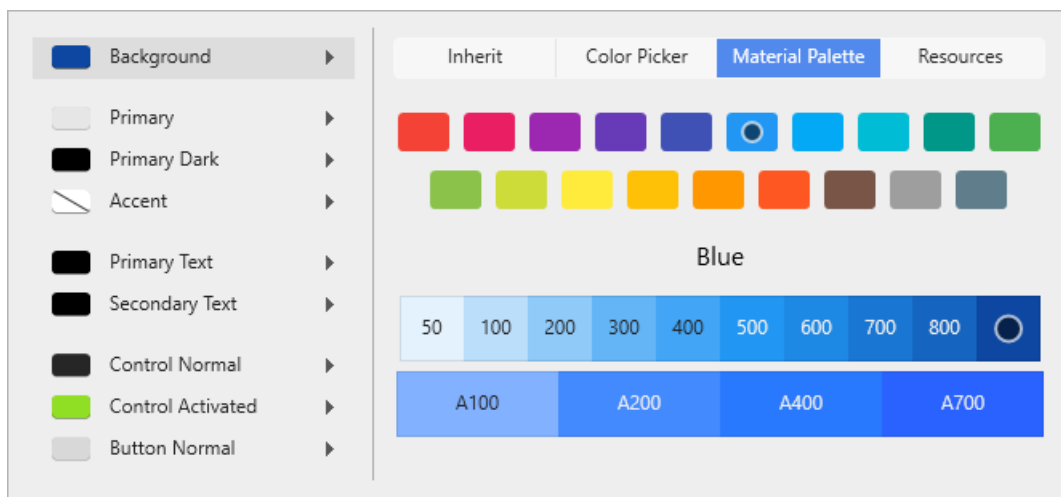
`Background Tint` 设置为的颜色代码 *Indigo 500* (`#ff3f51b5`)，并在设计器更新以反映此更改的背景色：



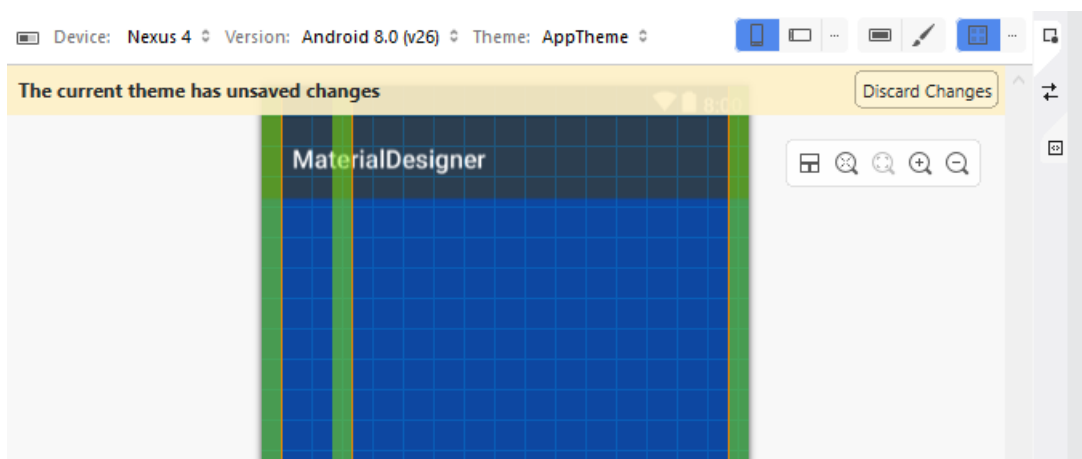
Material Design 的调色板的详细信息，请参阅 [Material Design 颜色调色板指南](#)。

创建新的主题

在以下示例中，我们将使用材料调色板可创建一个新的自定义主题。首先，我们将更改背景颜色蓝色 900:



更改颜色资源后，弹出一条消息的消息，*当前主题具有未保存的更改*。

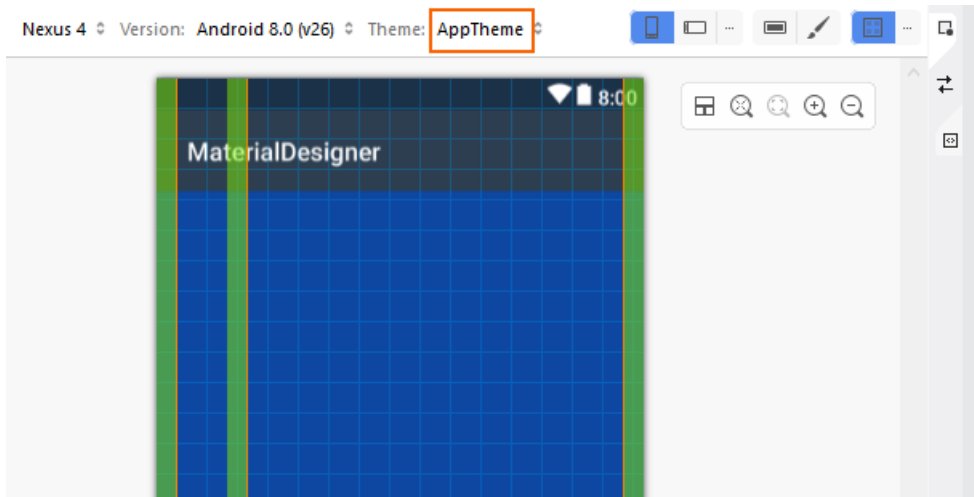


背景颜色在设计器中的已更改为新的颜色选择，但尚未保存此更改。此时，可以执行以下操作：

- 单击**放弃更改**放弃新的颜色选择（或选择）并恢复到你原始状态的主题。

- 按CTRL + S保存到所做的更改当前主题。

在以下示例中，CTRL + S，以便所做的更改保存到按下**AppTheme**:



总结

本主题介绍 Xamarin.Android 设计器中提供的材料设计功能。本文介绍了如何启用和配置材料设计网格，并介绍了如何使用主题编辑器来创建符合 Material Design 准则的新自定义主题。有关对 Material Design Xamarin.Android 支持的详细信息，请参阅[材料主题](#)。

相关链接

- [材料主题](#)
- [材料设计简介](#)

材料主题

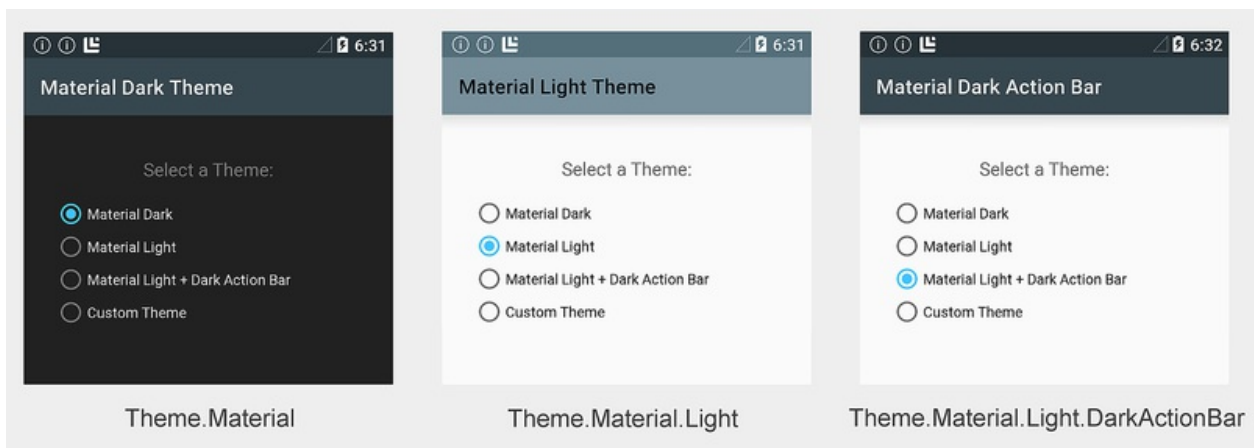
2018/10/19 • [Edit Online](#)

材料主题是一种用户界面样式，用于确定视图和开始使用 Android 5.0 (Lollipop) 的活动的`外观`。材料主题是内置到 Android 5.0，因此它通常由系统用户界面以及应用程序。材料主题不是"主题"意义上的用户可以动态设置菜单中选择一个系统范围外观选项。相反，可以为一系列可用于自定义您的应用程序的外观的相关内置的基本样式的认为材料主题。

Android 提供了三种材料主题风格：

- `Theme.Material` – 深色版本的材料主题使用;这是在 Android 5.0 默认风格。
- `Theme.Material.Light` – 材料主题的轻量版本。
- `Theme.Material.Light.DarkActionBar` – 材料主题，但使用深色操作栏的轻量版本。

此处显示这些材料主题类型的示例：



您可以从派生材料主题创建您自己的主题，重写部分或全部颜色特性。例如，可以创建派生自一个主题 `Theme.Material.Light`，但优先于应用栏颜色以匹配你的品牌颜色。您还可以设置的样式的各个视图;例如，可以创建的样式 `CardView` 会有更多的圆，并使用较暗的背景色。

可将单个主题用于整个应用程序，或可以为不同的屏幕（活动）在应用中使用不同的主题。上面的屏幕截图，例如，单个应用使用不同的主题针对每个活动来演示内置的颜色方案。单选按钮切换到不同的活动，该应用程序和结果，显示不同的主题。

因为仅在 Android 5.0 及更高版本支持材料主题，则不能用于它（或派生自材料主题自定义主题）主题到您的应用程序在早期版本的 Android 上运行。但是，可以配置应用以在 Android 5.0 设备上使用材料主题，并适当地回退到早期的主题在较旧版本的 Android 上运行时（请参阅[兼容性](#)部分中的详细信息）。

要求

以下是所需的基于 Xamarin 的应用中使用新的 Android 5.0 材料主题功能：

- **Xamarin.Android** – Xamarin.Android 4.20 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for mac。
- **Android SDK** – Android 5.0 (API 21) 或更高版本必须安装通过 Android SDK 管理器。
- **Java JDK 1.8** – 可以使用 JDK 1.7，如果您是专门针对 API 级别 23 和更早版本。提供了 JDK 1.8 [Oracle](#)。

若要了解如何配置 Android 5.0 应用程序项目，请参阅[设置了 Android 5.0 项目](#)。

使用内置的主题

使用材料主题的最简单方法是将应用配置为使用而无需自定义内置主题。如果不想显式配置了一个主题，您的应用程序将默认为 `Theme.Material`（深色主题）。如果你的应用程序只有一个活动，您可以在应用程序级别配置主题。如果应用具有多个活动，可以以使它使用相同的主题可跨所有活动，或可以将不同的主题分配给不同的活动应用程序级别的配置主题。以下部分介绍如何在应用级别和活动级别配置主题。

主题设置应用程序

若要配置整个应用程序以使用材料主题风格，设置 `android:theme` 属性中的应用程序节点 **AndroidManifest.xml** 到以下项之一：

- `@android:style/Theme.Material` – 深色主题。
- `@android:style/Theme.Material.Light` – 浅色主题。
- `@android:style/Theme.Material.Light.DarkActionBar` – 使用深色操作栏的浅色主题。

下面的示例配置应用程序 *MyApp* 使用浅色主题：

```
<application android:label="MyApp"
             android:theme="@android:style/Theme.Material.Light">
</application>
```

或者，可以将应用程序设置 `Theme` 中的属性 **AssemblyInfo.cs** (或 **Properties.cs**)。例如：

```
[assembly: Application(Theme="@android:style/Theme.Material.Light")]
```

如果应用程序主题设置为 `@android:style/Theme.Material.Light`，在每个活动 *MyApp* 将使用显示 `Theme.Material.Light`。

主题设置活动

将活动的主题，您将添加 `Theme` 设置为 `[Activity]` 活动声明上方特性并将分配 `Theme` 到你想要使用的材料主题风格。将活动与下面的示例主题 `Theme.Material.Light`：

```
[Activity(Theme = "@android:style/Theme.Material.Light",
         Label = "MyApp", MainLauncher = true, Icon = "@drawable/icon")]
```

此应用中的其他活动，则使用默认 `Theme.Material` 深色方案（或者，如果配置，应用程序主题设置）。

使用自定义主题

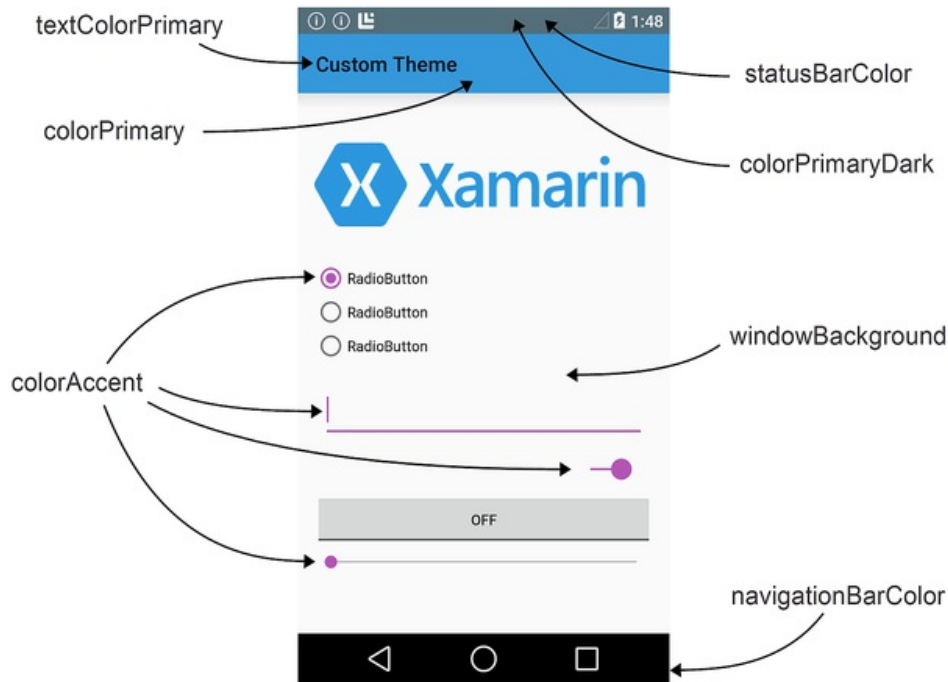
可以通过创建样式与自己的品牌应用的自定义主题增强自己的品牌'的颜色。若要创建自定义主题，您定义新的样式从内置的材料主题风格，派生的重写你想要更改的颜色属性。例如，可以定义派生的自定义主题 `Theme.Material.Light.DarkActionBar` 和屏幕背景色更改为米色而不是空白。

材料主题公开自定义的以下布局属性：

- `colorPrimary` – 应用栏的颜色。
- `colorPrimaryDark` – 状态栏和上下文应用栏的颜色这通常是深色版本 `colorPrimary`。
- `colorAccent` – UI 控件，如复选框、单选按钮和文本的编辑框的颜色。
- `windowBackground` – 屏幕背景的颜色。
- `textColorPrimary` – 在应用栏中的 UI 文本的颜色。

- `statusBarColor` – 状态栏的颜色。
- `navigationBarColor` – 导航栏的颜色。

下图中标记这些屏幕区域：



默认情况下 `statusBarColor` 设置的值为 `colorPrimaryDark`。可以设置 `statusBarColor` 为纯色，或将其设置为 `@android:color/transparent` 进行状态栏透明。在导航栏还可以使可透明通过设置 `navigationBarColor` 到 `@android:color/transparent`。

创建自定义应用主题

可以通过创建和修改文件中的创建自定义应用主题资源应用项目的文件夹。若要在设置应用与自定义主题样式，使用以下步骤：

- 创建 `colors.xml` 中的文件资源 `/values` 一此文件用于定义您的自定义主题颜色。例如，可以将以下代码粘贴 `colors.xml` 来帮助你入门：

```
<?xml version="1.0" encoding="UTF-8" ?>
<resources>
    <color name="my_blue">#3498DB</color>
    <color name="my_green">#77D065</color>
    <color name="my_purple">#B455B6</color>
    <color name="my_gray">#738182</color>
</resources>
```

- 修改此示例文件，以定义的名称和颜色资源将在您的自定义主题中使用的颜色代码。
- 创建资源/值 `-v21` 文件夹。在此文件夹中创建 `styles.xml` 文件：

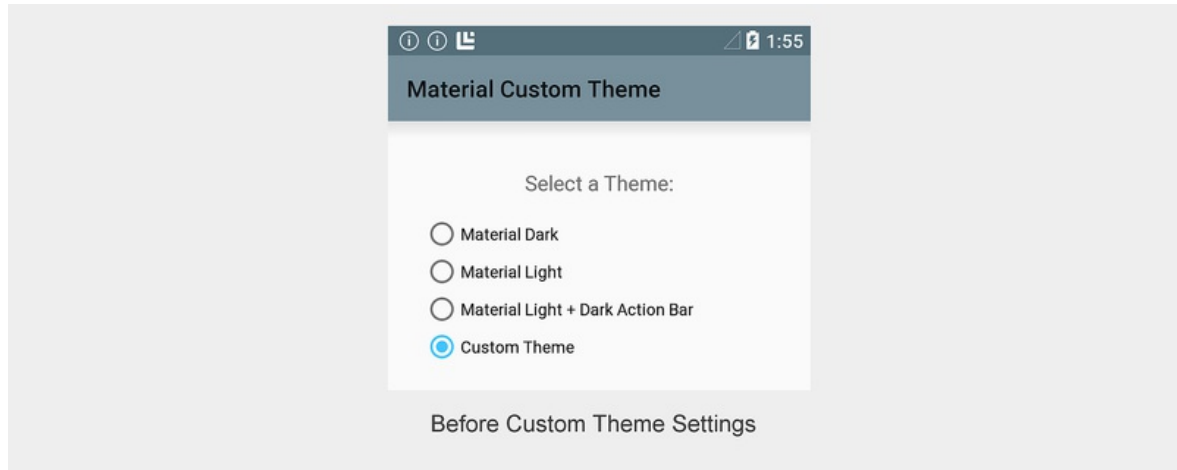


请注意, 资源/值 **-v21** 特定于 Android 5.0—较旧版本的 Android 不会读取此文件夹中的文件。

- 添加 `resources` 到节点 **styles.xml**, 并定义 `style` 节点与自定义主题的名称。例如, 下面是 **styles.xml** 文件, 用于定义 *MyCustomTheme* (派生自内置 `Theme.Material.Light` 主题样式):

```
<?xml version="1.0" encoding="UTF-8" ?>
<resources>
    <!-- Inherit from the light Material Theme -->
    <style name="MyCustomTheme" parent="android:Theme.Material.Light">
        <!-- Customizations go here -->
    </style>
</resources>
```

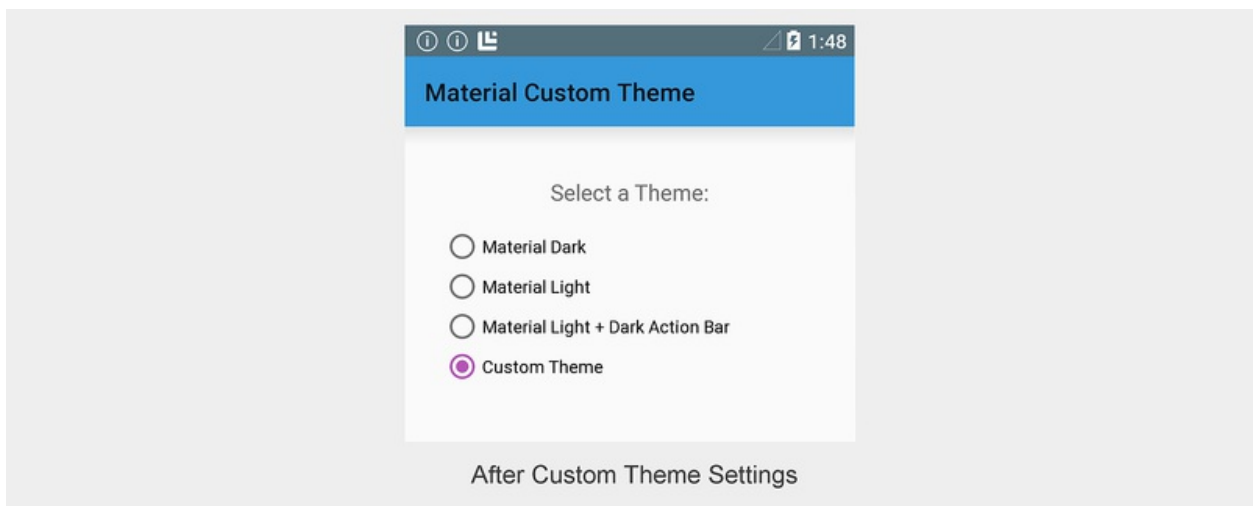
- 在此情况下, 使用的应用程序 *MyCustomTheme* 将显示股票 `Theme.Material.Light` 而无需自定义项的主题:



- 添加到颜色自定义 **styles.xml** 通过定义你想要更改的布局属性的颜色。例如, 若要更改到应用栏颜色 `my_blue` 并更改到 UI 控件的颜色 `my_purple`, 添加到颜色重写 **styles.xml**, 请参阅配置中的颜色资源 **colors.xml**:

```
<?xml version="1.0" encoding="UTF-8" ?>
<resources>
    <!-- Inherit from the light Material Theme -->
    <style name="MyCustomTheme" parent="android:Theme.Material.Light">
        <!-- Override the app bar color -->
        <item name="android:colorPrimary">@color/my_blue</item>
        <!-- Override the color of UI controls -->
        <item name="android:colorAccent">@color/my_purple</item>
    </style>
</resources>
```

这些更改后, 使用一个应用程序, 使用 *MyCustomTheme* 将显示在应用栏颜色 `my_blue` 和中的 UI 控件 `my_purple`, 但使用 `Theme.Material.Light` 其他地方的配色方案:



在此示例中，*MyCustomTheme*借用从颜色 `Theme.Material.Light` 背景颜色、状态栏和文本颜色，但它更改到应用栏的颜色 `my_blue` 设置单选按钮的颜色和 `my_purple`。

创建自定义视图样式

Android 5.0 还使您可以设置一个单独的视图的样式。在创建后 `colors.xml` 并 `styles.xml`（如在上一部分所述），可以添加到视图样式 `styles.xml`。若要设置样式的单个视图，使用以下步骤：

- 编辑 `Resources/values-v21/styles.xml` 并添加 `style` 节点与自定义视图样式的名称。设置为在此视图的自定义的颜色特性 `style` 节点。例如，若要创建自定义 `CardView` 更具有圆角和使用的样式 `my_blue` 卡片背景色，以添加 `style` 节点 `styles.xml` (内 `resources` 节点) 和配置的背景颜色和角半径：

```
<!-- Theme an individual view: -->
<style name="CardView.MyBlue">

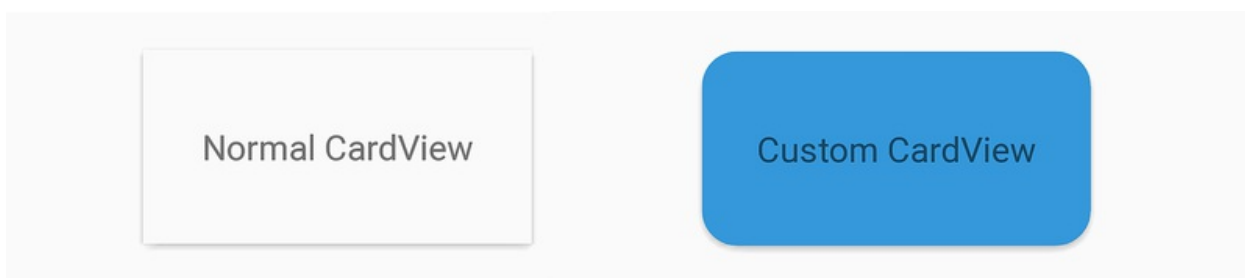
    <!-- Change the background color to Xamarin blue: -->
    <item name="cardBackgroundColor">@color/my_blue</item>

    <!-- Make the corners very round: -->
    <item name="cardCornerRadius">18dp</item>
</style>
```

- 在布局中，设置 `style` 该视图以匹配在上一步中选择的自定义样式名称的属性。例如：

```
<android.support.v7.widget.CardView
    style="@style/CardView.MyBlue"
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:layout_gravity="center_horizontal">
```

下面的屏幕截图举例说明默认值 `CardView`（所示在左侧）如相比 `CardView` 用自定义设计的 `CardView.MyBlue`（显示在右侧）的主题：



在此示例中，自定义 `CardView` 使用的背景色显示 `my_blue` 和 18dp 圆角半径。

兼容性

若要设置您的应用程序，以便它在 Android 5.0 上使用材料主题，但会自动恢复到较旧的 Android 版本的向下兼容样式的样式，使用以下步骤：

- 定义中的一个自定义主题 **Resources/values-v21/styles.xml** 派生材料主题样式。例如：

```
<resources>
    <style name="MyCustomTheme" parent="android:Theme.Material.Light">
        <!-- Your customizations go here -->
    </style>
</resources>
```

- 定义中的一个自定义主题 **Resources/values/styles.xml**，派生自一个较旧的主题，但使用与上述相同主题名称。例如：

```
<resources>
    <style name="MyCustomTheme" parent="android:Theme.Holo.Light">
        <!-- Your customizations go here -->
    </style>
</resources>
```

- 在中 **AndroidManifest.xml**，配置您的应用程序的自定义主题名称。例如：

```
<application android:label="MyApp"
    android:theme="@style/MyCustomTheme">
</application>
```

- 或者，你可以设置特定的活动中使用自定义主题样式：

```
[Activity(Label = "MyActivity", Theme = "@style/MyCustomTheme")]
```

如果您的主题使用在中定义的颜色 **colors.xml** 文件中，请务必将此文件中的放置资源/值(而非资源/值-v21)，以便这两个版本颜色定义可以访问您的自定义主题。

当你的应用在 Android 5.0 设备上运行时，它将使用中指定的主题定义 **Resources/values-v21/styles.xml**。当较旧的 Android 设备上运行此应用时，它将自动回退到指定的主题定义 **Resources/values/styles.xml**。

主题与较旧的 Android 版本的兼容性的详细信息，请参阅 [备用资源](#)。

总结

本文介绍了包括在 Android 5.0 (Lollipop) 的新材料主题用户界面样式。它描述可用于在设置应用样式三种内置材料主题风格、介绍了如何创建用于品牌打造您的应用程序，自定义主题和提供的示例为主题的单个视图。最后，本文介绍了如何同时向下兼容较旧版本的 Android 应用程序中使用材料主题。

相关链接

- [ThemeSwitcher \(示例\)](#)
- [棒棒糖形简介](#)
- [CardView](#)
- [备用资源](#)
- [Android 棒棒糖形](#)
- [Android 饼图开发人员](#)

- 材料设计
- 材料设计原则
- 保持兼容性

用户配置文件

2018/10/26 • [Edit Online](#)

Android 支持枚举联系人 `ContactsContract` 自 API 级别 5 的提供程序。例如，列出联系人非常简单，使用 `ContactsContract.Contacts` 类，如下面的代码示例中所示：

```
// Get the URI for the user's contacts:
var uri = ContactsContract.Contacts.ContentUri;

// Setup the "projection" (columns we want) for only the ID and display name:
string[] projection = {
    ContactsContract.Contacts.InterfaceConsts.Id,
    ContactsContract.Contacts.InterfaceConsts.DisplayName };

// Use a CursorLoader to retrieve the user's contacts data:
CursorLoader loader = new CursorLoader(this, uri, projection, null, null, null);
ICursor cursor = (ICursor)loader.LoadInBackground();

// Print the contact data to the console if reading back succeeds:
if (cursor != null)
{
    if (cursor.moveToFirst())
    {
        do
        {
            Console.WriteLine("Contact ID: {0}, Contact Name: {1}",
                               cursor.GetString(cursor.GetColumnIndex(projection[0])),
                               cursor.GetString(cursor.GetColumnIndex(projection[1])));
        } while (cursor.MoveNext());
    }
}
```

Android 4 (API 级别 14) 开头 `ContactsContract.Profile` 类是可通过 `ContactsContract` 提供程序。

`ContactsContract.Profile` 包括联系人数据，例如设备所有者的姓名和电话号码的设备的所有者提供对个人配置文件的访问。

所需权限

若要读取和写入的联系人数据，应用程序必须请求 `READ_CONTACTS` 和 `WRITE_CONTACTS` 权限，分别。此外，若要阅读和编辑用户配置文件，应用程序必须请求 `READ_PROFILE` 和 `WRITE_PROFILE` 权限。

更新配置文件数据

设置这些权限后，应用程序可以使用正常的 Android 技术与用户配置文件的数据进行交互。例如，若要更新配置文件的显示名称，请调用 `ContentResolver.Update` 与 `Uri` 通过检

索 `ContactsContract.Profile.ContentRawContactsUri` 属性，如所示如下：

```
var values = new ContentValues ();
values.Put (ContactsContract.Contacts.InterfaceConsts.DisplayName, "John Doe");

// Update the user profile with the name "John Doe":
ContentResolver.Update (ContactsContract.Profile.ContentRawContactsUri, values, null, null);
```

读取配置文件数据

发出到查询[ContactsContract.Profile.ContentUri](#)读回的配置文件数据。例如，下面的代码将读取用户配置文件的显示名称：

```
// Read the profile
var uri = ContactsContract.Profile.ContentUri;

// Setup the "projection" (column we want) for only the display name:
string[] projection = {
    ContactsContract.Contacts.InterfaceConsts.DisplayName };

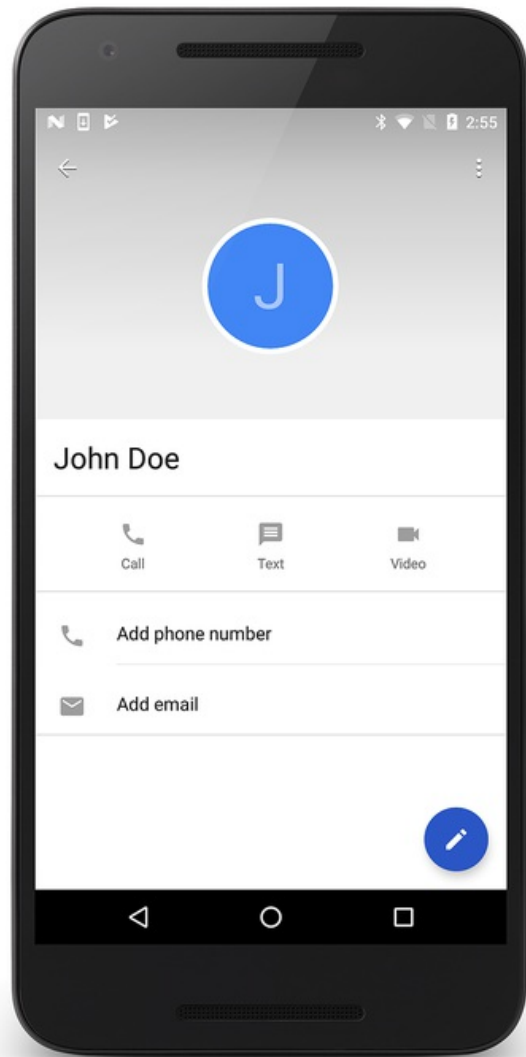
// Use a CursorLoader to retrieve the data:
CursorLoader loader = new CursorLoader(this, uri, projection, null, null, null);
ICursor cursor = (ICursor)loader.LoadInBackground();
if (cursor != null)
{
    if (cursor.MoveToFirst ())
    {
        Console.WriteLine(cursor.GetString (cursor.GetColumnIndex (projection [0])));
    }
}
```

导航到用户配置文件

最后，若要导航到用户配置文件，创建使用意向 `ActionView` 操作和一个 `ContactsContract.Profile.ContentUri` 然后将其传递给 `StartActivity` 方法如下：

```
var intent = new Intent (Intent.ActionView,
    ContactsContract.Profile.ContentUri);
StartActivity (intent);
```

当运行上面的代码，用户配置文件将显示如以下屏幕截图中所示：



使用用户配置文件是类似于与 Android 中的其他数据进行交互，它提供了一层额外的设备的个性化。

相关链接

- [ContactsProviderDemo \(示例\)](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

初始屏幕

2018/10/26 • [Edit Online](#)

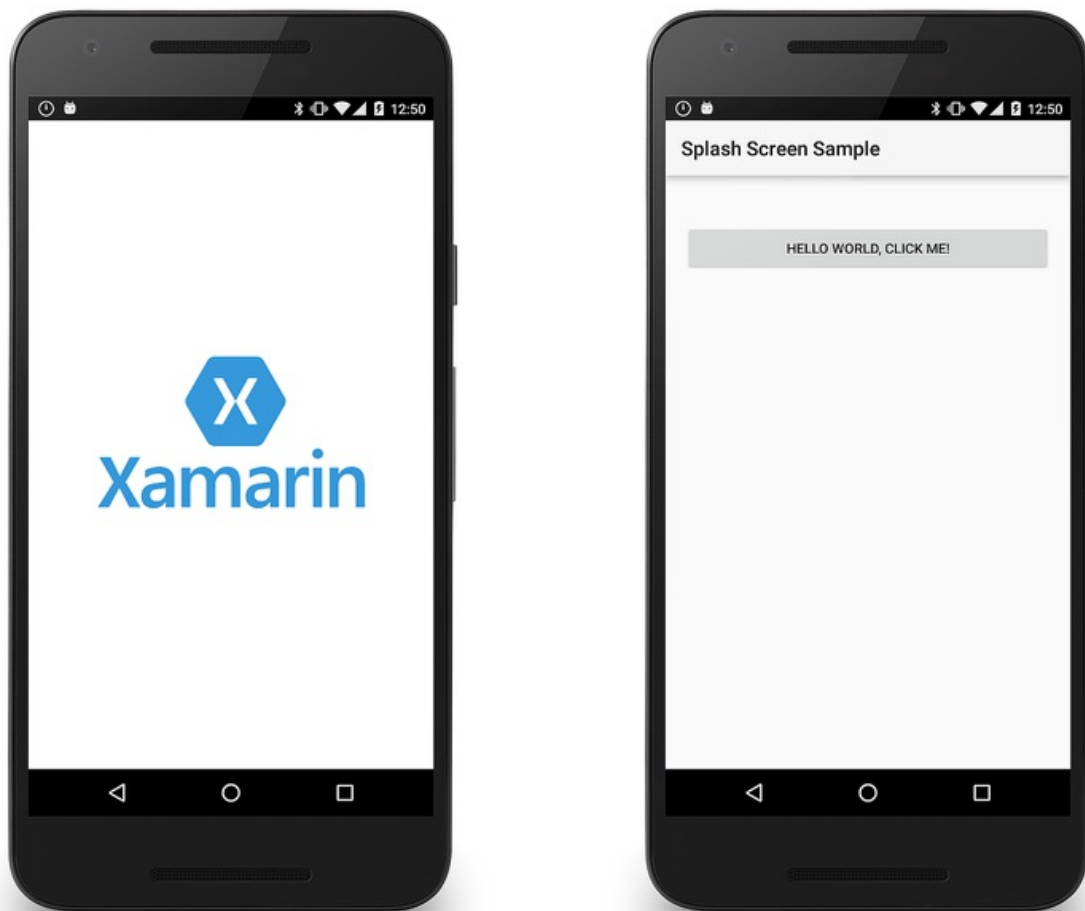
Android 应用程序需要一些时间才能启动，尤其是在设备上首次启动应用程序。初始屏幕可能会显示开始向上向用户或指示品牌的进度。

概述

Android 应用程序需要一些时间才能启动，尤其是在第一个期间应用程序运行在设备上 (有时这称为_冷启动_)。可能会显示初始屏幕启动进度给用户，或者根本不显示品牌信息以确定和推广应用程序。

本指南介绍 Android 应用程序中实现初始屏幕的一项技术。其中包括以下步骤：

1. 创建初始屏幕的可绘制资源。
2. 定义将显示可绘制资源的新主题。
3. 将新活动添加到将用作由上一步中创建的主题定义在初始屏幕的应用程序。



要求

本指南假定应用程序面向 Android API 级别 15 (Android 4.0.3) 或更高版本。应用程序还必须具

有 **Xamarin.Android.Support.v4** 并 **Xamarin.Android.Support.v7.AppCompat** NuGet 包添加到项目中。

中可能找到的所有代码和本指南中的 XML [初始屏幕](#) 用于本指南的示例项目。

实现初始屏幕

呈现和显示初始屏幕的最快方法是创建一个自定义主题，并将其应用到表现出的初始屏幕的活动。当呈现该活动时，它加载主题，并适用于活动的背景（引用主题）的可绘制资源。这种方法可以避免创建布局文件。

显示的品牌的活动作为实现初始屏幕可绘制，执行任何初始化和启动任何任务。一旦具有引导应用程序，初始屏幕活动启动主活动，并从应用程序的 back 堆栈中移除本身。

创建可绘制的初始屏幕

初始屏幕将显示 XML 可绘制背景的初始屏幕活动。需要使用图像的位图化图像（如 PNG 或 JPG）来显示它。

在本指南中，我们将使用 [层列表](#) 中心应用程序中的初始屏幕图像。以下代码片段示范了 `drawable` 资源使用

`layer-list`：

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <color android:color="@color/splash_background"/>
    </item>
    <item>
        <bitmap
            android:src="@drawable/splash"
            android:tileMode="disabled"
            android:gravity="center"/>
    </item>
</layer-list>
```

这 `layer-list` 初始图像将居中 **splash.png** 由指定的背景上 `@color/splash_background` 资源。将此 XML 文件中的放置资源 **/drawable** 文件夹 (例如，**Resources/drawable/splash_screen.xml**)。

创建初始屏幕可绘制后下，一步是创建初始屏幕的主题。

实现一个主题

若要创建初始屏幕活动的自定义主题，请编辑（或添加）文件 **values/styles.xml** 并创建新的 `style` 初始屏幕元素。一个示例 **values/style.xml** 如下所示的文件是与 `style` 名为 **MyTheme.Splash**：

```
<resources>
    <style name="MyTheme.Base" parent="Theme.AppCompat.Light">
    </style>

    <style name="MyTheme" parent="MyTheme.Base">
    </style>

    <style name="MyTheme.Splash" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="android:windowBackground">@drawable/splash_screen</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowFullscreen">true</item>
    </style>
</resources>
```

MyTheme.Splash 是非常 spartan—声明窗口背景，显式从窗口中，删除的标题栏并声明它是全屏。如果你想要创建初始屏幕，它模拟您的应用程序的 UI 之前活动增大第一个布局，则可以使用 `windowContentOverlay` 而非 `windowBackground` 样式定义中。在这种情况下，还必须修改 **splash_screen.xml** `drawable`，以便它显示你的 UI 的模拟。

创建初始活动

现在, 我们需要适用于 Android 启动我们的初始映像并执行的任何启动任务的新活动。以下代码是完整的初始屏幕实现的示例:

```
[Activity(Theme = "@style/MyTheme.Splash", MainLauncher = true, NoHistory = true)]
public class SplashActivity : AppCompatActivity
{
    static readonly string TAG = "X:" + typeof(SplashActivity).Name;

    public override void OnCreate(Bundle savedInstanceState, PersistableBundle persistentState)
    {
        base.OnCreate(savedInstanceState, persistentState);
        Log.Debug(TAG, "SplashActivity.OnCreate");
    }

    // Launches the startup task
    protected override void OnResume()
    {
        base.OnResume();
        Task startupWork = new Task(() => { SimulateStartup(); });
        startupWork.Start();
    }

    // Simulates background work that happens behind the splash screen
    async void SimulateStartup ()
    {
        Log.Debug(TAG, "Performing some startup work that takes a bit of time.");
        await Task.Delay (8000); // Simulate a bit of startup work.
        Log.Debug(TAG, "Startup work is finished - starting MainActivity.");
        StartActivity(new Intent(Application.Context, typeof (MainActivity)));
    }
}
```

`SplashActivity` 显式使用已创建在上一节中重写应用程序的默认主题的主题。无需加载的布局 `OnCreate` 正如主题所声明可绘制背景。

务必要设置 `NoHistory=true` 属性, 以便从 back 堆栈中移除活动。若要防止后退按钮取消启动过程, 您可以覆盖 `OnBackPressed`, 并让它不执行任何操作:

```
public override void OnBackPressed() { }
```

以异步方式在执行启动工作 `OnResume`。这是必需的因此, 启动工作不会变慢或延迟的启动屏幕的外观。完成工作后, `SplashActivity` 将启动 `MainActivity` 和用户可能会开始与应用交互。

这一新 `SplashActivity` 通过设置设置为应用程序的启动器活动 `MainLauncher` 属性为 `true`。因为 `SplashActivity` 启动器活动中, 您必须编辑 `MainActivity.cs`, 并删除 `MainLauncher` 属性从 `MainActivity`:

```
[Activity(Label = "@string/ApplicationName")]
public class MainActivity : AppCompatActivity
{
    // Code omitted for brevity
}
```

横向模式

在上一步骤中实现初始屏幕将在纵向和横向模式下正常显示。但是, 在某些情况下是需要具有单独的初始屏幕的纵向和横向模式 (例如, 如果初始图像是全屏幕)。

若要添加的横向模式下的初始屏幕, 请使用以下步骤:

1. 在中可绘制资源/ 文件夹中, 添加你想要使用初始屏幕图像的布局版本。在此示例中, **splash_logo_land.png**是上面的示例 (它使用白色时将其视为而不是蓝色) 中使用的徽标的横向版本。
2. 在资源/drawable文件夹中, 创建的布局版本 `layer-list` 可绘制的先前定义 (例如, **splash_screen_land.xml**)。此文件中设置初始屏幕图像横向形式的位图路径。在以下示例中, **splash_screen_land.xml**使用**splash_logo_land.png**:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <color android:color="@color/splash_background"/>
    </item>
    <item>
        <bitmap
            android:src="@drawable/splash_logo_land"
            android:tileMode="disabled"
            android:gravity="center"/>
    </item>
</layer-list>
```

3. 创建资源/值-land如果它尚不存在的文件夹。
4. 将文件添加**colors.xml**并**style.xml**到值 **land** (这些都可以复制和修改从现有**values/colors.xml**并**values/style.xml**文件)。
5. 修改值的土地/**style.xml**, 以便它使用横向形式的可绘制 `windowBackground`。在此示例中, **splash_screen_land.xml**使用:

```
<resources>
    <style name="MyTheme.Base" parent="Theme.AppCompat.Light">
    </style>
    <style name="MyTheme" parent="MyTheme.Base">
    </style>
    <style name="MyTheme.Splash" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="android:windowBackground">@drawable/splash_screen_land</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowFullscreen">true</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="android:windowActionBar">true</item>
    </style>
</resources>
```

6. 修改值的土地/**colors.xml**配置想要在初始屏幕的布局版本使用的颜色。在此示例中, 初始背景色更改为蓝色的横向模式下:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="primary">#2196F3</color>
    <color name="primaryDark">#1976D2</color>
    <color name="accent">#FFC107</color>
    <color name="window_background">#F5F5F5</color>
    <color name="splash_background">#3498DB</color>
</resources>
```

7. 生成并再次运行该应用。旋转为横向模式时仍显示初始屏幕的设备。初始屏幕更改为横向版本:



请注意，使用横向模式下初始屏幕不始终提供无缝体验。默认情况下，Android 纵向模式中启动该应用，并转换它为横向模式下，即使设备已在横向模式下。结果，启动应用时在设备处于横向模式时，如果该设备将简要显示纵向初始屏幕，然后对从纵向到横向初始屏幕旋转进行动画处理。遗憾的是，发生此初始纵向到横向转换，即使 `ScreenOrientation = Android.Content.PM.ScreenOrientation.Landscape` 指定在初始活动的标志。若要解决此限制的最佳方式是在纵向和横向模式中正确创建呈现单个初始屏幕图像。

总结

本指南介绍在 Xamarin.Android 应用程序中实现初始屏幕的一种方法也就是说，将自定义主题应用于启动活动。

相关链接

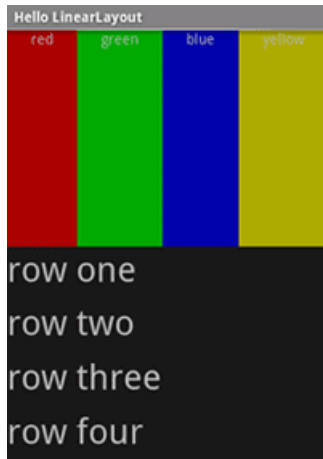
- [初始屏幕（示例）](#)
- [层列表 Drawable](#)
- [材料设计模式-启动屏幕](#)

布局

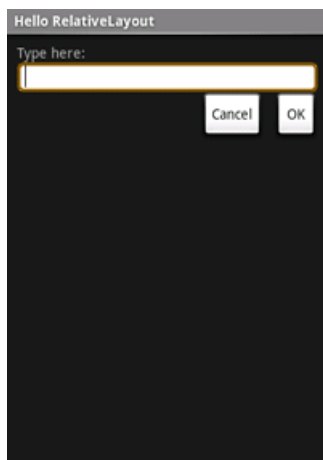
2018/10/26 • [Edit Online](#)

布局用于排列元素构成（例如活动）的屏幕的 UI 接口。以下部分介绍如何在 Xamarin.Android 应用中使用的最常用的布局。

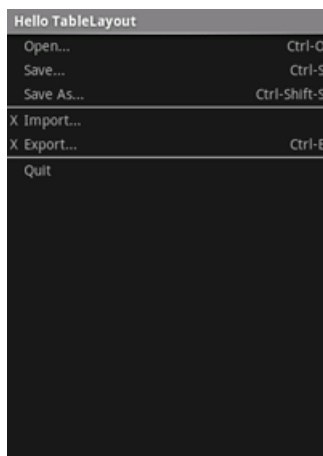
- [LinearLayout](#)是垂直或水平方向线性，显示子视图元素的视图组。



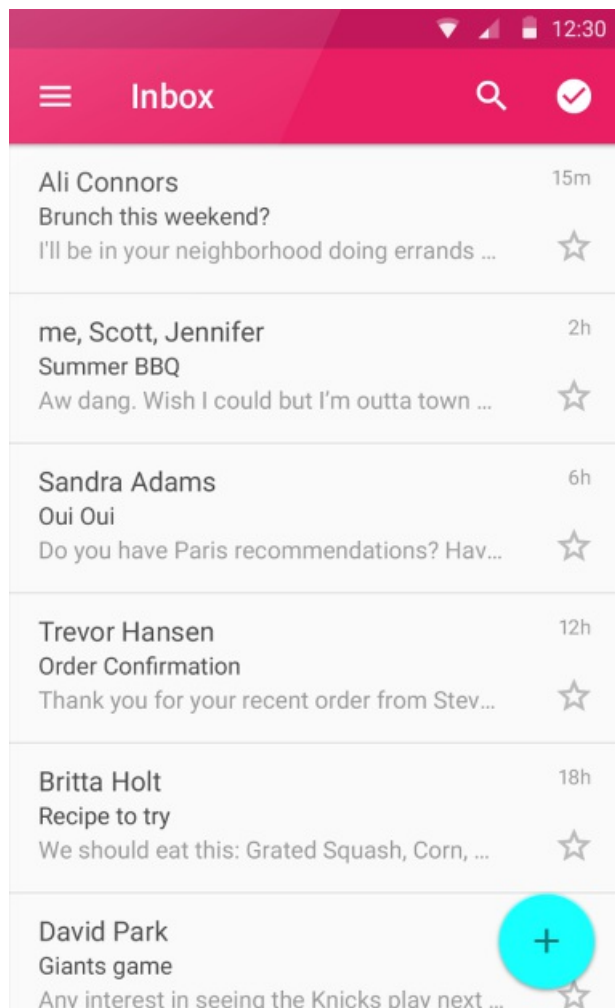
- [RelativeLayout](#)是查看组中的相对位置显示子视图元素。相对于同级元素，可以指定视图的位置。



- [TableLayout](#)是行和列中显示子视图元素的视图组。



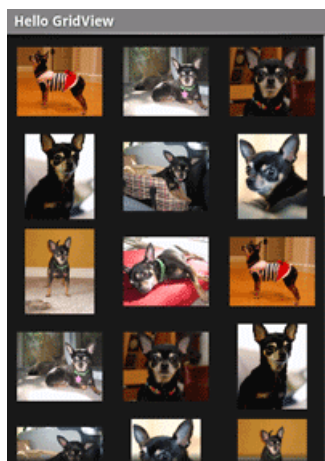
- [RecyclerView](#)是 UI 元素，它显示在列表或网格，使用户可以滚动浏览集合中项的集合。



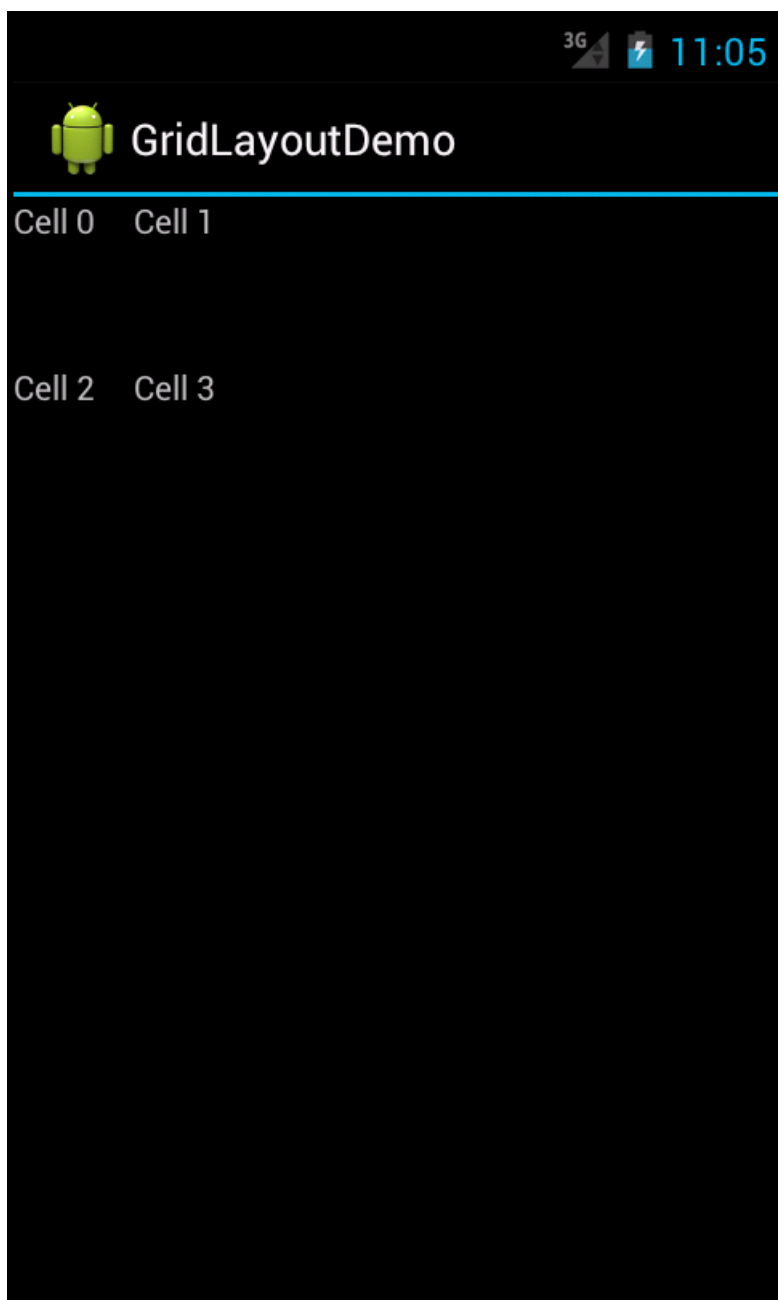
- **ListView** 是创建的可滚动项列表的视图组。列表项会自动插入到使用列表适配器列表中。`ListView` 是 Android 应用程序的重要 UI 组件，因为它用于无处不在从菜单选项的短列表到联系人或 internet 收藏夹的长列表。它提供了显示的行，可以使用内置样式格式化，也可以广泛地自定义滚动列表的简单方法。`ListView` 实例需要适配器以向它馈送行视图中包含的数据。



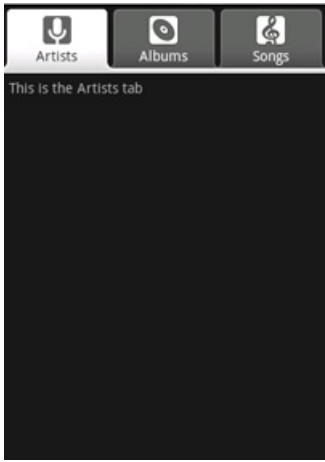
- **GridView** 是可滚动的二维网格中显示项的 UI 元素。



- [GridLayout](#)是支持布局类似于 HTML 表的 2D 网格中的视图的视图组。



- [选项卡式布局](#)由于其简单性和易用性是移动应用程序中的常见用户界面模式。它们提供一致、简单的方法的应用程序中的各种屏幕之间进行导航。



LinearLayout

2018/10/26 • [Edit Online](#)

`LinearLayout` 是 `ViewGroup` 显示子 `View` 线性方向中的元素垂直或水平。

您应该小心使用过度 `LinearLayout`。如果开始嵌套多个 `LinearLayout` s, 您可能需要考虑使用 `RelativeLayout` 改为。

启动一个名为的新项目 **HelloLinearLayout**。

打开 **Resources/Layout/Main.xml** 并插入以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" >
        <TextView
            android:text="red"
            android:gravity="center_horizontal"
            android:background="#aa0000"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1" />
        <TextView
            android:text="green"
            android:gravity="center_horizontal"
            android:background="#00aa00"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1" />
        <TextView
            android:text="blue"
            android:gravity="center_horizontal"
            android:background="#0000aa"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1" />
        <TextView
            android:text="yellow"
            android:gravity="center_horizontal"
            android:background="#aaaa00"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1" />
    </LinearLayout>

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" >
        <TextView
            android:text="row one"
            android:textSize="15pt"
            android:layout_width="match_parent"
```

```

        android:layout_width=        match_parent
        android:layout_height=       "wrap_content"
        android:layout_weight=       "1"    />
<TextView
    android:text=        "row two"
    android:textSize=    "15pt"
    android:layout_width=    "match_parent"
    android:layout_height=    "wrap_content"
    android:layout_weight=    "1"    />
<TextView
    android:text=        "row three"
    android:textSize=    "15pt"
    android:layout_width=    "match_parent"
    android:layout_height=    "wrap_content"
    android:layout_weight=    "1"    />
<TextView
    android:text=        "row four"
    android:textSize=    "15pt"
    android:layout_width=    "match_parent"
    android:layout_height=    "wrap_content"
    android:layout_weight=    "1"    />
</LinearLayout>

</LinearLayout>

```

请仔细检查此 XML。没有根 `LinearLayout` 它定义自己的方向为垂直—所有子 `View` s (它的具有两个) 将是堆积垂直。第一个子级是另一个 `LinearLayout` 使用水平方向, 并且第二个子级 `LinearLayout` 使用垂直方向。每个嵌套 `LinearLayout` s 包含多个 `TextView` 元素, 它们是由其父级定义的方式与每个其他面向 `LinearLayout` 。

现在, 打开 **HelloLinearLayout.cs** 并确保它将加载 **Resources/Layout/Main.xml** 中的布局 `OnCreate()` 方法:

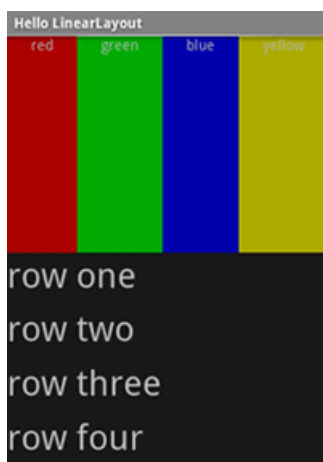
```

protected override void OnCreate (Bundle savedInstanceState)
{
    base.OnCreate (savedInstanceState);
    SetContentView (Resource.Layout.Main);
}

```

`SetContentView(int)` 方法加载的布局文件 `Activity`, 按资源 ID 指定—`Resources.Layout.Main` 指资源/布局 **/Main.xml** 布局文件。

运行该应用程序。你应看到以下信息:



请注意, XML 属性定义每个视图的行为的方法。尝试试验不同的值 `android:layout_weight` 若要查看如何分布的屏幕空间基于每个元素的权重。请参阅[常见布局对象](#)深入了解如何文档 `LinearLayout` 句柄 `android:layout_weight` 属性。

参考资料

- `LinearLayout`
- `TextView`

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#).

RelativeLayout

2018/10/26 • [Edit Online](#)

`RelativeLayout` 是 `ViewGroup` 显示子 `View` 中的相对位置的元素。位置 `View` 可以指定相对于同级元素（如有关的左侧或给定元素的下面），或在定位相对于 `RelativeLayout` 区域（如如对齐到底部，左侧的中心）。

一个 `RelativeLayout` 是一个非常强大的实用程序，对于设计用户界面，因为它可以消除嵌套 `ViewGroup` s。如果你发现自己使用多个嵌套 `LinearLayout` 组，您可能能够将它们替换成单个 `RelativeLayout`。

启动一个名为的新项目 **HelloRelativeLayout**。

打开 **Resources/Layout/Main.xml** 文件并插入以下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Type here:" />
    <EditText
        android:id="@+id/entry"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label" />
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

请注意，每个 `android:layout_*` 这样的特性 `layout_below`，`layout_alignParentRight`，和 `layout_toLeftOf`。使用时 `RelativeLayout`，可以使用这些属性来描述你想要放置每个 `View`。这些属性的每个定义一种不同的相对位置。某些属性使用的资源 ID 的同级 `View` 来定义其自己的相对位置。例如上，一次 `Button` 定义为位于左侧的和对齐--top-的 `View` 由 ID 标识 `ok` (即以前 `Button`)。

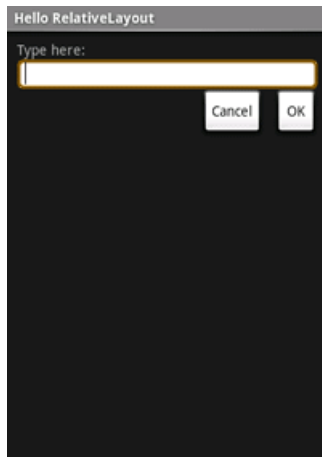
中定义的所有可用的布局属性 `RelativeLayout.LayoutParams`。

请确保加载在此布局 `onCreate()` 方法：

```
protected override void onCreate (Bundle savedInstanceState)
{
    base.OnCreate (savedInstanceState);
    SetContentView (Resource.Layout.Main);
}
```

`SetContentView(int)` 方法加载的布局文件 `Activity`，按资源 ID 指定——`Resource.Layout.Main` 指资源/布局 `/Main.xml` 布局文件。

运行该应用程序。应会看到以下布局：



资源

- `RelativeLayout`
- `RelativeLayout.LayoutParams`
- `TextView`
- `EditText`
- `Button`

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution](#) 许可证。

TableLayout

2018/10/26 • [Edit Online](#)

`TableLayout` 是 `ViewGroup` 显示子 `View` 行和列中的元素。

启动一个名为的新项目**HelloTableLayout**。

打开**Resources/Layout/Main.xml**文件并插入以下：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Open..."
            android:padding="3dip"/>
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
            android:padding="3dip"/>
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save..."
            android:padding="3dip"/>
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip"/>
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save As..."
            android:padding="3dip"/>
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip"/>
    </TableRow>

    <View
        android:layout_height="2dip"
        android:background="#FF909090"/>

    <TableRow>
        <TextView
            android:text="X"
            android:padding="3dip"/>
        <TextView
            android:text="Import..."
            android:padding="3dip"/>
    </TableRow>

    <TableRow>
```

```

        <TextView
            android:text="X"
            android:padding="3dip"/>
        <TextView
            android:text="Export..."
            android:padding="3dip"/>
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip"/>
    </TableRow>

    <View
        android:layout_height="2dip"
        android:background="#FF909090"/>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Quit"
            android:padding="3dip"/>
    </TableRow>
</TableLayout>

```

请注意, 这与 HTML 表的结构的相似。的 `TableLayout` 元素是类似于 HTML `<table>` 元素; `TableRow` 就像 `<tr>` 元素; 但的单元格, 你可以使用任何类型的 `View` 元素。在此示例中, `TextView` 用于每个单元格。在某些行之间还有一个基本 `View`, 用于绘制水平线。

请确保你 **HelloTableLayout** 活动加载在此布局 `onCreate()` 方法:

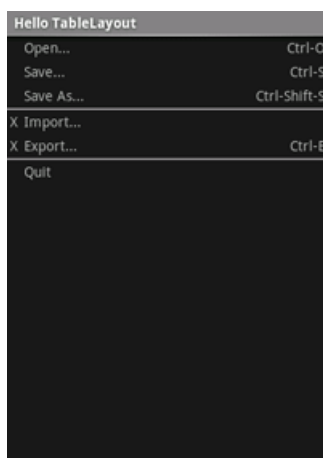
```

protected override void onCreate (Bundle savedInstanceState)
{
    base.onCreate (savedInstanceState);
    setContentView (Resource.Layout.Main);
}

```

`setContentView(int)` 方法加载的布局文件 `Activity`, 按资源 ID 指定——`Resource.Layout.Main` 指资源/布局 **/Main.xml** 布局文件。

运行该应用程序。你应看到以下信息:



参考资料

- `TableLayout`
- `TableRow`

- `TextView`

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution](#) 许可证.

RecyclerView

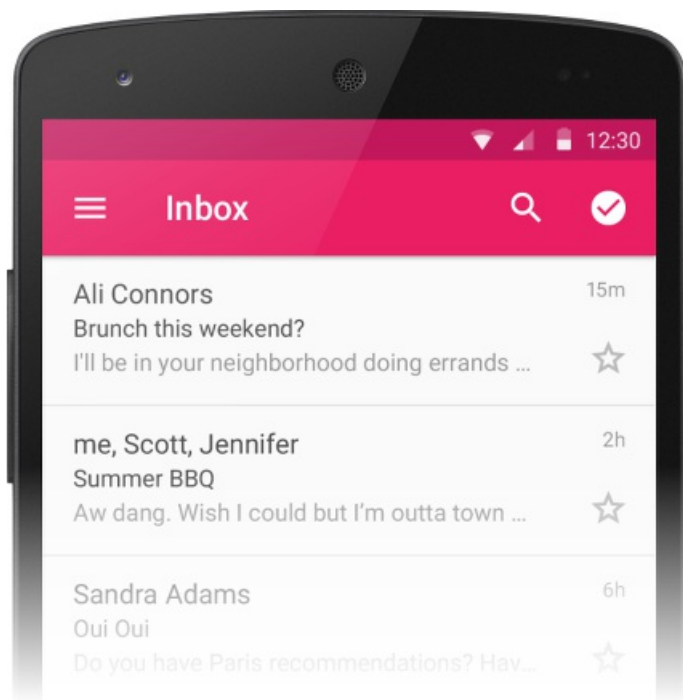
2018/10/26 • [Edit Online](#)

RecyclerView 用于显示集合, 是一组视图它旨在更灵活替代较旧的视图组, 如 ListView 和 GridView。本指南介绍如何使用和自 RecyclerView 定义在 Xamarin.Android 应用程序中。

RecyclerView

许多应用程序需要显示 (如邮件、联系人、图像或歌曲); 具有相同类型的集合通常, 此集合是太大, 无法容纳在屏幕上, 因此集合也会出现在一个小型窗口, 可以顺利地滚动浏览集合中的所有项。RecyclerView 是 Android 小组件显示在列表或网格, 使用户可以滚动浏览集合中项的集合。下面是示例中使用的应用程序的屏幕截图

RecyclerView 可以垂直滚动列表中显示电子邮件收件箱内容:



RecyclerView 提供两个极具吸引力的功能:

- 它具有灵活的体系结构, 以便您可以通过插入在您首选的组件中修改其行为。
- 它是高效地使用较大的集合, 因为它会重用项视图并需要使用 *查看持有者* 到缓存的视图引用。

本指南介绍如何使用 RecyclerView Xamarin.Android 应用程序; 在其中介绍了如何添加 RecyclerView 包到你的 Xamarin.Android 项目, 并介绍了如何 RecyclerView 典型的应用程序中的函数。提供了真实代码示例来演示如何将集成 RecyclerView 到你的应用程序、如何实现项视图, 请单击, 以及如何刷新 RecyclerView 在其基础数据发生更改。本指南假定您熟悉 Xamarin.Android 开发。

要求

尽管 RecyclerView 是通常与 Android 5.0 Lollipop 相关联, 它作为提供支持库-RecyclerView 作用于应用, 该目标 API 级别 (Android 2.1) 7 及更高版本。使用所需的以下 RecyclerView 基于 Xamarin 的应用程序:

- **Xamarin.Android** – Xamarin.Android 4.20 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for mac。
- 应用程序项目必须包括 **Xamarin.Android.Support.v7.RecyclerView** 包。有关安装 NuGet 包的详细信息, 请参阅 [演练: 在项目中包括 NuGet](#)。

概述

`RecyclerView` 可以视为替换 `ListView` 和 `GridView` 在 Android 中的小组件。与其前身一样，`RecyclerView` 设计用于在小型窗口中，显示大型数据集，但 `RecyclerView` 提供更多布局选项和更好地优化用于显示较大的集合。如果你熟悉 `ListView`，有几个重要区别之间 `ListView` 和 `RecyclerView`：

- `RecyclerView` 稍微更复杂，无法使用：您必须编写更多代码以使用 `RecyclerView` 相比 `ListView`。
- `RecyclerView` 不提供的预定义的适配器;必须实现用于访问你的数据源的适配器代码。但是，对于 Android 包含使用的几个预定义的适配器 `ListView` 和 `GridView`。
- `RecyclerView` 不提供项的单击事件时在用户点击项;相反，由帮助器类处理项单击事件。与此相反，`ListView` 提供项的单击事件。
- `RecyclerView` 通过回收视图和通过强制持有者的视图模式，从而消除了不必要的布局资源查找，可增强性能。中的持有者的视图模式的使用是可选 `ListView`。
- `RecyclerView` 基于模块化设计，可简化自定义。例如，您可以在不同的布局策略无需更改大量代码中插入到您的应用程序。与此相反，`ListView` 是相对比较整体化结构中。
- `RecyclerView` 包括内置动画项添加和删除。`ListView` 动画需要应用开发人员来说做一些额外工作。

部分

RecyclerView 部件和功能

本主题介绍如何 `Adapter`，`LayoutManager`，并 `ViewHolder` 共同用作帮助程序类以支持 `RecyclerView`。它提供的每个帮助程序类的高级概述并介绍了如何在应用中使用它们。

基本 RecyclerView 示例

本主题中提供的信息为基础 [RecyclerView 部件和功能](#) 提供真正的代码示例说明了如何通过各种 `RecyclerView` 元素实现以生成实际的照片浏览应用。

扩展 RecyclerView 示例

本主题将额外的代码添加到示例应用程序中显示 [基本 RecyclerView 示例](#) 若要演示如何以处理项的单击事件并更新 `RecyclerView` 当基础数据源发生更改。

总结

本指南介绍了 Android `RecyclerView` 小组件; 本文介绍了如何添加 `RecyclerView` 如何支持到 Xamarin.Android 项目的库 `RecyclerView` 如何强制实施的持有者的视图模式的效率，以及如何回收视图的各种帮助器类，它们组成 `RecyclerView` 开展协作，显示集合。它提供了示例代码演示如何 `RecyclerView` 集成到应用程序，它介绍了如何定制 `RecyclerView` 的布局策略通过插入不同的布局管理器，并介绍了如何处理项单击事件并通知 `RecyclerView` 的数据源更改。

有关详细信息 `RecyclerView`，请参阅 [RecyclerView 类引用](#)。

相关链接

- [RecyclerViewer \(示例\)](#)
- [棒棒糖形简介](#)
- [RecyclerView](#)

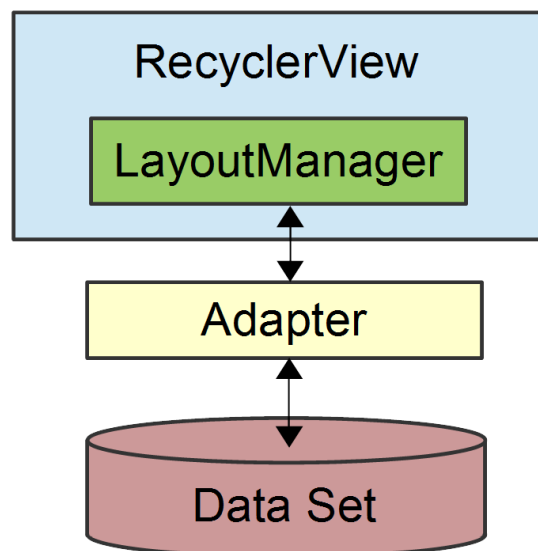
RecyclerView 部件和功能

2018/10/26 • [Edit Online](#)

`RecyclerView` 某些任务在内部（如滚动和回收的视图），但它的句柄是实质上是协调帮助程序类来显示集合的管理器。`RecyclerView` 委托到以下帮助器类的任务：

- `Adapter` – 增大项布局（实例化布局文件的内容），并将数据绑定到视图中显示 `RecyclerView`。适配器还报告项单击事件。
- `LayoutManager` – 度量值并将项中的视图 `RecyclerView` 并管理视图回收的策略。
- `ViewHolder` – 查找并存储视图的引用。视图持有者还有助于检测项目视图下鼠标。
- `ItemDecoration` – 允许应用将特殊的绘制和布局偏移量添加到项目中，突出显示和可视化分组边界之间绘制分隔线的特定视图。
- `ItemAnimator` – 定义动画的项的操作期间发生或进行更改时向适配器。

之间的关系 `RecyclerView`，`LayoutManager`，和 `Adapter` 类在下图中所示：



此图所示，`LayoutManager` 可视为之间的媒介 `Adapter` 和 `RecyclerView`。`LayoutManager` 对进行调用 `Adapter` 代表方法 `RecyclerView`。例如，`LayoutManager` 调用 `Adapter` 方法时就可以创建新视图中的特定项位置 `RecyclerView`。`Adapter` 增大该项的布局，并创建 `ViewHolder` 实例（未显示）来缓存对视图中的该位置的引用。当 `LayoutManager` 调用 `Adapter` 若要将某个特定项绑定到数据集，`Adapter` 定位的项的数据，检索从数据集中，并将其复制到关联的项目视图。

当使用 `RecyclerView` 在您的应用程序，创建以下类的派生的类型是必需的：

- `RecyclerView.Adapter` – 提供从应用程序的数据集（这是特定于您的应用程序）中显示的项视图到的绑定 `RecyclerView`。适配器知道如何将关联中的每个项目视图位置 `RecyclerView` 到数据源中的特定位置。此外，该适配器处理其中每个单独的项视图的内容的布局，并创建每个视图的视图持有者。适配器还报告检测到的项目视图的项的单击事件。
- `RecyclerView.ViewHolder` – 缓存对项布局文件中的视图的引用，以便不会不必要地重复资源查找。查看项单击事件，以在用户点击视图持有者关联的项目视图时将它们转发到适配器还排列视图持有者。

- `RecyclerView.LayoutManager` – 定位中的项 `RecyclerView`。可以使用多个预定义的布局管理器之一，也可以实现自己的自定义布局管理器。`RecyclerView` 委托到布局管理器，因此您可以在不同的布局管理器中而无需进行大量插入布局策略更改为您的应用程序。

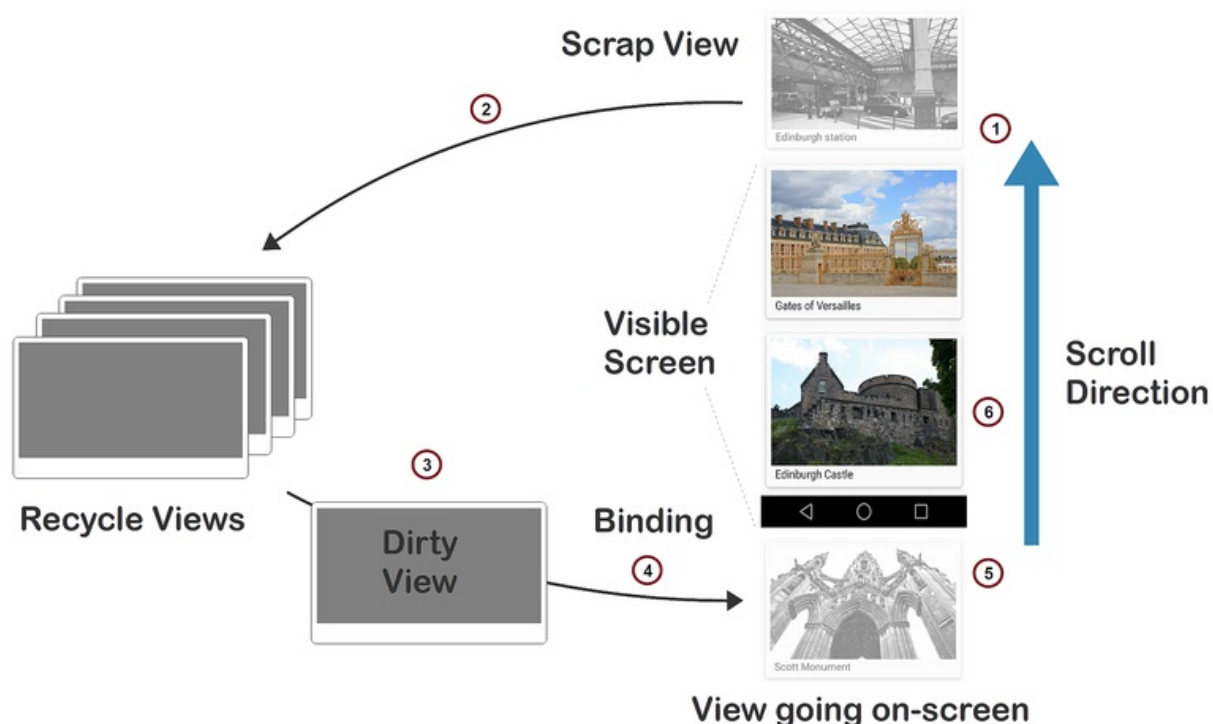
此外，可以选择性地扩展要更改的外观的以下类 `RecyclerView` 应用程序中：

- `RecyclerView.ItemDecoration`
- `RecyclerView.ItemAnimator`

如果不扩展 `ItemDecoration` 并 `ItemAnimator`，`RecyclerView` 使用默认实现。本指南不介绍如何创建自定义 `ItemDecoration` 并 `ItemAnimator` 类；有关这些类的详细信息，请参阅 [RecyclerView.ItemDecoration](#) 和 [RecyclerView.ItemAnimator](#)。

如何查看回收的工作原理

`RecyclerView` 不会为您的数据源中的每个项分配项目视图。相反，它会分配仅在屏幕上显示的项视图的数量，它重用这些项布局，当用户滚动。当不可见的第一次滚动视图时，会经历在下图中所示的回收过程：



1. 当不可见的滚动并不会再显示视图时，它将成为**报废视图**。
2. 片段视图放在池中，并成为**回收视图**。此池是缓存的显示相同类型的数据的视图。
3. 当新项时显示时，则视图取自回收池以供重复使用。因为此视图必须重新绑定适配器在显示之前，调用**脏视图**。
4. 已更新的视图是回收：适配器查找要显示的下一项的数据并将此数据复制到此项目的视图。从与回收的视图关联的视图所有者中检索这些视图的引用。
5. 回收的视图添加到中的项列表 `RecyclerView` 以转到屏幕上。
6. 回收的视图会屏幕上，因为在用户滚动 `RecyclerView` 到列表中的下一项。同时，另一个视图滚动视线之外，会根据上述步骤被回收。

除了项目视图重用 `RecyclerView` 还使用另一个效率优化：查看持有者。一个**视图持有者**是缓存查看引用一个简单类。适配器增大的项布局文件，每次它还会创建相应的视图持有者。视图持有人使用 `FindViewById` 夸大的项布局文件中获取对视图的引用。这些引用用于将新数据加载到视图，每次布局时回收显示新数据。

布局管理器

布局管理器负责定位中的项 `RecyclerView` 显示; 它确定表示类型 (列表或网格)、(是否项都显示垂直或水平) 的方向, 并应显示哪些方向项 (按正常顺序或按相反的顺序)。程序还负责计算的大小和位置中的每个项的布局管理器 `RecyclerView` 显示。

该布局管理器的其他用途: 它会确定何时回收不再对用户可见的项视图的策略。因为布局管理器识别哪些视图是可见 (, 哪些不是), 它是在要确定视图可以回收的最佳位置。若要回收视图, 布局管理器通常会对适配器调用, 以回收视图的内容替换为不同的数据, 如前面所述 [视图回收的工作原理](#)。

您可以扩展 `RecyclerView.LayoutManager` 创建自己的布局管理器中, 也可以使用预定义的布局管理器。

`RecyclerView` 提供了以下预定义的布局管理器:

- `LinearLayoutManager` – 排列项中可垂直滚动的列或行中, 可以水平滚动。
- `GridLayoutManager` – 在网格中显示项。
- `StaggeredGridLayoutManager` – 交错的网格, 其中某些项具有不同高度和宽度中显示的项。

若要指定布局管理器, 实例化所选的布局管理器, 并将其传递给 `SetLayoutManager` 方法。请注意, 您必须指定的布局管理器— `RecyclerView` 不会默认情况下选择一个预定义的布局管理器。

有关布局管理器的详细信息, 请参阅 [RecyclerView.LayoutManager 类引用](#)。

视图持有者

视图持有者是为缓存视图引用定义的类。适配器使用这些视图引用绑定到其内容的每个视图。中的每个项 `RecyclerView` 具有关联的视图持有者实例的缓存项的视图引用。若要创建视图持有者, 使用以下步骤来定义一个类来保存每个项的视图的这组:

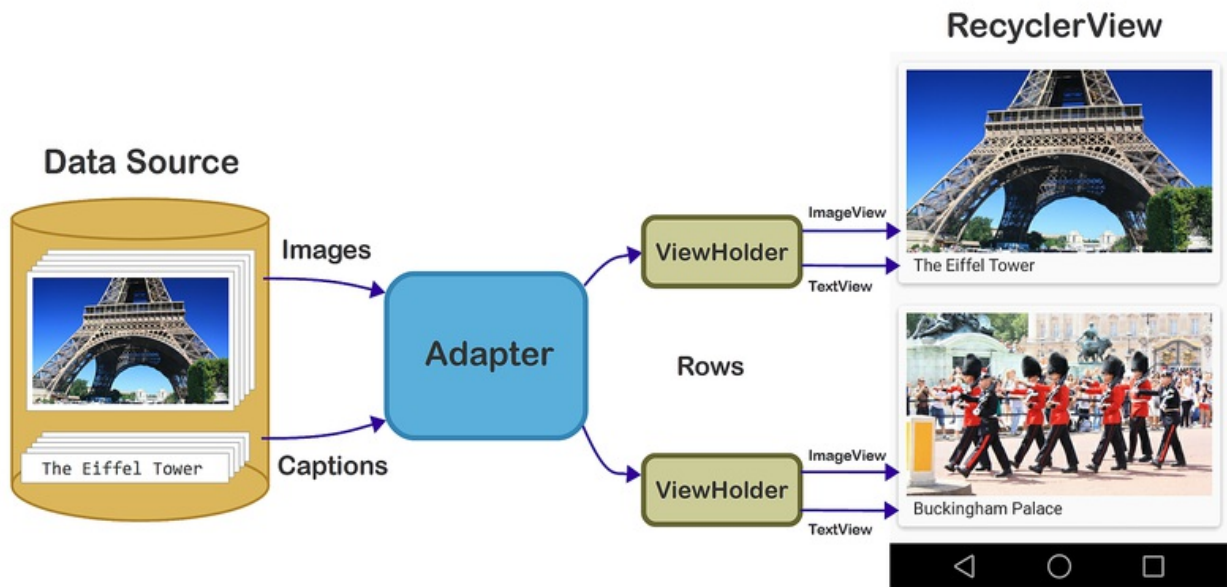
1. 子类 `RecyclerView.ViewHolder`。
2. 实现一个构造函数, 查找并存储视图的引用。
3. 实现该适配器可用于访问这些引用的属性。

详细的示例 `ViewHolder` 实现所示 [基本 RecyclerView 示例](#)。有关详细信息 `RecyclerView.ViewHolder`, 请参阅 [RecyclerView.ViewHolder 类引用](#)。

适配器

大部分"繁琐" `RecyclerView` 集成代码会在适配器中发生。 `RecyclerView` 要求提供适配器派生自 `RecyclerView.Adapter` 访问您的数据源并填充数据源的内容与每个项。由于数据源是特定于应用程序, 必须实现一个知道如何访问你的数据适配器功能。适配器从数据源中提取信息, 并将其加载到每个项 `RecyclerView` 集合。

下图阐释了适配器如何将数据源视图持有人通过中的内容映射到在每个行项中的各个视图 `RecyclerView`:



适配器将加载每个 `RecyclerView` 具有特定行项的数据行。有关行位置 P ，例如，适配器找到位置处的关联的数据 P 行到此数据在位置中的数据源和副本项 P 中 `RecyclerView` 集合。在上面的绘图中，例如，适配器使用视图持有者来查找其引用 `ImageView` 并 `TextView` 中的该位置，使其不包含重复调用 `findViewById` 这些视图作为用户滚动浏览集合和重用视图。

在实现适配器时，必须重写以下 `RecyclerView.Adapter` 方法：

- `onCreateViewHolder` – 实例化项布局文件和视图持有者。
- `onBindViewHolder` – 将指定位置处的数据加载到其引用存储在给定的视图持有者的视图。
- `ItemCount` – 在数据源中返回的项数。

布局管理器调用这些方法，它将定位中的项而 `RecyclerView`。

通知数据更改 `RecyclerView`

`RecyclerView` 不会自动更新其显示时其数据的内容源发生变更；适配器必须通知 `RecyclerView` 数据集中的更改时。数据集可能会更改在许多方面；例如，可以更改其中某个项的内容或数据整体结构可能会更改。

`RecyclerView.Adapter` 提供了多种可以调用的方法，以便 `RecyclerView` 响应数据更改的最高效的方式：

- `notifyItemChanged` – 位于指定位置处的项已更改的信号。
- `notifyItemRangeChanged` – 指定范围中的位置的项已更改的信号。
- `notifyItemInserted` – 发出信号，表明新插入指定位置中的项。
- `notifyItemRangeInserted` – 指定范围中的位置的项已插入新的信号。
- `notifyItemRemoved` – 已删除的指定位置中的项的信号。
- `notifyItemRangeRemoved` – 指定范围中的位置的项已删除的信号。
- `notifyDataSetChanged` – 数据集已更改的信号（强制执行完整的更新）。

如果您知道确切数据集的更改方式，可以调用适当的方法更高版本，若要刷新 `RecyclerView` 最高效的方式。如果不知道确切数据集的更改方式，可以调用 `notifyDataSetChanged`，大大减少效率更高因为 `RecyclerView` 必须刷新所有对用户可见的视图。有关这些方法的详细信息，请参阅 [RecyclerView.Adapter](#)。

在下一主题 [基本 RecyclerView 示例](#)，示例应用程序实现以演示的部件和上面所述的功能的实际代码示例。

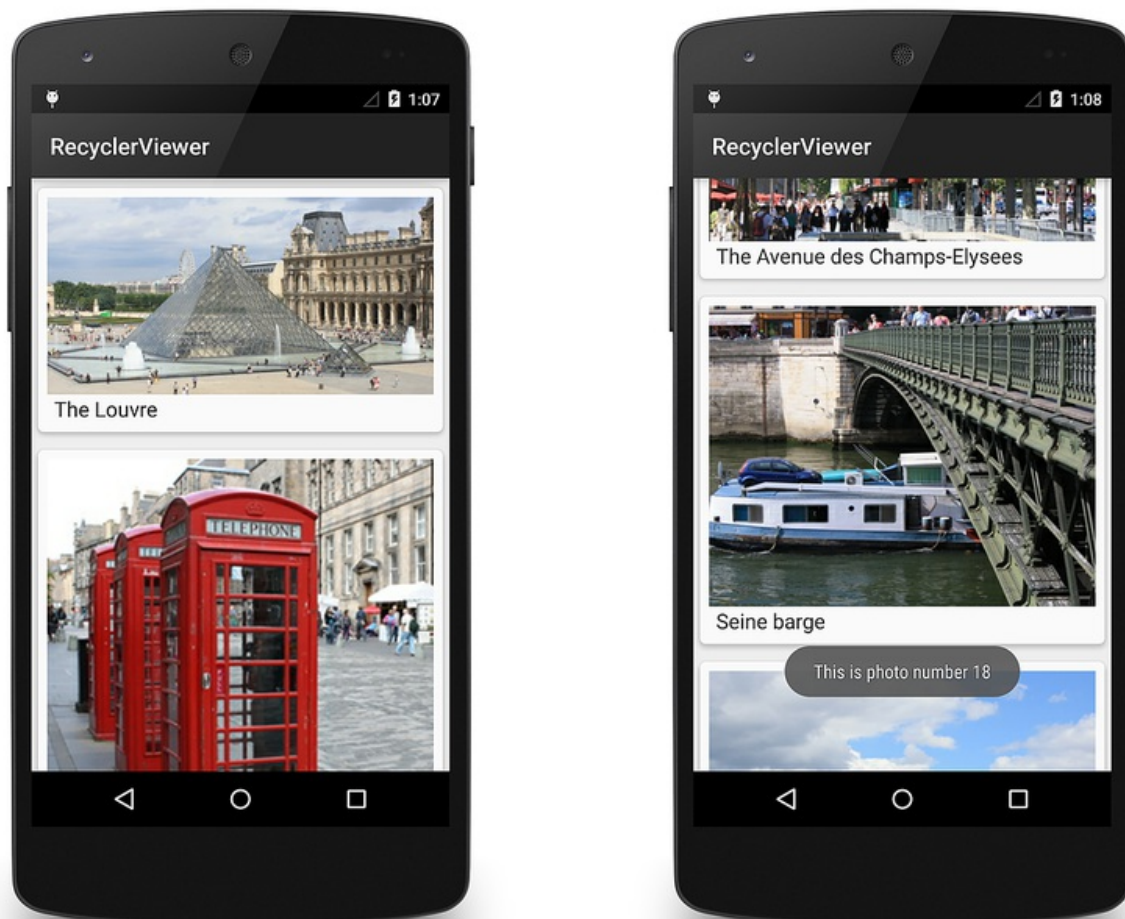
相关链接

- [RecyclerView](#)
- [基本 RecyclerView 示例](#)
- [扩展 RecyclerView 示例](#)
- [RecyclerView](#)

基本 RecyclerView 示例

2018/11/13 • [Edit Online](#)

若要了解如何 `RecyclerView` 典型的应用程序中的工作原理，本主题将探讨 `RecyclerView` 示例应用程序，使用一个简单的代码示例 `RecyclerView` 以显示大量的照片集合：



RecyclerView使用 `CardView` 实现中的每个照片项 `RecyclerView` 布局。由于 `RecyclerView` 的性能优势，此示例应用是能够顺利且不明显延迟的情况下快速滚动大型照片集合。

示例数据源

在此示例应用中，“相册”数据源（由 `PhotoAlbum` 类）提供 `RecyclerView` 项内容。`PhotoAlbum` 是一系列包含隐藏式字幕的照片在实例化它时，将获得现成 32 照片集合：

```
PhotoAlbum mPhotoAlbum = new PhotoAlbum ();
```

在每个照片实例 `PhotoAlbum` 公开的属性使您可以阅读其映像资源 ID `PhotoID`，和它的标题字符串，`Caption`。照片集合被组织方法规定索引器可以访问每张照片。例如，以下代码行访问映像资源 ID 和标题的集合中的第十个照片：

```
int imageId = mPhotoAlbum[9].ImageId;  
string caption = mPhotoAlbum[9].Caption;
```

`PhotoAlbum` 此外提供了 `RandomSwap` 可调用以交换第一张照片集合中与集合中的其他位置的随机选择照片的方

法：

```
mPhotoAlbum.RandomSwap ();
```

因为的实现细节 `PhotoAlbum` 了解与不相关 `RecyclerView`，则 `PhotoAlbum` 此处不提供源代码。向源代码 `PhotoAlbum` 网址 [PhotoAlbum.cs](#) 中 `RecyclerView` 示例应用程序。

布局 and 初始化

布局文件中，**Main.axml**，包含单个 `RecyclerView` 内 `LinearLayout`：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:scrollbars="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
</LinearLayout>
```

请注意，必须使用完全限定名称 **android.support.v7.widget.RecyclerView** 因为 `RecyclerView` 打包在支持库中。

`OnCreate` 方法的 `MainActivity` 初始化此布局，在实例化适配器，并准备基础数据源：

```
public class MainActivity : Activity
{
    RecyclerView mRecyclerView;
    RecyclerView.LayoutManager mLayoutManager;
    PhotoAlbumAdapter mAdapter;
    PhotoAlbum mPhotoAlbum;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Prepare the data source:
        mPhotoAlbum = new PhotoAlbum ();

        // Instantiate the adapter and pass in its data source:
        mAdapter = new PhotoAlbumAdapter (mPhotoAlbum);

        // Set our view from the "main" layout resource:
        SetContentView (Resource.Layout.Main);

        // Get our RecyclerView layout:
        mRecyclerView = FindViewById<RecyclerView> (Resource.Id.recyclerView);

        // Plug the adapter into the RecyclerView:
        mRecyclerView.SetAdapter (mAdapter);
    }
}
```

此代码将执行以下操作：

1. 实例化 `PhotoAlbum` 数据源。
2. 将照片唱片集数据源传递到构造函数的适配器，`PhotoAlbumAdapter`（稍后在本指南中定义）。请注意，它被视为最佳做法将数据源作为参数传递给适配器的构造函数。
3. 获取 `RecyclerView` 从布局。
4. 插入到适配器 `RecyclerView` 实例通过调用 `RecyclerView``SetAdapter` 方法，如上面所示。

布局管理器

中的每项 `RecyclerView` 组成 `CardView`，其中包含照片图像和照片标题（中介绍详细信息[视图持有者](#)下面一节）。预定义 `LinearLayoutManager` 用于每个布局 `CardView` 垂直滚动的排列方式：

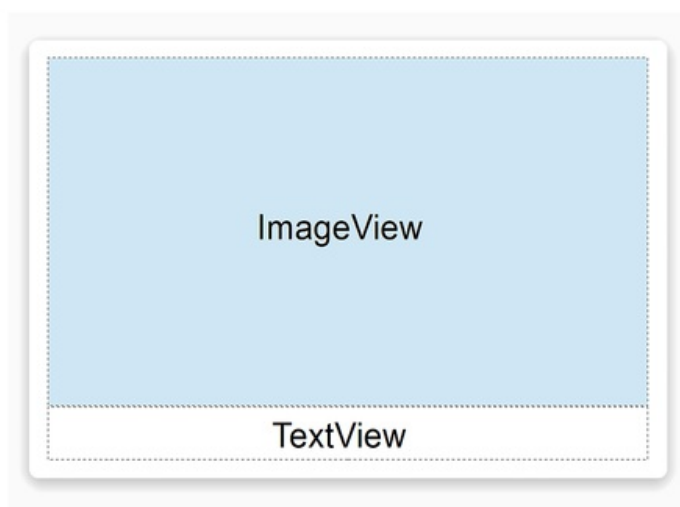
```
mLayoutManager = new LinearLayoutManager (this);
mRecyclerView.setLayoutManager (mLayoutManager);
```

此代码驻留在主活动 `OnCreate` 方法。布局管理器构造函数需要上下文，因此 `MainActivity` 使用传递 `this` 如上图所示。

而不是使用预定义 `LinearLayoutManager`，可以显示两个自定义布局管理器中插入 `CardView` 项的并排方案，实现要遍历的照片集合翻页动画效果。稍后在本指南中，将看到举例说明如何通过交换不同的布局管理器中修改布局。

视图持有者

调用视图持有者类 `PhotoViewHolder`。每个 `PhotoViewHolder` 实例包含对引用 `ImageView` 并 `TextView` 中的布局的相关的行项的 `CardView` 如图解此处：



`PhotoViewHolder` 派生自 `RecyclerView.ViewHolder` 并且包含用于存储对引用的属性 `ImageView` 和 `TextView` 上述布局中所示。`PhotoViewHolder` 包含两个属性和一个构造函数：

```
public class PhotoViewHolder : RecyclerView.ViewHolder
{
    public ImageView Image { get; private set; }
    public TextView Caption { get; private set; }

    public PhotoViewHolder (View itemView) : base (itemView)
    {
        // Locate and cache view references:
        Image = itemView.findViewById<ImageView> (Resource.Id.imageView);
        Caption = itemView.findViewById<TextView> (Resource.Id.textView);
    }
}
```

在此代码示例中，`PhotoViewHolder` 构造函数传递到父项目视图的引用（`CardView`）的 `PhotoViewHolder` 包装。请注意，你始终将转发父项视图到基构造函数。`PhotoViewHolder` 构造函数调用 `findViewById` 对父项视图来查找其子视图引用的每个 `ImageView` 并 `TextView`，将存储中的结果 `Image` 和 `Caption` 属性，分别。适配器更高版本中检索视图引用从这些属性时它会更新此 `CardView` 的子视图使用新数据。

有关详细信息 `RecyclerView.ViewHolder`，请参阅[RecyclerView.ViewHolder 类引用](#)。

适配器

适配器将加载每个 `RecyclerView` 具有特定照片的数据行。为给定行位置处照片 P ，例如，适配器找到位置处的关联的数据 P 行到此数据在位置中的数据源和副本项 P 在 `RecyclerView` 集合。适配器使用视图持有者来查找其引用 `ImageView` 并 `TextView` 中的该位置，使其不包含重复调用 `FindViewById` 的那些视图当用户滚动浏览照片集合和重用视图。

在中 **`RecyclerView`**，一种适配器类派生自 `RecyclerView.Adapter` 若要创建 `PhotoAlbumAdapter`：

```
public class PhotoAlbumAdapter : RecyclerView.Adapter
{
    public PhotoAlbum mPhotoAlbum;

    public PhotoAlbumAdapter (PhotoAlbum photoAlbum)
    {
        mPhotoAlbum = photoAlbum;
    }
    ...
}
```

`mPhotoAlbum` 成员包含传递到构造函数中的数据源（相册）；构造函数将照片唱片集复制到此成员变量。下列必需 `RecyclerView.Adapter` 方法实现：

- `onCreateViewHolder` – 实例化项布局文件和视图持有者。
- `onBindViewHolder` – 将指定位置处的数据加载到其引用存储在给定的视图持有者的视图。
- `getItemCount` – 在数据源中返回的项数。

布局管理器调用这些方法，它将定位中的项而 `RecyclerView`。以下各节中检查这些方法的实现。

`onCreateViewHolder`

布局管理器调用 `onCreateViewHolder` 时 `RecyclerView` 需要新的视图拥有者，代表项目。`onCreateViewHolder` 增大该视图的布局文件从项目视图并将包装在新视图 `PhotoViewHolder` 实例。`PhotoViewHolder` 构造函数查找并存储在布局中的子视图的引用，如前面所述[视图持有者](#)。

每个行项都由 `CardView`，其中包含 `ImageView`（适用于照片）和一个 `TextView`（适用于标题）。此布局文件中驻留 **`PhotoCardView.xml`**：

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <android.support.v7.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        card_view:cardElevation="4dp"
        card_view:cardUseCompatPadding="true"
        card_view:cardCornerRadius="5dp">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="8dp">
            <ImageView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:id="@+id/imageView"
                android:scaleType="centerCrop" />
            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:textColor="#333333"
                android:text="Caption"
                android:id="@+id/textView"
                android:layout_gravity="center_horizontal"
                android:layout_marginLeft="4dp" />
            </LinearLayout>
        </android.support.v7.widget.CardView>
    </FrameLayout>

```

此布局表示中的单个行项目 `RecyclerView`。 `onBindViewHolder` 方法（如下所述）将数据复制到数据源中 `ImageView` 和 `TextView` 的此布局。 `onCreateViewHolder` 增大此布局中的给定的照片位置 `RecyclerView`，并实例化一个新 `PhotoViewHolder` 实例（其定位和缓存对引用 `ImageView` 和 `TextView` 中关联的子视图 `CardView` 布局）：

```

public override RecyclerView.ViewHolder
    onCreateViewHolder (ViewGroup parent, int viewType)
{
    // Inflate the CardView for the photo:
    View itemView = LayoutInflater.From (parent.Context).
        Inflate (Resource.Layout.PhotoCardView, parent, false);

    // Create a ViewHolder to hold view references inside the CardView:
    PhotoViewHolder vh = new PhotoViewHolder (itemView);
    return vh;
}

```

生成的视图持有者实例， `vh`，返回到调用方（布局管理器）。

onBindViewHolder

布局管理器何时可以显示中的特定视图 `RecyclerView` 的屏幕可视区域中，它会调用适配器的 `onBindViewHolder` 方法来填充数据源中的内容的指定的行位置处的项。 `onBindViewHolder` 获取指定的行位置（照片的图像资源和照片的标题的字符串）的照片信息并将此数据复制到关联的视图。视图位于通过引用存储在视图持有者对象（其传递通过 `holder` 参数）：


```

public override void
    OnBindViewHolder (RecyclerView.ViewHolder holder, int position)
{
    PhotoViewHolder vh = holder as PhotoViewHolder;

    // Load the photo image resource from the photo album:
    vh.Image.SetImageResource (mPhotoAlbum[position].PhotoID);

    // Load the photo caption from the photo album:
    vh.Caption.Text = mPhotoAlbum[position].Caption;
}

```

传入的视图持有者对象必须先转换为派生的视图持有者类型 (在这种情况下, `PhotoViewHolder`) 使用它之前。该适配器将图像资源加载到视图持有者的引用的视图 `Image` 属性, 并将标题文本复制到引用的视图持有者的视图 `Caption` 属性。这绑定其数据与关联的视图。

请注意, `OnBindViewHolder` 是直接处理数据的结构的代码。在这种情况下, `OnBindViewHolder` 了解如何将映射 `RecyclerView` 项到其关联的数据项目在数据源中的位置。映射是直接在这种情况下由于位置可用作到相册; 数组索引但是, 更复杂的数据源可能需要额外的代码来建立此类映射。

ItemCount

`ItemCount` 方法返回数据集中项的数目。在示例照片查看器应用中, 项的计数是中的照片相册的照片数:

```

public override int ItemCount
{
    get { return mPhotoAlbum.NumPhotos; }
}

```

有关详细信息 `RecyclerView.Adapter`, 请参阅[RecyclerView.Adapter 类引用](#)。

汇总所有内容

得到 `RecyclerView` 的示例照片应用程序的实现包括 `MainActivity` 创建数据源、布局管理器和适配器的代码。

`MainActivity` 创建 `mRecyclerView` 实例, 实例化的数据源和适配器, 并插入布局管理器和适配器:

```

public class MainActivity : Activity
{
    RecyclerView mRecyclerView;
    RecyclerView.LayoutManager mLayoutManager;
    PhotoAlbumAdapter mAdapter;
    PhotoAlbum mPhotoAlbum;

    protected override void onCreate (Bundle bundle)
    {
        base.onCreate (bundle);
        mPhotoAlbum = new PhotoAlbum();
        SetContentView (Resource.Layout.Main);
        mRecyclerView = FindViewById<RecyclerView> (Resource.Id.recyclerView);

        // Plug in the linear layout manager:
        mLayoutManager = new LinearLayoutManager (this);
        mRecyclerView.SetLayoutManager (mLayoutManager);

        // Plug in my adapter:
        mAdapter = new PhotoAlbumAdapter (mPhotoAlbum);
        mRecyclerView.SetAdapter (mAdapter);
    }
}

```

`PhotoViewHolder` 查找并缓存视图引用:


```

public class PhotoViewHolder : RecyclerView.ViewHolder
{
    public ImageView Image { get; private set; }
    public TextView Caption { get; private set; }

    public PhotoViewHolder (View itemView) : base (itemView)
    {
        // Locate and cache view references:
        Image = itemView.FindViewById<ImageView> (Resource.Id.imageView);
        Caption = itemView.FindViewById<TextView> (Resource.Id.textView);
    }
}

```

PhotoAlbumAdapter 实现三个所需的方法重写：

```

public class PhotoAlbumAdapter : RecyclerView.Adapter
{
    public PhotoAlbum mPhotoAlbum;
    public PhotoAlbumAdapter (PhotoAlbum photoAlbum)
    {
        mPhotoAlbum = photoAlbum;
    }

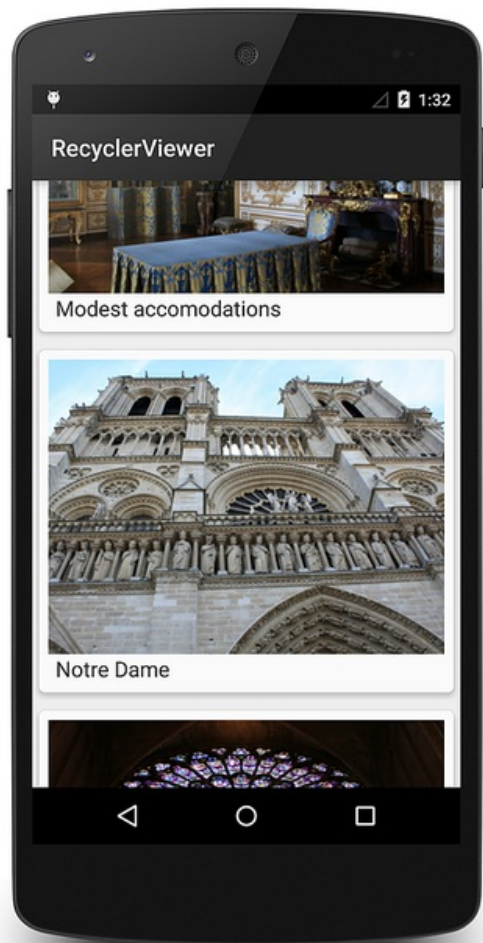
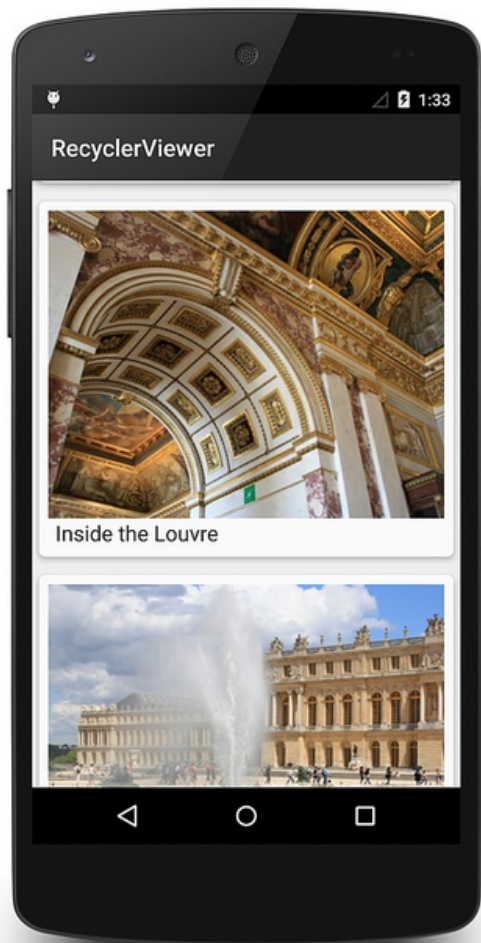
    public override RecyclerView.ViewHolder
        OnCreateViewHolder (ViewGroup parent, int viewType)
    {
        View itemView = LayoutInflater.From (parent.Context).
            Inflate (Resource.Layout.PhotoCardView, parent, false);
        PhotoViewHolder vh = new PhotoViewHolder (itemView);
        return vh;
    }

    public override void
        OnBindViewHolder (RecyclerView.ViewHolder holder, int position)
    {
        PhotoViewHolder vh = holder as PhotoViewHolder;
        vh.Image.SetImageResource (mPhotoAlbum[position].PhotoID);
        vh.Caption.Text = mPhotoAlbum[position].Caption;
    }

    public override int getItemCount
    {
        get { return mPhotoAlbum.NumPhotos; }
    }
}

```

编译并运行此代码将创建基本的照片查看应用程序，如以下屏幕截图中所示：



如果（如上面的屏幕截图中所示），将不会绘制阴影，编辑`properties/AndroidManifest.xml`并添加以下属性设置为 `<application>` 元素：

```
android:hardwareAccelerated="true"
```

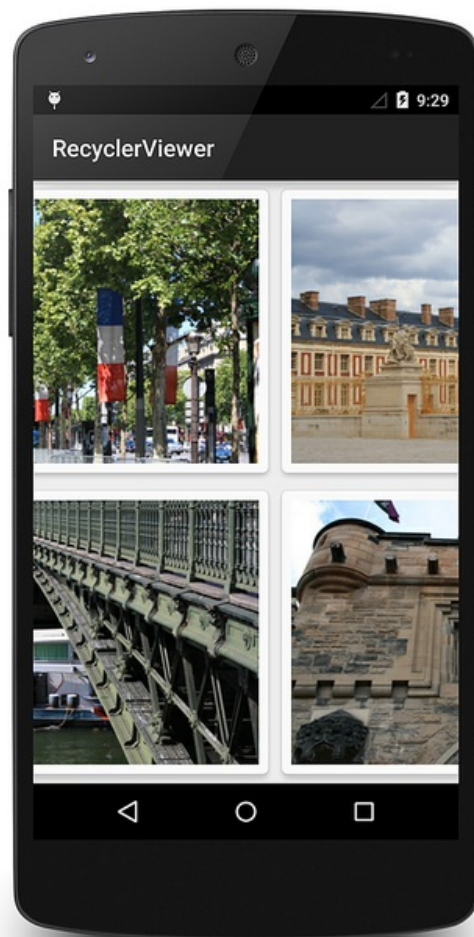
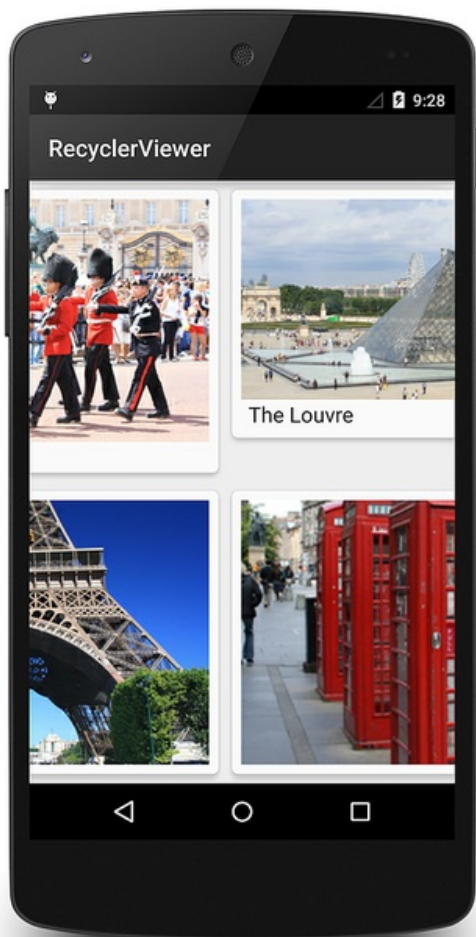
此基本应用仅支持浏览照片相册。它不响应以项的触摸事件，也不会处理基础数据中的更改。此功能中添加[扩展 RecyclerView 示例](#)。

更改 `LayoutManager`

由于 `RecyclerView` 的灵活性，很容易地修改应用程序以使用不同的布局管理器。在下面的示例，它修改为显示具有水平滚动的网格布局而不是具有垂直线性布局上的相册。若要执行此操作，布局管理器实例化改为使用 `GridLayoutManager`，如下所示：

```
mLayoutManager = new GridLayoutManager(this, 2, GridLayoutManager.Horizontal, false);
```

此代码更改取代了垂直 `LinearLayoutManager` 与 `GridLayoutManager` 这将显示在水平方向上滚动的两个行组成的网格。如果编译并再次运行该应用，将看到一个网格中显示的照片和滚动是水平，而不是垂直：



通过更改只有一行代码，则可以修改照片查看应用程序以不同的行为与使用不同的布局。请注意，适配器代码既不布局 XML 必须进行修改以更改布局样式。

在下一主题[扩展 RecyclerView 示例](#)，此基本示例应用程序扩展以处理项的单击事件并更新 `RecyclerView` 当基础数据源发生更改。

相关链接

- [RecyclerView \(示例\)](#)
- [RecyclerView](#)
- [RecyclerView 部件和功能](#)
- [扩展 RecyclerView 示例](#)
- [RecyclerView](#)

扩展 RecyclerView 示例

2018/10/26 • [Edit Online](#)

基本应用程序中所述[基本 RecyclerView 示例](#)实际上并未执行其他操作-它只需将滚动，并显示照片项，以方便浏览的固定的列表。在实际应用程序中，用户希望能够通过点击中显示的项与应用程序进行交互。此外，可以更改基础数据源（或由应用更改），并显示的内容必须保持与这些更改相一致。在以下部分中，您将学习如何以处理项的单击事件并更新 `RecyclerView` 当基础数据源发生更改。

处理项单击事件

当用户触摸中的项 `RecyclerView`，生成的项的单击事件以通知项已使用哪种应用。此事件不由生成 `RecyclerView` -相反，项视图（这包装在视图持有者）检测到收尾工作了，并报告这些收尾工作了单击事件。

为了说明如何处理项单击事件，以下步骤介绍如何向报表有哪些照片已使用过的用户修改基本的照片查看应用。示例应用程序中的项的单击事件时，按以下顺序发生：

1. 照片的 `CardView` 检测项 click 事件，并向适配器通知。
2. 该适配器将（具有项的位置信息）的事件转发到活动的项的单击处理程序。
3. 活动的项的单击处理程序对应项的单击事件。

首先，调用事件处理程序成员 `ItemClick` 添加到 `PhotoAlbumAdapter` 类定义：

```
public event EventHandler<int> ItemClick;
```

接下来，将一个项的单击事件处理程序方法添加到 `MainActivity`。此处理程序会短暂地显示 toast 通知，指示已使用哪些照片项：

```
void OnItemClick (object sender, int position)
{
    int photoNum = position + 1;
    Toast.MakeText(this, "This is photo number " + photoNum, ToastLength.Short).Show();
}
```

接下来，需要一行代码来注册 `OnItemClick` 处理程序替换 `PhotoAlbumAdapter`。执行此操作的好时机是后立即 `PhotoAlbumAdapter` 创建：

```
mAdapter = new PhotoAlbumAdapter (mPhotoAlbum);
mAdapter.ItemClick += OnItemClick;
```

在此基本示例中，处理程序注册会在主活动中的发生 `OnCreate` 方法，但生产应用程序可能注册中的处理程序 `OnResume` 并将其在注销 `OnPause` -请参阅[活动生命周期](#)有关详细信息。

`PhotoAlbumAdapter` 现在，将调用 `OnItemClick` 当它收到的项的单击事件。下一步是在引发此适配器中创建一个处理程序 `ItemClick` 事件。下面的方法， `OnClick`，紧跟在其后适配器的添加 `ItemCount` 方法：

```
void OnClick (int position)
{
    if (ItemClick != null)
        ItemClick (this, position);
}
```

这 `OnClick` 方法是适配器 `侦听器` 从项视图项的单击事件。项视图 (通过项目视图的视图持有者), 可以注册此侦听器之前 `PhotoViewHolder` 构造函数, 必须修改以接受作为附加参数, 此方法并注册 `OnClick` 与项目视图 `Click` 事件。下面是修改后 `PhotoViewHolder` 构造函数:

```
public PhotoViewHolder (View itemView, Action<int> listener)
    : base (itemView)
{
    Image = itemView.FindViewById<ImageView> (Resource.Id.imageView);
    Caption = itemView.FindViewById<TextView> (Resource.Id.textView);

    itemView.Click += (sender, e) => listener (base.LayoutPosition);
}
```

`itemView` 参数包含对引用 `CardView`, 已使用过的用户。请注意视图持有者基类知道项的布局位置 (`CardView`), 它表示 (通过 `LayoutPosition` 属性), 并且此位置传递给适配器的 `OnClick` 发生某项的单击事件的方法。该适配器 `OnCreateViewHolder` 方法修改传递适配器的 `OnClick` 视图所有者的构造函数的方法:

```
PhotoViewHolder vh = new PhotoViewHolder (itemView, OnClick);
```

现在当生成并运行示例照片查看应用, 点击显示在照片将导致 toast 通知显示报告已使用的照片:



此示例演示一种方法用于实现事件处理程序替换 `RecyclerView`。无法在此处使用的另一种方法是将事件放在视图持有者并让订阅这些事件的适配器。如果示例照片应用程序提供照片编辑功能, 将需要单独的事件 `ImageView` 并 `TextView` 在每个 `CardView`: 涉及 `TextView` 在启动时 `EditView` 使用户可以编辑的对话框标题和涉及 `ImageView` 在启动时使用户可以裁剪或旋转照片的照片 touchup 工具。具体取决于您的应用程序的需要, 您必须设计用于处理

和响应触摸事件的最佳做法。

若要演示如何 `RecyclerView` 时可以修改数据集发生更改，示例照片查看应用程序以随机数据源中选择一张照片，并与第一张照片，将其切换可更新。首先，随机选取按钮添加到示例照片应用 **Main.axml** 布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/randPickButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Random Pick" />
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:scrollbars="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
</LinearLayout>
```

接下来，代码将添加到主活动的末尾 `onCreate` 方法查找 `Random Pick` 按钮布局，并将一个处理程序附加到它：

```
Button randomPickBtn = findViewById<Button>(Resource.Id.randPickButton);

randomPickBtn.Click += delegate
{
    if (mPhotoAlbum != null)
    {
        // Randomly swap a photo with the first photo:
        int idx = mPhotoAlbum.RandomSwap();
    }
};
```

此处理程序调用照片相册 `RandomSwap` 方法时随机选取点击按钮。 `RandomSwap` 方法随机交换数据源中的第一个照片的照片，然后返回随机交换照片的索引。当编译和运行此代码示例应用时，点击随机选取因为，按钮不会导致显示更改 `RecyclerView` 并不知道对数据源的更改。

要保留 `RecyclerView` 后的数据源的更改，更新随机选取单击处理程序必须进行修改，以调用适配器的 `NotifyItemChanged` 方法的已更改的集合中每个项（在这种情况下，两个项具有更改：第一张照片和交换的照片）。这将导致 `RecyclerView` 以更新其显示，以便与数据源的新状态保持一致：

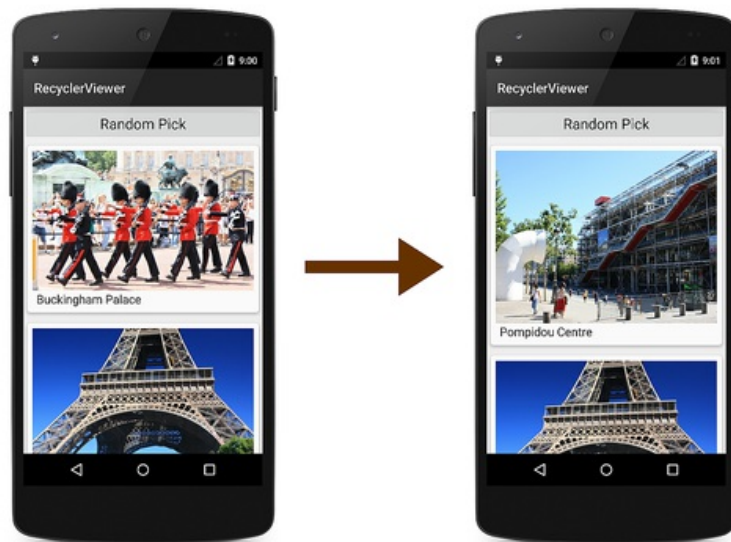

```
Button randomPickBtn = findViewById<Button>(Resource.Id.randPickButton);

randomPickBtn.Click += delegate
{
    if (mPhotoAlbum != null)
    {
        int idx = mPhotoAlbum.RandomSwap();

        // First photo has changed:
        mAdapter.NotifyItemChanged(0);

        // Swapped photo has changed:
        mAdapter.NotifyItemChanged(idx);
    }
};
```

现在，当随机选取点击按钮，`RecyclerView` 更新显示器以显示，照片进一步向下集合中具有尚未被替换集合中第一张照片：



当然 `NotifyDataSetChanged` 可能已调用而不是使两个对 `NotifyItemChanged`，但这样做因此将强制 `RecyclerView` 刷新整个集合，即使集合中的只有两个项已更改。调用 `NotifyItemChanged` 是相对于调用变得更加有效 `NotifyDataSetChanged`。

相关链接

- [RecyclerView \(示例\)](#)
- [RecyclerView](#)
- [RecyclerView 部件和功能](#)
- [基本 RecyclerView 示例](#)
- [RecyclerView](#)

ListView

2018/10/26 • [Edit Online](#)

ListView 是 Android 应用程序 的一个重要的 UI 组件它用于无处不在从菜单选项的短列表到联系人或 internet 收藏夹的长列表。它提供了显示的行，可以使用内置样式格式化，也可以广泛地自定义滚动列表的简单方法。

概述

列表视图和适配器包含在 Android 应用程序的最基本的构建基块。`ListView` 类提供了灵活的方式来呈现数据是否短菜单或长的滚动列表。它提供可用性功能，如快速滚动，索引和单个或多个选择，以帮助您构建您的应用程序便于移动访问的用户界面。`ListView` 实例需要 *适配器*，以向它馈送行视图中包含的数据。

本指南介绍如何实现 `ListView` 和各种 `Adapter` 在 Xamarin.Android 中的类。它还演示了如何自定义的外观 `ListView`，并讨论了行重复使用以减少内存消耗的重要性。此外，还有一些活动生命周期的影响方式的讨论 `ListView` 和 `Adapter` 使用。如果您正在从事跨平台应用程序通过 Xamarin.iOS `ListView` 控件是结构上类似于 iOS `UITableView` (和 Android `Adapter` 类似于 `UITableViewSource`)。

首先，简短的教程介绍了 `ListView` 一个基本代码的示例。接下来，提供更高级的主题的链接可帮助你使用 `ListView` 中实际的应用程序。

NOTE

`RecyclerView` 小组件是更高级、更灵活版本 `ListView`。因为 `RecyclerView` 设计为的后继 `ListView` (和 `GridView`)，我们建议你使用 `RecyclerView` 而非 `ListView` 新的应用程序开发。有关详细信息，请参阅 [RecyclerView](#)。

ListView 教程

`ListView` 是 `ViewGroup` 创建可滚动项的列表。列表项会自动插入到列表使用 `IListAdapter`。

在本教程中，将创建从一个字符串数组，读取的国家/地区名称的可滚动列表。选择列表项后的 toast 消息将显示在列表中项的位置。

启动一个名为的新项目 **HelloListView**。

创建一个名为 XML 文件 **list_item.xml** 并将其内部保存资源/布局/ 文件夹。插入以下：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp">
</TextView>
```

此文件定义将被放入每个项的布局 `ListView`。

打开 `MainActivity.cs` 并修改类来扩展 `ListActivity` (而不是 `Activity`):

```
public class MainActivity : ListActivity
{
```


插入下面的代码 `OnCreate()` 方法：

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    ListAdapter = new ArrayAdapter<string> (this, Resource.Layout.list_item, countries);

    ListView.TextFilterEnabled = true;

    ListView.ItemClick += delegate (object sender, AdapterView.ItemClickEventArgs args)
    {
        Toast.MakeText(Application, ((TextView)args.View).Text, ToastLength.Short).Show();
    };
}
```

请注意，这不会为该活动加载布局文件（通常执行此操作与 `SetContentView(int)` ）。相反，设置 `ListAdapter` 将自动添加属性 `ListView` 若要填充的整个屏幕 `ListActivity` 。此方法采用 `ArrayAdapter<T>` ，用于管理将被放入的列表项的数组 `ListView` 。的 `ArrayAdapter<T>` 构造函数采用应用程序 `Context` ，每个列表项（在上一步中创建）的布局说明和一个 `T[]` 或 `Java.Util.IList<T>` 在插入对象的数组 `ListView` （下一步定义的）。

的 `TextFilterEnabled` 属性启用筛选的文本 `ListView` ，以便当用户开始键入时，将筛选列表。

的 `ItemClick` 可以使用事件订阅的单击处理程序。中的项 `ListView` 是单击，调用该处理程序和一个 `Toast` 显示消息，使用从单击项的文本。

可以使用而不是定义您自己的布局文件的平台提供的列表项设计 `ListAdapter` 。例如，请尝试使用

`Android.Resource.Layout.SimpleListItem1` 而不是 `Resource.Layout.list_item` 。

以下代码添加到 `using` 语句：

```
using System;
```

接下来，将以下字符串数组添加为成员 `MainActivity`：

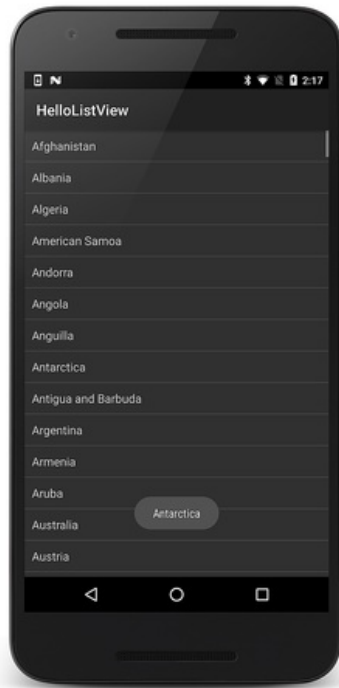
```

static readonly string[] countries = new String[] {
    "Afghanistan","Albania","Algeria","American Samoa","Andorra",
    "Angola","Anguilla","Antarctica","Antigua and Barbuda","Argentina",
    "Armenia","Aruba","Australia","Austria","Azerbaijan",
    "Bahrain","Bangladesh","Barbados","Belarus","Belgium",
    "Belize","Benin","Bermuda","Bhutan","Bolivia",
    "Bosnia and Herzegovina","Botswana","Bouvet Island","Brazil","British Indian Ocean Territory",
    "British Virgin Islands","Brunei","Bulgaria","Burkina Faso","Burundi",
    "Cote d'Ivoire","Cambodia","Cameroon","Canada","Cape Verde",
    "Cayman Islands","Central African Republic","Chad","Chile","China",
    "Christmas Island","Cocos (Keeling) Islands","Colombia","Comoros","Congo",
    "Cook Islands","Costa Rica","Croatia","Cuba","Cyprus","Czech Republic",
    "Democratic Republic of the Congo","Denmark","Djibouti","Dominica","Dominican Republic",
    "East Timor","Ecuador","Egypt","El Salvador","Equatorial Guinea","Eritrea",
    "Estonia","Ethiopia","Faeroe Islands","Falkland Islands","Fiji","Finland",
    "Former Yugoslav Republic of Macedonia","France","French Guiana","French Polynesia",
    "French Southern Territories","Gabon","Georgia","Germany","Ghana","Gibraltar",
    "Greece","Greenland","Grenada","Guadeloupe","Guam","Guatemala","Guinea","Guinea-Bissau",
    "Guyana","Haiti","Heard Island and McDonald Islands","Honduras","Hong Kong","Hungary",
    "Iceland","India","Indonesia","Iran","Iraq","Ireland","Israel","Italy","Jamaica",
    "Japan","Jordan","Kazakhstan","Kenya","Kiribati","Kuwait","Kyrgyzstan","Laos",
    "Latvia","Lebanon","Lesotho","Liberia","Libya","Liechtenstein","Lithuania","Luxembourg",
    "Macau","Madagascar","Malawi","Malaysia","Maldives","Mali","Malta","Marshall Islands",
    "Martinique","Mauritania","Mauritius","Mayotte","Mexico","Micronesia","Moldova",
    "Monaco","Mongolia","Montserrat","Morocco","Mozambique","Myanmar","Namibia",
    "Nauru","Nepal","Netherlands","Netherlands Antilles","New Caledonia","New Zealand",
    "Nicaragua","Niger","Nigeria","Niue","Norfolk Island","North Korea","Northern Marianas",
    "Norway","Oman","Pakistan","Palau","Panama","Papua New Guinea","Paraguay","Peru",
    "Philippines","Pitcairn Islands","Poland","Portugal","Puerto Rico","Qatar",
    "Reunion","Romania","Russia","Rwanda","Sgo Tome and Principe","Saint Helena",
    "Saint Kitts and Nevis","Saint Lucia","Saint Pierre and Miquelon",
    "Saint Vincent and the Grenadines","Samoa","San Marino","Saudi Arabia","Senegal",
    "Seychelles","Sierra Leone","Singapore","Slovakia","Slovenia","Solomon Islands",
    "Somalia","South Africa","South Georgia and the South Sandwich Islands","South Korea",
    "Spain","Sri Lanka","Sudan","Suriname","Svalbard and Jan Mayen","Swaziland","Sweden",
    "Switzerland","Syria","Taiwan","Tajikistan","Tanzania","Thailand","The Bahamas",
    "The Gambia","Togo","Tokelau","Tonga","Trinidad and Tobago","Tunisia","Turkey",
    "Turkmenistan","Turks and Caicos Islands","Tuvalu","Virgin Islands","Uganda",
    "Ukraine","United Arab Emirates","United Kingdom",
    "United States","United States Minor Outlying Islands","Uruguay","Uzbekistan",
    "Vanuatu","Vatican City","Venezuela","Vietnam","Wallis and Futuna","Western Sahara",
    "Yemen","Yugoslavia","Zambia","Zimbabwe"
};

```

这是将被放入的字符串数组 `ListView` 。

运行该应用程序。可以滚动列表, 或键入以筛选它, 然后单击要看到一条消息的项。将显示如下所示的内容:



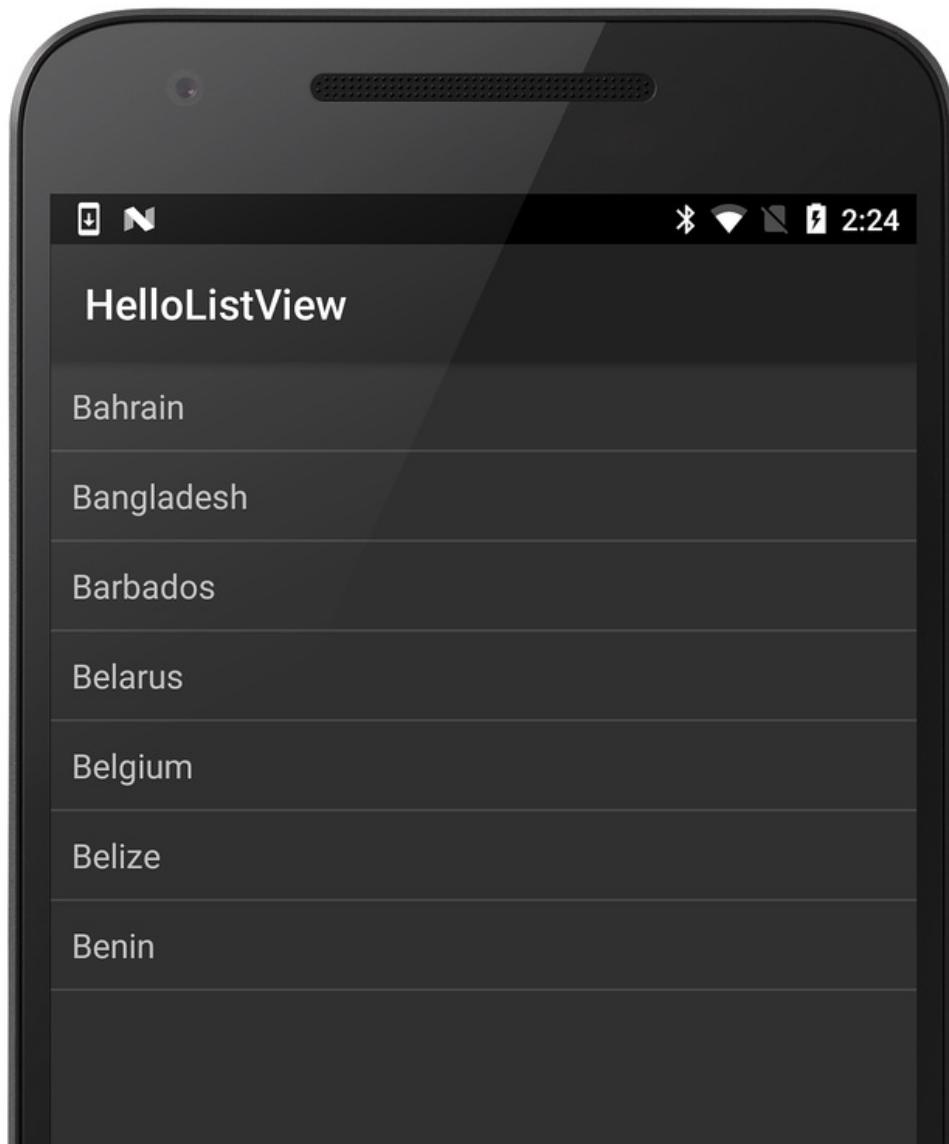
请注意，使用硬编码的字符串数组不是最佳的设计方案。一个用于为简单起见，本教程中演示 `ListView` 小组件。更好的做法是定义的外部资源，例如，使用一个字符串数组中引用 `string-array` 在项目中的资源 `Resources/Values/Strings.xml` 文件。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">HelloListView</string>
  <string-array name="countries_array">
    <item>Bahrain</item>
    <item>Bangladesh</item>
    <item>Barbados</item>
    <item>Belarus</item>
    <item>Belgium</item>
    <item>Belize</item>
    <item>Benin</item>
  </string-array>
</resources>
```

若要使用这些资源字符串 `ArrayAdapter`，替换原始 `ListAdapter` 使用以下行：

```
string[] countries = Resources.GetStringArray (Resource.Array.countries_array);
ListAdapter = new ArrayAdapter<string> (this, Resource.Layout.list_item, countries);
```

运行该应用程序。将显示如下所示的内容：



进一步深入了解 ListView

(链接见下) 的其余主题需要全面了解如何使用 `ListView` 类以及可用于的适配器类型的不同类型。该结构如下所示:

- 可视外观-组成部分 `ListView` 控件以及它们如何工作。
- 类-用来显示类概述 `ListView`。
- 在 `ListView` 中显示数据-如何显示的数据的简单列表; 如何实现 `ListView's` 可用性功能; 如何使用不同的内置行布局; 以及适配器如何通过重复使用行视图节省内存。
- 自定义外观-的样式更改 `ListView` 与自定义布局、字体和颜色。
- 使用 `SQLite` -如何显示与 `SQLite` 数据库中的数据 `CursorAdapter`。
- 活动生命周期-时实现的设计注意事项 `ListView` 活动, 包括其中生命周期中应填充你的数据以及何时释放资源。

讨论 (分解为六个部分) 开头的概述 `ListView` 类本身之前介绍的如何使用它以渐进方式更复杂的示例。

- [ListView 部件和功能](#)
- [填充 ListView 的数据](#)

- [自定义 ListView 的外观](#)
- [使用 CursorAdapters](#)
- [使用 ContentProvider](#)
- [ListView 和活动生命周期](#)

总结

此系列主题引入 `ListView` 并提供如何使用的内置功能的一些示例 `ListActivity`。它介绍了的自定义实现 `ListView` 允许的色彩缤纷的布局并使用 SQLite 数据库，并简要谈及活动生命周期的相关性你 `ListView` 实现。

相关链接

- [AccessoryViews \(示例\)](#)
- [BasicTableAndroid \(示例\)](#)
- [BasicTableAdapter \(示例\)](#)
- [BuiltInViews \(示例\)](#)
- [CustomRowView \(示例\)](#)
- [FastScroll \(示例\)](#)
- [SectionIndex \(示例\)](#)
- [SimpleCursorTableAdapter \(示例\)](#)
- [CursorTableAdapter \(示例\)](#)
- [活动生命周期教程](#)
- [使用表和单元格 \(在 Xamarin.iOS\)](#)
- [ListView 类引用](#)
- [ListActivity 类引用](#)
- [BaseAdapter 类引用](#)
- [ArrayAdapter 类引用](#)
- [CursorAdapter 类引用](#)

ListView 部件和功能

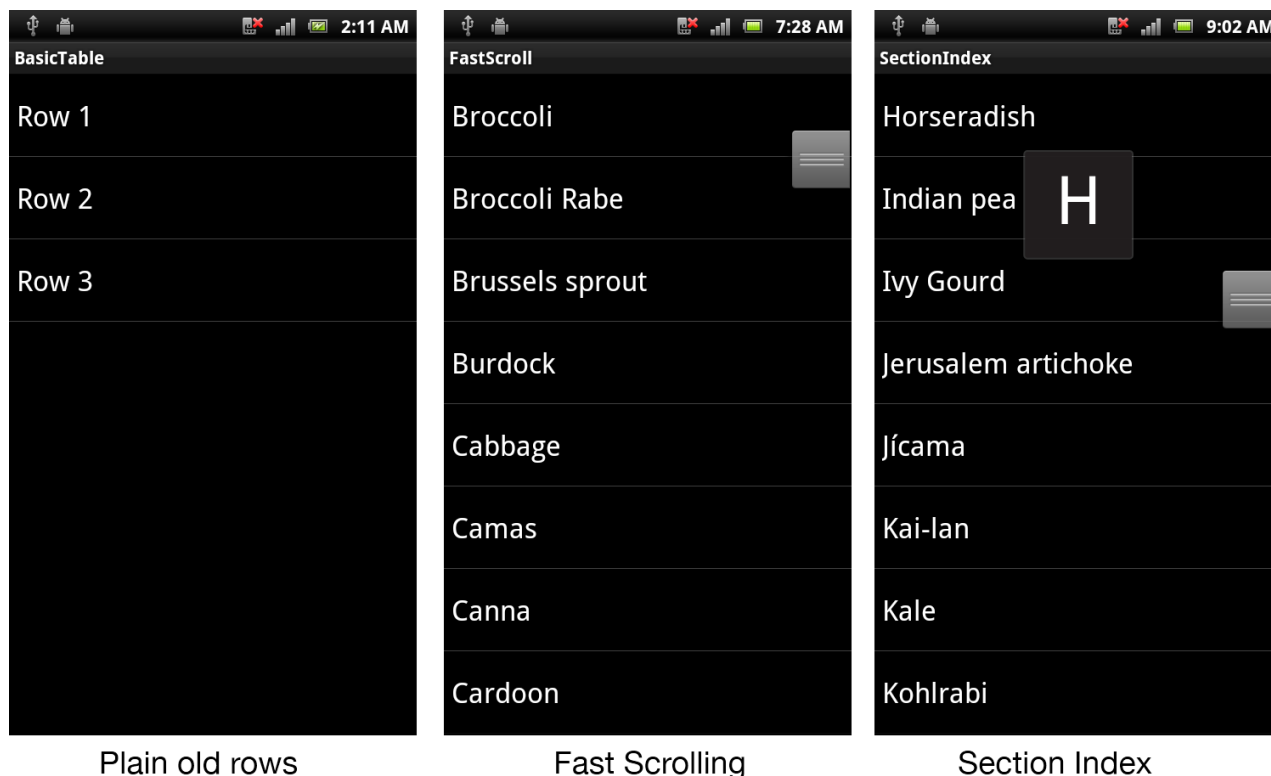
2018/10/26 • [Edit Online](#)

概述

一个 `ListView` 由以下几个部分组成：

- 行-列表中的数据的数据的可视表示形式。
- 适配器-将数据源绑定到列表视图的非可视类。
- 快速滚动-使用户可以滚动列表的长度的句柄。
- 部分索引-通过滚动浮动的用户界面元素以指示列表中的当前行的位置的行。

这些屏幕截图使用基本 `ListView` 控件以显示快速滚动和部分索引的呈现方式：



构成元素 `ListView` 下面更详细地介绍：

行

每一行都具有其自己 `View`。该视图可以是内置在中定义的视图之一 `Android.Resources`，或自定义视图。每个行可以使用相同的视图布局或它们可以各不相同。使用内置的布局和其他解释了如何定义自定义布局的本文档中有示例。

适配器

`ListView` 控件需要 `Adapter` 提供带格式 `View` 每个行。Android 具有内置适配器和视图，可以使用，也可以创建自定义类。

快速滚动

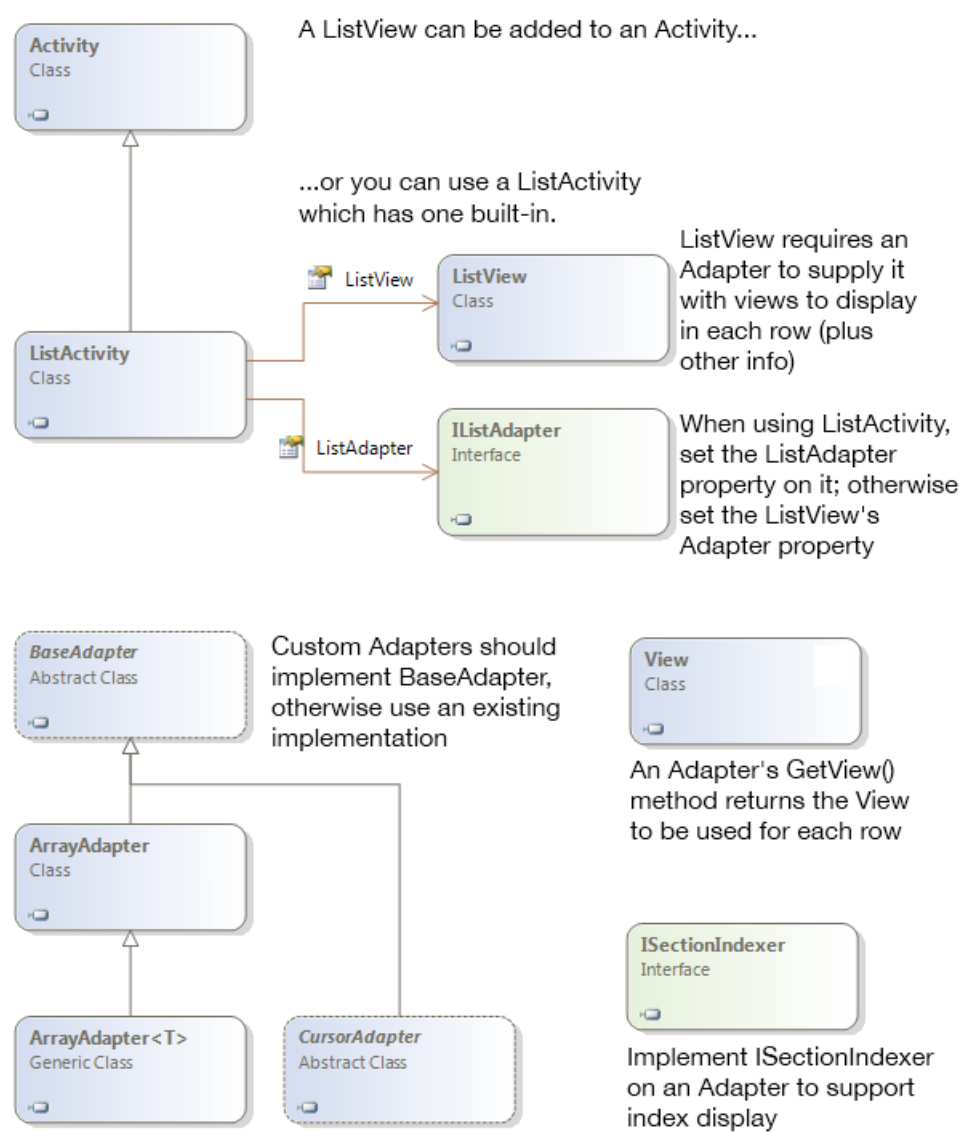
当 `ListView` 包含许多行的数据快速滚动可以启用，以帮助用户导航到列表的任何部分。快速滚动滚动条可以根据需要启用（和自定义在 API 级别 11 和更高版本）。

部分索引

在滚动较长的列表时，可选部分索引提供反馈的用户列表的哪一部分它们当前正在查看。它只是适用于长列表，通常在与快速滚动一起使用。

类概述

用于显示的主要类 `ListView` 如下所示：



每个类的用途如下所述：

- **`ListView`** –显示可滚动的行集合的用户界面元素。在手机它通常将占用整个屏幕（在这种情况下，`ListActivity` 类可用于）也可能是在手机或平板电脑设备上可查看大布局的一部分。
- 视图–在 Android 中的视图可以是任何用户界面元素，但上下文中的 `ListView` 它需要 `View` 提供每个行。
- **`BaseAdapter`** –基类的适配器实现绑定 `ListView` 到数据源。
- **`ArrayAdapter`** –内置适配器类，将绑定到字符串数组 `ListView` 进行显示。泛型 `ArrayAdapter<T>` 对于其他类型执行相同的操作。
- **`CursorAdapter`** –使用 `CursorAdapter` 或 `SimpleCursorAdapter` 显示基于 SQLite 的查询的数据。

本文档包含使用的简单示例 `ArrayAdapter` 以及更复杂的示例需要的自定义实现 `BaseAdapter` 或 `CursorAdapter` 。

填充 ListView 的数据

2018/10/26 • [Edit Online](#)

概述

若要将行添加到 `ListView` 你需要将其添加到你的布局并实现 `ListAdapter` 方法的 `ListView` 调用来填充本身。Android 包括内置 `ListActivity` 和 `ArrayAdapter` 可以不用定义任何自定义布局 XML 或代码的情况下使用的类。`ListActivity` 类会自动创建 `ListView`，并公开 `ListAdapter` 提供行视图以显示通过适配器的属性。

内置适配器采用视图资源 ID 作为参数，获取用于每个行。你可以使用内置的资源，如 `Android.Resource.Layout` 以便您无需编写您自己。

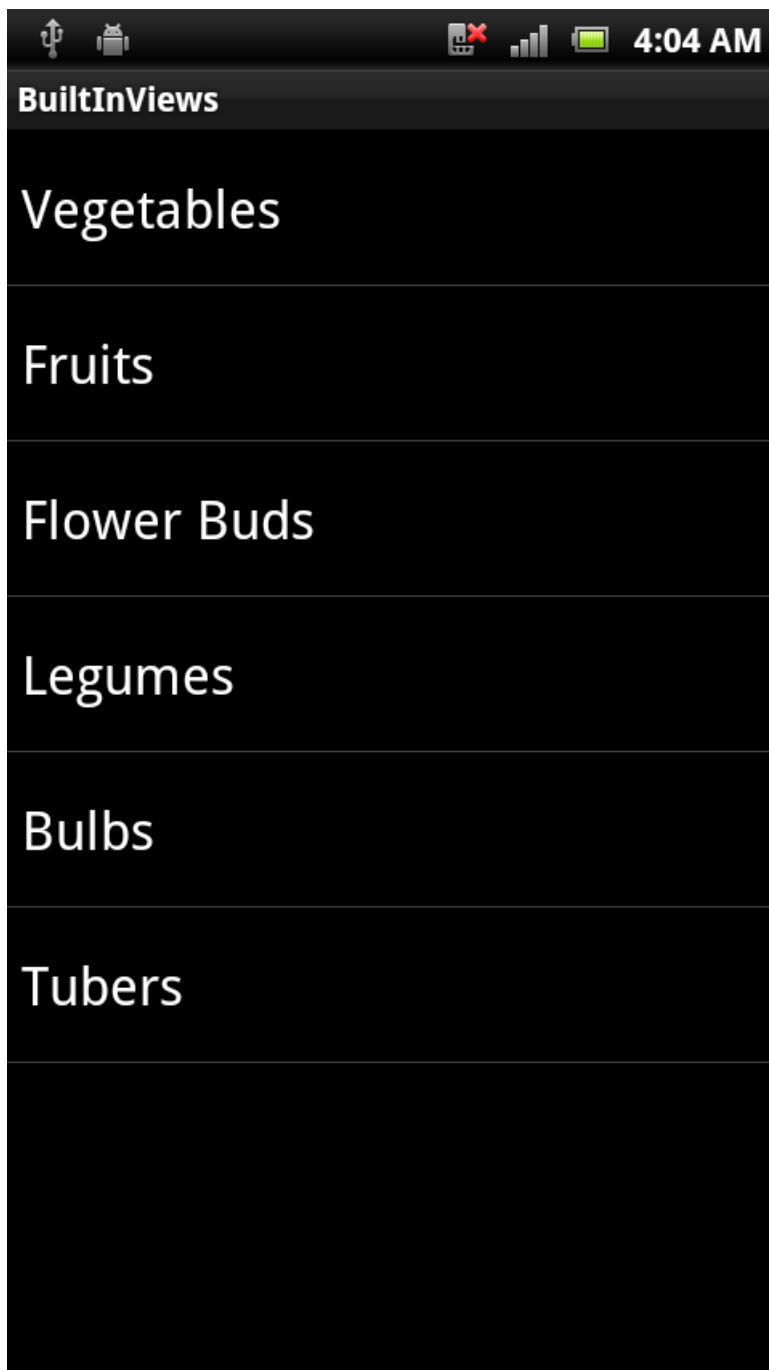
使用 ListActivity 和 ArrayAdapter<字符串>

该示例 **BasicTable/HomeScreen.cs** 演示了如何使用这些类以显示 `ListView` 中只有少量的代码行：

```
[Activity(Label = "BasicTable", MainLauncher = true, Icon = "@drawable/icon")]
public class HomeScreen : ListActivity {
    string[] items;
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        items = new string[] { "Vegetables", "Fruits", "Flower Buds", "Legumes", "Bulbs", "Tubers" };
        ListAdapter = new ArrayAdapter<String>(this, Android.Resource.Layout.SimpleListItem1, items);
    }
    protected override void OnListItemClick(ListView l, View v, int position, long id)
    }
}
```

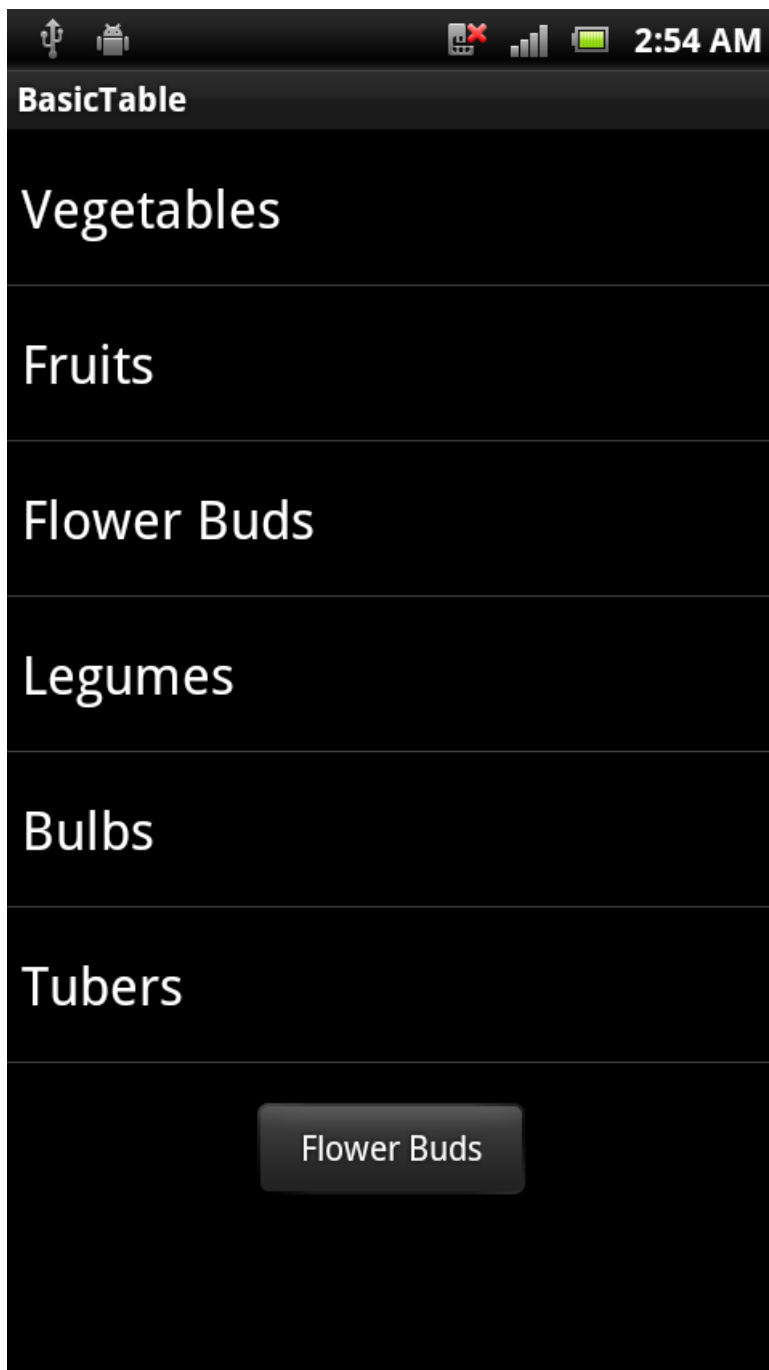
处理行单击

通常 `ListView` 还使用户有机会接触的行执行某些操作（如播放一首歌曲，或调用一个联系人，或显示另一个屏幕）。以响应用户收尾工作了需要有一个更多方法中实现 `ListActivity` - `OnListItemClick` - 如下所示：



```
protected override void OnListItemClick(ListView l, View v, int position, long id)
{
    var t = items[position];
    Android.Widget.Toast.MakeText(this, t, Android.Widget.ToastLength.Short).Show();
}
```

现在，用户可以触摸行和 `Toast` 将会出现警告：



实现 ListAdapter

`ArrayAdapter<string>` 非常适合它很简单，但由于严格限制。但是，通常情况下，有一系列业务实体，而不是只是想要绑定的字符串。例如，如果你的数据包含的员工类的集合，您可能想要只显示每个雇员的名称的列表。若要自定义的行为 `ListView` 来控制显示的数据必须实现的子类 `BaseAdapter` 重写的以下四个项目：

- **计数**—告诉该控件的数据中有多少行。
- **GetView**—返回每个行，一个视图填充数据。此方法具有一个参数为 `ListView` 以便重复使用现有的未使用的行中传递。
- **GetItemId**—返回的行标识符（通常行数字，但也可以是任何您喜欢的长整型值）。
- **此 [int] 索引器**—以返回与特定行号关联的数据。

中的示例代码 `BasicTableAdapter/HomeScreenAdapter.cs` 演示了如何创建子类 `BaseAdapter`：

```

public class HomeScreenAdapter : BaseAdapter<string> {
    string[] items;
    Activity context;
    public HomeScreenAdapter(Activity context, string[] items) : base() {
        this.context = context;
        this.items = items;
    }
    public override long GetItemId(int position)
    {
        return position;
    }
    public override string this[int position] {
        get { return items[position]; }
    }
    public override int Count {
        get { return items.Length; }
    }
    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        View view = convertView; // re-use an existing view, if one is available
        if (view == null) // otherwise create a new one
            view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);
        view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = items[position];
        return view;
    }
}

```

使用自定义适配器

使用自定义适配器是类似于内置 `ArrayAdapter`，并传入 `context` 和 `string[]` 要显示的值：

```
ListAdapter = new HomeScreenAdapter(this, items);
```

因为此示例使用相同的行布局（`SimpleListItem1`）生成的应用程序看起来与前面的示例相同。

重复使用行视图

在此示例中仅有六个项。由于屏幕可容纳八个，因此没有行重复使用必需。在显示时数百或数千个行，但是，它是在浪费的内存来创建数百或数千个 `View` 对象时仅包括八种每次都适合在屏幕上。若要避免这种情况下，行从其视图放到队列中以重复使用的屏幕上消失时。当用户滚动时，`ListView` 调用 `GetView` 若要请求显示新视图—如果可用它将传递中的未使用的视图 `convertView` 参数。如果此值为 `null`，则你的代码应创建新的视图实例，否则为可以重新设置该对象的属性并重新使用它。

`GetView` 方法应遵循此模式重复使用行视图：

```

public override View GetView(int position, View convertView, ViewGroup parent)
{
    View view = convertView; // re-use an existing view, if one is supplied
    if (view == null) // otherwise create a new one
        view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);
    // set view properties to reflect data for the given row
    view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = items[position];
    // return the view, populated with data, for display
    return view;
}

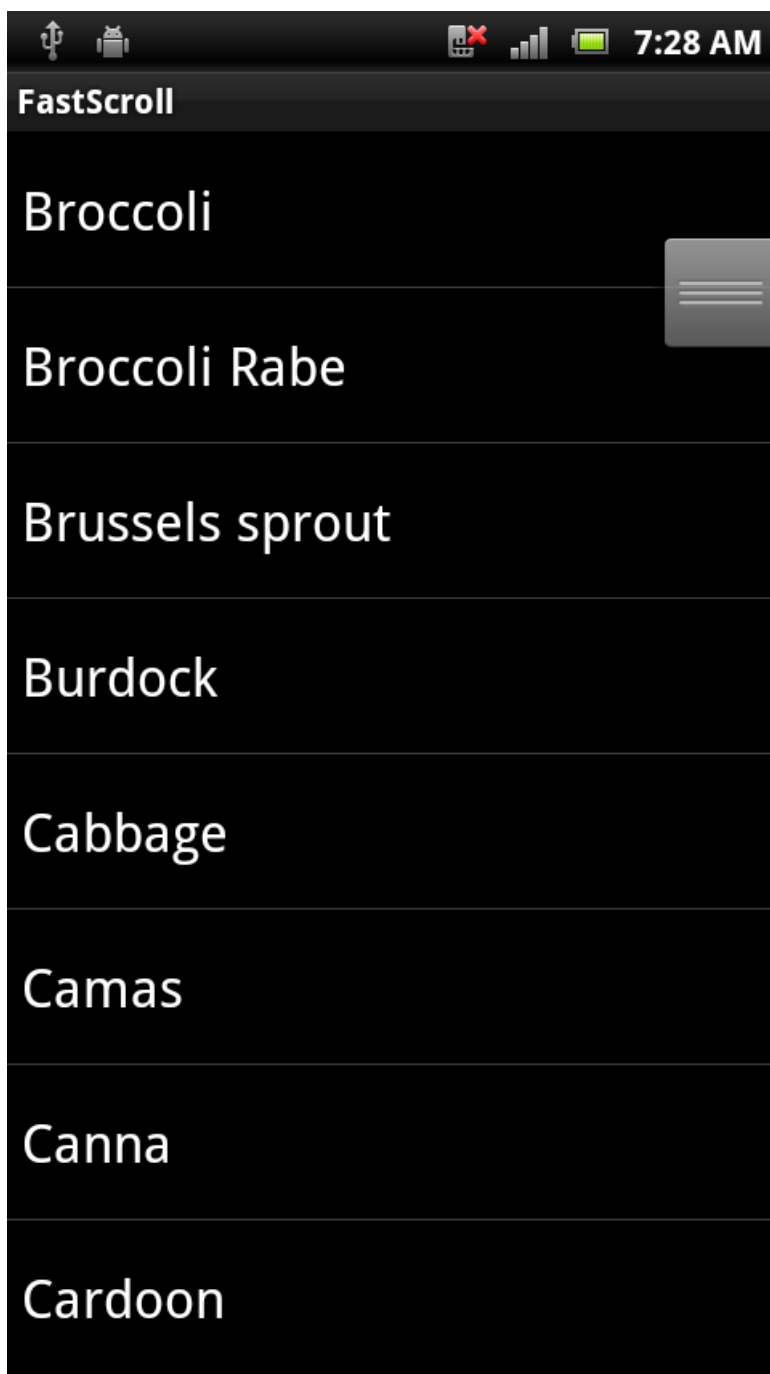
```

自定义适配器实现应该始终重复使用 `convertView` 对象之前创建新视图，以确保它们不会运行内存不足时显示较长的列表。

某些适配器实现（如 `CursorAdapter`）没有 `GetView` 方法，而不是它们需要两种不同方法 `NewView` 并 `BindView` 其中强制执行重复使用行分隔的职责 `GetView` 为两个方法。没有 `CursorAdapter` 文档后面的示例。

启用快速滚动

快速滚动可帮助用户通过提供的其他句柄，它就像直接访问列表的一部分的滚动条来滚动查看较长的列表。此屏幕截图显示了快速滚动句柄：



导致要显示的快速滚动句柄就像设置一样简单 `FastScrollEnabled` 属性设置为 `true`：

```
ListView.FastScrollEnabled = true;
```

添加部分索引

部分索引为用户提供其他反馈，当它们是快速滚动浏览一长串-显示他们已滚动到哪些 section。若要使其显示适配器子类必须实现的部分索引 `ISectionIndexer` 接口提供具体取决于所显示的行索引文本：



若要实现 `ISectionIndexer` 需要三个方法添加到适配器：

- **GetSections** –提供索引可能会显示的标题部分的完整列表。此方法需要 Java 对象的数组，因此该代码需要创建 `Java.Lang.Object[]` 从.NET 集合。在本示例中它将返回一组在列表中的初始字符 `Java.Lang.String`。
- **GetPositionForSection** –返回给定的部分索引的第一个行位置。
- **GetSectionForPosition** –返回部分索引，要为某一给定行显示。

该示例 `SectionIndex/HomeScreenAdapter.cs` 文件的构造函数中实现这些方法和一些附加代码。构造函数生成的节索引循环通过每个行并提取的标题（项必须已进行排序为实现此目的）的第一个字符。

```

alphaIndex = new Dictionary<string, int>();
for (int i = 0; i < items.Length; i++) { // loop through items
    var key = items[i][0].ToString();
    if (!alphaIndex.ContainsKey(key))
        alphaIndex.Add(key, i); // add each 'new' letter to the index
}
sections = new string[alphaIndex.Keys.Count];
alphaIndex.Keys.CopyTo(sections, 0); // convert letters list to string[]

// Interface requires a Java.Lang.Object[], so we create one here
sectionsObjects = new Java.Lang.Object[sections.Length];
for (int i = 0; i < sections.Length; i++) {
    sectionsObjects[i] = new Java.Lang.String(sections[i]);
}

```

使用创建的数据结构 `ISectionIndexer` 方法都是非常简单：

```

public Java.Lang.Object[] GetSections()
{
    return sectionsObjects;
}
public int GetPositionForSection(int section)
{
    return alphaIndexer[sections[section]];
}
public int GetSectionForPosition(int position)
{
    // this method isn't called in this example, but code is provided for completeness
    int prevSection = 0;
    for (int i = 0; i < sections.Length; i++)
    {
        if (GetPositionForSection(i) > position)
        {
            break;
        }
        prevSection = i;
    }
    return prevSection;
}

```

在部分索引标题不需要将 1 对 1 映射到实际部分。这就是为什么 `GetPositionForSection` 方法存在。

`GetPositionForSection` 使你有机会将映射到任何部分都是在列表视图中的索引列表中的任何索引。例如，您可能必须在索引中的"z"，但你可能不具有用于每个字母的表节，因此可能会映射到而不是"z"映射到 26, 25 或 24，或其他任何部分索引"z"应映射到。

相关链接

- [BasicTableAndroid \(示例\)](#)
- [BasicTableAdapter \(示例\)](#)
- [FastScroll \(示例\)](#)

自定义 ListView 的外观

2018/10/26 • [Edit Online](#)

概述

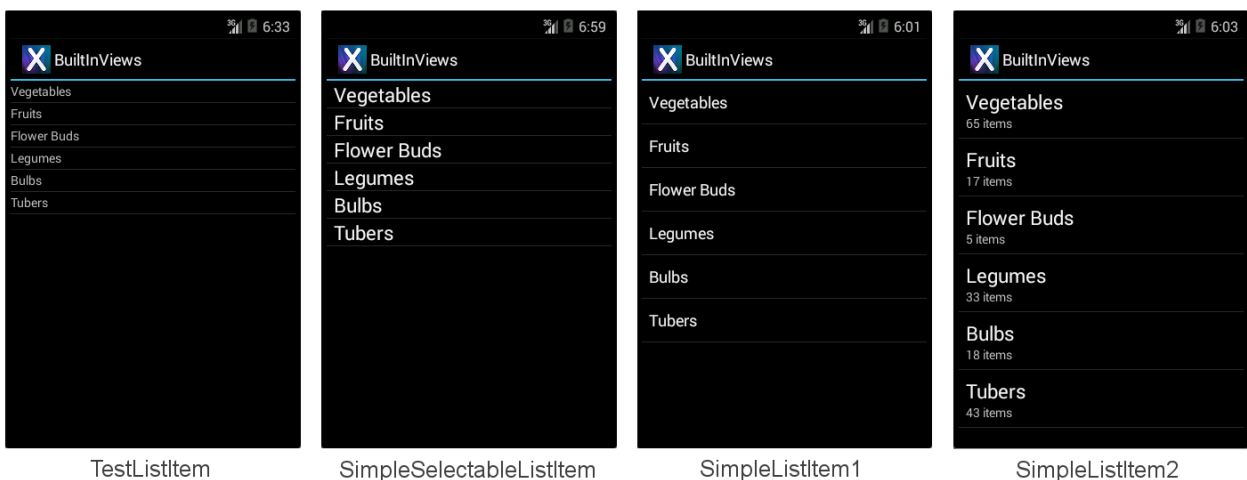
正在显示的行的布局由指定 ListView 的外观。若要更改的外观 `ListView`，使用不同的行的布局。

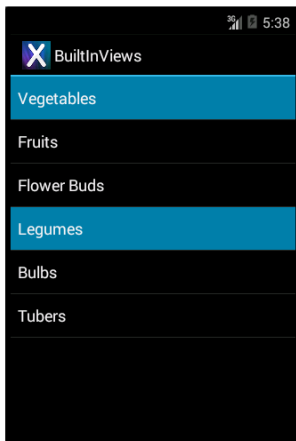
内置行视图

有十二个视图，可以使用引用 **Android.Resource.Layout**:

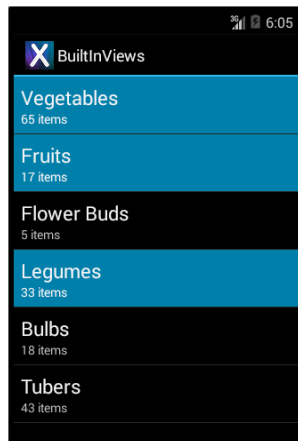
- **TestListItem** – 单一具有最少的格式的文本的行。
- **SimpleListItem1** – 的单行文本。
- **SimpleListItem2** – 两行文本。
- **SimpleSelectableListItem** – 单一的支持 (API 级别 11 中已加入) 的单个或多个项目选择的文本行。
- **SimpleListItemActivated1** – SimpleListItem1, 与相似, 但背景颜色指示当选择行 (API 级别 11 中添加)。
- **SimpleListItemActivated2** – SimpleListItem2, 与相似, 但背景颜色指示当选择行 (API 级别 11 中添加)。
- **SimpleListItemChecked** – 显示复选标记以指示选定内容。
- **SimpleListItemMultipleChoice** – 显示复选框以指示多重选择。
- **SimpleListItemSingleChoice** – 显示单选按钮, 以指示排斥的选定内容。
- **TwoLineListItem** – 两行文本。
- **ActivityListItem** – 的单行文本与图像。
- **SimpleExpandableListItem** – 可以展开或折叠组按类别和每个组的行。

每个内置行视图有与之关联的内置的样式。这些屏幕截图显示了每个视图的显示方式:

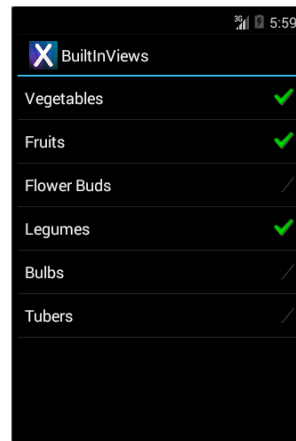




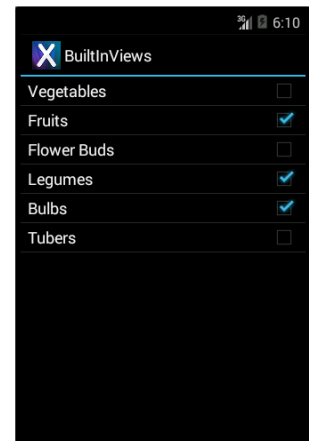
SimpleListItemActivated1



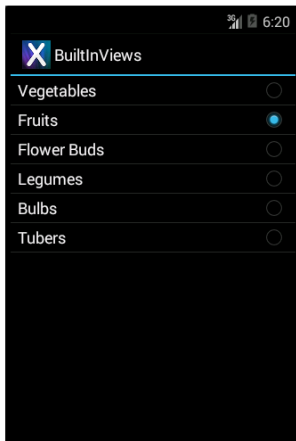
SimpleListItemActivated2



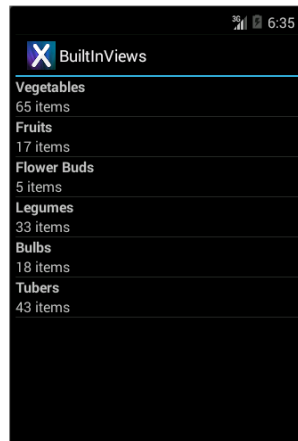
SimpleListItemChecked



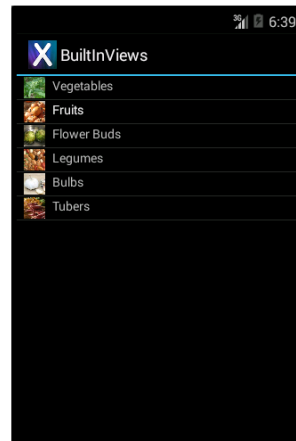
SimpleListItemMultipleChoice



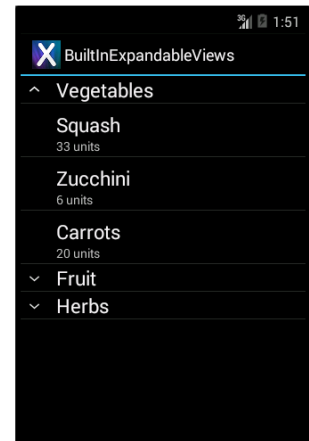
SimpleListItemSingleChoice



TwoLineListItem



ActivityListItem



SimpleExpandableListItem

BuiltInViews/HomeScreenAdapter.cs 示例文件 (在 **BuiltInViews** 解决方案) 包含用于生成不可展开列表项屏幕的代码。在视图中设置 `GetView` 方法如下：

```
view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);
```

然后可以通过引用标准控件标识符来设置视图的属性 `Text1`，`Text2` 并 `Icon` 下 `Android.Resource.Id` (未设置属性的视图不包含，否则将引发异常)：

```
view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = item.Heading;
view.FindViewById<TextView>(Android.Resource.Id.Text2).Text = item.SubHeading;
view.FindViewById<ImageView>(Android.Resource.Id.Icon).SetImageResource(item.ImageResourceId); // only use
with ActivityListItem
```

BuiltInExpandableViews/ExpandableScreenAdapter.cs 示例文件 (在 **BuiltInViews** 解决方案) 包含用于生成 `SimpleExpandableListItem` 屏幕的代码。在中设置的组视图 `GetGroupView` 方法如下：

```
view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleExpandableListItem1, null);
```

在中设置子视图 `GetChildView` 方法如下：

```
view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleExpandableListItem2, null);
```

然后可以通过引用标准来设置的组视图和子视图的属性 `Text1` 和 `Text2` 控制标识符，如上所示。

`SimpleExpandableListItem` 屏幕截图 (如上所示) 提供了一个行组视图 (`SimpleExpandableListItem1`) 和两行代码子

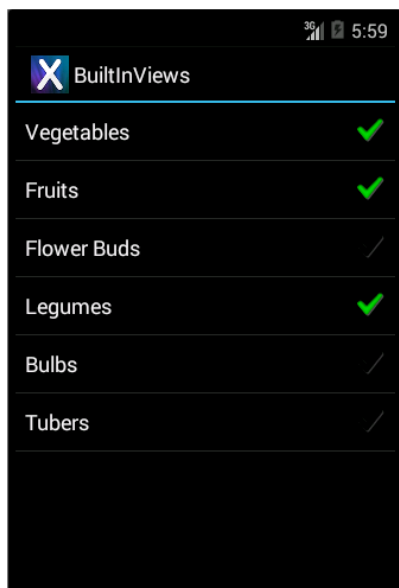
视图 (SimpleExpandableListItem2) 的示例。或者, 可以为两行 (SimpleExpandableListItem2) 配置的组视图和子视图可以配置为一个行 (SimpleExpandableListItem1), 或这两组视图和子视图可以具有相同数目的行。

附件

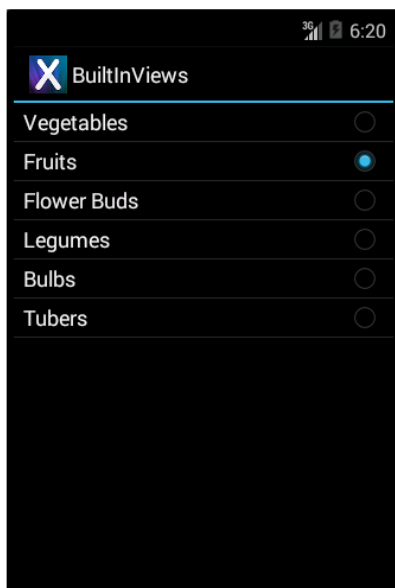
行可以具有附件添加到视图的右侧, 以指示选择状态:

- **SimpleListItemChecked** -创建单项选择列表, 其中检查作为指示器。
- **SimpleListItemSingleChoice** -创建只有一个选择是可能的单选按钮类型列表。
- **SimpleListItemMultipleChoice** -创建多个选项是可能的复选框类型列表。

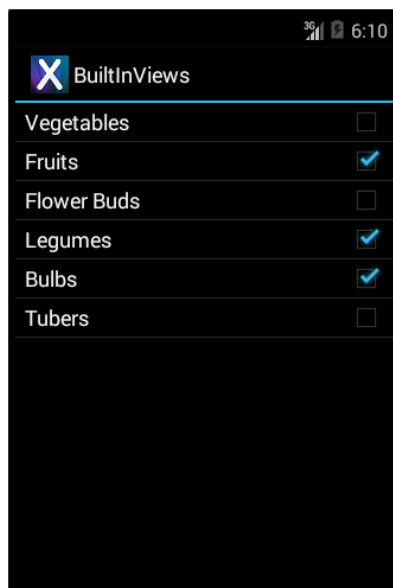
前面提到的附件其各自的顺序中的以下屏幕所示:



SimpleListItemChecked



SimpleListItemSingleChoice



SimpleListItemMultipleChoice

若要显示一个这些附件传递到适配器的必需的布局资源 ID 然后手动设置所需的行的选择状态。这行代码显示了如何创建和分配 `Adapter` 使用这些布局之一:

```
ListAdapter = new ArrayAdapter<String>(this, Android.Resource.Layout.SimpleListItemChecked, items);
```

`ListView` 本身支持不同的选择模式, 而不考虑所显示的附件。若要避免混淆, 请使用 `Single` 使用的选择模式 `SingleChoice` 附件和 `Checked` 或 `Multiple` 模式 `MultipleChoice` 样式。选择模式受 `ChoiceMode` 属性的 `ListView`。

处理 API 级别

早期版本的 Xamarin.Android 实现为整数属性的枚举。最新版本引入了合适的 .NET 枚举类型从而得更轻松地发现潜在的选项。

具体取决于哪个 API 级别为目标, `ChoiceMode` 是一个整数或枚举。示例文件 **AccessoryViews/HomeScreen.cs** 已块注释掉如果你想要面向 Gingerbread API:

```
// For targeting Gingerbread the ChoiceMode is an int, otherwise it is an
// enumeration.

lv.ChoiceMode = Android.Widget.ChoiceMode.Single; // 1
//lv.ChoiceMode = Android.Widget.ChoiceMode.Multiple; // 2
//lv.ChoiceMode = Android.Widget.ChoiceMode.None; // 0

// Use this block if targeting Gingerbread or lower
/*
lv.ChoiceMode = 1; // Single
//lv.ChoiceMode = 0; // none
//lv.ChoiceMode = 2; // Multiple
//lv.ChoiceMode = 3; // MultipleModal
*/
```

以编程方式选择项

手动将设置为哪些项选择通过 `SetItemChecked` 方法（它可以多次调用了多重选择）：

```
// Set the initially checked row ("Fruits")
lv.SetItemChecked(1, true);
```

该代码还需要检测以不同的方式从多个选择选择一个选项。若要确定在选择了哪一行 `Single` 模式下使用 `CheckedItemPosition` 整数属性：

```
FindViewById<ListView>(Android.Resource.Id.List).CheckedItemPosition
```

若要确定在选择了哪些行 `Multiple` 模式下需要遍历 `CheckedItemPositions` `SparseBooleanArray`。稀疏数组就像一个字典，其中只包含条目的值已更改，因此您必须遍历整个数组寻找 `true` 值要知道什么具有已选择列表中，如下面的代码段中所示：

```
var sparseArray = FindViewById<ListView>(Android.Resource.Id.List).CheckedItemPositions;
for (var i = 0; i < sparseArray.Size(); i++ )
{
    Console.Write(sparseArray.KeyAt(i) + "=" + sparseArray.ValueAt(i) + ",");
}
Console.WriteLine();
```

创建自定义行布局

四个内置行视图是非常简单。要显示更复杂的布局（如电子邮件或推文或联系信息的列表）将需要自定义视图。自定义视图通常声明为 `AXML 文件资源/布局` 目录，然后再加载，使用其资源的自定义适配器的 `Id`。该视图可以包含任意数量的显示类（如 `TextViews`、`ImageViews` 和其他控件）使用自定义颜色、字体和布局。

此示例与前面的示例通过多种方式不同：

- 继承自 `Activity`，而不 `ListActivity`。你可以为任何自定义行 `ListView`，但也可以在包含其他控件 `Activity` 布局（如标题、按钮或其他用户界面元素）。此示例将上面的标题 `ListView` 来说明。
- 屏幕; 需要一个 `AXML 布局文件` 在前面的示例 `ListActivity` 不需要的布局文件。包含此 `AXML ListView` 控制声明。
- 需要一个 `AXML 布局文件`，可呈现每个行。此 `AXML 文件` 包含自定义字体和颜色设置的文本和图像控件。
- 使用可选的自定义选择器 `XML 文件` 选择此项时设置的行的外观。
- `Adapter` 实现返回从自定义布局 `GetView` 重写。

- `ItemClickListener` 必须以不同的方式声明 (事件处理程序附加到 `ListView.OnItemClickListener` 而不是重写 `OnListItemClick` 中 `ListActivity`)。

这些更改下面详细介绍, 开始创建活动的视图和自定义行视图, 然后介绍对适配器和活动来呈现它们的修改。

将 `ListView` 添加到活动布局

因为 `HomeScreen` 不再继承 `ListActivity` 它没有默认视图中, 因此必须为 `HomeScreen` 的视图创建布局 AXML 文件。对于此示例, 该视图将有一个标题 (使用 `TextView`) 和一个 `ListView` 以显示数据。在中定义布局 `Resources/Layout/HomeScreen.xml` 文件如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/Heading"
        android:text="Vegetable Groups"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#00000000"
        android:textSize="30dp"
        android:textColor="#FF267F00"
        android:textStyle="bold"
        android:padding="5dp"
    />
    <ListView android:id="@+id/List"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:cacheColorHint="#FFDAFF7F"
    />
</LinearLayout>
```

使用的好处 `Activity` 与自定义布局 (而不是 `ListActivity`) 在于能够将其他控件添加到屏幕上, 如标题 `TextView` 在此示例中。

创建自定义行布局

另一个 AXML 布局文件需包含会在列表视图中显示每个行的自定义布局。在此示例中的行, 将有绿色背景、棕色、文本和右对齐图像。Android XML 标记来声明此布局中所述 `Resources/Layout/CustomView.xml`:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#FFDAFF7F"
    android:padding="8dp">
    <LinearLayout android:id="@+id/Text"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp">
        <TextView
            android:id="@+id/Text1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#FF7F3300"
            android:textSize="20dp"
            android:textStyle="italic"
        />
        <TextView
            android:id="@+id/Text2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="14dp"
            android:textColor="#FF267F00"
            android:paddingLeft="100dp"
        />
    </LinearLayout>
    <ImageView
        android:id="@+id/Image"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:padding="5dp"
        android:src="@drawable/icon"
        android:layout_alignParentRight="true" />
</RelativeLayout >

```

而自定义行布局可以包含许多不同的控件，滚动性能可能受到复杂的设计和使用映像（尤其是如果他们有要通过网络加载）。有关解决滚动性能问题，请参阅 Google 的文章了解详细信息。

引用自定义视图

自定义适配器示例的实现 `HomeScreenAdapter.cs`。关键方法是 `GetView`，它会加载自定义 AXML 使用的资源 ID `Resource.Layout.CustomView`，然后在每个视图，然后再返回它在控件上设置属性。完整的适配器类所示：

```

public class HomeScreenAdapter : BaseAdapter<TableItem> {
    List<TableItem> items;
    Activity context;
    public HomeScreenAdapter(Activity context, List<TableItem> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }
    public override long GetItemId(int position)
    {
        return position;
    }
    public override TableItem this[int position]
    {
        get { return items[position]; }
    }
    public override int Count
    {
        get { return items.Count; }
    }
    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];
        View view = convertView;
        if (view == null) // no view to re-use, create new
            view = context.LayoutInflater.Inflate(Resource.Layout.CustomView, null);
        view.FindViewById<TextView>(Resource.Id.Text1).Text = item.Heading;
        view.FindViewById<TextView>(Resource.Id.Text2).Text = item.SubHeading;
        view.FindViewById<ImageView>(Resource.Id.Image).SetImageResource(item.ImageResourceId);
        return view;
    }
}

```

引用活动中自定义的 ListView

因为 `HomeScreen` 类现在继承自 `Activity`、`ListView` 要保存对 AXML 中声明的控件的引用的类中声明字段：

```

ListView listView;

```

然后，类必须加载活动的自定义布局 AXML 使用 `SetContentView` 方法。然后，它可以找到 `ListView` 布局中的控件然后创建并分配该适配器并分配的单击处理程序。OnCreate 方法中的代码如下所示：

```

SetContentView(Resource.Layout.HomeScreen); // loads the HomeScreen.xml as this activity's view
listView = FindViewById<ListView>(Resource.Id.List); // get reference to the ListView in the layout

// populate the listview with data
listView.Adapter = new HomeScreenAdapter(this, tableItems);
listView.ItemClick += OnListItemClick; // to be defined

```

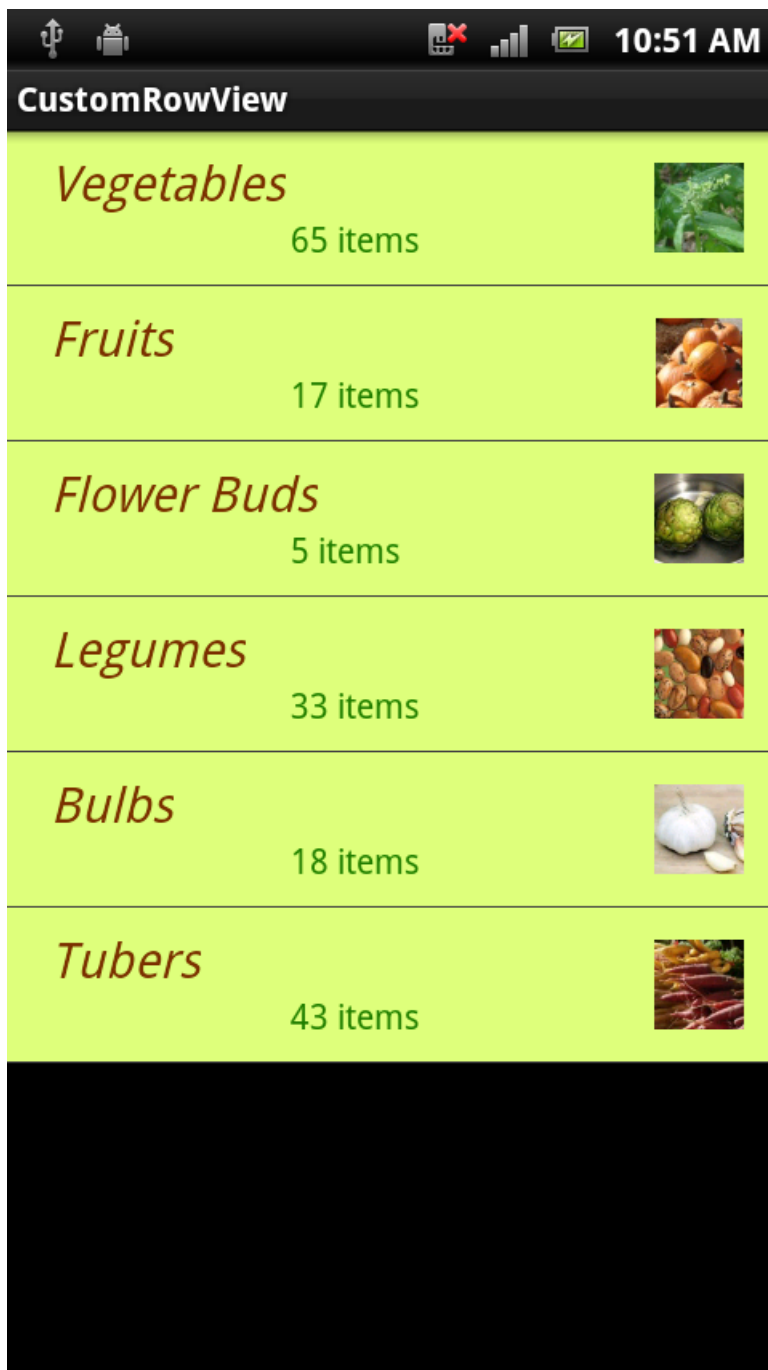
最后 `ItemClick` 必须定义处理程序; 在这种情况下它只是显示 `Toast` 消息：

```

void OnListItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
    var listView = sender as ListView;
    var t = tableItems[e.Position];
    Android.Widget.Toast.MakeText(this, t.Heading, Android.Widget.ToastLength.Short).Show();
}

```

生成屏幕如下所示：



自定义的行选择器颜色

触摸某一行时它应会突出显示用户反馈。自定义视图时指定为作为背景色**CustomView.axml**，它还重写选择突出显示。这行代码中**CustomView.axml**背景的浅绿色，但它也意味着没有可视的指示器时接触的行集：

```
android:background="#FFDAFF7F"
```

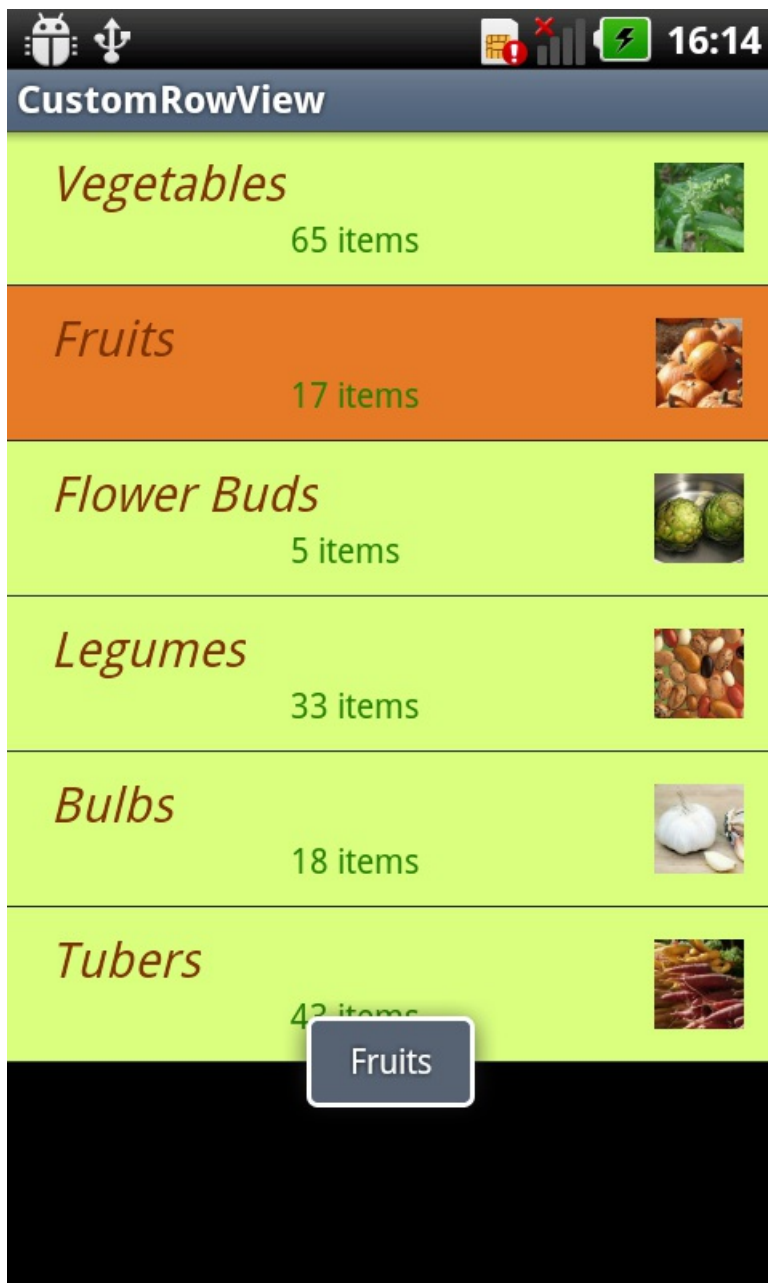
若要重新启用突出显示行为，以及自定义使用的颜色，请改为将背景属性设置为自定义选择器。选择器将声明的默认背景色以及突出显示颜色。该文件**Resources/Drawable/CustomSelector.xml**包含以下声明：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item android:state_pressed="false"
    android:state_selected="false"
    android:drawable="@color/cellback" />
<item android:state_pressed="true" >
    <shape>
        <gradient
            android:startColor="#E77A26"
            android:endColor="#E77A26"
            android:angle="270" />
        </shape>
    </item>
<item android:state_selected="true"
    android:state_pressed="false"
    android:drawable="@color/cellback" />
</selector>
```

若要引用自定义选择器，更改中的背景特性**CustomView.xml**到：

```
android:background="@drawable/CustomSelector"
```

所选的行和相应 **Toast** 消息现在如下所示：



阻止在自定义布局上闪烁

Android 尝试改进的性能 `ListView` 滚动通过缓存布局信息。如果您有长时间滚动数据的列表还应设置 `android:cacheColorHint` 属性上的 `ListView` (为相同作为自定义行布局的背景的颜色值) 的活动的 AXML 定义中的声明。不包括此提示可能会导致闪烁当用户滚动浏览列表自定义行背景色。

相关链接

- [BuiltInViews \(示例\)](#)
- [AccessoryViews \(示例\)](#)
- [CustomRowView \(示例\)](#)

使用 CursorAdapters

2018/10/26 • [Edit Online](#)

概述

Android 提供了适配器类，特别是要显示 SQLite 数据库查询中的数据：

SimpleCursorAdapter – 类似到 `ArrayAdapter` 因为不子类化的情况下可以使用它。只需在构造函数中提供所需的参数（如游标和布局信息中），然后将分配给 `ListView`。

CursorAdapter – 基类继承时需要更好地控制通过数据绑定的值复制到布局控件（例如，显示/隐藏控件或更改其属性）。

游标适配器提供高性能的方式来滚动查看存储在 SQLite 中的数据的长列表。使用代码必须定义中的 SQL 查询 `Cursor` 对象，然后介绍如何创建和填充的视图的每个行。

创建一个 SQLite 数据库

若要演示游标适配器需要简单的 SQLite 数据库实现。中的代

码 **SimpleCursorTableAdapter/VegetableDatabase.cs** 包含代码和 SQL 来创建一个表并填充一些数据。完整 `VegetableDatabase` 类如下所示：

```
class VegetableDatabase : SQLiteOpenHelper {
    public static readonly string create_table_sql =
        "CREATE TABLE [vegetables] ([_id] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, [name] TEXT NOT NULL UNIQUE)";
    public static readonly string DatabaseName = "vegetables.db";
    public static readonly int DatabaseVersion = 1;
    public VegetableDatabase(Context context) : base(context, DatabaseName, null, DatabaseVersion) { }
    public override void OnCreate(SQLiteDatabase db)
    {
        db.ExecSQL(create_table_sql);
        // seed with data
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Vegetables')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Fruits')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Flower Buds')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Legumes')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Bulbs')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Tubers')");
    }
    public override void OnUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        // not required until second version :)
        throw new NotImplementedException();
    }
}
```

`VegetableDatabase` 中进行实例化类 `OnCreate` 方法的 `HomeScreen` 活动。`SQLiteOpenHelper` 基类管理数据库文件的安装程序并确保在 SQL 其 `OnCreate` 方法只运行一次。在以下两个示例中为使用此类 `SimpleCursorAdapter` 和 `CursorAdapter`。

游标查询必须有一个整数列 `_id` 为 `CursorAdapter` 工作。如果基础表不具有名为的整数列 `_id` 然后使用列别名中的另一个唯一整数 `RawQuery` 构成光标。请参阅 [Android 文档](#) 有关进一步信息。

创建游标

这些示例使用 `RawQuery` 若要打开 SQL 查询到 `Cursor` 对象。返回从游标的列列表定义可用于在光标适配器中显示

的数据列。创建的数据库中的代码 `SimpleCursorTableAdapter/HomeScreen.cs` `OnCreate` 方法如下所示：

```
vdb = new VegetableDatabase(this);
cursor = vdb.ReadableDatabase.RawQuery("SELECT * FROM vegetables", null); // cursor query
StartManagingCursor(cursor);
// use either SimpleCursorAdapter or CursorAdapter subclass here!
```

调用的任何代码 `StartManagingCursor` 还应调用 `StopManagingCursor`。这些示例使用 `OnCreate` 若要开始，和 `OnDestroy` 以关闭游标。 `OnDestroy` 方法包含此代码：

```
StopManagingCursor(cursor);
cursor.Close();
```

一旦应用程序都有一个 SQLite 数据库可用并已创建的光标对象，因为所示，它可以利用 `SimpleCursorAdapter` 或其子 `CursorAdapter` 以显示行 `ListView`。

使用 SimpleCursorAdapter

`SimpleCursorAdapter` 类似于 `ArrayAdapter`，但专用于与 SQLite 一起使用。其不需要子类化 – 只需创建对象时将设置一些简单的参数，然后将其分配给 `ListView` 的 `Adapter` 属性。

`SimpleCursorAdapter` 构造函数的参数包括：

上下文 – 对包含的活动的引用。

布局 – 要使用行视图的资源 ID。

要求 ICursor – 其中包含要显示的数据的 SQLite 查询的游标。

从字符串数组-数组与游标中的列的名称相对应的字符串。

到整数数组-数组的对应行布局中控件的布局 Id。中指定的列的值 from 数组将绑定到相同的索引处的此数组中指定了 ControllID。

from 和 to 数组必须具有相同的条目数，因为其形成从数据源向布局控件的映射视图中。

`SimpleCursorTableAdapter/HomeScreen.cs` 向上示例代码电线 `SimpleCursorAdapter` 如下所示：

```
// which columns map to which layout controls
string[] fromColumns = new string[] { "name" };
int[] toControlIDs = new int[] { Android.Resource.Id.Text1 };
// use a SimpleCursorAdapter
listView.Adapter = new SimpleCursorAdapter (this, Android.Resource.Layout.SimpleListItem1, cursor,
    fromColumns,
    toControlIDs);
```

`SimpleCursorAdapter` 是快速简单地显示在 SQLite 数据 `ListView`。主要限制是，它只能绑定列的值以显示控件，它不允许你更改（例如，显示/隐藏控件或更改属性）的行布局的其他方面。

子类化 CursorAdapter

一个 `CursorAdapter` 子类具有相同的性能优点为 `SimpleCursorAdapter` 显示数据从 SQLite，但它还提供完全控制创建和每个行视图的布局。 `CursorAdapter` 实现有很大差异子类化 `BaseAdapter` 因为它不重写 `GetView`， `GetItemId`， `Count` 或 `this[]` 索引器。

提供一个有效的 SQLite 数据库，您只需重写两个方法来创建 `CursorAdapter` 子类：

- **BindView** – 给定一个视图，将其更新为显示提供的游标中的数据。
- **NewItem** – 时调用 `ListView` 需要新的视图来显示。 `CursorAdapter` 将会负责的回收视图 (不同于 `getView` 常规适配器上的方法)。

前面的示例中的适配器子类具有方法返回的行数以及如何检索当前项 – `CursorAdapter` 不需要这些方法，因为该信息可以从此类本身的光标。通过将拆分的创建和填充到这两种方法，每个视图的 `CursorAdapter` 强制执行视图重新使用。这是与此相反到常规适配器，则可以忽略 `convertView` 参数的 `BaseAdapter.getView` 方法。

实现 `CursorAdapter`

中的代码 `CursorTableAdapter/HomeScreenCursorAdapter.cs` 包含 `CursorAdapter` 子类。它存储传入构造函数，以便它可以访问的上下文引用 `LayoutInflater` 在 `NewItem` 方法。完整的类如下所示：

```
public class HomeScreenCursorAdapter : CursorAdapter {
    Activity context;
    public HomeScreenCursorAdapter(Activity context, ICursor c)
        : base(context, c)
    {
        this.context = context;
    }
    public override void BindView(View view, Context context, ICursor cursor)
    {
        var textView = view.FindViewById<TextView>(Android.Resource.Id.Text1);
        textView.Text = cursor.GetString(1); // 'name' is column 1 in the cursor query
    }
    public override View NewView(Context context, ICursor cursor, ViewGroup parent)
    {
        return this.context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, parent, false);
    }
}
```

分配 `CursorAdapter`

在中 `Activity` 将显示 `ListView`，创建该游标和 `CursorAdapter` 然后将其分配到列表视图。

执行此操作中的代码 `CursorTableAdapter/HomeScreen.cs` `OnCreate` 方法如下所示：

```
// create the cursor
vdb = new VegetableDatabase(this);
cursor = vdb.ReadableDatabase.RawQuery("SELECT * FROM vegetables", null);
StartManagingCursor(cursor);

// create the CursorAdapter
listView.Adapter = (ListAdapter)new HomeScreenCursorAdapter(this, cursor, false);
```

`OnDestroy` 方法包含 `StopManagingCursor` 前面所述的方法调用。

相关链接

- [SimpleCursorTableAdapter \(示例\)](#)
- [CursorTableAdapter \(示例\)](#)

使用 ContentProvider

2018/10/26 • [Edit Online](#)

此外可以使用 CursorAdapters 以显示来自 ContentProvider 数据。Contentprovider 允许你访问由其他应用程序公开的数据 (包括 Android 系统数据, 如联系人、媒体和日历信息)。

访问 ContentProvider 的首选的方法是使用 LoaderManager CursorLoader 使用。在 Android 3.0 (API 级别 11, Honeycomb) 移动关闭主线程正在阻塞的任务中引入了 LoaderManager, 并使用 CursorLoader 允许要绑定到 ListView 以便显示之前在线程中加载数据。

请参阅[简介 Contentprovider](#)有关详细信息。

ListView 和活动生命周期

2018/10/26 • [Edit Online](#)

活动通过某些状态转在应用程序运行，如启动、运行、暂停和停止。有关详细信息，以及有关处理状态转换的特定指导原则，请参阅[活动生命周期教程](#)。务必要了解活动生命周期和位置在 `ListView` 正确的位置中的代码。

在活动的此文档中的示例的所有执行安装任务 `OnCreate` 方法和（如果需要）中执行清除 `OnDestroy`。这些示例通常使用小型数据集不会更改，因此是不必要的更频繁地重新加载数据。

但是，如果数据频繁更改，或者使用大量的内存可能适合使用不同的生命周期方法来填充和刷新你 `ListView`。例如，如果基础数据不断更改（或可能会受到其他活动上的更新）然后创建中的适配器 `OnStart` 或 `OnResume` 将确保最新的数据显示了每次显示活动。

如果适配器使用资源，如内存、或托管的游标，请记住需要释放这些资源中其中实例化时所（例如在补充方法。中创建的对象 `OnStart` 可以中释放 `OnStop`）。

配置更改

务必记住该配置更改-尤其是屏幕上旋转和键盘的可见性-可能会导致销毁并重新创建当前活动（除非指定其他方式使用 `ConfigurationChanges` 属性）。这意味着，正常情况下，旋转设备将导致 `ListView` 并 `Adapter` 重新创建和（除非您编写代码 `OnPause` 和 `OnResume`）滚动位置和行选择状态都将丢失。

以下属性会阻止将活动从会销毁并重新创建由于配置更改：

```
[Activity(ConfigurationChanges="keyboardHidden|orientation")]
```

活动应然后替代 `OnConfigurationChanged` 适当地响应这些更改。有关如何处理配置更改的更多详细信息请参阅文档。

GridView

2018/10/26 • [Edit Online](#)

`GridView` 是 `ViewGroup` 二维、可滚动的网格中显示项。网格项自动插入布局使用 `ListAdapter`。

在本教程中，将创建的图像缩略图的网格。选择某个项后的 toast 消息将显示的图像的位置。

启动一个名为的新项目 **HelloGridView**。

找到你想要使用，一些照片或[下载这些示例图像](#)。将图像文件添加到项目的资源/**Drawable**目录。在中属性窗口中，将生成操作设置为**AndroidResource**。

打开**Resources/Layout/Main.xml**文件并插入以下：

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

这 `GridView` 将填充整个屏幕。而是可以自我说明包含以下属性。有关有效的特性的详细信息，请参阅 `GridView` 引用。

打开 `HelloGridView.cs` 并插入以下代码 `OnCreate()` 方法：

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    SetContentView (Resource.Layout.Main);

    var gridview = FindViewById<GridView> (Resource.Id.gridview);
    gridview.Adapter = new ImageAdapter (this);

    gridview.ItemClick += delegate (object sender, AdapterView.ItemClickEventArgs args) {
        Toast.MakeText (this, args.Position.ToString (), ToastLength.Short).Show ();
    };
}
```

之后**Main.xml**的内容视图中，设置布局 `GridView` 与布局从捕获 `FindViewById`。的 `Adapter` 然后使用属性来设置自定义适配器 (`ImageAdapter`) 作为要在网格中显示的所有项的源。 `ImageAdapter` 在下一步中创建。

若要在网格中的某个项单击时执行某些操作，一个匿名委托，它订阅 `ItemClick` 事件。它显示 `Toast`，它显示选定项的索引位置（从零开始）（在实际方案中，位置可用来获取另一项任务的实际尺寸的图像）。请注意 Java 样式侦听器类，可以使用而不是.NET 事件。

创建一个名为的新类 `ImageAdapter` 子类 `BaseAdapter`：

```

public class ImageAdapter : BaseAdapter
{
    Context context;

    public ImageAdapter (Context c)
    {
        context = c;
    }

    public override int Count {
        get { return thumbIds.Length; }
    }

    public override Java.Lang.Object GetItem (int position)
    {
        return null;
    }

    public override long GetItemId (int position)
    {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public override View GetView (int position, View convertView, ViewGroup parent)
    {
        ImageView imageView;

        if (convertView == null) { // if it's not recycled, initialize some attributes
            imageView = new ImageView (context);
            imageView.LayoutParameters = new GridView.LayoutParams (85, 85);
            imageView.SetScaleType (ImageView.ScaleType.CenterCrop);
            imageView.SetPadding (8, 8, 8, 8);
        } else {
            imageView = (ImageView)convertView;
        }

        imageView.SetImageResource (thumbIds[position]);
        return imageView;
    }

    // references to our images
    int[] thumbIds = {
        Resource.Drawable.sample_2, Resource.Drawable.sample_3,
        Resource.Drawable.sample_4, Resource.Drawable.sample_5,
        Resource.Drawable.sample_6, Resource.Drawable.sample_7,
        Resource.Drawable.sample_0, Resource.Drawable.sample_1,
        Resource.Drawable.sample_2, Resource.Drawable.sample_3,
        Resource.Drawable.sample_4, Resource.Drawable.sample_5,
        Resource.Drawable.sample_6, Resource.Drawable.sample_7,
        Resource.Drawable.sample_0, Resource.Drawable.sample_1,
        Resource.Drawable.sample_2, Resource.Drawable.sample_3,
        Resource.Drawable.sample_4, Resource.Drawable.sample_5,
        Resource.Drawable.sample_6, Resource.Drawable.sample_7
    };
}

```

首先，这会实现继承自某些所需的方法 `BaseAdapter`。构造函数和 `Count` 属性都很容易理解。通常情况下，`GetItem(int)` 应返回在适配器中，指定的位置的实例对象，但它对于此示例中，将忽略。同样，`GetItemId(int)` 应返回的行 id 的项，但此处不需要它。

第一种方法需要 `GetView()`。此方法创建一个新 `View` 每个映像添加到 `ImageAdapter`。此调用时，`View` 传递中，这通常是回收的对象 (至少一次已被调用后)，因此可以进行检查以查看该对象是否为 null。如果它是为 null，`ImageView` 实例化并使用映像演示文稿的所需属性配置：

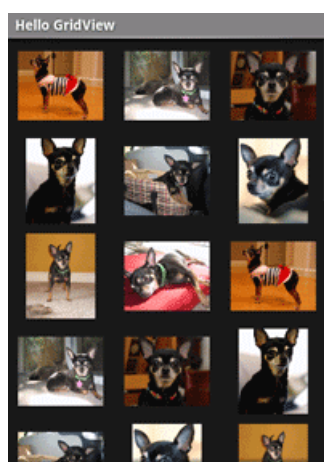
- `LayoutParams` 设置视图的高度和宽度—这可确保, 无论可绘制的大小, 每个图像进行大小调整和裁剪这些维度, 根据需要中容纳不下。
- `setScaleType()` 声明应向中心裁剪图像, 就 (如有必要)。
- `SetPadding(int, int, int, int)` 定义所有边的填充。(请注意, 是否映像具有不同的纵横比, 则更少填充将导致为多个裁剪图像的结果提供给 `imageView` 各自的维数不匹配。)

如果 `View` 传递给 `getView()` 是不为 `null`, 则本地 `ImageView` 初始化与回收 `View` 对象。

在末尾 `getView()` 方法中, `position` 整数传递给该方法用于选择中的映像 `thumbIds` 数组, 它被设置为映像资源 `ImageView`。

剩下的就是定义 `thumbIds` 可绘制资源的数组。

运行该应用程序。网格布局应如下所示:



尝试试验的行为 `GridView` 和 `ImageView` 通过调整它们的属性的元素。例如, 而不是使用 `LayoutParams` 尝试使用 `SetAdjustViewBounds()`。

参考资料

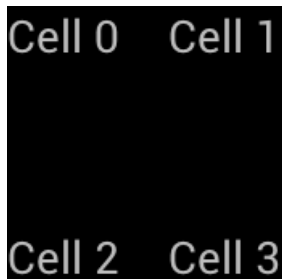
- `GridView`
- `ImageView`
- `BaseAdapter`。

此页的部分是基于工作创建和共享通过 *Android* 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#)。

GridLayout

2018/10/26 • [Edit Online](#)

`GridLayout` 是一种新 `ViewGroup` 支持布局类似于 HTML 表的 2D 网格中的视图，如下所示的子类：



`GridLayout` 适用于平面视图层次结构，其中子视图及其位置的网格中设置通过指定行和其应有的列。这样一来，`GridLayout`能够在网格中定位视图，而无需任何中间视图提供的表结构，如 `TableLayout` 中使用的表行中所示。通过维护平面层次结构中，`GridLayout`能够将更迅速布局其子视图。让我们看看示例演示了这一概念的实际含义在代码中。

创建网格布局

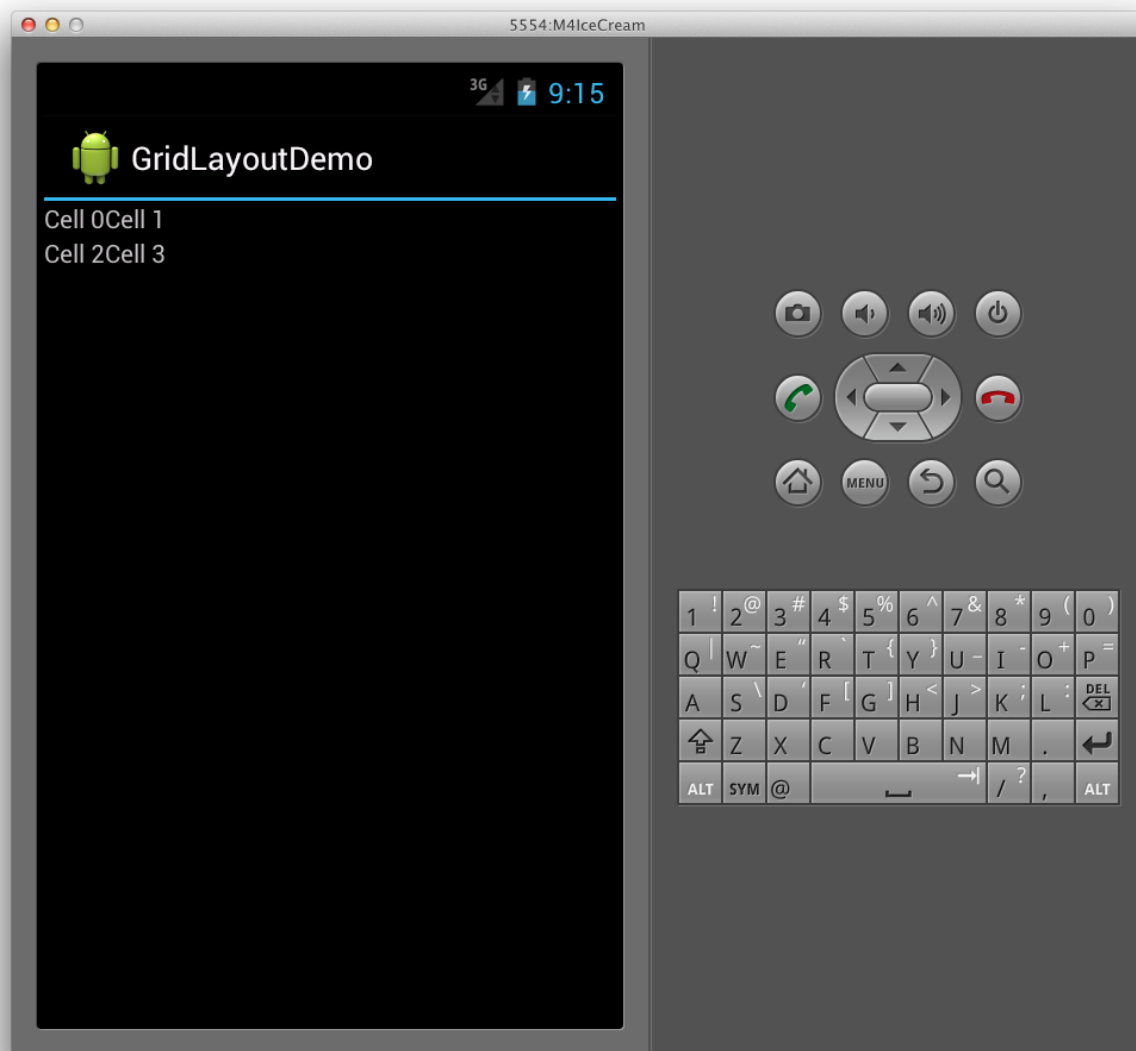
以下 XML 添加了多种 `TextView` 控件添加到`GridLayout`。

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2">
    <TextView
        android:text="Cell 0"
        android:textSize="14dip" />
    <TextView
        android:text="Cell 1"
        android:textSize="14dip" />
    <TextView
        android:text="Cell 2"
        android:textSize="14dip" />
    <TextView
        android:text="Cell 3"
        android:textSize="14dip" />
</GridLayout>
```

布局将调整行和列大小以便单元格可以容纳其内容，如以下关系图所示：

| | |
|--------|--------|
| Cell 0 | Cell 1 |
| Cell 2 | Cell 3 |

这会导致以下用户界面的应用程序中运行时：

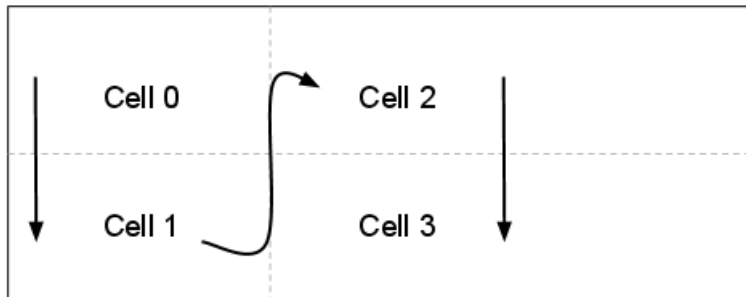


指定方向

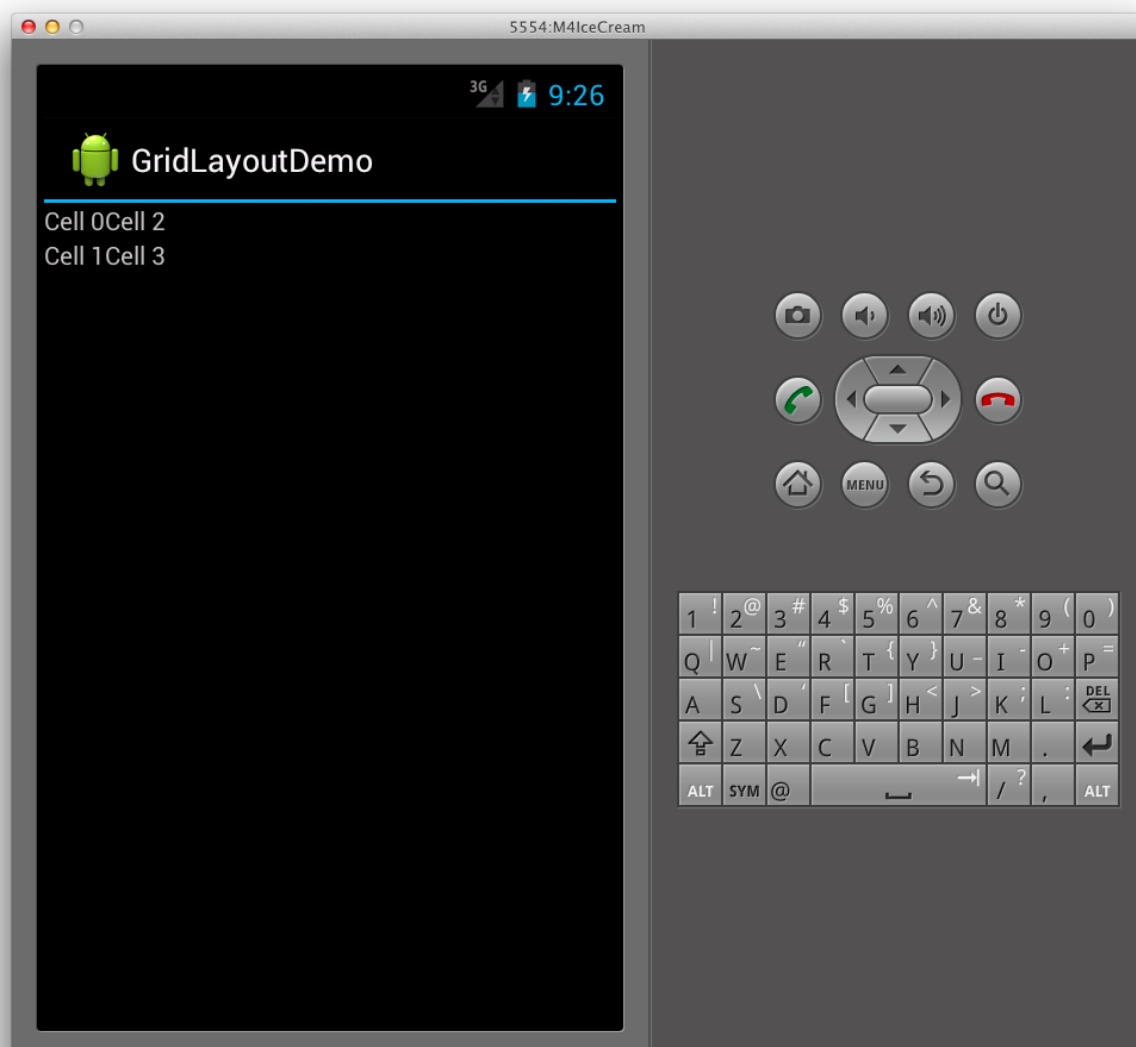
请注意，在 XML 更高版本，每个 `TextView` 未指定的行或列。如果这些未指定，`GridLayout` 分配每个按顺序，根据方向的子视图。例如，让我们更改从默认情况下，水平、此类为垂直的 `GridLayout` 的方向：

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2"
    android:orientation="vertical">
</GridLayout>
```

现在，`GridLayout` 将放置从上到下每个列，而不是从左到右中的单元格，如下所示：



这会导致在运行时的以下用户界面：



指定显式位置

如果我们想要显式控制中的子视图的位置 `GridLayout`，我们可以设置其 `layout_row` 和 `layout_column` 属性。例如，以下 XML 将导致第一个屏幕截图所示（如上所示），而不考虑方向的布局。

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="2">
    <TextView
        android:text="Cell 0"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="0" />
    <TextView
        android:text="Cell 1"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="1" />
    <TextView
        android:text="Cell 2"
        android:textSize="14dip"
        android:layout_row="1"
        android:layout_column="0" />
    <TextView
        android:text="Cell 3"
        android:textSize="14dip"
        android:layout_row="1"
        android:layout_column="1" />
</GridLayout>
```

指定的间距

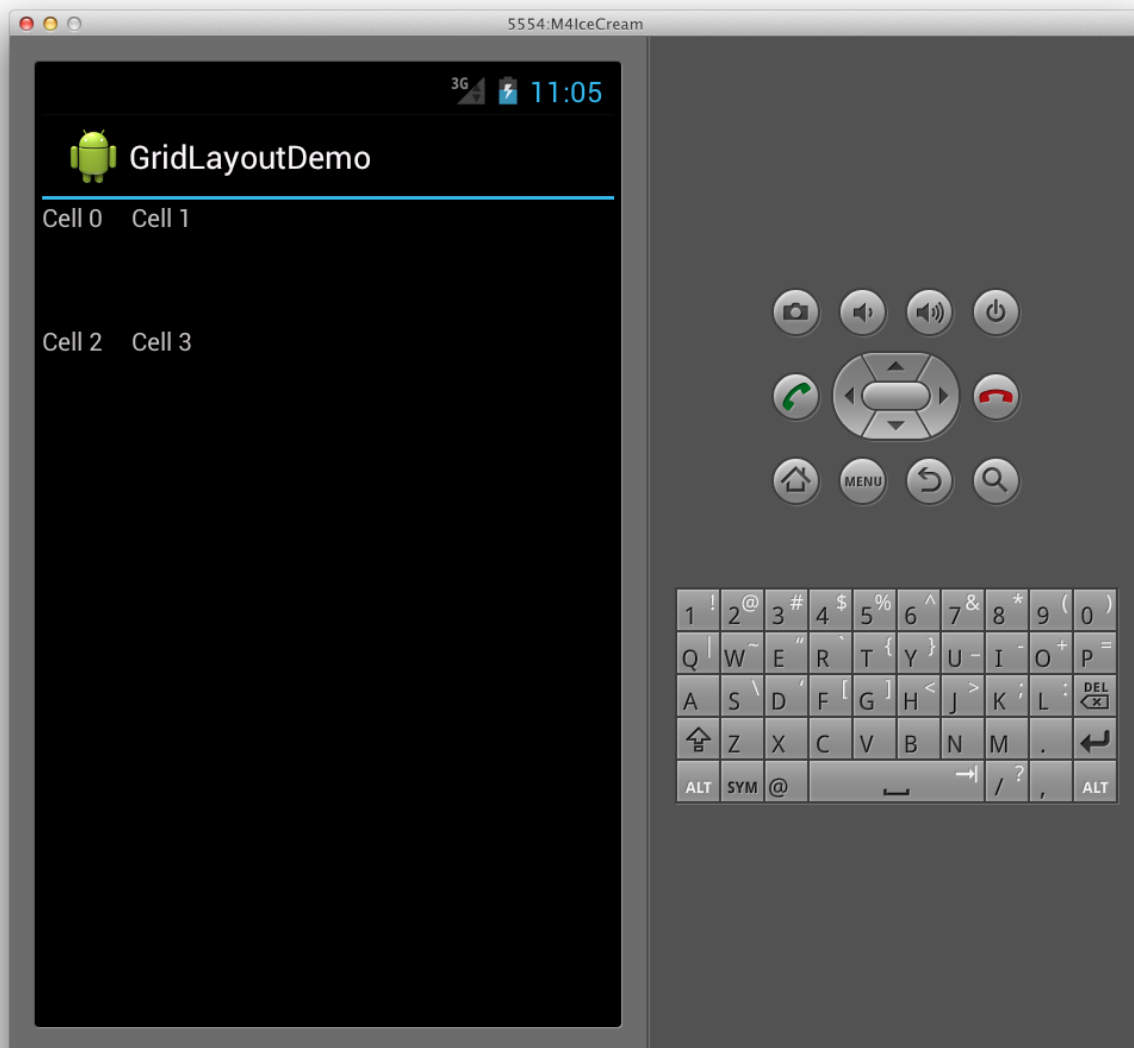
我们有几个选项可将提供的视图子级之间的间距 `GridLayout`。我们可以使用 `layout_margin` 属性来直接设置的边上每个子视图，如下所示

```
<TextView
    android:text="Cell 0"
    android:textSize="14dip"
    android:layout_row="0"
    android:layout_column="0"
    android:layout_margin="10dp" />
```

此外，在 Android 4 中，新的通用间距视图名为 `Space` 现已推出。若要使用它，只需将其添加为子视图中。例如，下面的 XML 添加到一个附加行 `GridLayout` 通过设置其 `rowCount` 为 3，并添加 `Space` 提供之间的间距的视图 `TextViews`。

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="3"
    android:columnCount="2"
    android:orientation="vertical">
    <TextView
        android:text="Cell 0"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="0" />
    <TextView
        android:text="Cell 1"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="1" />
    <Space
        android:layout_row="1"
        android:layout_column="0"
        android:layout_width="50dp"
        android:layout_height="50dp" />
    <TextView
        android:text="Cell 2"
        android:textSize="14dip"
        android:layout_row="2"
        android:layout_column="0" />
    <TextView
        android:text="Cell 3"
        android:textSize="14dip"
        android:layout_row="2"
        android:layout_column="1" />
</GridLayout>
```

此 XML 创建间距以 `GridLayout`，如下所示：



使用新的好处 `Space` 视图是它允许间距, 并且不需要我们对每个子视图设置属性。

跨越行和列

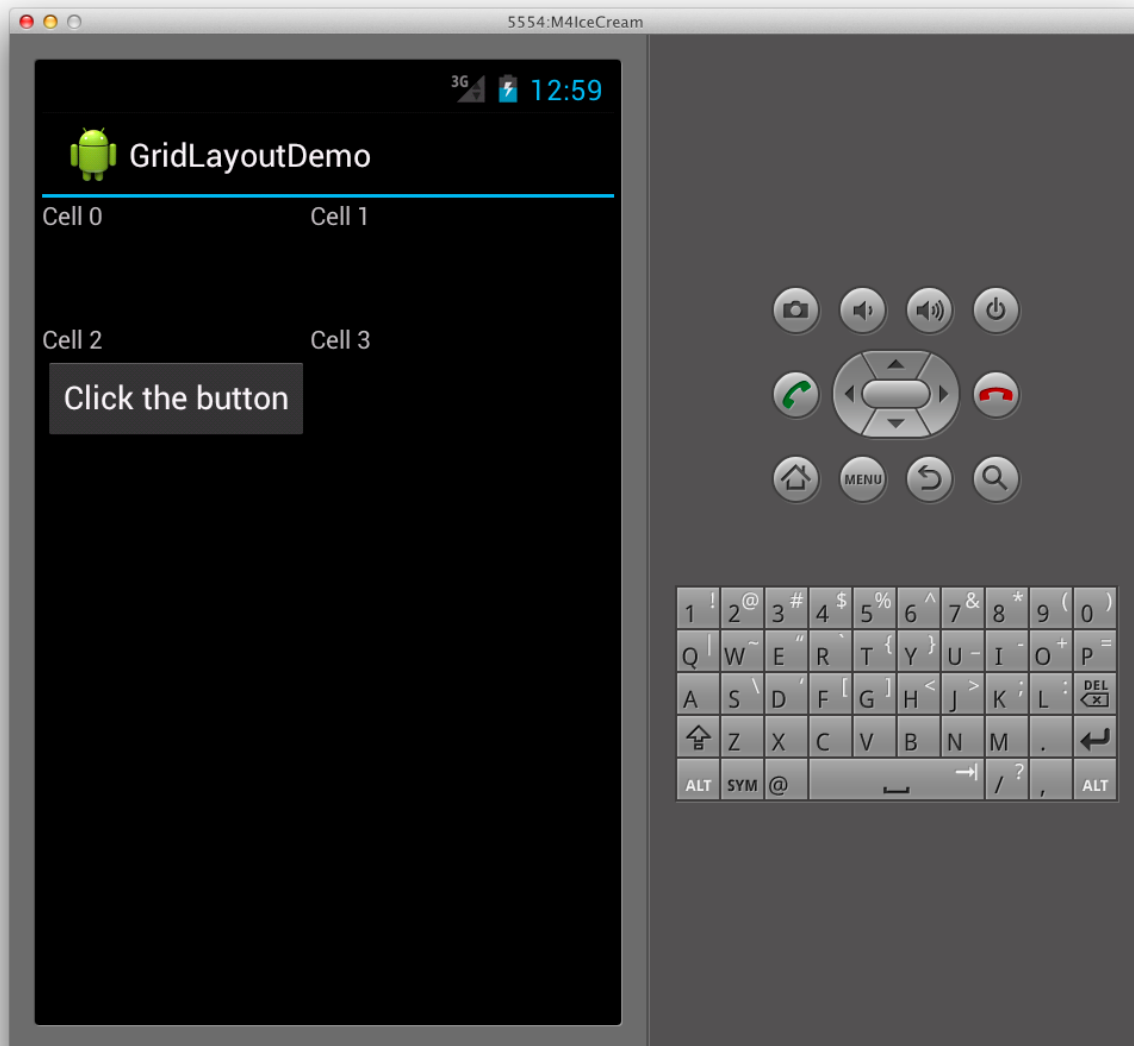
`GridLayout` 还支持跨多个列和行的单元格。例如, 假设我们添加另一行包含一个按钮 `GridLayout`, 如下所示:

```

<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="4"
    android:columnCount="2"
    android:orientation="vertical">
    <TextView
        android:text="Cell 0"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="0" />
    <TextView
        android:text="Cell 1"
        android:textSize="14dip"
        android:layout_row="0"
        android:layout_column="1" />
    <Space
        android:layout_row="1"
        android:layout_column="0"
        android:layout_width="50dp"
        android:layout_height="50dp" />
    <TextView
        android:text="Cell 2"
        android:textSize="14dip"
        android:layout_row="2"
        android:layout_column="0" />
    <TextView
        android:text="Cell 3"
        android:textSize="14dip"
        android:layout_row="2"
        android:layout_column="1" />
    <Button
        android:id="@+id/myButton"
        android:text="@string/hello"
        android:layout_row="3"
        android:layout_column="0" />
</GridLayout>

```

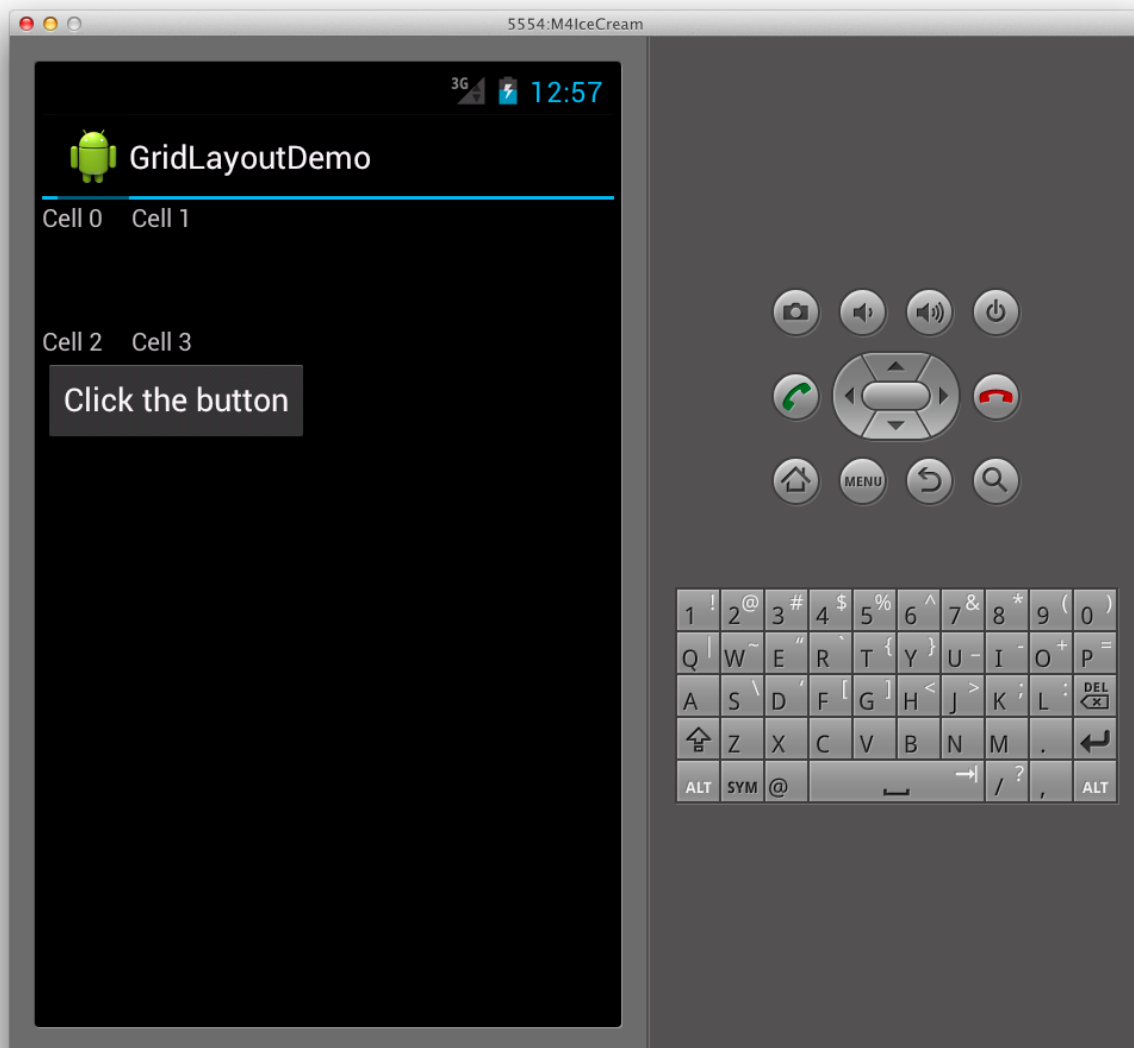
这将导致的第一列 `GridLayout` 被拉伸以适应大小的按钮，正如我们在此处看到：



若要防止拉伸的第一列，我们可以设置按钮以通过设置此类及其 `columnspan` 跨越两个列：

```
<Button
    android:id="@+id/myButton"
    android:text="@string/hello"
    android:layout_row="3"
    android:layout_column="0"
    android:layout_columnSpan="2" />
```

执行此操作导致的布局 `TextViews` 类似于我们已经有了更早版本，使用添加到底部的按钮布局 `GridLayout`，如下所示：



相关链接

- [GridLayoutDemo \(示例\)](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

选项卡式的布局

2018/10/26 • [Edit Online](#)

概述

选项卡是由于其简单性和可用性的移动应用程序中的常见用户界面模式。它们提供一致、简单的方法的应用程序中的各种屏幕之间进行导航。Android 有几个 API 的选项卡式接口：

- **ActionBar** –这是一组新的 API 的目标是提供一致的 Android 3.0 (API 级别为 11) 中引入的一部分导航和切换视图的接口。已为 Android 2.2 (API 级别 8) 与返回移植[Android 支持库 v7](#)。
- **PagerTabStrip** –指示当前、下一步, 和上一个页面的 `ViewPager`。 `ViewPager` 仅可[Android 支持库 v4](#)。有关详细信息 `PagerTabStrip`, 请参阅[ViewPager](#)。
- **工具栏** – `Toolbar` 是取代的更新、更灵活的操作栏组件 `ActionBar`。 `Toolbar` 在 Android 5.0 Lollipop 或更高版本, 并且仍可用于较旧版本的 Android 通过[Android 支持库 v7](#) NuGet 包。 `Toolbar` 目前在 Android 应用中使用的建议的操作栏组件。有关详细信息, 请参阅[工具栏](#)。

相关链接

- [材料设计-选项卡- ActionBar](#)
- [Android 支持库 v7 AppCompat NuGet 包](#)
- [v7 appcompat 库](#)

使用 ActionBar 的选项卡式的布局

2018/11/13 • [Edit Online](#)

本指南介绍, 并介绍了如何使用 ActionBar Api 创建 Xamarin.Android 应用程序中的选项卡式的用户界面。

概述

在操作栏是 Android UI 模式, 它用于为选项卡、应用程序标识、菜单和搜索等主要功能提供一致的用户界面。在 Android 3.0 (API 级别为 11) 中, Google 引入了 ActionBar Api 向 Android 平台。ActionBar Api 引入了用户界面主题, 以提供一致的外观和允许的选项卡式的用户界面的类。本指南介绍如何将操作栏选项卡添加到 Xamarin.Android 应用程序。它还讨论了如何使用 Android 支持库 v7 到向后移植到面向 Android 2.1 到 Android 2.3 的 Xamarin.Android 应用程序的操作栏选项卡。

请注意, `Toolbar` 是一个更高版本和更通用的操作栏组件, 应使用而不是 `ActionBar` (`Toolbar` 旨在替换 `ActionBar`)。有关详细信息, 请参阅[工具栏](#)。

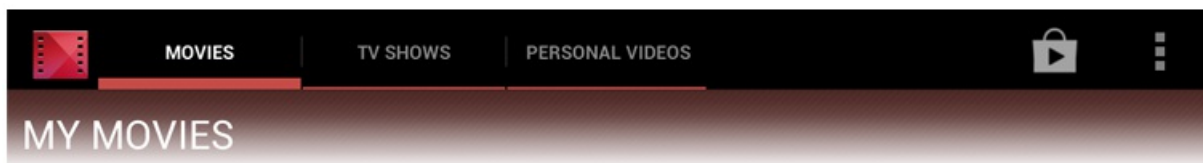
要求

所有 Xamarin.Android 应用程序面向 API 级别 11 (Android 3.0) 或更高版本的本机 Android Api 一部分中有权访问 ActionBar Api。

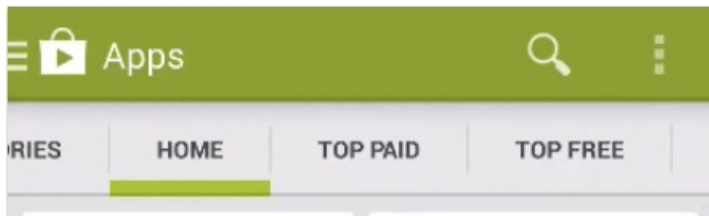
某些 ActionBar Api 返回移植到了 API 级别 7 (Android 2.1), 通过[V7 AppCompat 库](#), 可由适用于 Xamarin.Android 应用通过[Xamarin Android 支持库-V7](#)包。

引入了操作栏中的选项卡

在操作栏尝试同时显示所有其选项卡并使所有选项卡的大小基于最宽的选项卡标签的宽度相同。这是下面的屏幕截图所示:



当 ActionBar 无法显示所有选项卡后时, 它将设置可水平滚动视图中的选项卡。左侧或右侧才能看到其余的选项卡, 用户可能往下轻扫。从 Google Play 此屏幕截图显示了此示例:



应与之关联的操作栏中的每个选项卡 [片段](#)。当用户选择一个选项卡时, 则应用程序将显示选项卡与相关联的片段。ActionBar 不负责向用户显示相应的片段。相反, ActionBar 会通知应用程序通过实现 `ActionBar.ITabListener` 接口的类的选项卡中的状态更改。此接口提供了三个选项卡的状态更改时, Android 将调用的回调方法:

- **OnTabSelected** - 当用户选择选项卡时调用此方法。它应该会显示该片段。
- **OnTabReselected** - 当选项卡已被选中, 但用户再次选择时调用此方法。通常使用此回叫来刷新/更新显示的片段。

- **OnTabUnselected** - 当用户选择另一个选项卡时调用此方法。此回调用于将状态保存在显示的片段中之前就会消失。

Xamarin.Android 包装 `ActionBar.ITabListener` 上的事件与 `ActionBar.Tab` 类。应用程序可能会将事件处理程序分配给一个或多个这些事件。有三个事件（一个用于在每个方法 `ActionBar.ITabListener`）将引发操作栏选项卡：

- `TabSelected`
- `TabReselected`
- `TabUnselected`

向 `ActionBar` 添加选项卡

`ActionBar` 是本地 Android 3.0（API 级别为 11）和更高版本，并且可供任何针对此 API 是一个最少的 Xamarin.Android 应用程序。

以下步骤演示了如何将 `ActionBar` 选项卡添加到 Android 活动：

1. 在中 `OnCreate` 方法的活动-之前初始化任何 UI 小部件-应用程序必须设置 `NavigationMode` 上 `ActionBar` 到 `ActionBar.NavigationModeTabs` 此代码中所示代码片段：

```
ActionBar.NavigationMode = ActionBarNavigationMode.Tabs;
SetContentView(Resource.Layout.Main);
```

2. 创建新选项卡上使用 `ActionBar.NewTab()`。
3. 分配事件处理程序或提供自定义 `ActionBar.ITabListener` 将会响应用户交互操作栏选项卡时引发的事件的实现。
4. 添加到在上一步中创建的选项卡 `ActionBar`。

以下代码是使用以下步骤将选项卡添加到使用事件处理程序的状态更改进行响应的应用程序的一个示例：

```
protected override void OnCreate(Bundle bundle)
{
    ActionBar.NavigationMode = ActionBarNavigationMode.Tabs;
    SetContentView(Resource.Layout.Main);

    ActionBar.Tab tab = ActionBar.NewTab();
    tab.SetText(Resources.GetString(Resource.String.tab1_text));
    tab.SetIcon(Resource.Drawable.tab1_icon);
    tab.TabSelected += (sender, args) => {
        // Do something when tab is selected
    };
    ActionBar.AddTab(tab);

    tab = ActionBar.NewTab();
    tab.SetText(Resources.GetString(Resource.String.tab2_text));
    tab.SetIcon(Resource.Drawable.tab2_icon);
    tab.TabSelected += (sender, args) => {
        // Do something when tab is selected
    };
    ActionBar.AddTab(tab);
}
```

事件处理程序 vs `ActionBar.ITabListener`

应用程序应使用事件处理程序和 `ActionBar.ITabListener` 为不同的方案。事件处理程序确实提供一定量的语法上方便;他们无需创建一个类并实现保存您 `ActionBar.ITabListener`。这种便利确实代价-Xamarin.Android 执行此转换的创建一个类和实现 `ActionBar.ITabListener` 为您。当应用程序具有有限的数量的选项卡时，这是没问题。

多个选项卡上，在处理时共享操作栏选项卡之间的常见功能，也可以是内存和性能，以创建一个自定义类，实现更高

效 `ActionBar.ITabListener`，和共享单个类的实例。这将减少 GREF 的 Xamarin.Android 应用程序使用的数目。

向后兼容性较旧的设备

[Android 支持库 v7 AppCompat](#) 后端口操作栏选项卡添加到 Android 2.1（API 级别 7）。此组件添加到项目后，选项卡是可在 Xamarin.Android 应用程序中访问。

若要使用 ActionBar，活动必须子类 `ActionBarActivity` 和使用 AppCompat 主题，如下面的代码段中所示：

```
[Activity(Label = "@string/app_name", Theme = "@style/Theme.AppCompat", MainLauncher = true, Icon =
"@drawable/ic_launcher")]
public class MainActivity: ActionBarActivity
```

活动可能会获取对从其 ActionBar 的引用 `ActionBarActivity.SupportActionBar` 属性。下面的代码段说明了设置 ActionBar 在活动中的示例：

```
[Activity(Label = "@string/app_name", Theme = "@style/Theme.AppCompat", MainLauncher = true, Icon =
"@drawable/ic_launcher")]
public class MainActivity : ActionBarActivity, ActionBar.ITabListener
{
    static readonly string Tag = "ActionBarTabsSupport";

    public void OnTabReselected(ActionBar.Tab tab, FragmentTransaction ft)
    {
        // Optionally refresh/update the displayed tab.
        Log.Debug(Tag, "The tab {0} was re-selected.", tab.Text);
    }

    public void OnTabSelected(ActionBar.Tab tab, FragmentTransaction ft)
    {
        // Display the fragment the user should see
        Log.Debug(Tag, "The tab {0} has been selected.", tab.Text);
    }

    public void OnTabUnselected(ActionBar.Tab tab, FragmentTransaction ft)
    {
        // Save any state in the displayed fragment.
        Log.Debug(Tag, "The tab {0} as been unselected.", tab.Text);
    }

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SupportActionBar.NavigationMode = ActionBar.NavigationModeTabs;
        SetContentView(Resource.Layout.Main);
    }

    void AddTabToActionBar(int labelResourceId, int iconResourceId)
    {
        ActionBar.Tab tab = SupportActionBar.NewTab()
            .SetText(labelResourceId)
            .SetIcon(iconResourceId)
            .SetTabListener(this);

        SupportActionBar.AddTab(tab);
    }
}
```

总结

在本指南中我们讨论了如何在 Xamarin.Android 中使用 ActionBar 创建选项卡式的用户界面。我们介绍了如何将选项卡添加到 ActionBar 和活动与通过选项卡事件的交互方式 `ActionBar.ITabListener` 接口。我们还了解到 Android 支持库 v7 AppCompat 包 backports ActionBar 与旧版本的 Android 的选项卡。

相关链接

- [ActionBarTabs \(示例\)](#)
- [工具栏](#)
- [片段](#)
- [ActionBar](#)
- [ActionBarActivity](#)
- [操作栏模式](#)
- [Android v7 AppCompat](#)
- [Xamarin.Android 支持库 v7 AppCompat NuGet 包](#)

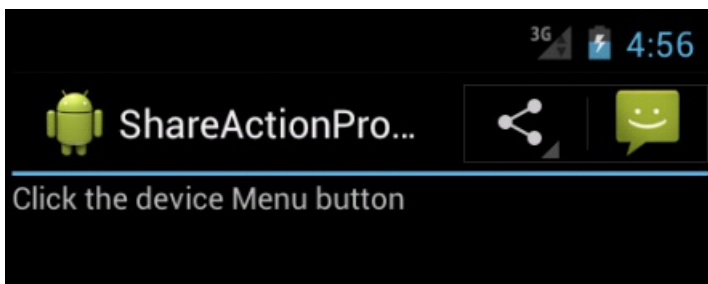
Android 控件（小组件）

2018/10/25 • [Edit Online](#)

Xamarin.Android 会公开所有由 Android 提供的本机用户界面控件（小组件）。使用 Android 设计器的 Xamarin.Android 应用，或以编程方式通过 XML 布局文件，可以轻松地添加这些控件。无论选择哪种方法，Xamarin.Android 公开的所有用户界面对象属性和 C# 中的方法。以下各节介绍最常见的 Android 用户界面控件，并说明如何将它们合并到 Xamarin.Android 应用。

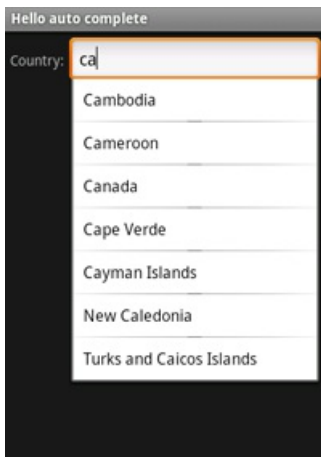
ActionBar

`ActionBar` 是活动标题、导航接口和其他交互式项将显示一个工具栏。通常情况下，`ActionBar` 显示在活动的窗口的顶部。



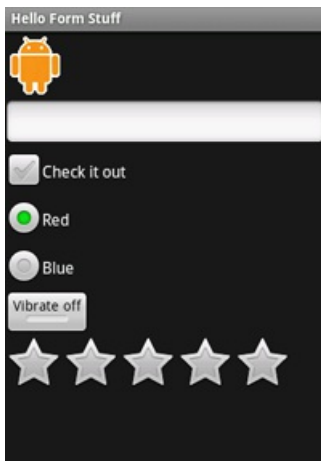
AutoCompleteTextView

`AutoCompleteTextView` 是一个可编辑的文本视图元素，当用户输入时自动显示完成建议。下拉列表菜单，用户可以选择要替换的编辑框的内容的项中显示的建议列表。



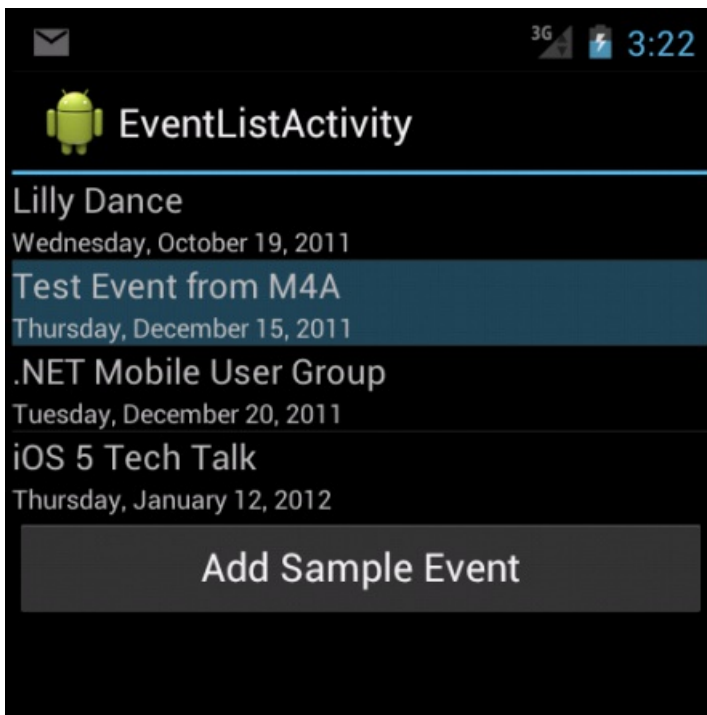
Button

`Button` 是用户点击执行操作的 UI 元素。



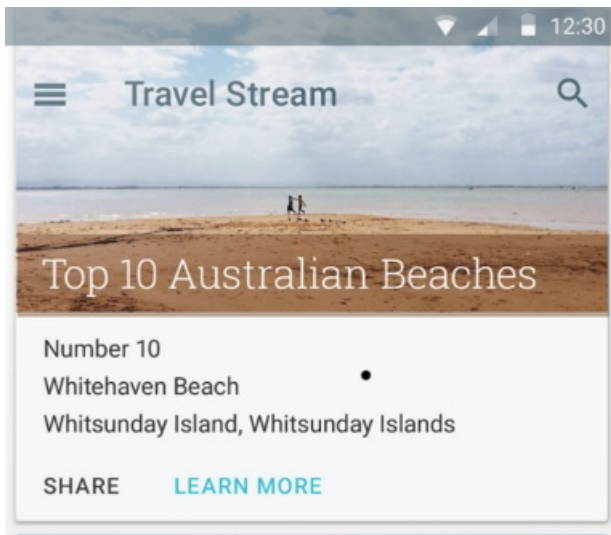
Calendar

`Calendar` 类用于转换中的特定实例的值，例如年、月、小时、数月，日和下一周的日期时间（偏离 epoch 的毫秒值）。`Calendar` 支持丰富的交互选项的日历数据，包括能够读取和写入事件、与会者和提醒。通过在应用程序中使用的日历提供程序，通过 API 添加的数据将出现在附带了 Android 内置日历应用程序。



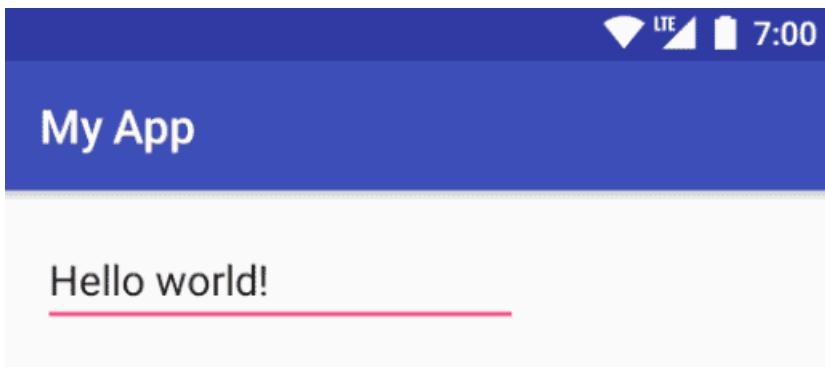
CardView

`CardView` 是类似于卡的视图中显示的文本和图像内容的 UI 组件。`CardView` 作为实现 `FrameLayout` 带有圆的角和阴影的小组件。通常情况下，`CardView` 用来表示中的单个行项 `ListView` 或 `GridView` 查看组。



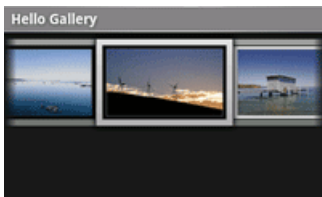
EditText

`EditText` 是用于输入和修改文本的 UI 元素。



Gallery

`Gallery` 是用于在水平滚动的列表; 中显示项的布局小组件它将在视图的中心定位在当前所选内容。



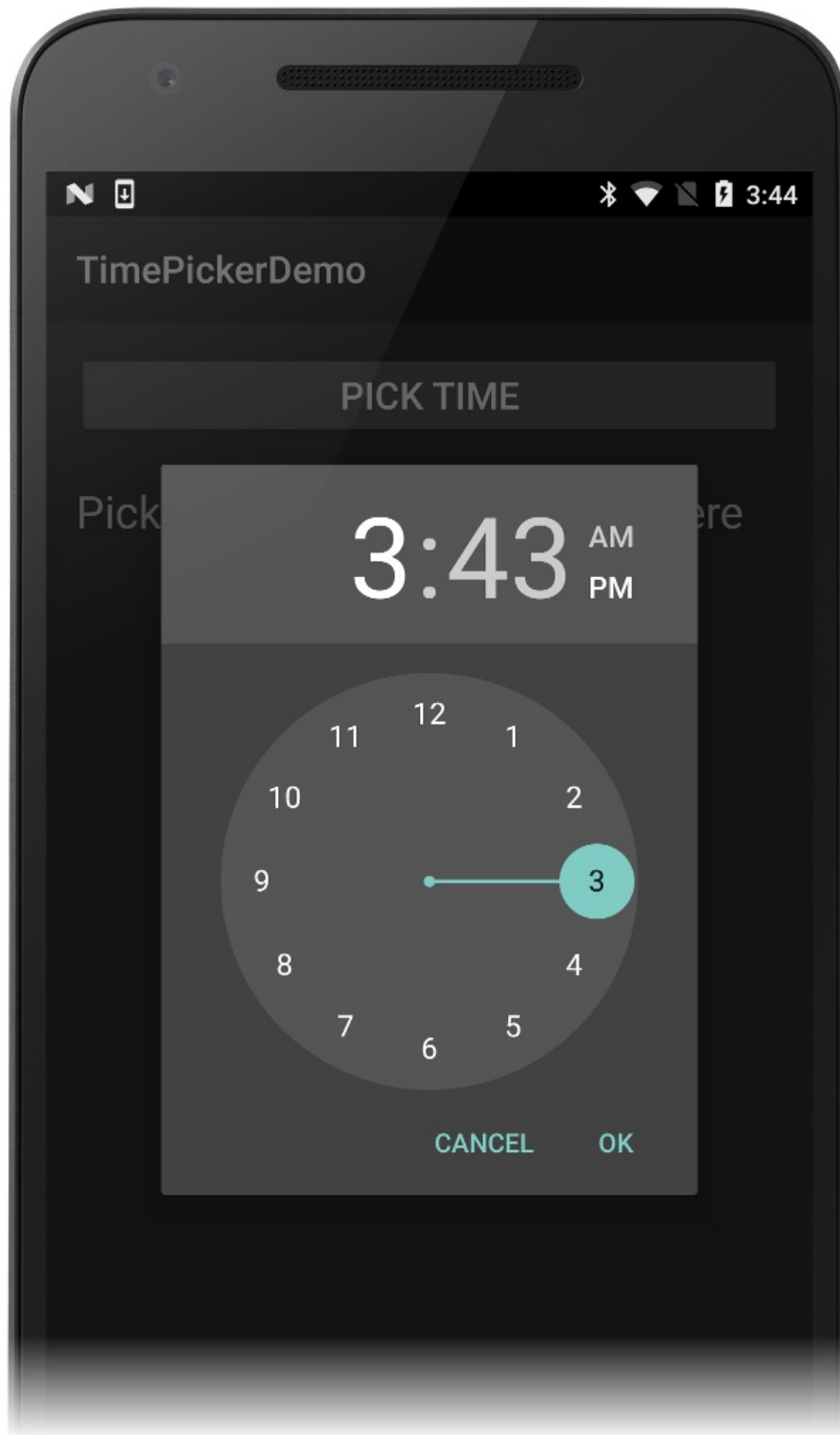
Navigation Bar

Navigation Bar 提供在不包括有关的硬件按钮的设备上的导航控件主页，返回，和菜单。



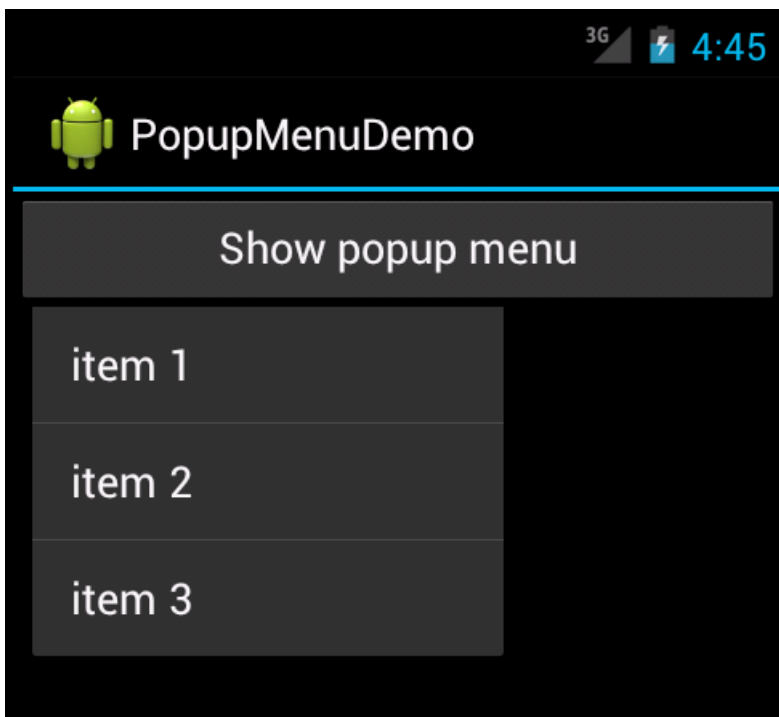
Picker

*选取器*是允许用户通过使用由 Android 提供的对话框中选择一个日期或时间的 UI 元素。



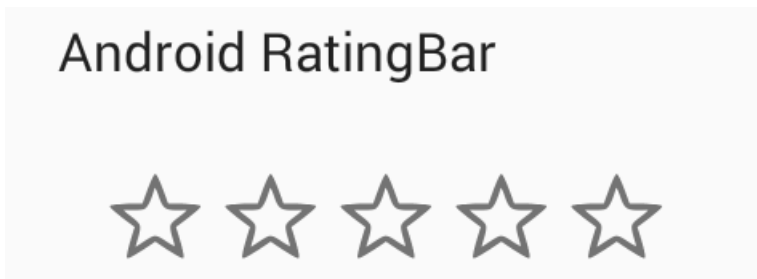
PopupMenu

`PopupMenu` 用于显示附加到特定视图的弹出菜单。



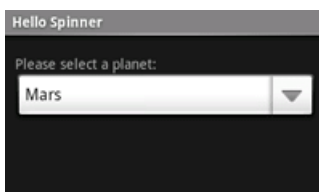
RatingBar

一个 `RatingBar` 是显示在星评级的 UI 元素。



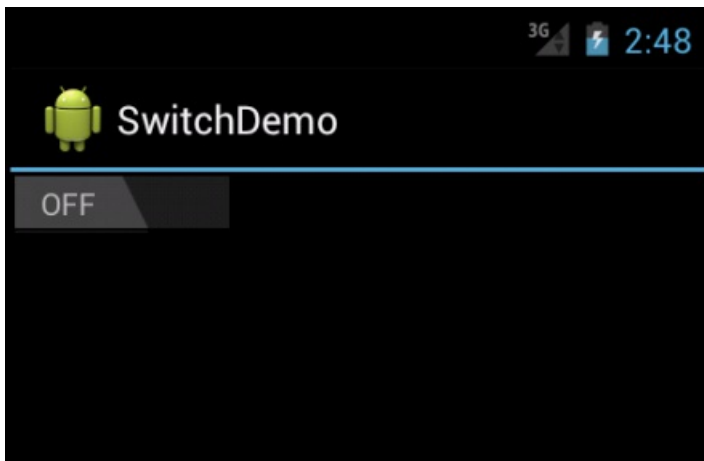
Spinner

`Spinner` 是，可以快速从一组选择一个值的 UI 元素。它是 similar 到下拉列表。



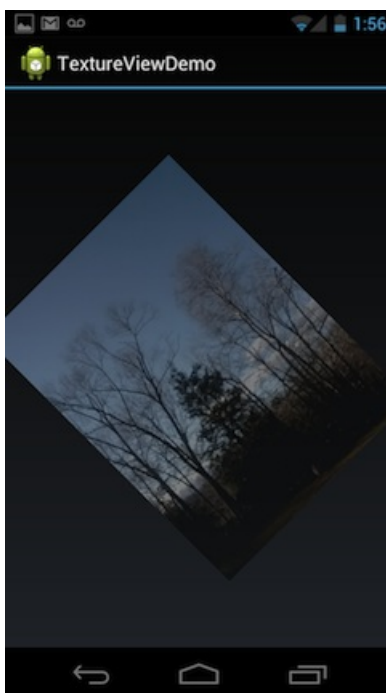
Switch

`Switch` 是允许用户以两个状态，例如在之间切换或关闭的 UI 元素。 `Switch` 默认值为 OFF。



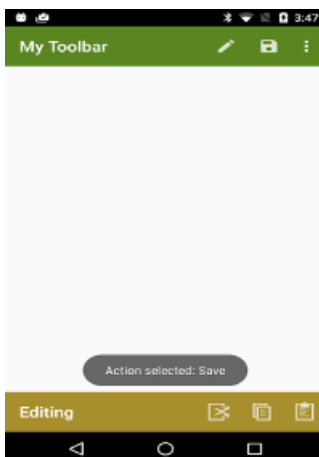
TextureView

`TextureView` 是一个使用硬件加速 2D 呈现若要启用的视频或 OpenGL 内容流, 要显示的视图。



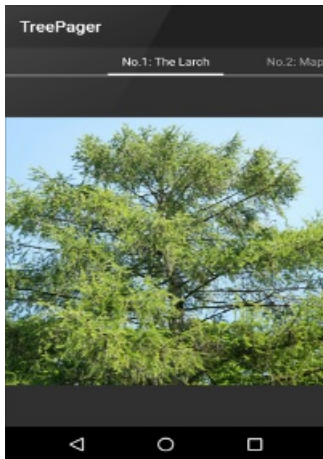
Toolbar

`Toolbar` (在 Android 5.0 Lollipop 中引入) 的组件可以看作操作栏接口的泛化-它旨在替换操作栏。`Toolbar` 可以应用的布局中, 在任何地方使用, 它更自定义操作栏比。



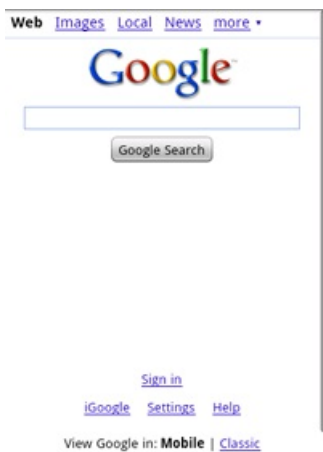
ViewPager

ViewPager 是允许用户翻转左侧和右侧的数据页的布局管理器。



WebView

WebView 是一个 UI 元素，您可以创建自己的窗口用于查看的网页（或甚至开发完整的浏览器）。



操作栏

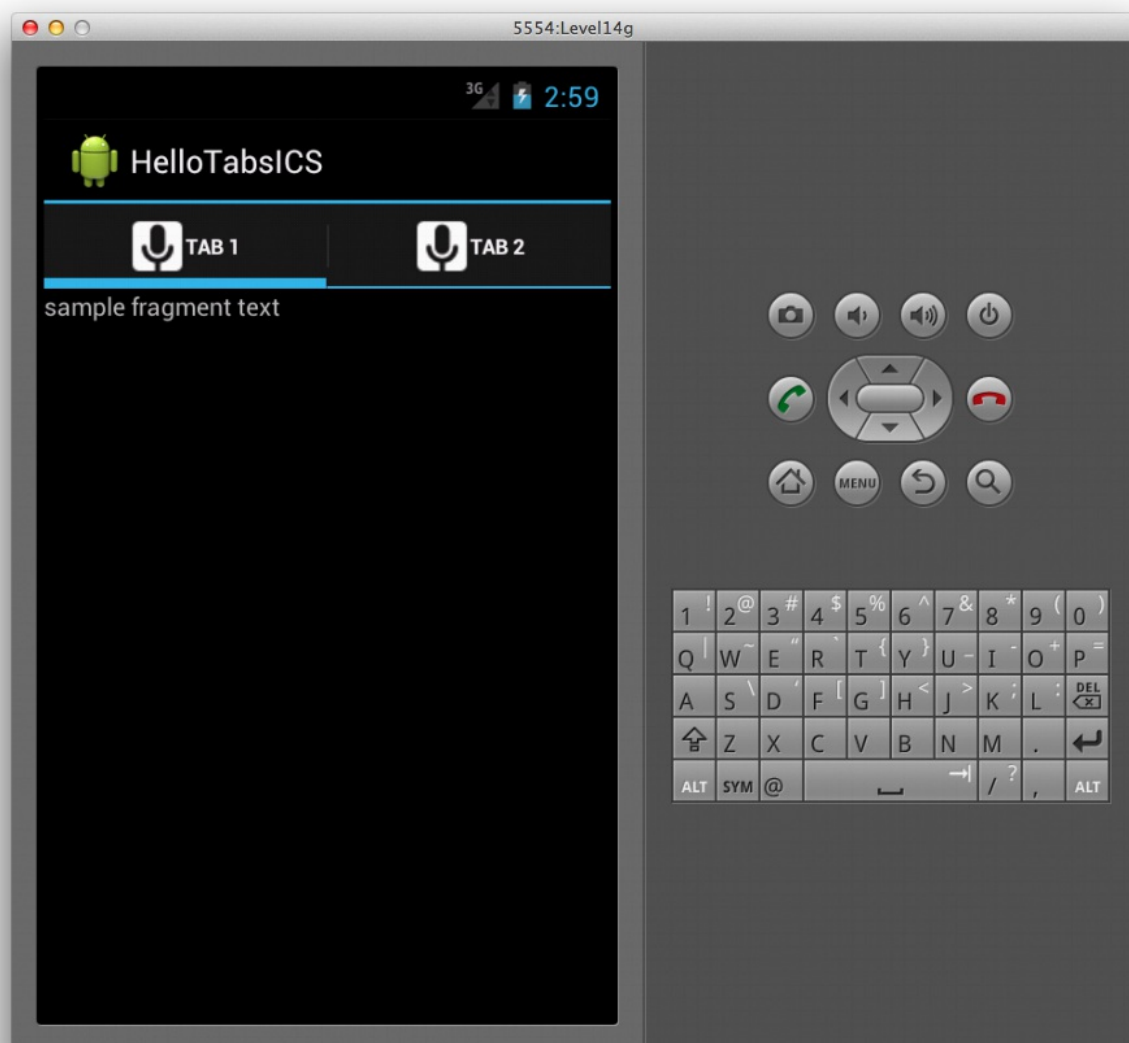
2018/10/26 • [Edit Online](#)

概述

当使用 `TabActivity`，代码以创建选项卡图标具有针对 Android 4.0 框架运行时不起作用。尽管它在功能上非常像以前那样在 2.3 开始之前, 的 Android 版本 `TabActivity` 类本身已在 4.0 中弃用。创建选项卡式的界面的新方法引入了使用操作栏中, 我们接下来要讨论的。

操作栏选项卡

在操作栏包括支持 Android 4.0 中添加选项卡式的接口。以下屏幕截图显示了此类接口的示例。



若要在操作栏中创建选项卡, 我们首先需要设置其 `NavigationMode` 属性以支持的选项卡。在 Android 4 中, `ActionBar` 属性是可在活动类, 我们可以用来设置上 `NavigationMode` 如下所示:

```
this.ActionBar.NavigationMode = ActionBarNavigationMode.Tabs;
```

完成此操作后，我们可以通过调用创建一个选项卡 `NewTab` 操作栏上的方法。使用此选项卡实例，我们可以调用 `SetText` 和 `setIcon` 方法来设置选项卡的标签文本和图标，如下所示的代码中按顺序进行这些调用：

```
var tab = this.ActionBar.NewTab ();
tab.SetText (tabText);
tab.SetIcon (Resource.Drawable.ic_tab_white);
```

但是，我们可以添加选项卡之前，我们需要处理 `TabSelected` 事件。在此处理程序，我们可以创建选项卡的内容。操作栏选项卡用于处理片段，表示在活动中的用户界面的一部分的类。对于此示例中，片段的视图包含单个 `TextView`，其中我们放在大量我们 `Fragment` 子类如下：

```
class SampleTabFragment: Fragment
{
    public override View onCreateView (LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState)
    {
        base.onCreateView (inflater, container, savedInstanceState);

        var view = inflater.Inflate (
            Resource.Layout.Tab, container, false);

        var sampleTextView =
            view.FindViewById<TextView> (Resource.Id.sampleTextView);
        sampleTextView.Text = "sample fragment text";

        return view;
    }
}
```

事件自变量传入 `TabSelected` 事件属于类型 `TabEventArgs`，其中包括 `FragmentManager` 属性，我们可以使用添加该片段，如下所示：

```
tab.TabSelected += delegate(object sender, ActionBar.TabEventArgs e) {
    e.FragmentManager.Add (Resource.Id.fragmentContainer,
        new SampleTabFragment ());
};
```

最后，我们可以将标签添加到操作栏通过调用 `AddTab` 方法在此代码中所示：

```
this.ActionBar.AddTab (tab);
```

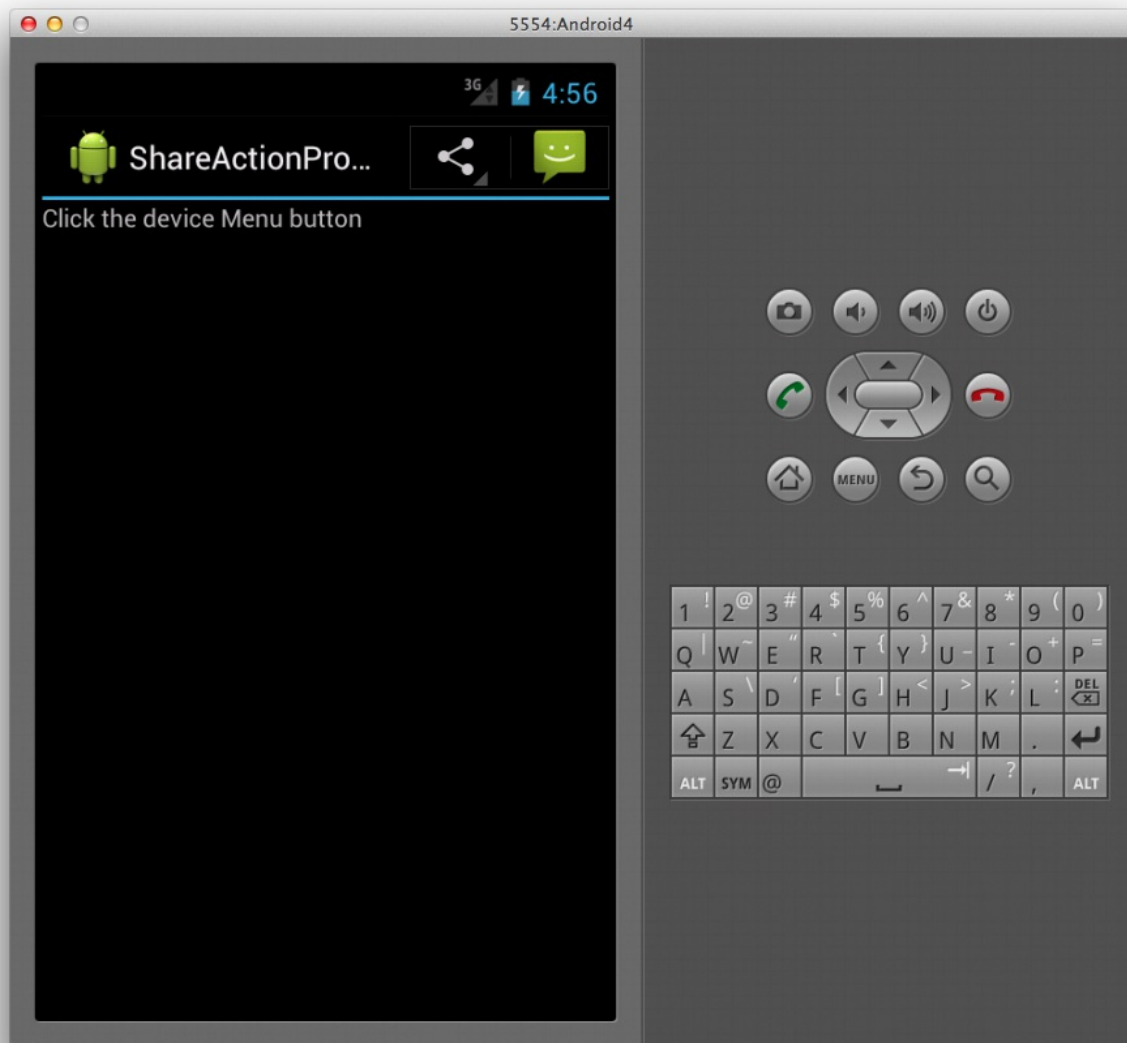
有关完整示例，请参阅 *Hello Tabs/CS* 本文档的示例代码中的项目。

ShareActionProvider

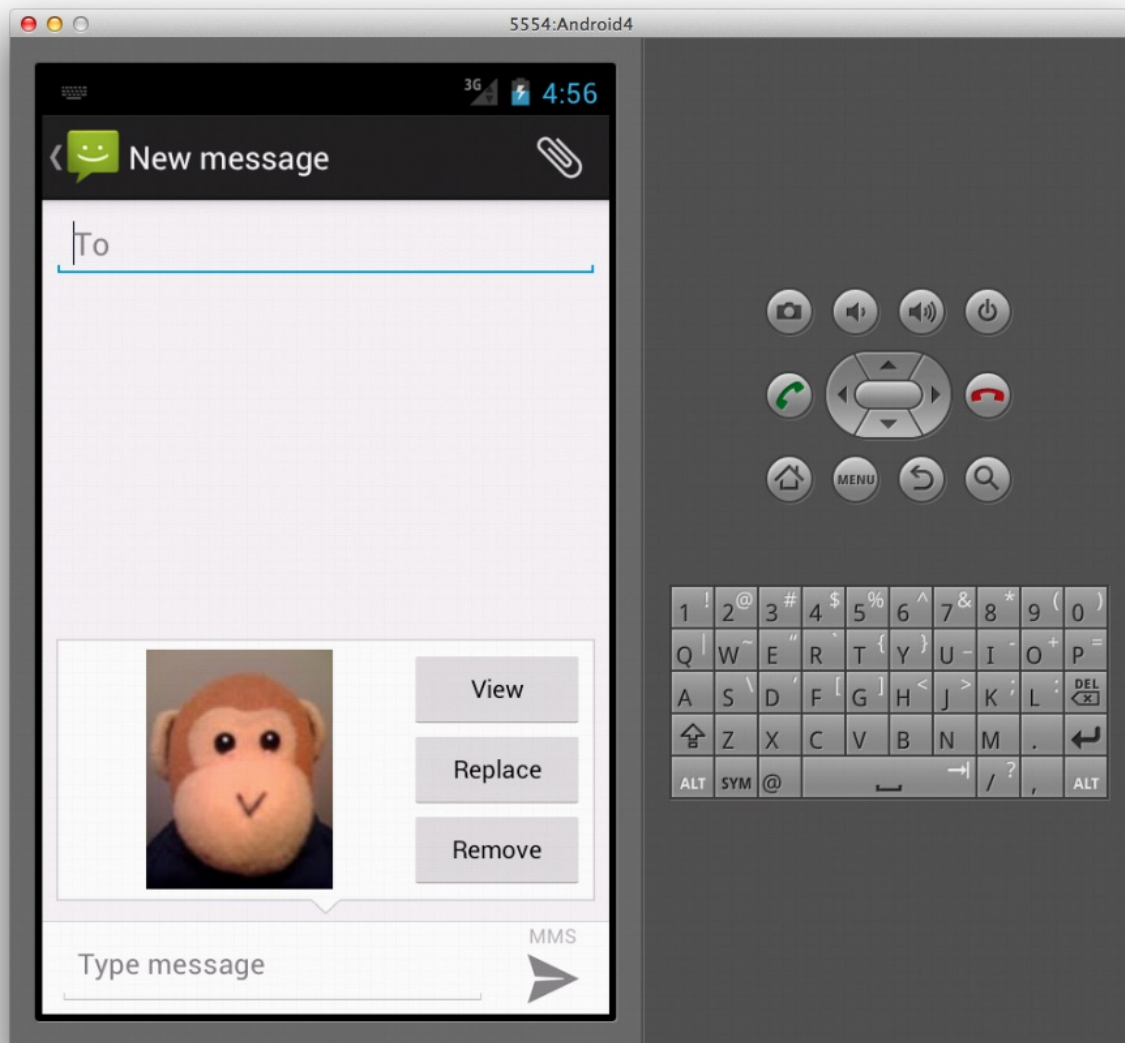
`ShareActionProvider` 类，用于执行操作栏中的共享操作。它负责使用的应用可以处理共享意向并从操作栏更高版本到这些保留的以前用过的应用程序，以方便访问的历史记录列表创建操作视图。这允许应用程序通过在 Android 保持一致的用户体验来共享数据。

映像共享示例

例如，下面是操作栏与菜单项来共享映像的屏幕截图 (摘自 [ShareActionProvider](#) 示例)。当用户点击操作栏上的菜单项时，`ShareActionProvider` 加载应用程序能够处理与关联的意向 `ShareActionProvider`。在此示例中，消息传送应用程序之前使用，以便在操作栏上显示。



当用户单击操作栏中的项时，会启动消息传递应用，其中包含共享的映像，如下所示：



指定的操作提供程序类

若要使用 `ShareActionProvider`，请将 `android:actionProviderClass` 属性在操作栏菜单的 XML 中的菜单项，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/shareMenuItem"
        android:showAsAction="always"
        android:title="@string/sharePicture"
        android:actionProviderClass="android.widget.ShareActionProvider" />
</menu>
```

以下菜单

若要放大量菜单上，我们重写 `onCreateOptionsMenu` 活动子类中。对菜单上的引用后，我们可以获取

`ShareActionProvider` 从 `ActionProvider` 属性的菜单项，然后使用 `SetShareIntent` 方法以设置 `ShareActionProvider` 的意图，如下所示：

```
public override bool OnCreateOptionsMenu (IMenu menu)
{
    MenuInflater.Inflate (Resource.Menu.ActionBarMenu, menu);

    var shareMenuItem = menu.FindItem (Resource.Id.shareMenuItem);
    var shareActionProvider =
        (ShareActionProvider)shareMenuItem.ActionProvider;
    shareActionProvider.SetShareIntent (CreateIntent ());
}
```

创建意向

`ShareActionProvider` 将使用意向，传递给 `SetShareIntent` 在上面的代码，以启动适当的活动的方法。在这种情况下，我们创建使用以下代码将发送映像的意图：

```
Intent CreateIntent ()
{
    var sendPictureIntent = new Intent (Intent.ActionSend);
    sendPictureIntent.SetType ("image/*");
    var uri = Android.Net.Uri.FromFile (GetFileStreamPath ("monkey.png"));
    sendPictureIntent.PutExtra (Intent.ExtraStream, uri);
    return sendPictureIntent;
}
```

包含为与应用程序的资产并复制到可公开访问的位置时创建活动，使其可供其他应用程序，如消息传递应用上面的代码示例中的图像。本文附带的示例代码包含此示例中，说明其用途的完整源代码。

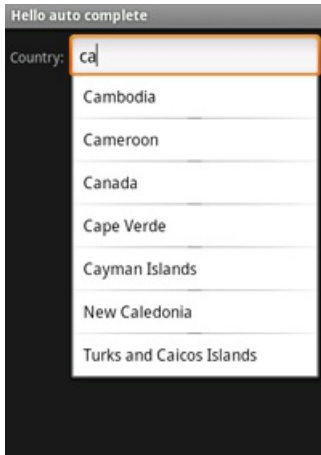
相关链接

- [Hello 选项卡 ICS（示例）](#)
- [ShareActionProvider 演示（示例）](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

AutoCompleteTextView

2018/10/25 • [Edit Online](#)

`AutoCompleteTextView` 是一个可编辑的文本视图元素，当用户输入时自动显示完成建议。下拉列表菜单，用户可以选择要替换的编辑框的内容的项中显示的建议列表。



概述

若要创建一个文本项小组件，提供自动完成的建议，请使用 `AutoCompleteTextView` 小组件。建议从与通过小组件相关联的字符串的集合中接收 `ArrayAdapter` 。

在本教程中，您将创建 `AutoCompleteTextView` 提供有关国家/地区名称的建议的小组件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Country" />
    <AutoCompleteTextView android:id="@+id/autocomplete_country"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"/>
</LinearLayout>
```

`TextView` 是引入了一个标签 `AutoCompleteTextView` 小组件。

教程

启动一个名为的新项目 *HelloAutoComplete*。

创建一个名为 `list_item.xml` 的 XML 文件，然后将其保存在 **Resources/Layout** 文件夹中。将此文件的“生成操作”设置为 `AndroidResource`。编辑文件，完成后应如下所示：

```
<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp"
    android:textColor="#000">

</TextView>
```

此文件定义了一个简单 `TextView` 将用于在建议列表中显示的每个项。

打开 **Resources/Layout/Main.xml** 并插入以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Country" />
    <AutoCompleteTextView android:id="@+id/autocomplete_country"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"/>
</LinearLayout>
```

打开 **MainActivity.cs** 并插入以下代码 `OnCreate()` 方法：

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Set our view from the "Main" layout resource
    SetContentView (Resource.Layout.Main);

    AutoCompleteTextView textView = FindViewById<AutoCompleteTextView> (Resource.Id.autocomplete_country);
    var adapter = new ArrayAdapter<String> (this, Resource.Layout.list_item, COUNTRIES);

    textView.Adapter = adapter;
}
```

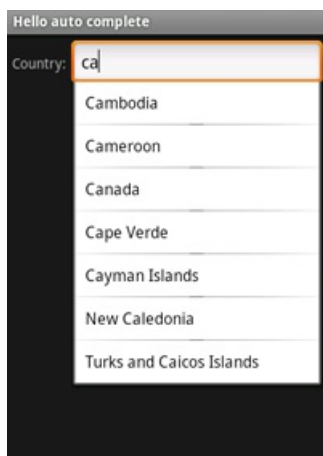
内容视图设置为之后 `main.xml` 布局，`AutoCompleteTextView` 从与布局捕获小组件 `FindViewById`。一个新 `ArrayAdapter` 然后初始化绑定 `list_item.xml` 中每个列表项的布局 `COUNTRIES`（在下一步中定义）的字符串数组。最后，`SetAdapter()` 调用以将相关联 `ArrayAdapter` 与 `AutoCompleteTextView` 小组件，以便字符串数组将填充建议列表。

内部 `MainActivity` 类中，添加的字符串数组：

```
static string[] COUNTRIES = new string[] {
    "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra",
    "Angola", "Anguilla", "Antarctica", "Antigua and Barbuda", "Argentina",
    "Armenia", "Aruba", "Australia", "Austria", "Azerbaijan",
    "Bahrain", "Bangladesh", "Barbados", "Belarus", "Belgium",
    "Belize", "Benin", "Bermuda", "Bhutan", "Bolivia",
    "Bosnia and Herzegovina", "Botswana", "Bouvet Island", "Brazil", "British Indian Ocean Territory",
    "British Virgin Islands", "Brunei", "Bulgaria", "Burkina Faso", "Burundi",
    "Cote d'Ivoire", "Cambodia", "Cameroon", "Canada", "Cape Verde",
    "Cayman Islands", "Central African Republic", "Chad", "Chile", "China",
    "Christmas Island", "Cocos (Keeling) Islands", "Colombia", "Comoros", "Congo",
    "Cook Islands", "Costa Rica", "Croatia", "Cuba", "Cyprus", "Czech Republic",
    "Democratic Republic of the Congo", "Denmark", "Djibouti", "Dominica", "Dominican Republic",
    "East Timor", "Ecuador", "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea",
    "Estonia", "Ethiopia", "Faeroe Islands", "Falkland Islands", "Fiji", "Finland",
    "Former Yugoslav Republic of Macedonia", "France", "French Guiana", "French Polynesia",
    "French Southern Territories", "Gabon", "Georgia", "Germany", "Ghana", "Gibraltar",
    "Greece", "Greenland", "Grenada", "Guadeloupe", "Guam", "Guatemala", "Guinea", "Guinea-Bissau",
    "Guyana", "Haiti", "Heard Island and McDonald Islands", "Honduras", "Hong Kong", "Hungary",
    "Iceland", "India", "Indonesia", "Iran", "Iraq", "Ireland", "Israel", "Italy", "Jamaica",
    "Japan", "Jordan", "Kazakhstan", "Kenya", "Kiribati", "Kuwait", "Kyrgyzstan", "Laos",
    "Latvia", "Lebanon", "Lesotho", "Liberia", "Libya", "Liechtenstein", "Lithuania", "Luxembourg",
    "Macau", "Madagascar", "Malawi", "Malaysia", "Maldives", "Mali", "Malta", "Marshall Islands",
    "Martinique", "Mauritania", "Mauritius", "Mayotte", "Mexico", "Micronesia", "Moldova",
    "Monaco", "Mongolia", "Montserrat", "Morocco", "Mozambique", "Myanmar", "Namibia",
    "Nauru", "Nepal", "Netherlands", "Netherlands Antilles", "New Caledonia", "New Zealand",
    "Nicaragua", "Niger", "Nigeria", "Niue", "Norfolk Island", "North Korea", "Northern Marianas",
    "Norway", "Oman", "Pakistan", "Palau", "Panama", "Papua New Guinea", "Paraguay", "Peru",
    "Philippines", "Pitcairn Islands", "Poland", "Portugal", "Puerto Rico", "Qatar",
    "Reunion", "Romania", "Russia", "Rwanda", "Sgo Tome and Principe", "Saint Helena",
    "Saint Kitts and Nevis", "Saint Lucia", "Saint Pierre and Miquelon",
    "Saint Vincent and the Grenadines", "Samoa", "San Marino", "Saudi Arabia", "Senegal",
    "Seychelles", "Sierra Leone", "Singapore", "Slovakia", "Slovenia", "Solomon Islands",
    "Somalia", "South Africa", "South Georgia and the South Sandwich Islands", "South Korea",
    "Spain", "Sri Lanka", "Sudan", "Suriname", "Svalbard and Jan Mayen", "Swaziland", "Sweden",
    "Switzerland", "Syria", "Taiwan", "Tajikistan", "Tanzania", "Thailand", "The Bahamas",
    "The Gambia", "Togo", "Tokelau", "Tonga", "Trinidad and Tobago", "Tunisia", "Turkey",
    "Turkmenistan", "Turks and Caicos Islands", "Tuvalu", "Virgin Islands", "Uganda",
    "Ukraine", "United Arab Emirates", "United Kingdom",
    "United States", "United States Minor Outlying Islands", "Uruguay", "Uzbekistan",
    "Vanuatu", "Vatican City", "Venezuela", "Vietnam", "Wallis and Futuna", "Western Sahara",
    "Yemen", "Yugoslavia", "Zambia", "Zimbabwe"
};
```

这是当用户键入到将在下拉列表中提供的建议的列表 `AutoCompleteTextView` 小组件。

运行该应用程序。键入时，应看到类似如下：



详细信息

请注意，使用硬编码的字符串数组并不是建议的设计做法，因为应用程序代码应侧重于行为，而不是内容。应用程序内容，如字符串应从代码中简化对内容进行修改并简化本地化内容的外部化。硬编码的字符串在本教程中仅用于进行简单并重点介绍 `AutoCompleteTextView` 小组件。相反，你的应用程序应声明一个 XML 文件中的此类字符串数组。这可以通过 `<string-array>` 在项目中的资源 `res/values/strings.xml` 文件。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Bahrain</item>
        <item>Bangladesh</item>
        <item>Barbados</item>
        <item>Belarus</item>
        <item>Belgium</item>
        <item>Belize</item>
        <item>Benin</item>
    </string-array>
</resources>
```

若要使用这些资源字符串 `ArrayAdapter`，替换原始 `ArrayAdapter` 构造函数使用以下行：

```
string[] countries = Resources.GetStringArray (Resource.array.countries_array);
var adapter = new ArrayAdapter<String> (this, Resource.layout.list_item, countries);
```

参考资料

- [AutoCompleteTextView 食谱](#) – Xamarin.Android 示例项目，以便 `AutoCompleteTextView`。
- `ArrayAdapter`
- `AutoCompleteTextView`

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#)。本教程基于 [Android 自动完成教程](#)。

Xamarin.Android 中的 Button

2018/10/26 • [Edit Online](#)

`Button` 类用来表示 Android 屏幕上各种不同的按钮样式。本部分介绍在 Xamarin 中使用按钮的不同选项：

- `RadioButton` 允许用户从一组中选择一个选项。
- `ToggleButton` 允许用户在两个状态设置之间翻转(切换)。
- `CheckBox` 是一种特殊类型的按钮，可以通过将其选中或取消选中来指示两种可能的状态之一。
- 你还可以创建 [自定义按钮](#)，它使用图片而不是文本。

RadioButton

2018/10/26 • [Edit Online](#)

在本部分中，您将创建两个互斥的单选按钮（启用一个禁用的其他），使用 `RadioGroup` 和 `RadioButton` 小组件。按下任意单选按钮时，将显示的 toast 消息。

打开 **Resources/layout/Main.xml** 文件，并添加两个 `RadioButton` s，嵌套在 `RadioGroup` (内 `LinearLayout`):

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red" />
    <RadioButton android:id="@+id/radio_blue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Blue" />
</RadioGroup>
```

非常重要的 `RadioButton` s 组合在一起由 `RadioGroup` 元素，以便一次可以选择不超过一个。由 Android 系统自动处理此逻辑。当一个 `RadioButton` 在选择了一个组，所有其他用户已取消自动选择。

若要执行某些操作时每个 `RadioButton` 是选择，我们需要编写一个事件处理程序：

```
private void RadioButtonClick (object sender, EventArgs e)
{
    RadioButton rb = (RadioButton)sender;
    Toast.MakeText (this, rb.Text, ToastLength.Short).Show ();
}
```

首先，传递在发件人是转换为单选按钮。然后 `Toast` 选定的单选按钮的文本将显示消息。

现在，在底部 `OnCreate()` 方法中，添加以下：

```
RadioButton radio_red = FindViewById<RadioButton>(Resource.Id.radio_red);
RadioButton radio_blue = FindViewById<RadioButton>(Resource.Id.radio_blue);

radio_red.Click += RadioButtonClick;
radio_blue.Click += RadioButtonClick;
```

这会将每个捕获 `RadioButton` 布局中，并将添加每个新创建的事件 handler 到。

运行该应用程序。

提示：如果您需要自行更改的状态（例如，当加载一个已保存 `CheckBoxPreference`），使用 `Checked` 属性 setter 或 `Toggle()` 方法。

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#)。

ToggleButton

2018/10/26 • [Edit Online](#)

在本部分中，您将创建专用于使用的两个状态之间切换按钮 `ToggleButton` 小组件。此小组件是单选按钮的绝佳替代方法，如果您有两个互相排斥的简单状态 ("on"和"关闭", 例如)。Android 4.0 (API 级别 14) 中引入了名为的切换按钮的替代方法 `Switch`。

左边的一对图像为 **ToggleButton** 的示例，而右边的一对图像则展示的是 **Switch** 的示例：



应用程序使用哪个控件是个风格问题。这两个小组件在功能上等效。

打开 **Resources/layout/Main.xml** 文件，并添加 `ToggleButton` 元素 (内 `LinearLayout`)：

若要更改状态时执行某些操作，请将以下代码添加到末尾 `onCreate()` 方法：

```
ToggleButton togglebutton = findViewById<ToggleButton>(Resource.Id.togglebutton);

togglebutton.Click += (o, e) => {
    // Perform action on clicks
    if (togglebutton.Checked)
        Toast.MakeText(this, "Checked", ToastLength.Short).Show ();
    else
        Toast.MakeText(this, "Not checked", ToastLength.Short).Show ();
};
```

这会将捕获 `ToggleButton` 元素在布局，然后处理单击事件，用于定义要在单击按钮时执行的操作。在此示例中，该方法将检查新状态的按钮，然后显示 `Toast` 条消息，指示当前状态。

注意，`ToggleButton` 在选中和未选中之间处理自己的状态更改，因此你只需要获取当前是哪一种状态。

运行该应用程序。

提示：如果您需要自行更改的状态 (例如，当加载一个已保存 `CheckBoxPreference`)，使用 `Checked` 属性 setter 或 `Toggle()` 方法。

相关链接

- [ToggleButton](#)
- [Switch](#)

CheckBox

2018/10/26 • [Edit Online](#)

在本部分中，您将创建一个复选框，用于选择项，使用 `CheckBox` 小组件。按下复选框时的 toast 消息将指示复选框的当前状态。

打开 **Resources/layout/Main.xml** 文件，并添加 `CheckBox` 元素 (内 `LinearLayout`):

```
<CheckBox android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="check it out" />
```

若要更改状态时执行某些操作，请将以下代码添加到末尾 `onCreate()` 方法：

```
CheckBox checkbox = findViewById<CheckBox>(Resource.Id.checkbox);

checkbox.Click += (o, e) => {
    if (checkbox.Checked)
        Toast.MakeText (this, "Selected", ToastLength.Short).Show ();
    else
        Toast.MakeText (this, "Not selected", ToastLength.Short).Show ();
};
```

这会将捕获 `CheckBox` 元素在布局，然后处理单击事件，用于定义要单击相应的复选框时进行的操作。单击时，`Checked` 属性被调用以检查新状态的复选框。如果选中，则 `Toast` 显示消息"已选"，否则为它将显示"未选择"。的 `CheckBox` 处理它自己的状态更改，因此只需查询的当前状态。

运行它。

提示：如果您需要自行更改的状态（例如，当加载一个已保存 `CheckBoxPreference`，使用 `Checked` 属性 setter 或 `Toggle()` 方法。

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#)。

自定义按钮

2018/10/26 • [Edit Online](#)

在本部分中，您将创建一个按钮与自定义映像而不是文本，使用 `Button` 小组件和定义三个不同的映像用于不同按钮状态的 XML 文件。按下按钮时，将显示一条短消息。

右键单击并下载下面的三个图像，然后将其复制到项目的 **Resources/drawable** 目录。这些图像将用于不同的按钮状态。



在 **Resources/drawable** 目录中创建名为 **android_button.xml** 的新文件。插入以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/android_pressed"
        android:state_pressed="true" />
    <item android:drawable="@drawable/android_focused"
        android:state_focused="true" />
    <item android:drawable="@drawable/android_normal" />
</selector>
```

此项定义单个可绘制资源，这将更改按钮的当前状态基于其映像。第一个 `<item>` 定义 **android_pressed.png** 作为按下按钮的图像（激活）；第二个 `<item>` 定义 **android_focused.png** 作为映像时已设定焦点的按钮（当突出显示该按钮时使用轨迹球或方向键）；第三个 `<item>` 定义 **android_normal.png**（时没有按下或已设定焦点）的正常状态的图像。此 XML 文件现在表示单个可绘制资源和通过引用时 `Button` 对于其背景显示的图像将更改基于以下三种状态。

NOTE

`<item>` 元素顺序很重要。当引用此图像时，`<item>` 将遍历这些项，以确定哪个项适合当前按钮状态。“默认”的图像是最后一个，因此只在 `android:state_pressed` 和 `android:state_focused` 这两个条件都为 `false` 时才应用它。

打开 **Resources/layout/Main.xml** 文件，并添加 `Button` 元素：

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:background="@drawable/android_button" />
```

`android:background` 属性指定要用于按钮背景的可绘制资源（其中，在保存时 **Resources/drawable/android.xml**，作为引用 `@drawable/android`）。这会替换整个系统的按钮所使用的普通的背景图像。为了使可绘制要更改其根据按钮状态的图像，图像必须应用于背景。

若要执行操作时按下按钮，在末尾添加下面的代码 `onCreate()` 方法：

```
Button button = findViewById<Button>(Resource.Id.button);

button.Click += (o, e) => {
    Toast.MakeText (this, "Beep Boop", ToastLength.Short).Show ();
};
```

这会将捕获 `Button` 从布局, 然后添加 `Toast` 时要显示的消息 `Button` 单击。

现在运行应用程序。

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution](#) 许可证。

日历 API

一组新的日历 Android 4 中引入的 Api 支持应用程序，可用于读取或写入到的日历提供程序的数据。这些 Api 支持丰富的交互选项的日历数据，包括能够读取和写入事件、与会者和提醒。通过在应用程序中使用的日历提供程序，通过 API 添加的数据将出现在 Android 4 附带内置日历应用程序。

添加权限

在使用应用程序中的新日历 Api，需要执行第一件事是将适当的权限添加到 Android 清单。若要添加所需的权限都 `android.permission.READ_CALENDAR` 和 `android.permission.WRITE_CALENDAR`，具体取决于是否您读取和/或写入日历数据。

使用协定的日历

后设置这些权限，您可以通过使用日历数据进行交互 `CalendarContract` 类。此类提供了与日历提供程序交互时，可以使用应用程序的数据模型。 `CalendarContract`，应用程序可以将 Uri 解析为日历实体，如日历和事件。它还提供一种方法与在每个实体，如日历的名称和 ID，或某一事件的开始和结束日期的各个字段进行交互。

让我们看一个示例，使用日历 API。在此示例中，我们将介绍如何枚举日历和它们的事件，以及如何将新事件添加到日历。

列出日历

首先，让我们看一下如何枚举已注册的日历应用中的日历。若要执行此操作，我们可以实例化 `CursorLoader`。在 Android 3.0 (API 11) 中引入 `CursorLoader` 是首选的方法来使用 `ContentProvider`。至少，我们将需要对日历和我们想要返回; 的列指定内容 Uri 此列规范称为_投影_。

调用 `CursorLoader.LoadInBackground` 方法可用于查询数据，如日历提供程序的内容提供程序。 `LoadInBackground` 执行实际的加载操作，并返回 `Cursor` 与查询的结果。

`CalendarContract` 可帮助我们指定这两个内容 Uri 和投影。若要获取的内容 Uri 进行查询的日历，我们只需使用 `CalendarContract.Calendars.ContentUri` 如下属性：

```
var calendarsUri = CalendarContract.Calendars.ContentUri;
```

使用 `CalendarContract` 来指定哪些日历我们需要的列也同样简单。我们只需添加中的字段

`CalendarContract.Calendars.InterfaceConsts` 到一个数组的类。例如，下面的代码包括日历的 ID、显示名称和帐户名称：

```
string[] calendarsProjection = {
    CalendarContract.Calendars.InterfaceConsts.Id,
    CalendarContract.Calendars.InterfaceConsts.CalendarDisplayName,
    CalendarContract.Calendars.InterfaceConsts.AccountName
};
```

`Id` 必须包含如果使用的 `SimpleCursorAdapter` 将数据绑定到 UI 中，正如我们稍后将看到。使用内容 Uri 和投影后，我们实例化 `CursorLoader`，并调用 `CursorLoader.LoadInBackground` 方法以返回日历数据的光标，如下所示：

```
var loader = new CursorLoader(this, calendarsUri, calendarsProjection, null, null, null);
var cursor = (ICursor)loader.LoadInBackground();
```

此示例中的 UI 包含 `ListView`，与表示单个日历的列表中每个项。以下 XML 显示了包含的标记 `ListView`：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

此外，我们需要指定每个列表项，我们将放在单独的 XML 文件，如下所示的 UI：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView android:id="@+id/calDisplayName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16dip" />
    <TextView android:id="@+id/calAccountName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="12dip" />
</LinearLayout>
```

从现在开始，它是只是正常的 Android 代码，以将数据从光标位置绑定到 UI。我们将使用 `SimpleCursorAdapter`，如下所示：

```
string[] sourceColumns = {
    CalendarContract.Calendars.InterfaceConsts.CalendarDisplayName,
    CalendarContract.Calendars.InterfaceConsts.AccountName };

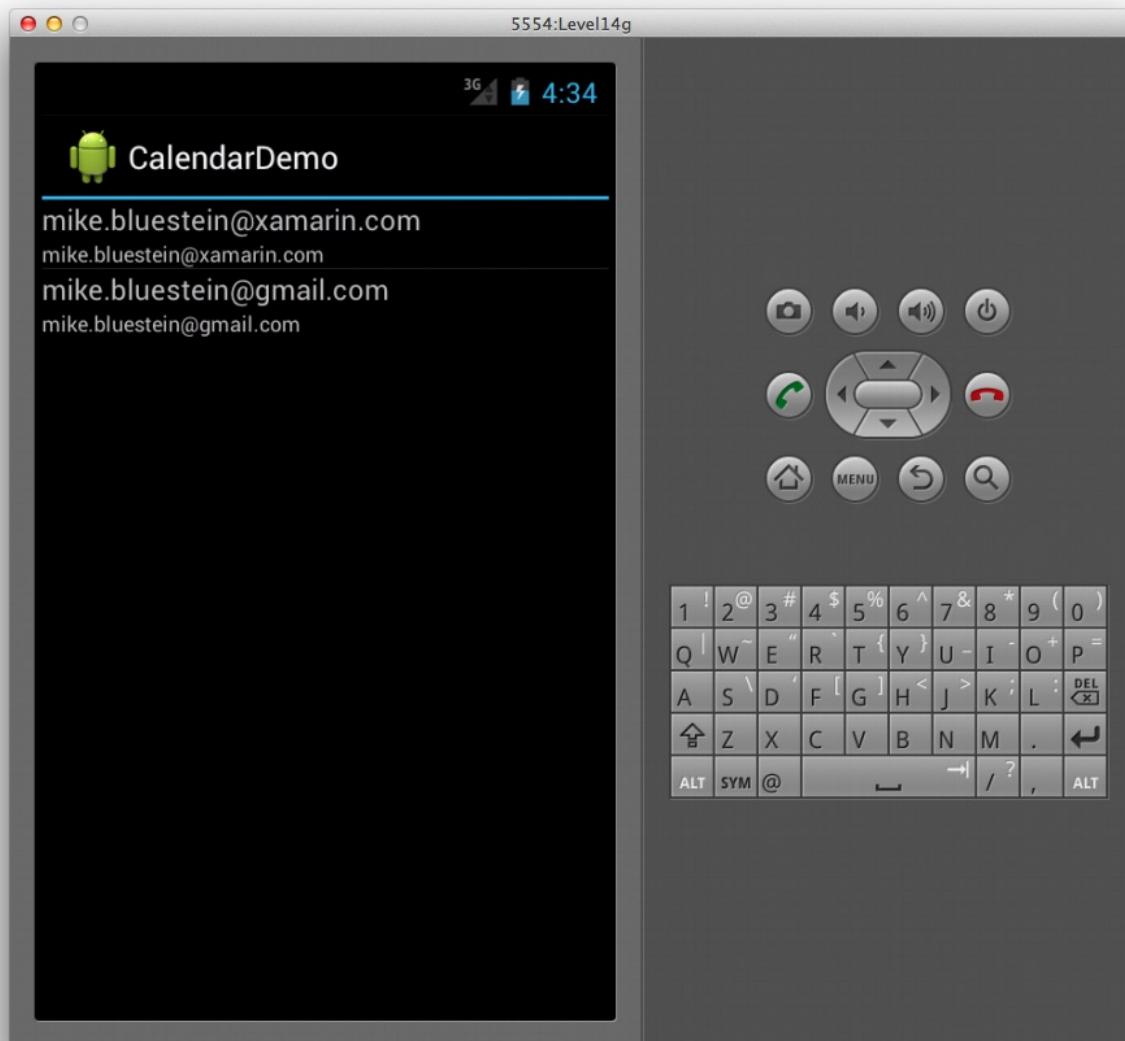
int[] targetResources = {
    Resource.Id.calDisplayName, Resource.Id.calAccountName };

SimpleCursorAdapter adapter = new SimpleCursorAdapter (this,
    Resource.Layout.CallListItem, cursor, sourceColumns, targetResources);

ListAdapter = adapter;
```

在上述代码中，适配器将中指定的列 `sourceColumns` 数组并将其写入到中的用户界面元素 `targetResources` 光标中每个日历项的数组。此处使用的活动是一个的子类 `ListActivity`；它包括 `ListAdapter` 为我们设置适配器的属性。

下面是显示最终结果中，使用日历信息显示在屏幕截图 `ListView`：



列出日历事件

下一步让我们看看如何枚举给定日历事件。上面的示例基于构建的我们将提供的事件时用户选择一个日历的列表。因此，我们将需要处理前面的代码中的项选择：

```
ListView.ItemClick += (sender, e) => {  
    int i = (e as ItemEventArgs).Position;  
  
    cursor.MoveToPosition(i);  
    int calId =  
        cursor.GetInt (cursor.GetColumnIndex (calendarsProjection [0]));  
  
    var showEvents = new Intent(this, typeof(EventListActivity));  
    showEvents.PutExtra("calId", calId);  
    StartActivity(showEvents);  
};
```

在此代码中，我们创建打开类型的一个活动的意图 `EventListActivity`，传递中意向的日历的 ID。我们将需要知道哪些日历查询事件的 ID。在中 `EventListActivity` 的 `OnCreate` 方法中，我们可以检索从 ID `Intent`，如下所示：

```
_calId = Intent.GetIntExtra ("calId", -1);
```

现在让我们为此查询事件日历 id。查询事件过程是我们查询的更早版本，日历的列表的方法类似但这次我们将使

用 `CalendarContract.Events` 类。以下代码将创建一个查询来检索事件：

```
var eventsUri = CalendarContract.Events.ContentUri;

string[] eventsProjection = {
    CalendarContract.Events.InterfaceConsts.Id,
    CalendarContract.Events.InterfaceConsts.Title,
    CalendarContract.Events.InterfaceConsts.Dtstart
};

var loader = new CursorLoader(this, eventsUri, eventsProjection,
    String.Format ("calendar_id={0}", _calId), null, "dtstart ASC");
var cursor = (ICursor)loader.LoadInBackground();
```

在此代码中，我们首先获取内容 `Uri` 中的事件 `CalendarContract.Events.ContentUri` 属性。然后我们指定我们想要检索 `eventsProjection` 数组中的事件列。最后，我们实例化 `CursorLoader` 使用此信息和调用加载程序 `LoadInBackground` 方法以返回 `Cursor` 与事件数据。

若要在 UI 中显示的事件数据，我们可以使用标记和代码就像我们以前用来显示日历的列表。同样，使用 `SimpleCursorAdapter` 要绑定到数据 `ListView` 如下面的代码中所示：

```
string[] sourceColumns = {
    CalendarContract.Events.InterfaceConsts.Title,
    CalendarContract.Events.InterfaceConsts.Dtstart };

int[] targetResources = {
    Resource.Id.eventTitle,
    Resource.Id.eventStartDate };

var adapter = new SimpleCursorAdapter (this, Resource.Layout.EventListItem,
    cursor, sourceColumns, targetResources);

adapter.ViewBinder = new ViewBinder ();
ListAdapter = adapter;
```

此代码，我们使用前面显示的日历列表的代码之间的主要区别是使用 `ViewBinder`，这行上设置：

```
adapter.ViewBinder = new ViewBinder ();
```

`ViewBinder` 类可用于进一步控制如何将我们绑定到视图的值。在这种情况下，我们使用它从毫秒的事件开始时间转换为日期字符串，如下实现中所示：

```

class ViewBinder : Java.Lang.Object, SimpleCursorAdapter.IViewBinder
{
    public bool SetViewValue (View view, Android.Database.ICursor cursor,
        int columnIndex)
    {
        if (columnIndex == 2) {
            long ms = cursor.GetLong (columnIndex);

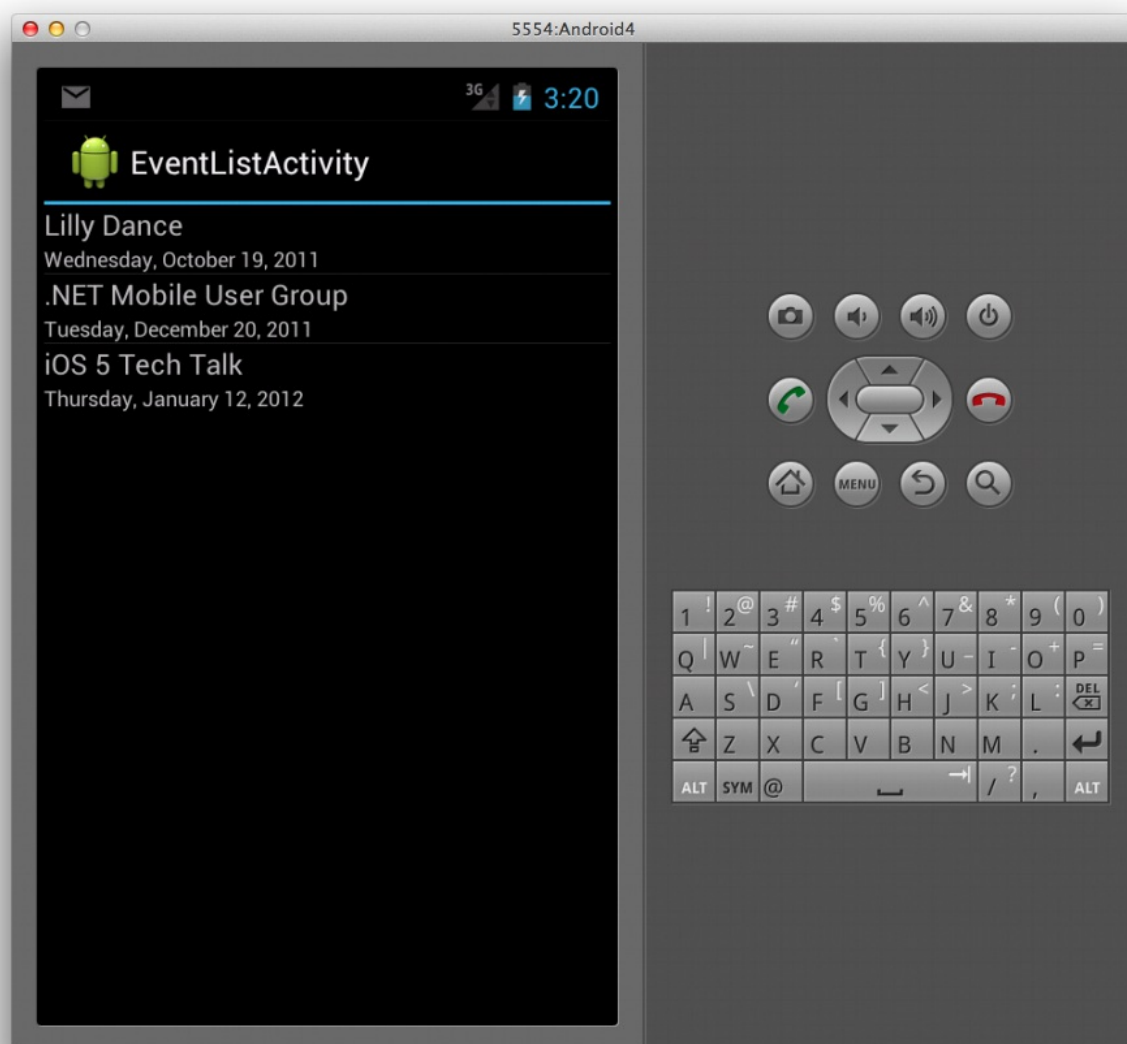
            DateTime date = new DateTime (1970, 1, 1, 0, 0, 0,
                DateTimeKind.Utc).AddMilliseconds (ms).ToLocalTime ();

            TextView textView = (TextView)view;
            textView.Text = date.ToLongDateString ();

            return true;
        }
        return false;
    }
}

```

这将显示事件的列表，如下所示：



添加日历事件

我们已了解如何读取日历数据。现在让我们了解如何将事件添加到日历。为实现此目的，请务必包括

`android.permission.WRITE_CALENDAR` 我们前面所述的权限。若要添加到日历事件，我们将：

1. 创建 `ContentValues` 实例。
2. 使用中的密钥 `CalendarContract.Events.InterfaceConsts` 类，以填充 `ContentValues` 实例。
3. 设置时区的事件的开始和结束时间。
4. 使用 `ContentResolver` 要插入到日历的事件数据。

下面的代码说明了这些步骤：

```
ContentValues eventValues = new ContentValues ();

eventValues.Put (CalendarContract.Events.InterfaceConsts.CalendarId,
    _calId);
eventValues.Put (CalendarContract.Events.InterfaceConsts.Title,
    "Test Event from M4A");
eventValues.Put (CalendarContract.Events.InterfaceConsts.Description,
    "This is an event created from Xamarin.Android");
eventValues.Put (CalendarContract.Events.InterfaceConsts.Dtstart,
    GetDateTimeMS (2011, 12, 15, 10, 0));
eventValues.Put (CalendarContract.Events.InterfaceConsts.Dtend,
    GetDateTimeMS (2011, 12, 15, 11, 0));

eventValues.Put(CalendarContract.Events.InterfaceConsts.EventTimezone,
    "UTC");
eventValues.Put(CalendarContract.Events.InterfaceConsts.EventEndTimezone,
    "UTC");

var uri = ContentResolver.Insert (CalendarContract.Events.ContentUri,
    eventValues);
```

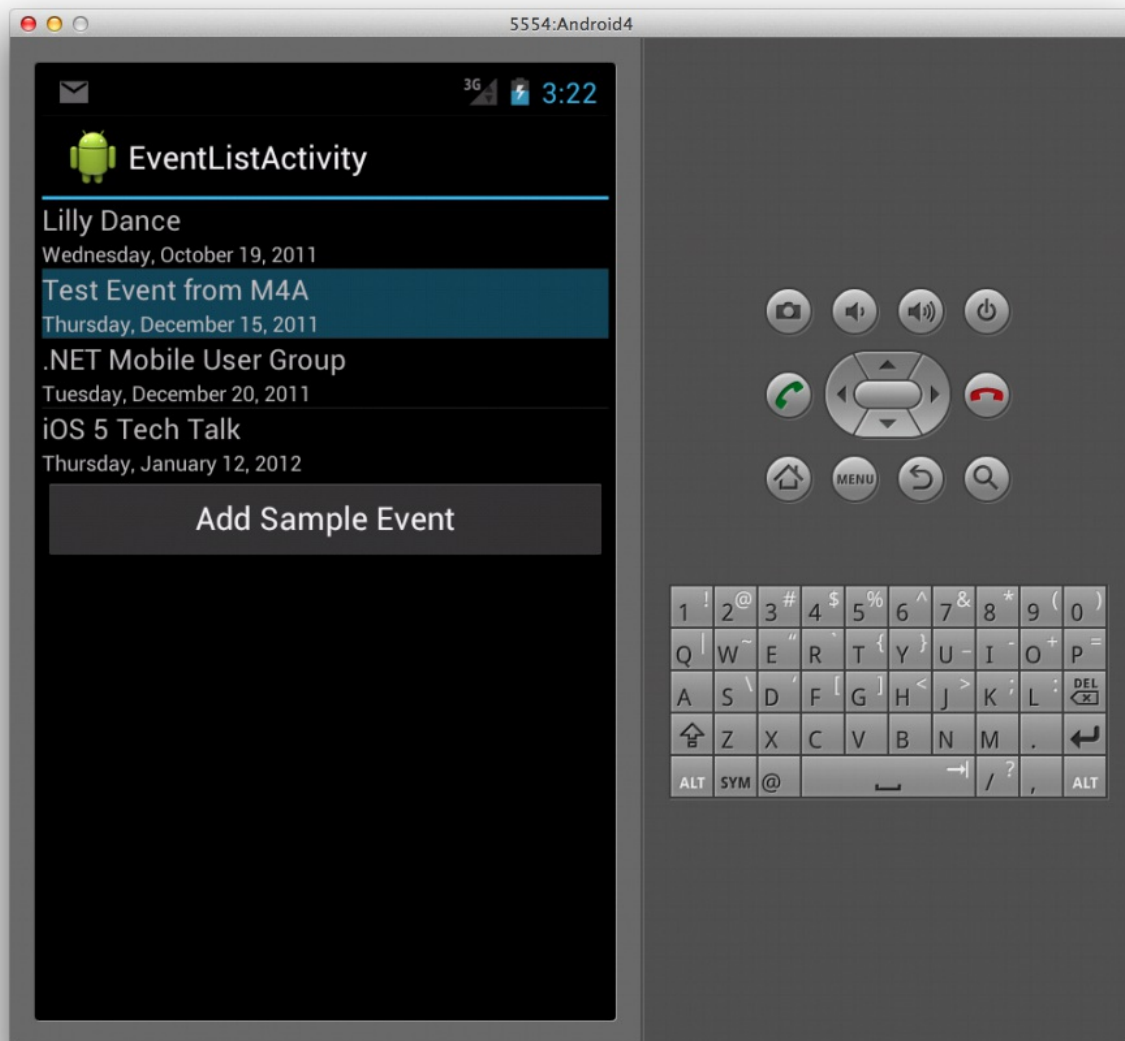
请注意，如果我们未设置时区，类型的异常 `Java.Lang.IllegalArgumentException` 将引发。由于事件时间值必须在自 epoch 以来的以毫秒为单位表示，我们创建 `GetDateTimeMS` 方法（在 `EventListActivity`）将我们的日期规范转换毫秒格式：

```
long GetDateTimeMS (int yr, int month, int day, int hr, int min)
{
    Calendar c = Calendar.GetInstance (Java.Util.TimeZone.Default);

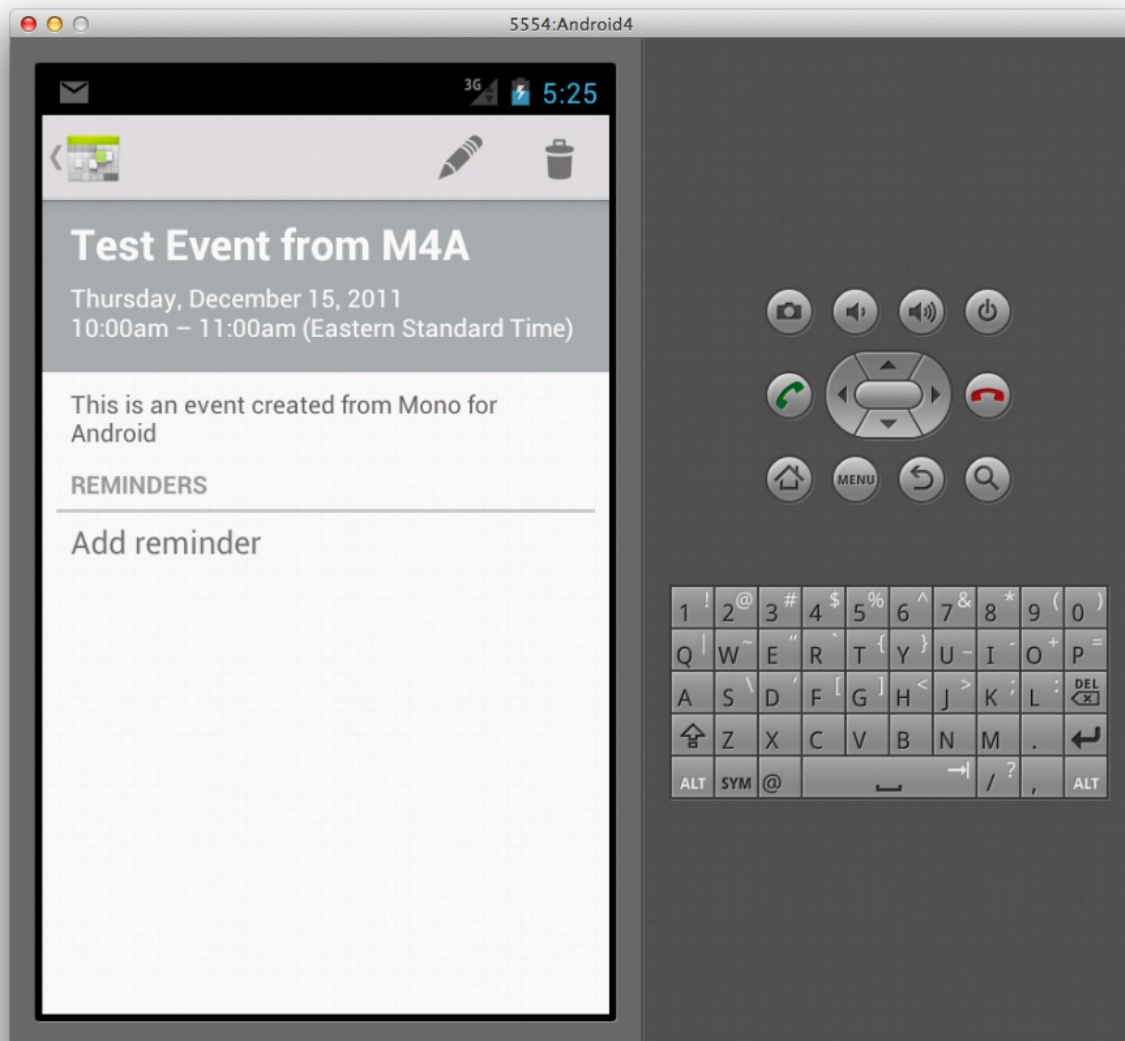
    c.Set (Java.Util.CalendarField.DayOfMonth, 15);
    c.Set (Java.Util.CalendarField.HourOfDay, hr);
    c.Set (Java.Util.CalendarField.Minute, min);
    c.Set (Java.Util.CalendarField.Month, Calendar.December);
    c.Set (Java.Util.CalendarField.Year, 2011);

    return c.TimeInMillis;
}
```

如果我们向事件列表用户界面添加一个按钮，并运行上面的代码中按钮的单击事件处理程序，该事件将添加到日历并更新列表中，如下所示：



如果我们打开日历应用, 我们将看到该事件将被写入存在以及:



正如您所看到的 Android 可让功能强大且易于访问，以检索和持久保存日历数据，从而将日历功能无缝集成的应用程序。

相关链接

- [日历演示（示例）](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

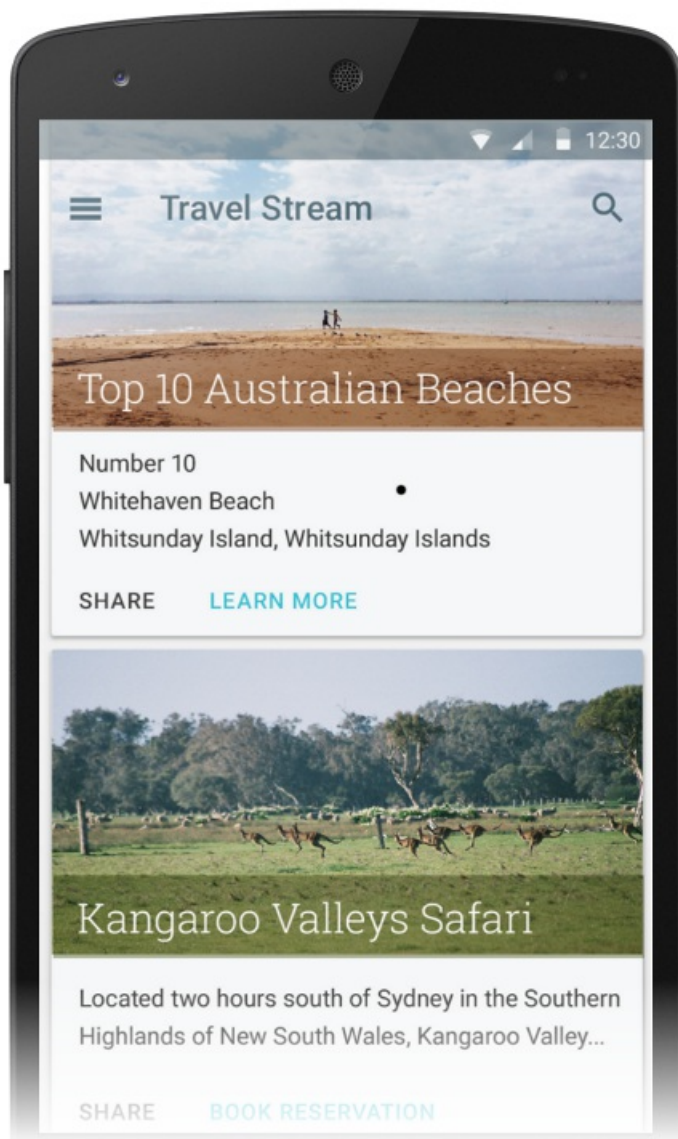
CardView

2018/10/26 • [Edit Online](#)

Cardview 小组件是在类似于卡的视图中显示的文本和图像内容的 UI 组件。本指南介绍如何使用和自 CardView Xamarin.Android 应用程序中定义同时保持向后兼容早期版本的 Android。

概述

Cardview 小组件中, 在 Android 5.0 (Lollipop) 中引入的是一个 UI 组件, 类似于卡的视图中显示的文本和图像的内容。CardView 作为实现 FrameLayout 带有圆的角和阴影的小组件。通常情况下, CardView 用来表示中的单个行项 ListView 或 GridView 查看组。例如, 下面的屏幕截图是实现一个旅行预订应用的示例 CardView -基于在可滚动的旅行目标卡 ListView:



本指南介绍如何添加 CardView 包到你的 Xamarin.Android 项目, 如何添加 CardView 到你的布局, 以及如何自定义的外观 CardView 应用程序中。此外, 本指南提供的详细的列表 CardView 您可以更改, 包括可帮助你使用的属性的特性 CardView 早于 Android 5.0 Lollipop 版本的 Android 上。

要求

以下所需使用新的 Android 5.0 及更高版本的功能 (包括 `CardView`) 的基于 Xamarin 的应用中:

- **Xamarin.Android** – Xamarin.Android 4.20 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for mac。
- **Android SDK** – Android 5.0 (API 21) 或更高版本必须安装通过 Android SDK 管理器。
- **Java JDK 1.8** – 可以使用 JDK 1.7, 如果您是专门针对 API 级别 23 和更早版本。提供了 JDK 1.8 [Oracle](#)。

您的应用程序还必须包括 `Xamarin.Android.Support.v7.CardView` 包。若要添加

`Xamarin.Android.Support.v7.CardView` Visual Studio for Mac 中的包:

1. 打开你的项目, 右键单击包, 然后选择**添加包**。
2. 在**中添加包**对话框中, 搜索**CardView**。
3. 选择**Xamarin 支持库 v7 CardView**, 然后单击**添加包**。

若要添加 `Xamarin.Android.Support.v7.CardView` Visual Studio 中的包:

1. 打开你的项目, 右键单击**引用节点 (在解决方案资源管理器窗格中)**, 然后选择**管理 NuGet 包...**
2. 当**管理 NuGet 包**显示对话框中, 输入**CardView**在搜索框中。
3. 当**Xamarin 支持库 v7 CardView**出现, 请单击**安装**。

若要了解如何配置 Android 5.0 应用程序项目, 请参阅[设置了 Android 5.0 项目](#)。有关安装 NuGet 包的详细信息, 请参阅[演练: 在项目中包括 NuGet](#)。

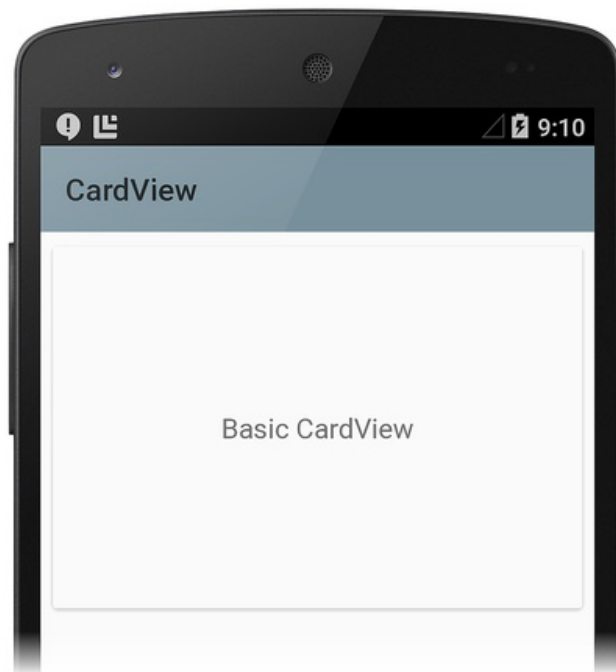
引入 CardView

默认值 `CardView` 类似于带有最少圆的角和一些阴影白色卡。下面的示例**Main.xml**布局显示单个 `CardView` 包含的小组件 `TextView`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal"
    android:padding="5dp">
    <android.support.v7.widget.CardView
        android:layout_width="fill_parent"
        android:layout_height="245dp"
        android:layout_gravity="center_horizontal">
        <TextView
            android:text="Basic CardView"
            android:layout_marginTop="0dp"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:layout_centerVertical="true"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true" />
        </android.support.v7.widget.CardView>
    </LinearLayout>
```

如果使用此 XML 替换现有内容**Main.xml**, 请务必注释掉中的任何代码**MainActivity.cs**, 是指在之前的 XML 的资源。

此布局的示例创建一个默认 `CardView` 通过一行文本, 如以下屏幕截图中所示:



在此示例中，应用样式设置为浅材料主题（`Theme.Material.Light`），以便 `CardView` 阴影和边缘都更轻松地查看。有关主题 Android 5.0 应用程序的详细信息，请参阅[材料主题](#)。在下一步部分中，我们将了解如何自定义 `CardView` 应用程序。

自定义 CardView

您可以修改基本 `CardView` 特性以自定义的外观 `CardView` 应用程序中。例如，提升 `CardView` 可以增加要强制转换更大的卷影（这会使卡看起来为浮点型上面在后台更高版本）。此外，可以增加圆角半径进行更多的卡片的舍入的角。

在下一步布局示例中，自定义 `CardView` 用于创建打印照片（“快照”）的模拟。`ImageView` 添加到 `CardView` 用于显示图像和一个 `TextView` 定位下面 `ImageView` 用于显示的图像标题。在此示例布局中，`CardView` 具有以下自定义设置：

- `cardElevation` 增加到 4dp 转换更大的卷影。
- `cardCornerRadius` 增加到 5dp 以使显示更加圆角的边角。

因为 `CardView` 提供通过 Android v7 支持库，其属性未提供的 `android:` 命名空间。因此，您必须定义 XML 命名空间，并使用为该命名空间 `CardView` 属性前缀。在下面的布局示例中，我们将使用这行来定义命名空间 `cardview`：

```
xmlns:cardview="http://schemas.android.com/apk/res-auto"
```

您可以调用此命名空间 `card_view` 甚至 `myapp` 如果你选择（它是只能在此文件的作用域中访问）。无论您选择要调用此命名空间，则必须使用其前缀 `CardView` 你想要修改的属性。在此布局的示例中，`cardview` 命名空间是前缀 `cardElevation` 和 `cardCornerRadius`：


```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:cardview="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal"
    android:padding="5dp">
    <android.support.v7.widget.CardView
        android:layout_width="fill_parent"
        android:layout_height="245dp"
        android:layout_gravity="center_horizontal"
        cardview:cardElevation="4dp"
        cardview:cardCornerRadius="5dp">
        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="240dp"
            android:orientation="vertical"
            android:padding="8dp">
            <ImageView
                android:layout_width="fill_parent"
                android:layout_height="190dp"
                android:id="@+id/imageView"
                android:scaleType="centerCrop" />
            <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:textColor="#333333"
                android:text="Photo Title"
                android:id="@+id/textView"
                android:layout_gravity="center_horizontal"
                android:layout_marginLeft="5dp" />
            </LinearLayout>
        </android.support.v7.widget.CardView>
    </LinearLayout>

```

此布局示例用于在照片查看应用中，显示的图像时 `CardView` 具有照片快照的外观，以下屏幕截图中所示：

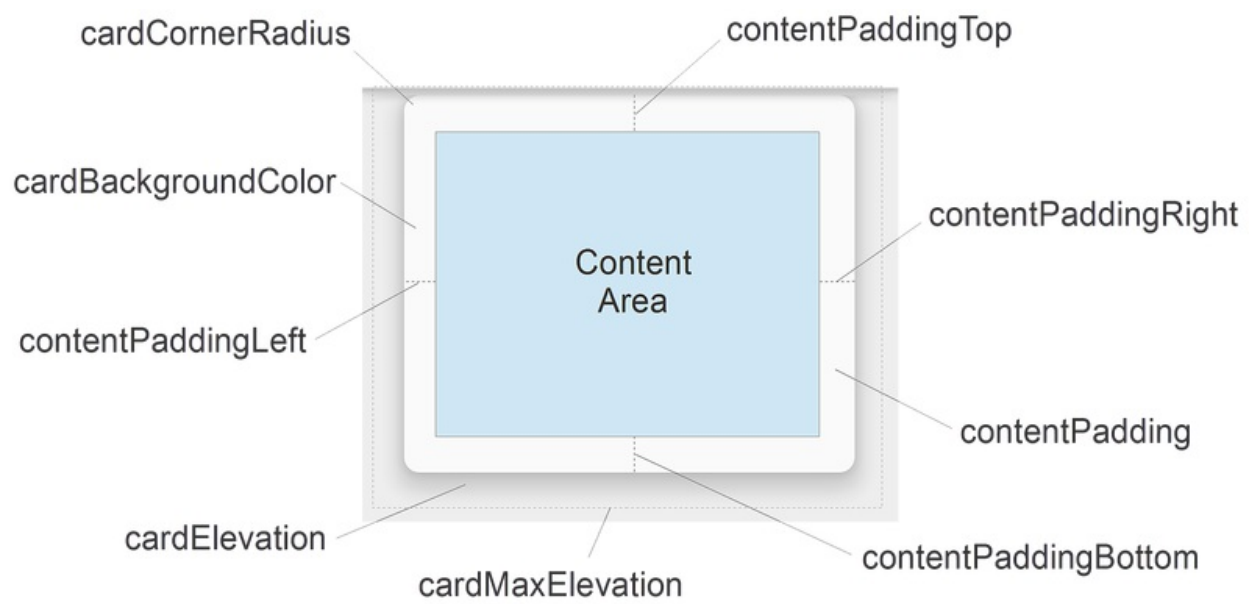


此屏幕截图取自 [RecyclerView](#) 示例应用，使用 `RecyclerView` 小部件显示的滚动列表 `CardView` 查看照片的映像。有关详细信息 `RecyclerView`，请参阅 [RecyclerView](#) 指南。

请注意，`CardView` 可以在其内容区域中显示多个子视图。例如，在上面的照片查看应用示例，在内容区域组成 `ListView`，其中包含 `ImageView` 和一个 `TextView`。尽管 `CardView` 实例通常垂直排列，也可以水平排列它们 (请参阅 [创建自定义视图样式示例屏幕快照](#))。

CardView 布局选项

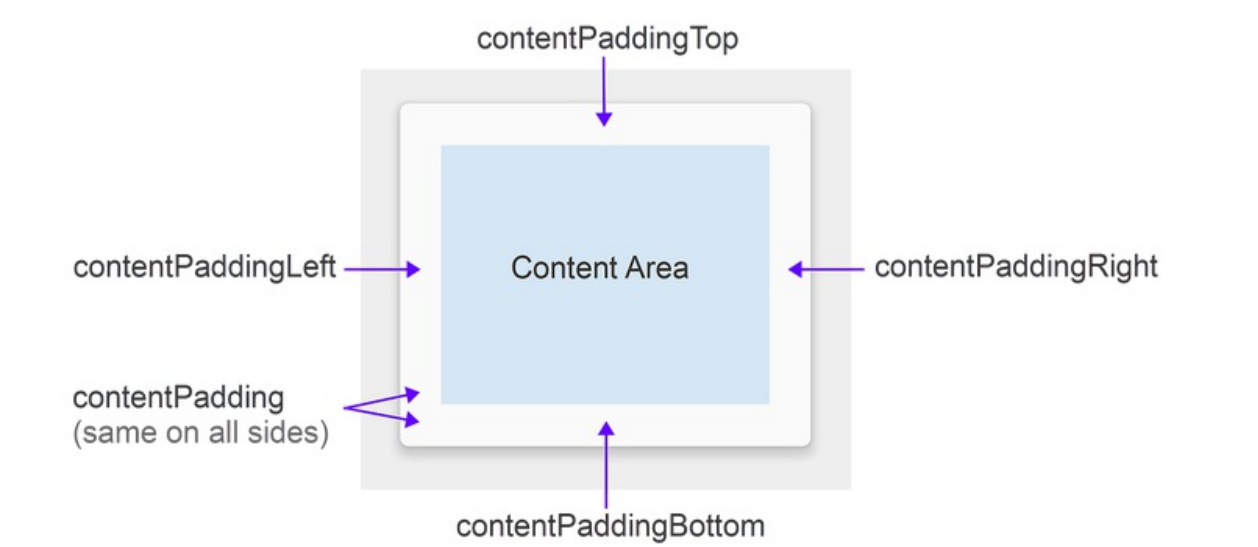
`CardView` 通过设置影响其填充、提升、圆角半径和背景色的一个或多个属性，可以自定义布局：



每个属性还可以更改动态通过调用对应 `CardView` 方法 (有关详细信息 `CardView` 方法，请参阅 [CardView 类引用](#))。请注意，这些属性 (除外的背景色) 接受一个维度值，该值是跟单位的十进制数字。例如，`11.5dp` 指定 11.5 密度无关的像素。

填充

`CardView` 提供了五个填充属性来定位数据卡中的内容。可以将它们设置在布局 XML 中也可以在代码中调用类似方法：



填充特性进行了说明，如下所示：

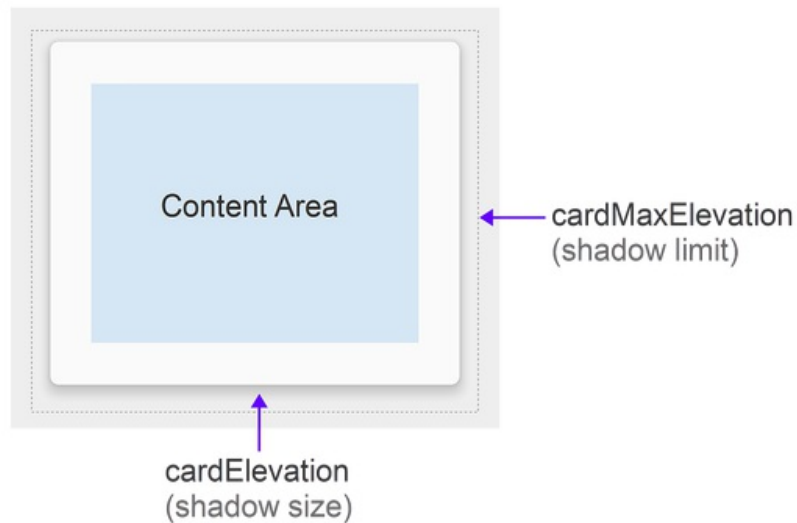
- `contentPadding` – 内部填充的子视图之间 `CardView` 和所有边缘的卡。
- `contentPaddingBottom` – 内部填充的子视图之间 `CardView` 和卡的下边缘。
- `contentPaddingLeft` – 内部填充的子视图之间 `CardView` 和卡片的左边的缘。
- `contentPaddingRight` – 内部填充的子视图之间 `CardView` 卡片的右边缘。

- `contentPaddingTop` – 内部填充的子视图之间 `CardView` 卡的上边缘。

内容填充属性是相对于边界的内容区域，而不是位于内容区域内的任何给定小组件。例如，如果 `contentPadding` 在照片查看应用程序，充分地增加了 `CardView` 会裁剪图像和卡片上显示的文本。

提升

`CardView` 提供了两个提升属性来控制其进行提升，因此，其阴影的大小：



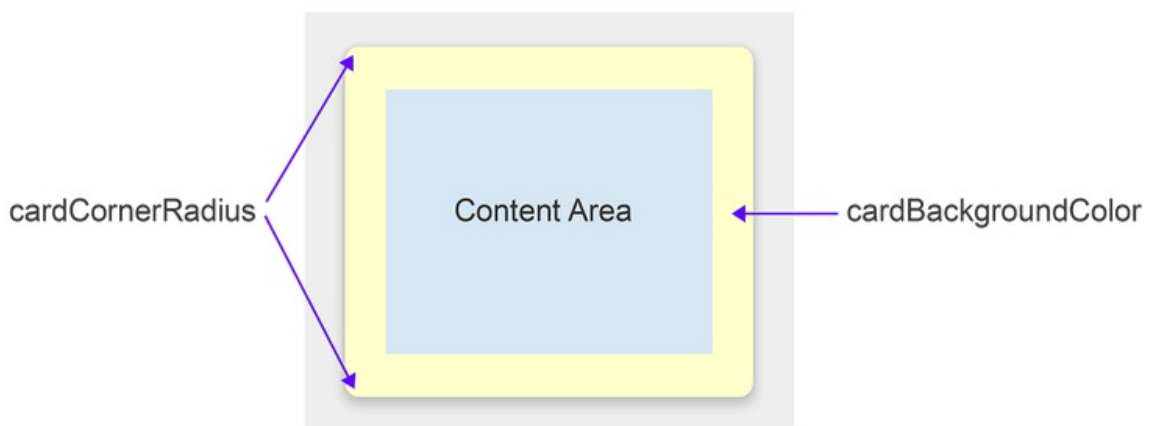
提升属性进行了说明，如下所示：

- `cardElevation` – 提升 `CardView` (表示其 Z 轴)。
- `cardMaxElevation` – 最大值 `CardView` 的提升。

较大的值 `cardElevation` 增加卷影大小以使 `CardView` 看起来为浮点型上面在后台更高版本。`cardElevation` 属性还确定重叠的视图的绘制顺序；这就是，则 `CardView` 将绘制具有较高的提升权限设置及更高版本具有较低的提升设置任何重叠视图的另一个重叠视图下。`cardMaxElevation` 设置可用于您的应用程序发生更改时提升动态-它会阻止阴影扩展过去使用此设置定义的限制。

圆角半径和背景色

`CardView` 提供了可用于控制其圆角半径和其背景色的属性。这两个属性允许您更改的整体样式 `CardView`：



这些属性解释如下：

- `cardCornerRadius` – 所有的角的圆角半径 `CardView`。
- `cardBackgroundColor` – 背景色 `CardView`。

在此图中，`cardCornerRadius` 设置为舍入 10dp 并 `cardBackgroundColor` 设置为 "#FFFFCC" (浅黄色)。

兼容性

可以使用 `CardView` 早于 Android 5.0 Lollipop 版本的 Android 上。因为 `CardView` 属于的 Android v7 支持库，可以使用 `CardView` Android 2.1（API 级别 7）及更高版本。但是，必须安装 `Xamarin.Android.Support.v7.CardView` 包中所述[要求](#)、更高版本。

`CardView` 表现出之前 Lollipop（API 级别 21）的设备上略有不同的行为：

- `CardView` 使用添加附加的填充的卷影以编程方式实现。
- `CardView` 不会剪切与相交的子视图 `CardView` 的圆角。

若要帮助管理这些兼容性差异，`CardView` 提供了几个可以配置在布局中的其他属性：

- `cardPreventCornerOverlap` – 将此属性设置为 `true` 以您的应用程序运行之前的 Android 版本（API 级别 20 及更早版本）时，将添加填充。此设置可防止 `CardView` 内容从与相交 `CardView` 的圆角。
- `cardUseCompatPadding` – 将此属性设置为 `true` 时您的应用程序运行在 Android 或大于 API 级别 21 的版本中添加填充。如果你想要使用 `CardView` 预棒棒糖形设备上，并让它看起来一样棒棒糖形上（或更高版本），将此属性设置为 `true`。启用此特性后，`CardView` 添加附加的填充要绘制阴影时在预棒棒糖形设备上运行。这有助于克服填充预棒棒糖形以编程方式隐藏实现生效时引入的差异。

有关保持与早期版本的 Android 的兼容性的详细信息，请参阅[维护兼容性](#)。

总结

本指南介绍了新 `CardView` 小组件包含在 Android 5.0 (Lollipop)。它演示了默认值 `CardView` 外观，并说明了如何自定义 `CardView` 通过更改其提升，角圆度，内容填充和背景色。它列出 `CardView` 布局特性（与引用关系图），并介绍了如何使用 `CardView` 早于 Android 5.0 Lollipop 的 Android 设备上。有关详细信息 `CardView`，请参阅[CardView 类引用](#)。

相关链接

- [RecyclerView（示例）](#)
- [棒棒糖形简介](#)
- [CardView 类引用](#)

编辑文本

2018/10/26 • [Edit Online](#)

在本部分中，你将使用[EditText](#)小组件创建的用户输入的文本字段。一旦到字段中，输入文本**Enter**密钥将显示文本中的 toast 消息。

打开**Resources/layout/activity_main.xml**并添加[EditText](#)到包含布局元素。下面的示例**activity_main.xml**已

[EditText](#) 已添加到 [LinearLayout](#) :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/edittext"
        android:layout_width="match_parent"
        android:imeOptions="actionGo"
        android:inputType="text"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在此代码示例中，[EditText](#) 特性 [android:imeOptions](#) 设置为 [actionGo](#) 。此设置更改的默认**完成**操作**转**操作，以便点击**Enter**密钥触发器 [KeyPress](#) 输入的处理程序。(通常情况下，[actionGo](#) 使用，以便**Enter**密钥以使用户可以在中键入的 URL 目标。)

若要处理用户的文本输入，请将以下代码添加到末尾**OnCreate**中的方法**MainActivity.cs**:

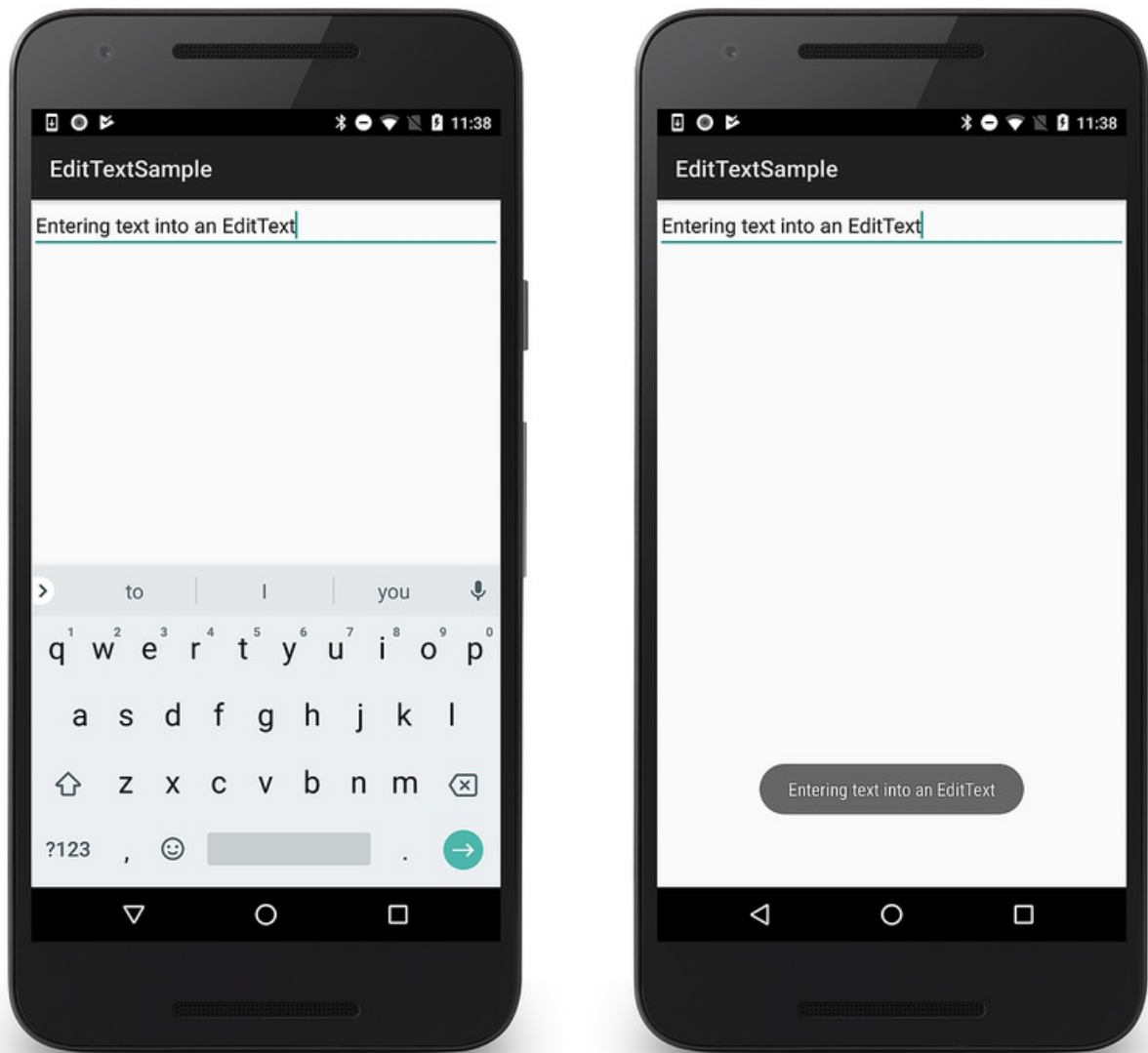
```
EditText edittext = FindViewById<EditText>(Resource.Id.edittext);
edittext.KeyPress += (object sender, View.KeyEventArgs e) => {
    e.Handled = false;
    if (e.Event.Action == KeyEventActions.Down && e.KeyCode == KeyCode.Enter)
    {
        Toast.MakeText(this, edittext.Text, ToastLength.Short).Show();
        e.Handled = true;
    }
};
```

此外，添加以下 [using](#) 语句的页首**MainActivity.cs**如果尚不存在：

```
using Android.Views;
```

此代码示例增大[EditText](#)元素在布局，并添加[KeyPress](#)定义小组件获得焦点时按下某个键时进行的操作的处理程序。在这种情况下，定义方法来侦听**Enter**密钥（当点击），然后弹出[Toast](#)与已输入的文本消息。请注意，[Handled](#)属性应始终为 [true](#) 处理该事件。这是有必要阻止从浮升事件向上（这将产生的文本字段中返回一个回车符）。

运行应用程序和文本字段中输入一些文本。当您按下**Enter**键、toast 通知将显示在右侧所示：



此页的部分是基于创建的工作的修改并 [由 Android 的开放源项目共享](#) 中所述的条款, 并用 2.5 的 [creative Commons Attribution 许可证](#)。本教程基于 [Android 窗体内容教程](#)。

相关链接

- [EditTextSample](#)

库

2018/10/26 • [Edit Online](#)

`Gallery` 是用于在水平滚动的列表中显示项的布局小组件并将当前所选内容放置在视图的中心。

IMPORTANT

在 Android 4.1 (API 级别 16) 中已弃用此小组件。

在本教程中，将创建的照片库，然后选择库项每次显示的 toast 消息。

之后 `Main.xml` 为内容视图中，设置布局 `Gallery` 与布局从捕获 `findViewById` 的 `Adapter` 然后使用属性来设置自定义适配器 (`ImageAdapter`) 作为显示在 dallery 中所有项目的源。 `ImageAdapter` 在下一步中创建。

若要单击库中的项时执行某些操作，一个匿名委托，它订阅 `ItemClick` 事件的参数。它显示了 `Toast` 显示选项的索引位置 (从零开始) (在实际方案中，位置可用来获取另一项任务的实际尺寸的图像)。

首先，有几个成员变量，包括的引用保存在可绘制资源目录中的映像的 Id 的数组 (可绘制资源/)。

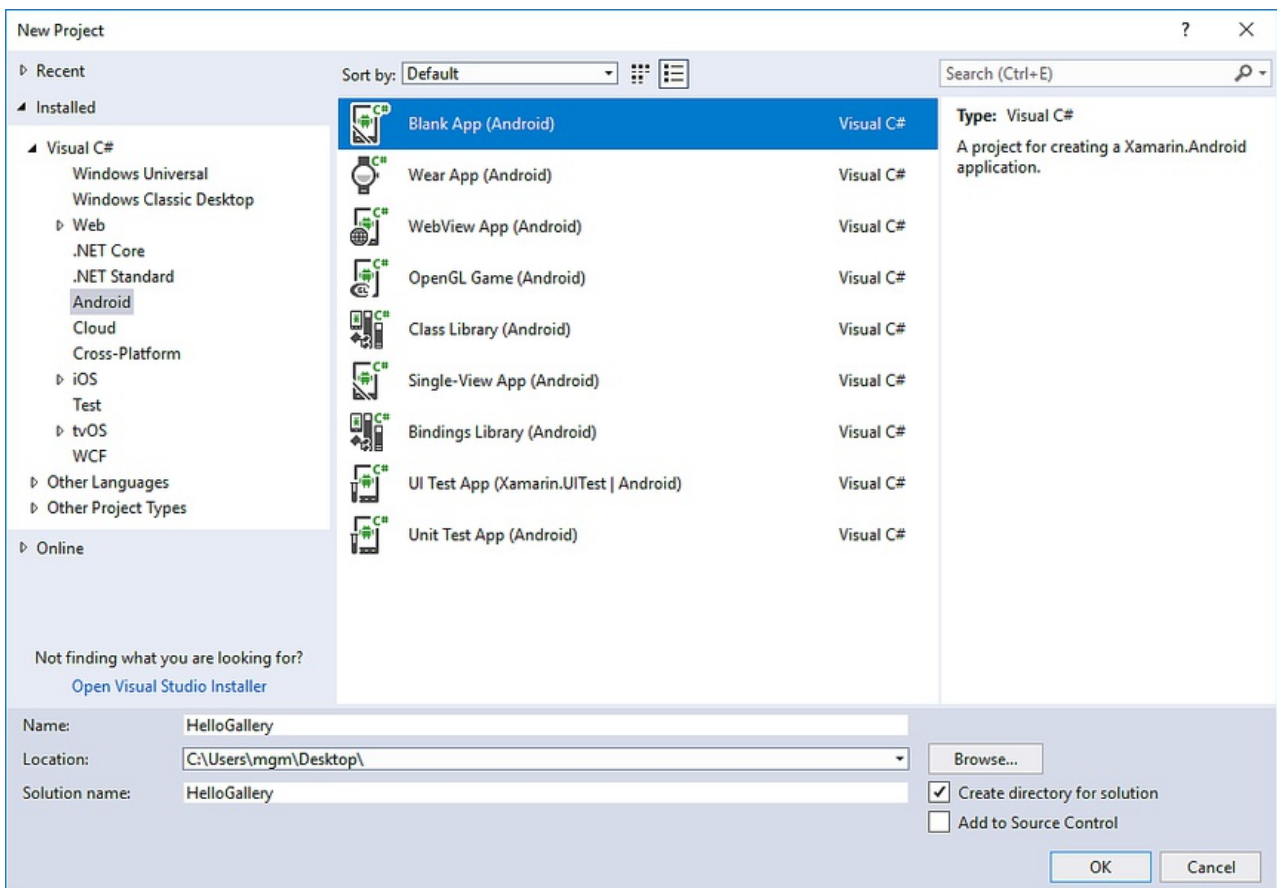
接下来是类构造函数，其中 `Context` 有关 `ImageAdapter` 定义实例并将其保存到本地字段。接下来，这会实现继承自某些所需的方法 `BaseAdapter` 。构造函数和 `Count` 属性都很容易理解。通常情况下， `getItem(int)` 应返回在适配器中，指定的位置的实例对象，但它对于此示例中，将忽略。同样， `getItemId(int)` 应返回的行 id 的项，但此处不需要它。

方法执行的工作要应用到图像 `ImageView` 将嵌入的 `Gallery` 在此方法中，成员 `Context` 用于创建一个新 `ImageView` 。的 `ImageView` 通过应用的可绘制资源，设置本地数组中的映像准备 `Gallery.LayoutParams` 高度和宽度设置缩放以适合的图像 `ImageView` 维度，然后再最后设置使用促升属性构造函数中获取的背景。

请参阅 `ImageView.ScaleType` 其他图像缩放选项的。

演练

启动一个名为的新项目 *HelloGallery* 。



找到你想要使用，一些照片或[下载这些示例图像](#)。将图像文件添加到项目的资源/**Drawable**目录。在中属性窗口中，将生成操作设置为**AndroidResource**。

打开 **Resources/Layout/Main.axml** 并插入以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<Gallery xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gallery"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

打开 **MainActivity.cs** 并插入以下代码 **OnCreate()** 方法：

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Set our view from the "main" layout resource
    SetContentView (Resource.Layout.Main);

    Gallery gallery = (Gallery) FindViewById<Gallery>(Resource.Id.gallery);

    gallery.Adapter = new ImageAdapter (this);

    gallery.ItemClick += delegate (object sender, Android.Widget.AdapterView.ItemClickEventArgs args) {
        Toast.MakeText (this, args.Position.ToString (), ToastLength.Short).Show ();
    };
}
```

创建一个名为的新类 **ImageAdapter** 子类 **BaseAdapter**：


```

public class ImageAdapter : BaseAdapter
{
    Context context;

    public ImageAdapter (Context c)
    {
        context = c;
    }

    public override int Count { get { return thumbIds.Length; } }

    public override Java.Lang.Object GetItem (int position)
    {
        return null;
    }

    public override long GetItemId (int position)
    {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public override View GetView (int position, View convertView, ViewGroup parent)
    {
        ImageView i = new ImageView (context);

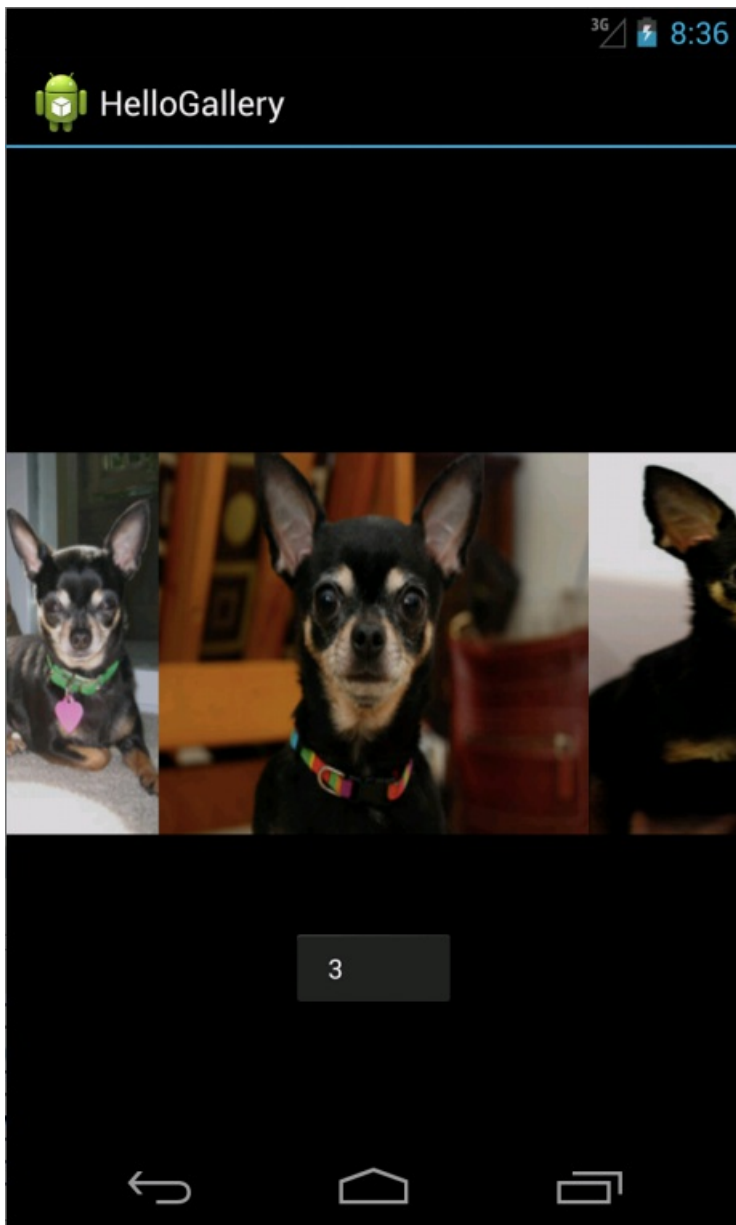
        i.SetImageResource (thumbIds[position]);
        i.LayoutParameters = new Gallery.LayoutParams (150, 100);
        i.SetScaleType (ImageView.ScaleType.FitXy);

        return i;
    }

    // references to our images
    int[] thumbIds = {
        Resource.Drawable.sample_1,
        Resource.Drawable.sample_2,
        Resource.Drawable.sample_3,
        Resource.Drawable.sample_4,
        Resource.Drawable.sample_5,
        Resource.Drawable.sample_6,
        Resource.Drawable.sample_7
    };
}

```

运行该应用程序。它应如以下屏幕截图所示：



参考资料

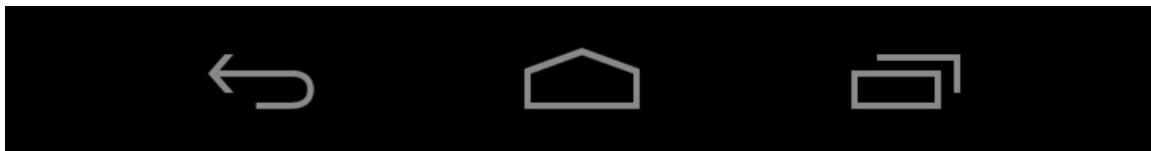
- [BaseAdapter](#)
- [Gallery](#)
- [ImageView](#)

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#).

导航栏

2018/10/26 • [Edit Online](#)

Android 4 引入了名为的新系统用户界面功能 **导航栏**, 这样就不包含的硬件按钮的设备上的导航控件主页, **返回**, 并**菜单**。下面的屏幕截图显示了从 Nexus 素数设备导航栏:

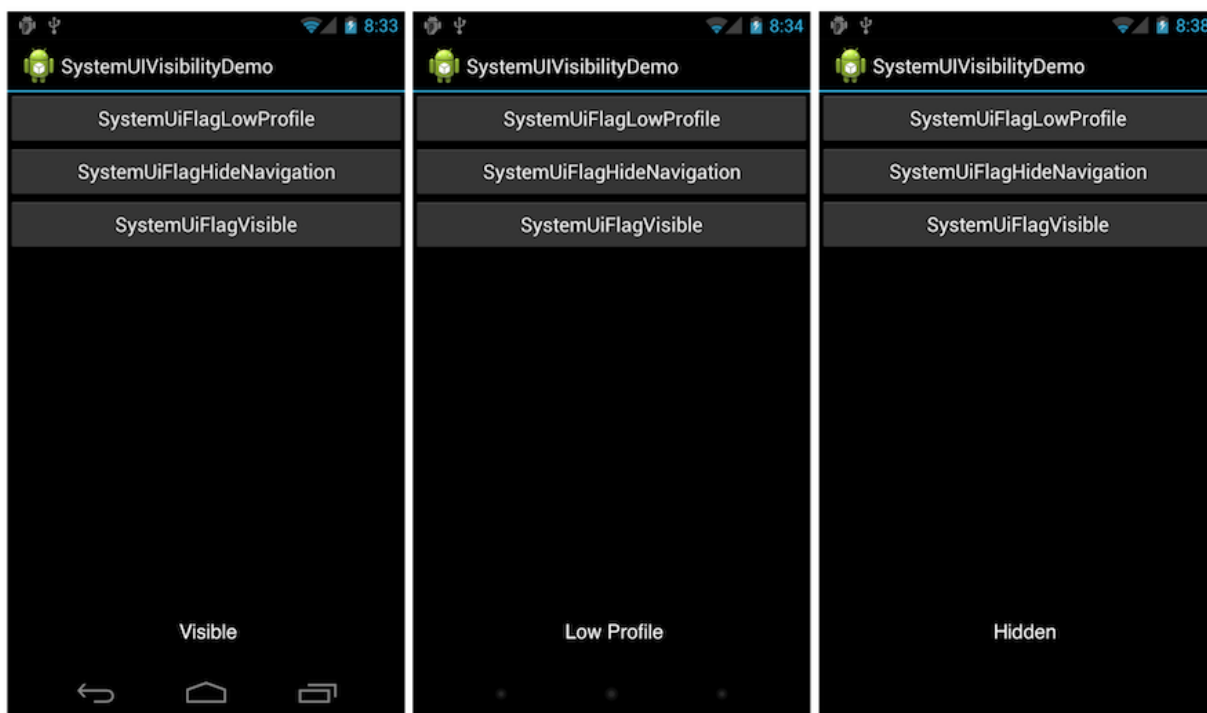


可以使用几个新的标记, 以控制导航栏和其控件的可见性以及 Android 3 中引入的系统栏的可见性。在定义标志 `Android.View.View` 类, 如下所示:

- `SystemUiFlagVisible` - 使导航栏可见。
- `SystemUiFlagLowProfile` - 在导航栏中的控件进行测试, dims。
- `SystemUiFlagHideNavigation` - 隐藏导航栏。

这些标志可以通过设置应用于的任意视图中的视图层次结构 `SystemUiVisibility` 属性。如果多个视图将此属性设置, 系统将它们与或运算结合起来, 并将其应用, 只要设置了标志窗口保留焦点。当您删除一个视图时, 它已设置任何标志也将被删除。

下面的示例演示一个简单的应用程序在其中单击任一按钮更改 `SystemUiVisibility`:



若要更改的代码 `SystemUiVisibility` 上的属性设置 `TextView` 从每个按钮的单击事件处理程序, 如下所示:

```
var tv = FindViewById<TextView> (Resource.Id.systemUiFlagTextView);
var lowProfileButton = FindViewById<Button>(Resource.Id.lowProfileButton);
var hideNavButton = FindViewById<Button> (Resource.Id.hideNavigation);
var visibleButton = FindViewById<Button> (Resource.Id.visibleButton);

lowProfileButton.Click += delegate {
    tv.SystemUiVisibility =
        (StatusBarVisibility)View.SystemUiFlagLowProfile;
};

hideNavButton.Click += delegate {
    tv.SystemUiVisibility =
        (StatusBarVisibility)View.SystemUiFlagHideNavigation;
};

visibleButton.Click += delegate {
    tv.SystemUiVisibility = (StatusBarVisibility)View.SystemUiFlagVisible;
}
```

此外，`SystemUiVisibility` 更改引发 `SystemUiVisibilityChange` 事件。就像设置 `SystemUiVisibility` 属性中，处理程序 `SystemUiVisibilityChange` 层次结构中的任何视图，可以注册事件。例如，以下代码使用 `TextView` 要注册的事件实例：

```
tv.SystemUiVisibilityChange +=
    delegate(object sender, View.SystemUiVisibilityChangeEventArgs e) {
        tv.Text = String.Format ("Visibility = {0}", e.Visibility);
    };
};
```

相关链接

- [SystemUiVisibilityDemo \(示例\)](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

选取器

2018/10/26 • [Edit Online](#)

选取器是允许用户通过使用由 Android 提供的对话框中选择一个日期或时间的 UI 元素：

- **日期选取器**用于选择日期（年、月 and 天）。

2016

Fri, Apr 1



April 2016



S

M

T

W

T

F

S

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

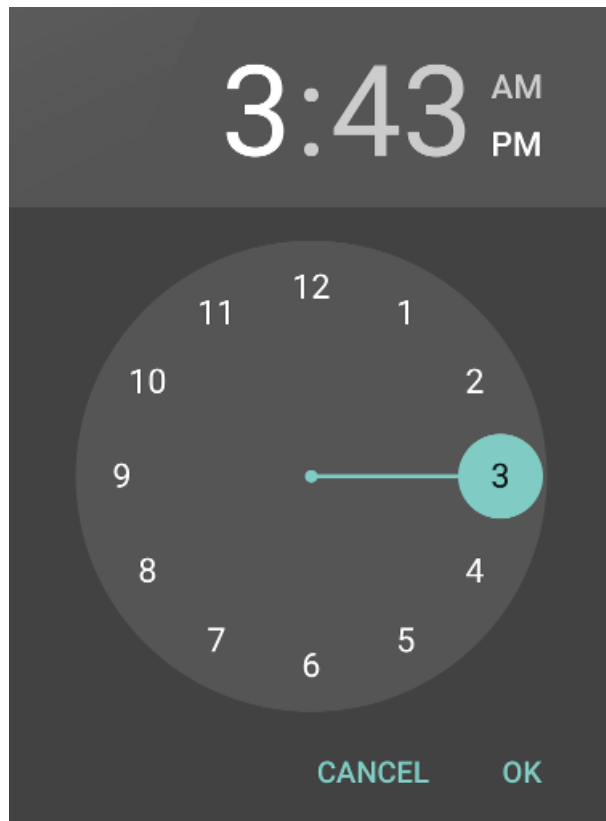
29

30

CANCEL

OK

- **时间选取器**用于选择的时间（小时、分钟和 AM/PM）。



日期选取器

2018/10/26 • [Edit Online](#)

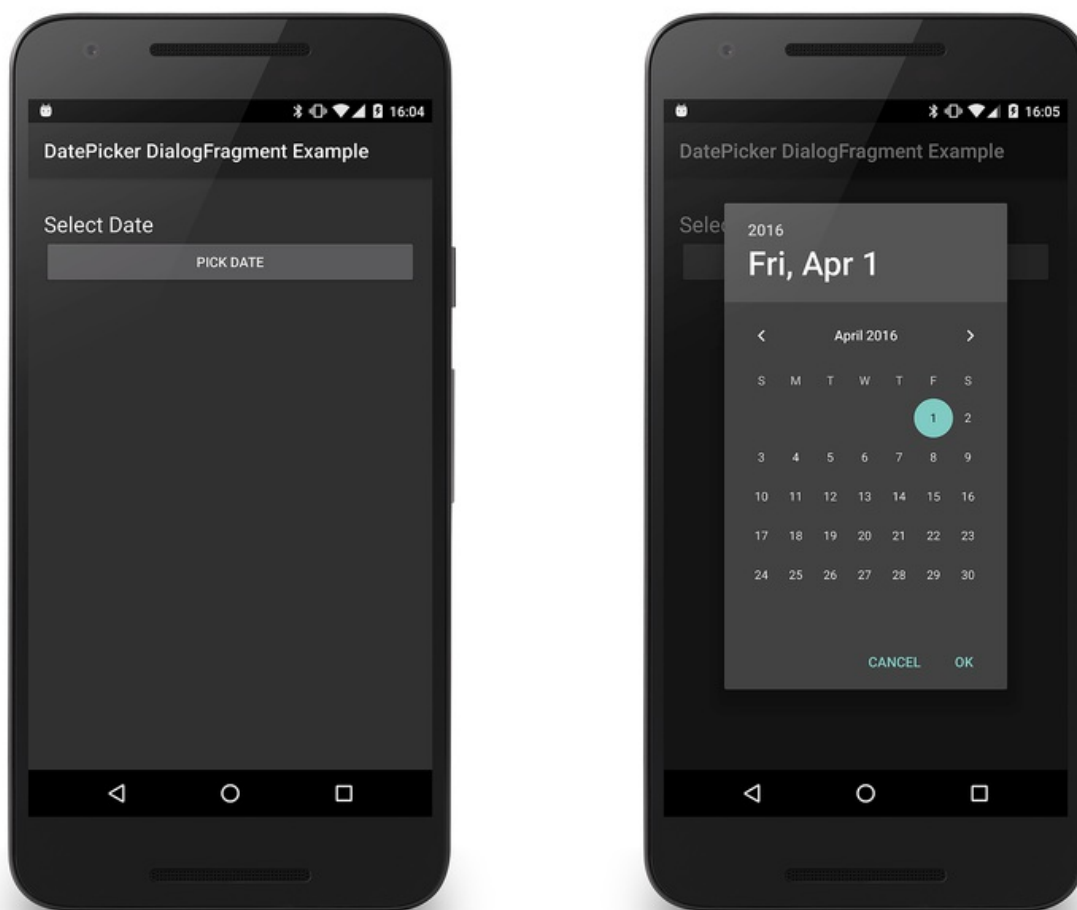
概述

没有用户时必须数据输入到 Android 应用程序的情况。为了帮助解决这个问题, Android 框架提供了 `DatePicker` 小组件和 `DatePickerDialog`。`DatePicker`, 用户可以跨设备和应用程序一致的界面中选择年、月和日。

`DatePickerDialog` 是一个帮助器类, 用于封装 `DatePicker` 在对话框中。

现代的 Android 应用程序应显示 `DatePickerDialog` 中 `DialogFragment`。这将允许应用程序将显示为弹出对话框中的 `DatePicker` 或嵌入在活动中。此外, `DialogFragment` 将管理的生命周期和显示的对话框中, 减少必须实现的代码量。

本指南将演示如何使用 `DatePickerDialog`、包装在 `DialogFragment`。示例应用程序将显示 `DatePickerDialog` 为模式对话框, 当用户单击活动上的按钮。当由用户设置日期时 `TextView` 将更新所选的日期。



要求

本指南的示例应用程序面向 Android 4.1 (API 级别 16) 或更高版本, 但适用于 Android 3.0 (API 级别为 11 或更高版本)。它是 android 的可以支持较旧版本项目和一些代码更改 Android 支持库 v4 加。

使用 DatePicker

此示例将扩展 `DialogFragment`。将托管子类并将其显示 `DatePickerDialog`：

2016

Fri, Apr 1



April 2016



| S | M | T | W | T | F | S |
|----|----|----|----|----|----|----|
| | | | | | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |

CANCEL

OK

当用户选择日期并单击**确定按钮**，`DatePickerDialog` 将调用的方法 `IOndateSetListener.OnDateSet` 。此接口由宿主实现 `DialogFragment` 。如果用户单击**取消按钮**，则片段和对话框将关闭本身。

有几种方法 `DialogFragment` 可以返回到托管的活动的所选的日期：

1. **调用方法或设置属性**–活动可以提供的属性或方法专门为此值设置。
2. **引发事件** – `DialogFragment` 可以定义将为事件时引发 `OnDateSet` 调用。
3. **使用 `Action`** – `DialogFragment` 可以调用 `Action<DateTime>` 显示在活动中的日期。该活动将提供 `Action<DateTime>` 实例化时 `DialogFragment` 。此示例将使用第三种方法，并要求提供活动 `Action<DateTime>` 到 `DialogFragment` 。

扩展 `DialogFragment`

显示的第一步 `DatePickerDialog` 是子类化 `DialogFragment` ，并将其实现 `IOndateSetListener` 接口：

```
public class DatePickerFragment : DialogFragment,
                                DatePickerDialog.IOnDateSetListener
{
    // TAG can be any string of your choice.
    public static readonly string TAG = "X:" + typeof (DatePickerFragment).Name.ToUpper();

    // Initialize this value to prevent NullReferenceExceptions.
    Action<DateTime> _dateSelectedHandler = delegate { };

    public static DatePickerFragment NewInstance(Action<DateTime> onDateSelected)
    {
        DatePickerFragment frag = new DatePickerFragment();
        frag._dateSelectedHandler = onDateSelected;
        return frag;
    }

    public override Dialog OnCreateDialog(Bundle savedInstanceState)
    {
        DateTime currently = DateTime.Now;
        DatePickerDialog dialog = new DatePickerDialog(Activity,
                                                        this,
                                                        currently.Year,
                                                        currently.Month - 1,
                                                        currently.Day);

        return dialog;
    }

    public void OnDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth)
    {
        // Note: monthOfYear is a value between 0 and 11, not 1 and 12!
        DateTime selectedDate = new DateTime(year, monthOfYear + 1, dayOfMonth);
        Log.Debug(TAG, selectedDate.ToString());
        _dateSelectedHandler(selectedDate);
    }
}
```

`NewInstance` 方法调用来实例化一个新 `DatePickerFragment` 。此方法采用 `Action<DateTime>` ，将在用户单击时调用**确定按钮** `DatePickerDialog` 。

当显示片段，Android 将调用的方法 `OnCreateDialog` 。此方法将创建一个新 `DatePickerDialog` 对象，并使用当前日期和回调的对象对其进行初始化（这是当前实例的 `DatePickerFragment` ）。

NOTE

请注意，月份中的值时 `onDateSetListener.onDateSet` 调用在 0 到 11 和不在 1 到 12 范围内。每月天数会在 1 到 31（具体取决于选择了月）的范围内。

显示 DatePickerFragment

现在，`DialogFragment` 已实现，本部分将说明如何在活动中使用片段。在本指南附带示例应用，该活动将实例化 `DialogFragment` 使用 `newInstance` 工厂方法，然后显示时，它调用 `DialogFragment.show`。实例化的一部分 `DialogFragment`，活动会 `onDateSet`，随后会显示在日期 `TextView` 承载活动：

```
[Activity(Label = "@string/app_name", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
{
    TextView _dateDisplay;
    Button _dateSelectButton;

    protected override void onCreate(Bundle bundle)
    {
        base.onCreate(bundle);
        setContentView(Resource.Layout.Main);

        _dateDisplay = FindViewById<TextView>(Resource.Id.date_display);
        _dateSelectButton = FindViewById<Button>(Resource.Id.date_select_button);
        _dateSelectButton.Click += DateSelect_OnClick;
    }

    void DateSelect_OnClick(object sender, EventArgs eventArgs)
    {
        DatePickerFragment frag = DatePickerFragment.newInstance(delegate(DateTime time)
        {
            _dateDisplay.Text =
time.ToLongDateString();
        });
        frag.Show(FragmentManager, DatePickerFragment.TAG);
    }
}
```

总结

此示例介绍了如何显示 `DatePicker` 为弹出模式对话框的 Android 活动一部分的小组件。它提供示例 `DialogFragment` 实现，并讨论 `onDateSetListener` 接口。此示例还演示了如何 `DialogFragment` 可能与主机的活动来显示所选的日期进行交互。

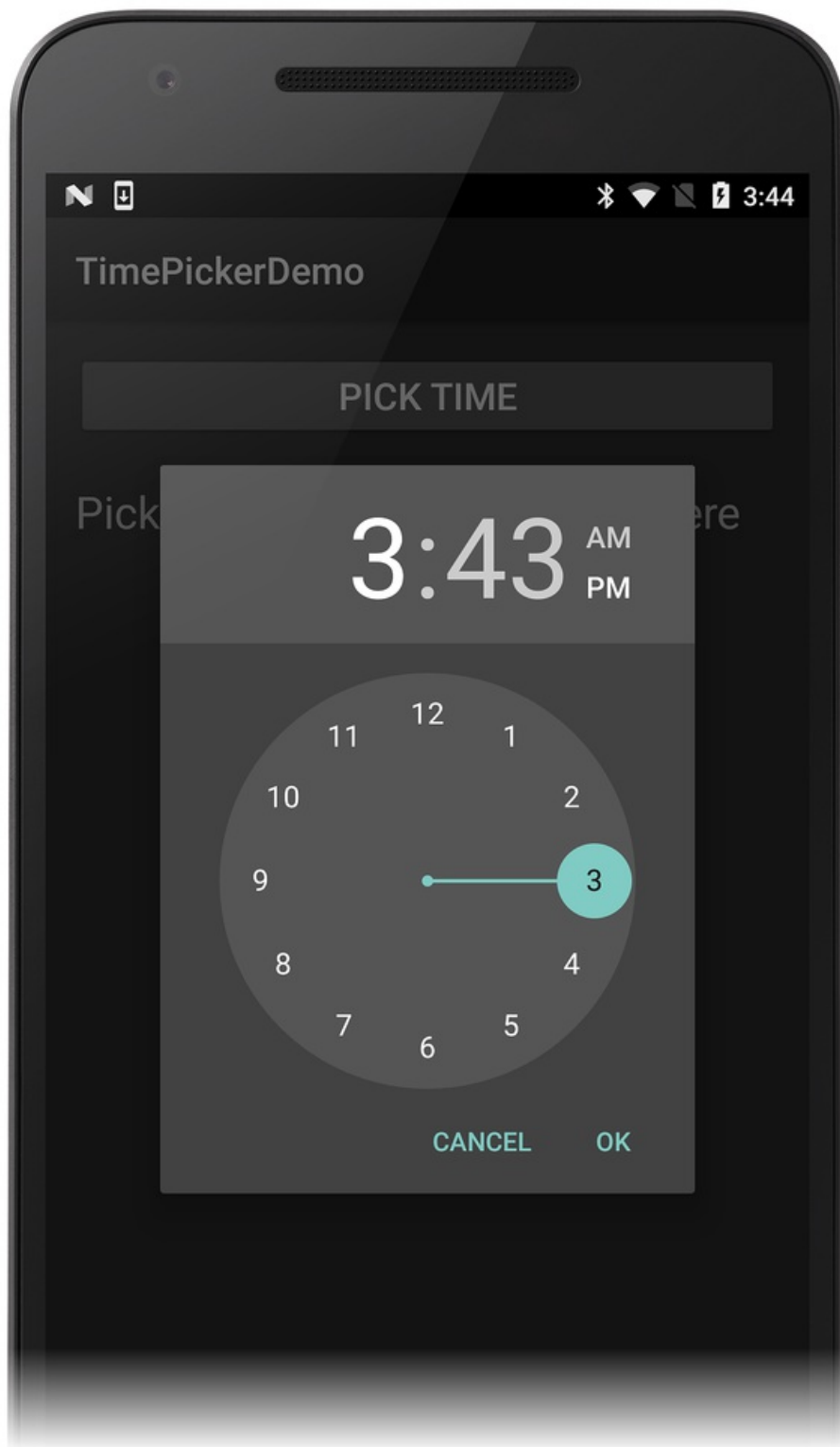
相关链接

- [DialogFragment](#)
- [DatePicker](#)
- [DatePickerDialog](#)
- [DatePickerDialog.OnDateSetListener](#)
- [选择日期](#)

时间选取器

2018/11/14 • [Edit Online](#)

若要提供某种方式让用户选择的时间, 可以使用`TimePicker`。Android 应用程序通常使用`TimePicker`与`TimePickerDialog`选择时间值-这有助于确保跨设备和应用程序一致的接口。`TimePicker` 允许用户在 24 小时或 12 小时上午/下午模式中选择一天的时间。`TimePickerDialog` 是一个帮助器类, 用于封装`TimePicker` 在对话框中。



概述

现代的 Android 应用程序显示 `TimePickerDialog` 中 `DialogFragment`。这使得应用程序以显示 `TimePicker` 作为系统弹出一个对话框或将其嵌入在活动中。此外，`DialogFragment` 管理的生命周期和显示的对话框中，减少必须实现的代码量。

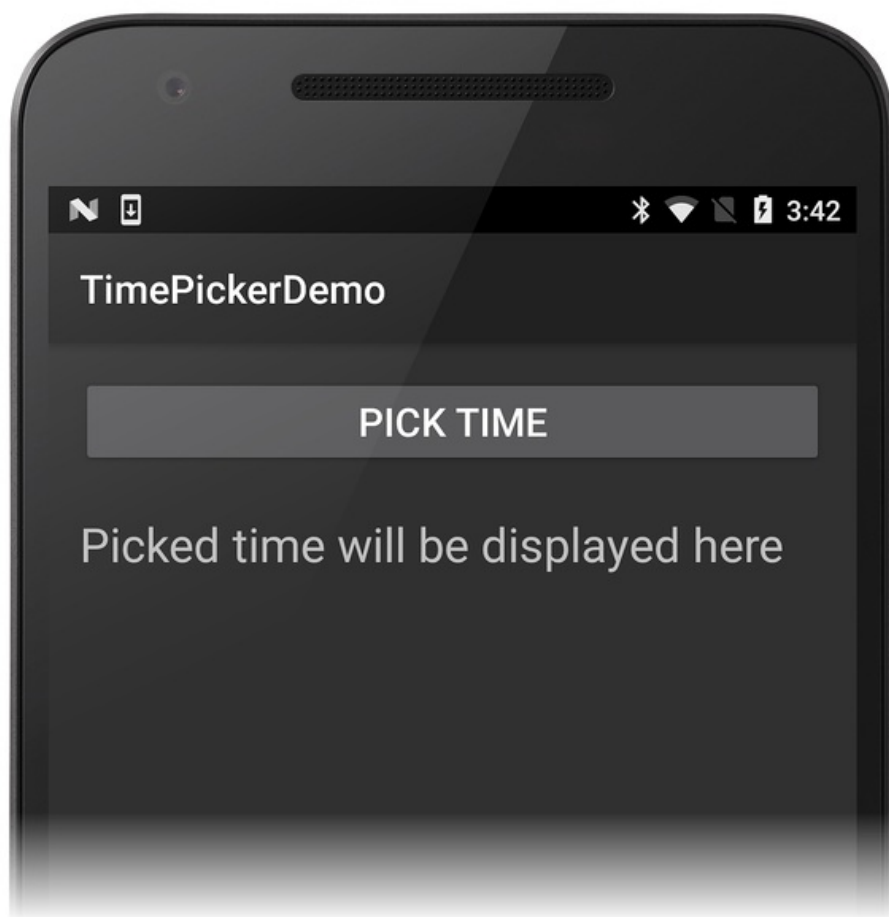
本指南演示如何使用 `TimePickerDialog`、包装在 `DialogFragment`。示例应用程序将显示 `TimePickerDialog` 为模式对话框，当用户单击活动上的按钮。当由用户设置时，对话框中退出，并处理程序更新 `TextView` 与所选的时间在活动屏幕上。

要求

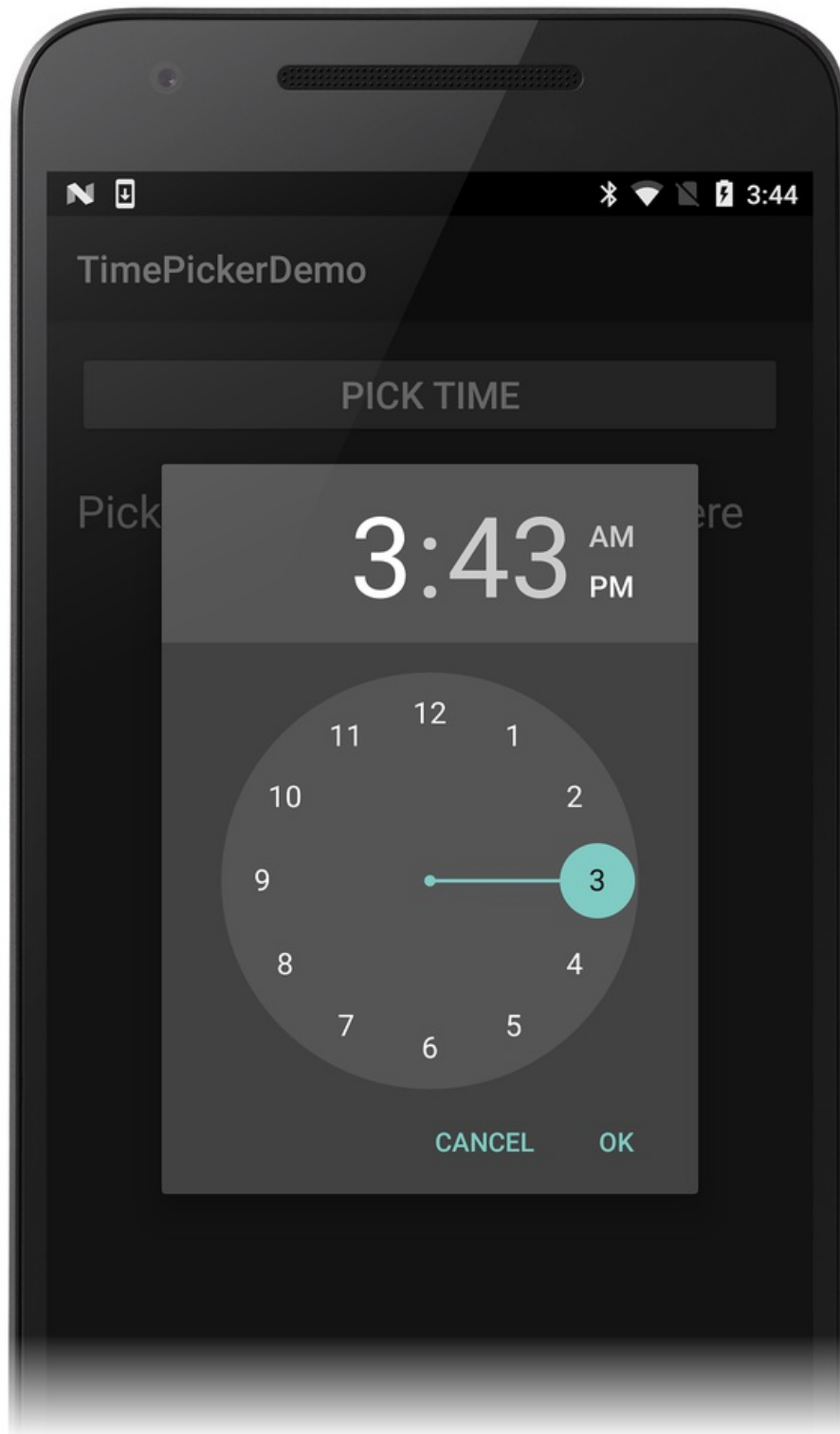
本指南的示例应用程序面向 Android 4.1 (API 级别 16) 或更高版本，但可以在 Android 3.0 (API 级别为 11 或更高版本)。它是 android 的可以支持较旧版本项目和一些代码更改 Android 支持库 v4 加。

使用 TimePicker

此示例扩展 `DialogFragment`；的子类实现 `DialogFragment` (称为 `TimePickerFragment` 下面) 驻留和显示 `TimePickerDialog`。如果首次启动示例应用程序，则会显示选取时间上方的按钮 `TextView`，将用来显示所选的时间：



当您单击选取时间按钮，示例应用程序启动 `TimePickerDialog` 此屏幕截图中所示：



在中 `TimePickerDialog`，选择一次，然后单击**确定按钮**会导致 `TimePickerDialog` 来调用该方法 `OnTimeSetListener.OnTimeSet`。此接口由宿主实现 `DialogFragment` (`TimePickerFragment`，如下所示)。单击**取消按钮**会导致片段和对话框，可以忽略。

`DialogFragment` 托管 Activity 中有三种方法将返回所选的时间：

1. **调用的方法或设置属性**—活动可以提供的属性或方法专门为此值设置。
2. **引发事件** — `DialogFragment` 可以定义将为事件时引发 `OnTimeSet` 调用。
3. **使用 `Action`** — `DialogFragment` 可以调用 `Action<DateTime>` 来显示在活动中的时间。该活动将提供 `Action<DateTime>` 实例化时 `DialogFragment`。

此示例将使用第三个方法，此操作需要活动供应 `Action<DateTime>` 处理程序 `DialogFragment`。

启动应用程序项目

启动一个名为新的 Android 项目 **TimePickerDemo** (如果不熟悉创建 Xamarin.Android 项目, 请参见[Hello, Android](#)若要了解如何创建一个新项目)。

编辑 **Resources/layout/Main.axml** 并将其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center_horizontal"
    android:padding="16dp">
    <Button
        android:id="@+id/select_button"
        android:paddingLeft="24dp"
        android:paddingRight="24dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="PICK TIME"
        android:textSize="20dp" />
    <TextView
        android:id="@+id/time_display"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:paddingTop="22dp"
        android:text="Picked time will be displayed here"
        android:textSize="24dp" />
</LinearLayout>
```

这是一个简单 [LinearLayout](#) 与 [TextView](#) 显示的时间和一个 [按钮](#) 这将打开 `TimePickerDialog`。请注意此布局, 使用硬编码的字符串和维度来使应用程序更简单且更易理解-生产应用程序通常使用这些值进行的资源 (如中所示 [DatePicker](#) 的代码示例)。

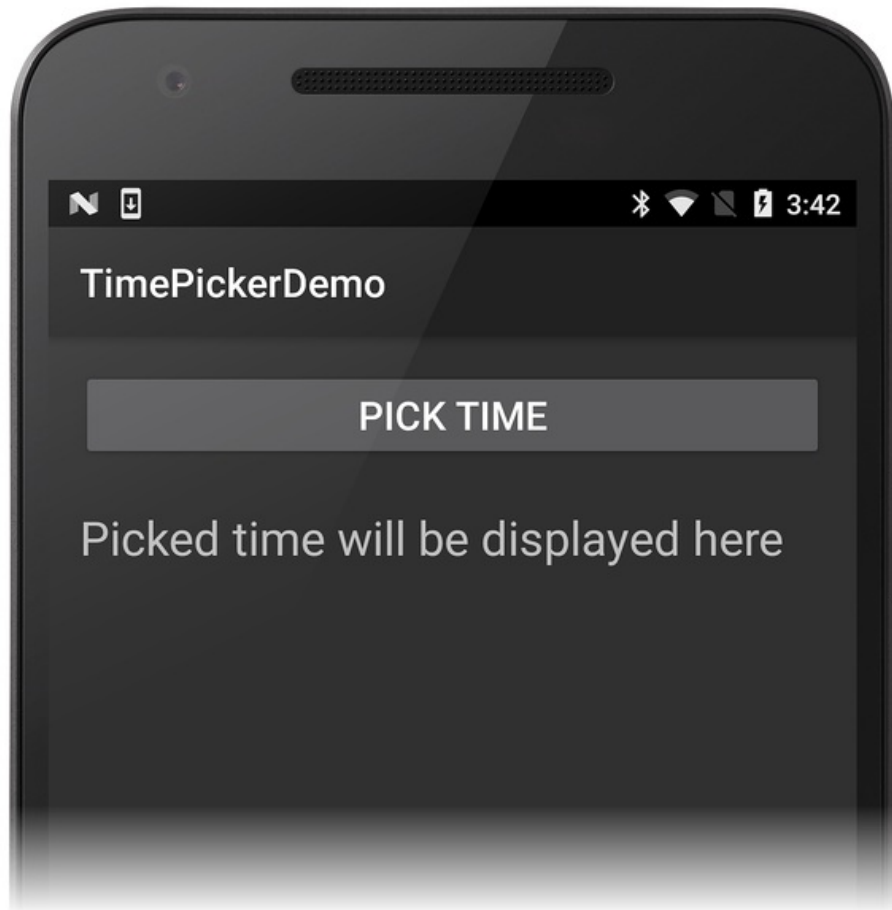
编辑 **MainActivity.cs** 和其内容替换为以下代码:

```
using Android.App;
using Android.Widget;
using Android.OS;
using System;
using Android.Util;
using Android.Text.Format;

namespace TimePickerDemo
{
    [Activity(Label = "TimePickerDemo", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        TextView timeDisplay;
        Button timeSelectButton;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView(Resource.Layout.Main);
            timeDisplay = FindViewById<TextView>(Resource.Id.time_display);
            timeSelectButton = FindViewById<Button>(Resource.Id.select_button);
        }
    }
}
```

当生成并运行此示例中时, 您应看到类似于以下屏幕截图的初始屏幕:



单击选取时间按钮不执行任何操作因为 `DialogFragment` 尚未实现以显示 `TimePicker`。下一步是创建此 `DialogFragment`。

扩展 DialogFragment

若要扩展 `DialogFragment` 用于 `TimePicker`，需要创建一个派生自的子类 `DialogFragment` 并实现 `TimePickerDialog.OnTimeSetListener`。添加以下类 **MainActivity.cs**:

```

public class TimePickerFragment : DialogFragment, TimePickerDialog.IOnTimeSetListener
{
    public static readonly string TAG = "MyTimePickerFragment";
    Action<DateTime> timeSelectedHandler = delegate { };

    public static TimePickerFragment NewInstance(Action<DateTime> onTimeSelected)
    {
        TimePickerFragment frag = new TimePickerFragment();
        frag.timeSelectedHandler = onTimeSelected;
        return frag;
    }

    public override Dialog OnCreateDialog (Bundle savedInstanceState)
    {
        DateTime currentTime = DateTime.Now;
        bool is24HourFormat = DateFormat.Is24HourFormat(Activity);
        TimePickerDialog dialog = new TimePickerDialog
            (Activity, this, currentTime.Hour, currentTime.Minute, is24HourFormat);
        return dialog;
    }

    public void OnTimeSet(TimePicker view, int hourOfDay, int minute)
    {
        DateTime currentTime = DateTime.Now;
        DateTime selectedTime = new DateTime(currentTime.Year, currentTime.Month, currentTime.Day, hourOfDay,
minute, 0);
        Log.Debug(TAG, selectedTime.ToString());
        timeSelectedHandler (selectedTime);
    }
}

```

这 `TimePickerFragment` 类分解为较小的部分和下一节中所述。

DialogFragment 实现

`TimePickerFragment` 实现多个方法：工厂方法中，对话框实例化方法，并 `OnTimeSet` 所需的处理程序方法 `TimePickerDialog.IOnTimeSetListener`。

- `TimePickerFragment` 是子类 `DialogFragment`。它还实现 `TimePickerDialog.IOnTimeSetListener` 接口 (即，它会提供所需 `OnTimeSet` 方法)：

```

public class TimePickerFragment : DialogFragment, TimePickerDialog.IOnTimeSetListener

```

- `TAG` 初始化用于日志记录 (*MyTimePickerFragment* 可以更改想要使用到的任何字符串)。
`timeSelectedHandler` 操作将初始化为一个空的委托，以防止 null 引用异常：

```

public static readonly string TAG = "MyTimePickerFragment";
Action<DateTime> timeSelectedHandler = delegate { };

```

- `NewInstance` 调用工厂方法来实例化一个新 `TimePickerFragment`。此方法采用 `Action<DateTime>` 当用户单击时调用处理程序确定按钮 `TimePickerDialog`：

```

public static TimePickerFragment NewInstance(Action<DateTime> onTimeSelected)
{
    TimePickerFragment frag = new TimePickerFragment();
    frag.timeSelectedHandler = onTimeSelected;
    return frag;
}

```

- Android 片段将会显示，当调用 `DialogFragment` 方法 `OnCreateDialog`。此方法创建一个新 `TimePickerDialog` 对象并初始化该活动，回调对象 (这是当前实例的 `TimePickerFragment`)，以及当前时间：

```
public override Dialog OnCreateDialog (Bundle savedInstanceState)
{
    DateTime currentTime = DateTime.Now;
    bool is24HourFormat = DateFormat.Is24HourFormat(Activity);
    TimePickerDialog dialog = new TimePickerDialog
        (Activity, this, currentTime.Hour, currentTime.Minute, is24HourFormat);
    return dialog;
}
```

- 当用户更改的时间设置 `TimePicker` 对话框中，`OnTimeSet` 调用方法。`OnTimeSet` 创建 `DateTime` 对象使用当前日期和时间（小时和分钟）中的合并用户选定的：

```
public void OnTimeSet(TimePicker view, int hourOfDay, int minute)
{
    DateTime currentTime = DateTime.Now;
    DateTime selectedTime = new DateTime(currentTime.Year, currentTime.Month, currentTime.Day,
        hourOfDay, minute, 0);
}
```

- 这 `DateTime` 对象传递给 `timeSelectedHandler` 向注册 `TimePickerFragment` 在创建时的对象。`OnTimeSet` 调用此处理程序以更新活动的时间显示为所选的时间（在下一节中实现此处理程序）：

```
timeSelectedHandler (selectedTime);
```

显示 TimePickerFragment

既然 `DialogFragment` 已实现的就可以实例化时 `DialogFragment` 使用 `NewInstance` 工厂方法并将其显示通过调用 `DialogFragment.Show`:

添加以下方法 `MainActivity`：

```
void TimeSelectOnClick (object sender, EventArgs eventArgs)
{
    TimePickerFragment frag = TimePickerFragment.NewInstance (
        delegate (DateTime time)
        {
            timeDisplay.Text = time.ToShortTimeString();
        });

    frag.Show(FragmentManager, TimePickerFragment.TAG);
}
```

之后 `TimeSelectOnClick` 实例化 `TimePickerFragment`，它会创建并使用传入的时间值更新活动的时间显示的匿名方法将传递在委托。最后，它将启动 `TimePicker` 对话框片段 (通过 `DialogFragment.Show`) 以显示 `TimePicker` 给用户。

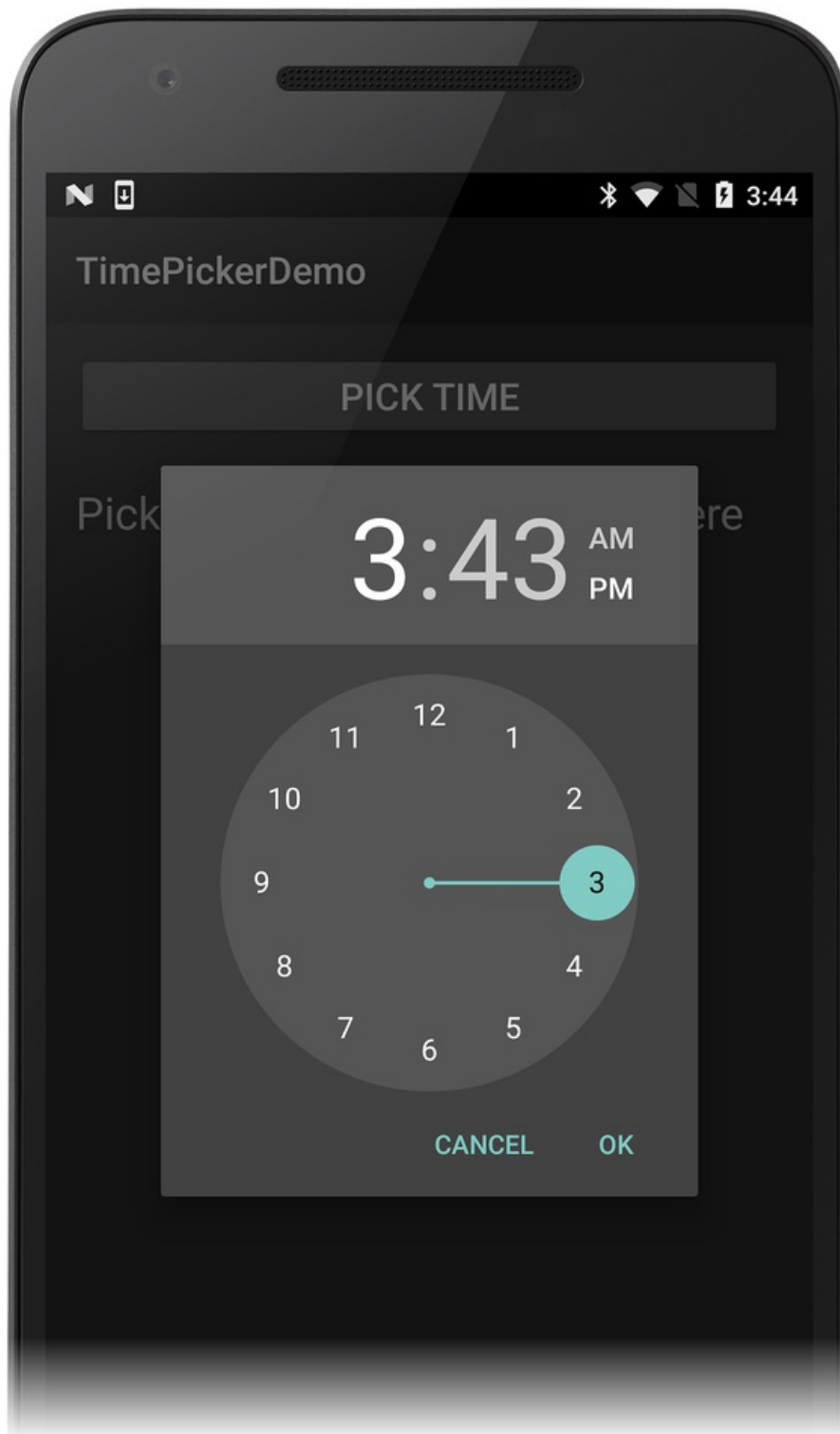
在末尾 `OnCreate` 方法中，添加以下行以附加到的事件处理程序选取时间启动对话框的按钮：

```
timeSelectButton.Click += TimeSelectOnClick;
```

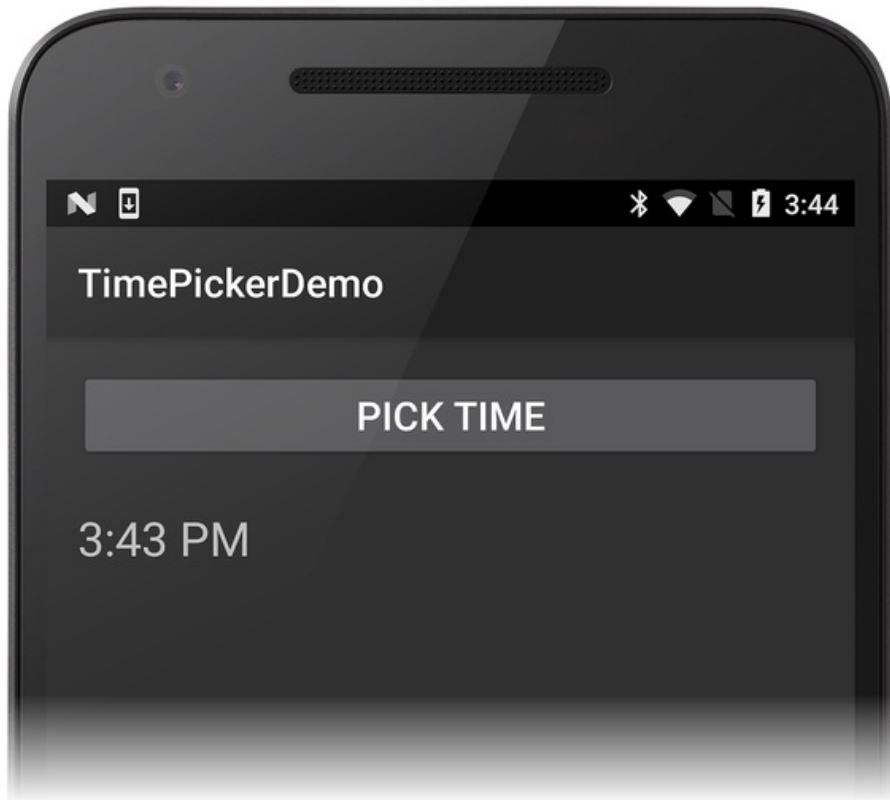
当选取时间单击按钮时，`TimeSelectOnClick` 将调用以显示 `TimePicker` 对话框向用户的片段。

尝试一下！

生成并运行应用。当您单击**选取时间**按钮，`TimePickerDialog` 显示在默认的时间格式的活动（在此示例中为 12 小时 AM/PM 模式下）：



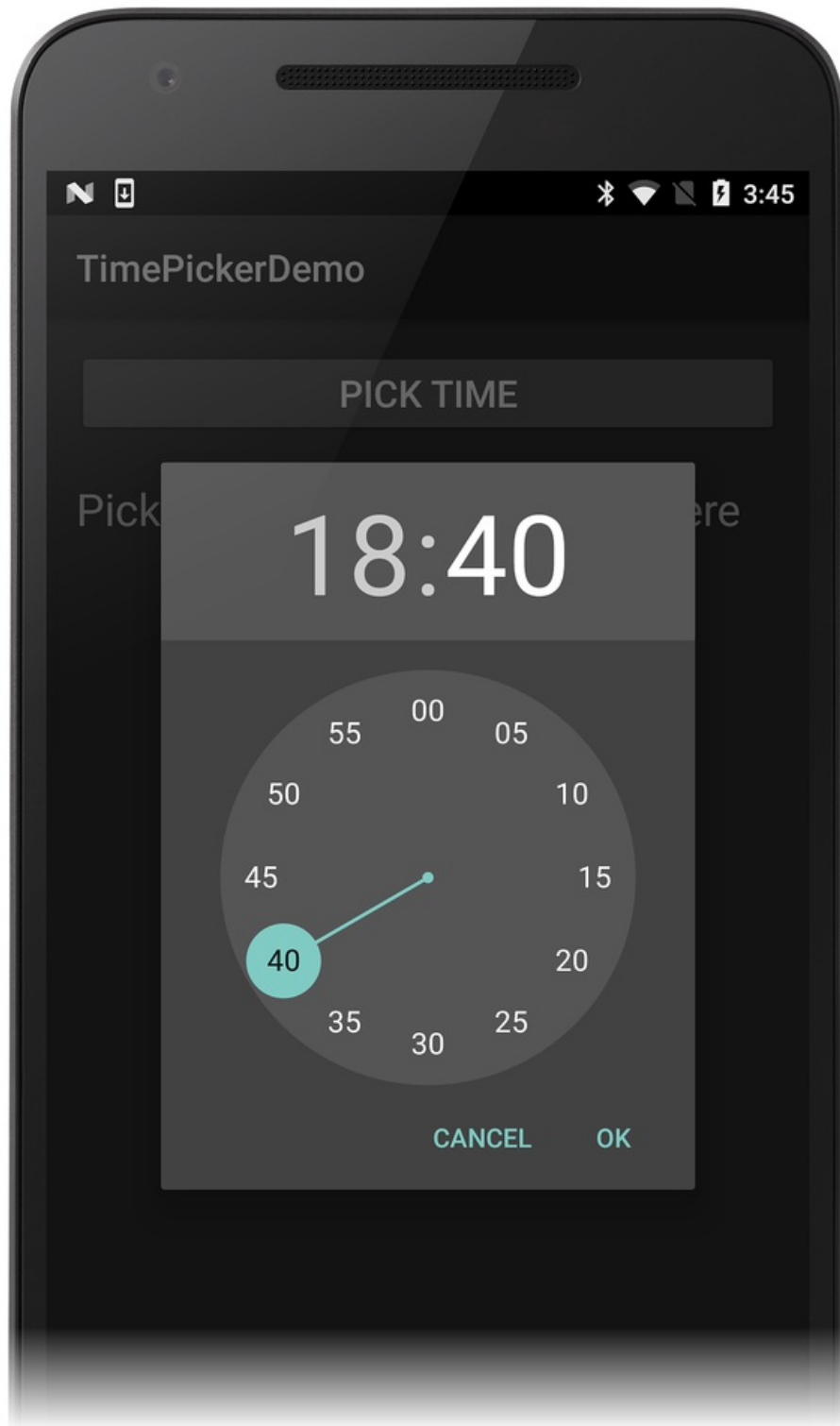
当您单击**确定**中 `TimePicker` 对话框中，该处理程序更新活动的 `TextView` 与所选的时间，然后退出：



接下来，添加以下代码行 `onCreateDialog` 后立即 `is24HourFormat` 进行声明和初始化：

```
is24HourFormat = true;
```

此更改会强制标志传递给 `TimePickerDialog` 构造函数为 `true` 因此 24 小时制模式的使用而不是托管的活动的时间格式。当生成并再次运行该应用时，单击**选取时间**按钮，`TimePicker` 对话框现在显示在 24 小时格式：



因为该处理程序调用 `DateTime.ToShortTimeString` 打印到活动的时间 `TextView`，时间仍将以默认 12 小时 AM/PM 格式打印。

总结

本文介绍了如何显示 `TimePicker` 为弹出模式对话框从 Android 活动的小组件。它提供一个示例 `DialogFragment` 实现和讨论了 `OnTimeSetListener` 接口。此示例还演示了如何将 `DialogFragment` 可以与主机的活动来显示所选的时间进行交互。

相关链接

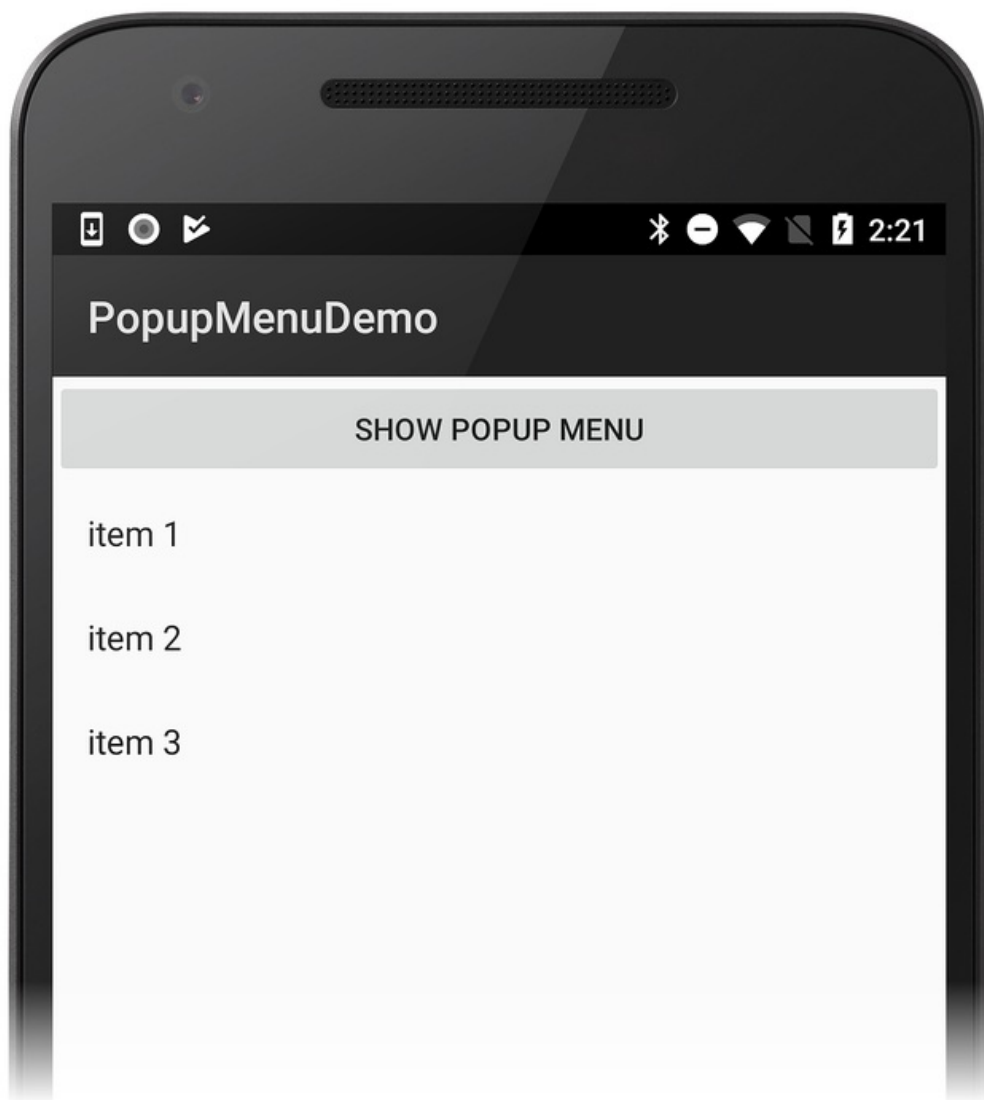
- [DialogFragment](#)
- [TimePicker](#)

- [TimePickerDialog](#)
- [TimePickerDialog.OnTimeSetListener](#)
- [TimePickerDemo \(示例\)](#)

弹出菜单

2018/10/26 • [Edit Online](#)

`PopupMenu` (也称为_快捷菜单_) 是一个菜单，定位到特定视图。在以下示例中，单个活动包含一个按钮。当用户点击按钮时，将显示三个项弹出菜单：



创建弹出菜单

第一步是创建菜单的菜单资源文件并将其置于资源/菜单。例如，以下 XML 是在上面的屏幕截图中显示的两个项菜单的代码 `Resources/menu/popup_menu.xml`：


```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
        android:title="item 1" />
    <item android:id="@+id/item1"
        android:title="item 2" />
    <item android:id="@+id/item1"
        android:title="item 3" />
</menu>
```

接下来, 创建的实例 `PopupMenu` 和锚定到其视图。创建的实例时 `PopupMenu`, 将其构造函数传递的引用 `Context` 以及菜单将附加到的视图。因此, 弹出菜单锚定到此视图在其构造过程。

在以下示例中, `PopupMenu` 在按钮的单击事件处理程序中创建 (其名为 `showPopupMenu`)。此按钮也是到视图 `PopupMenu` 定位, 如下面的代码示例中所示:

```
showPopupMenu.Click += (s, arg) => {
    PopupMenu menu = new PopupMenu (this, showPopupMenu);
};
```

最后, 必须在弹出菜单 夸大与先前创建的菜单资源。在下面的示例中, 调用的菜单 `充气` 方法将添加并将其 `显示` 调用方法来显示它:

```
showPopupMenu.Click += (s, arg) => {
    PopupMenu menu = new PopupMenu (this, showPopupMenu);
    menu.Inflate (Resource.Menu.popup_menu);
    menu.Show ();
};
```

处理菜单事件

当用户选择菜单项, `MenuItemClick` 单击将引发事件并将已解除的菜单。点击菜单外的任意位置将只需关闭它。在任一情况下, 当消除菜单上, 其 `DismissEvent` 将会引发。下面的代码将事件处理程序添加两个 `MenuItemClick` 和 `DismissEvent` 事件:

```
showPopupMenu.Click += (s, arg) => {
    PopupMenu menu = new PopupMenu (this, showPopupMenu);
    menu.Inflate (Resource.Menu.popup_menu);

    menu.MenuItemClick += (s1, arg1) => {
        Console.WriteLine ("{0} selected", arg1.Item.TitleFormatted);
    };

    menu.DismissEvent += (s2, arg2) => {
        Console.WriteLine ("menu dismissed");
    };
    menu.Show ();
};
```

相关链接

- [PopupMenuDemo \(示例\)](#)

RatingBar

2018/10/26 • [Edit Online](#)

RatingBar 是介于 1 到 5 星的分级将显示的 UI 组件。用户可能通过点按的星形本部分中选择某一等级，你将创建一个组件，允许用户向某一评级，提供 `RatingBar` 组件。

Android RatingBar



创建 RatingBar

1. 打开 **Resource/layout/Main.xml** 文件，并添加 `RatingBar` 元素 (内 `LinearLayout`):

```
<RatingBar android:id="@+id/ratingbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="1.0"/>
```

`android:numStars` 属性定义了多少星的评级栏显示。`android:stepSize` 属性定义每个星型的粒度 (例如，值为 `0.5` 将允许半颗星评级)。

2. 若要设置新的分级时可以执行一些操作，请将以下代码添加到末尾 `onCreate()` 方法:

```
RatingBar ratingbar = findViewById<RatingBar>(Resource.Id.ratingbar);

ratingbar.RatingBarChange += (o, e) => {
    Toast.makeText(this, "New Rating: " + ratingbar.Rating.ToString (), ToastLength.Short).Show ();
};
```

这会将捕获 `RatingBar` 组件从与布局 `findViewById` 并设置事件方法，然后定义用户设置分级时要执行的操作。在这种情况下，一个简单 `Toast` 消息将显示新的分级。

3. 运行该应用程序。

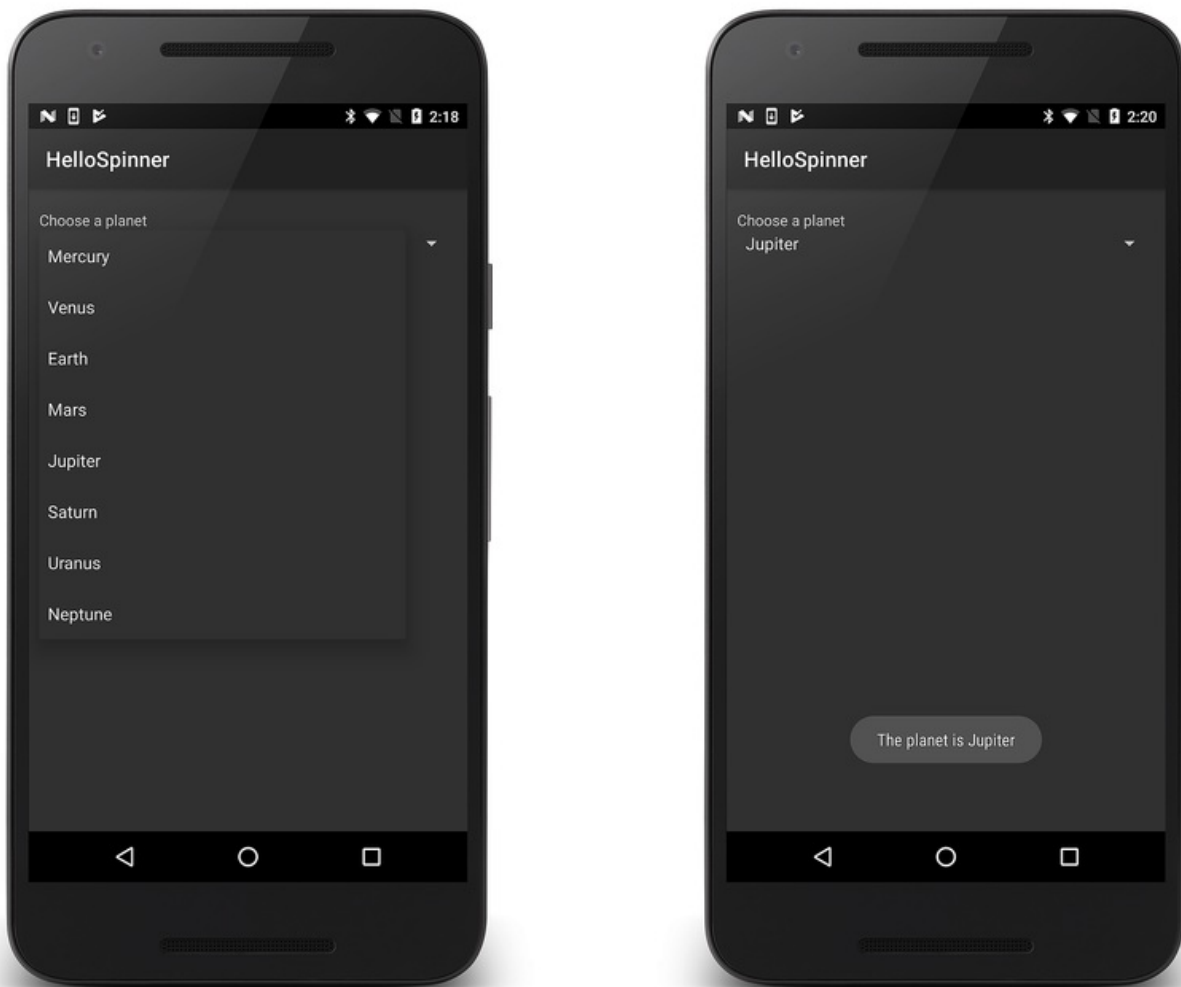
Spinner

2018/10/26 • [Edit Online](#)

Spinner 是一个小组件，提供了用于选择项的下拉列表。本指南介绍如何创建简单的应用程序中微调控件，显示与所选的选择相关联的其他值的修改后跟显示选项列表。

基本的微调框

在本教程的第一部分，你将创建一个简单的微调控件小组件，显示行星的列表。选中一颗后的 toast 消息显示所选的项：



启动一个名为的新项目 **HelloSpinner**。

打开 **Resources/Layout/Main.axml** 并插入以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dip"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dip"
        android:text="@string/planet_prompt"
    />
    <Spinner
        android:id="@+id/spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:prompt="@string/planet_prompt"
    />
</LinearLayout>
```

请注意，`TextView` 的 `android:text` 属性并 `Spinner` 的 `android:prompt` 属性这两个引用相同的字符串资源。此文本表现为小组件的标题。当应用于 `Spinner`，会在选择小组件后，会显示的选择对话框中显示的标题文本。

编辑 **Resources/Values/Strings.xml** 和修改该文件应如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloSpinner</string>
    <string name="planet_prompt">Choose a planet</string>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

第二个 `<string>` 元素定义引用的标题字符串 `TextView` 并 `Spinner` 上述布局中。`<string-array>` 元素定义将显示为列表中的字符串的列表 `Spinner` 小组件。

现在，打开 **MainActivity.cs** 并添加以下 `using` 语句：

```
using System;
```

接下来，插入以下代码 `OnCreate()` 方法：

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Set our view from the "Main" layout resource
    SetContentView (Resource.Layout.Main);

    Spinner spinner = FindViewById<Spinner> (Resource.Id.spinner);

    spinner.ItemSelected += new EventHandler<AdapterView.ItemSelectedEventArgs> (spinner_ItemSelected);
    var adapter = ArrayAdapter.CreateFromResource (
        this, Resource.Array.planets_array, Android.Resource.Layout.SimpleSpinnerItem);

    adapter.SetDropDownViewResource (Android.Resource.Layout.SimpleSpinnerDropDownItem);
    spinner.Adapter = adapter;
}
```

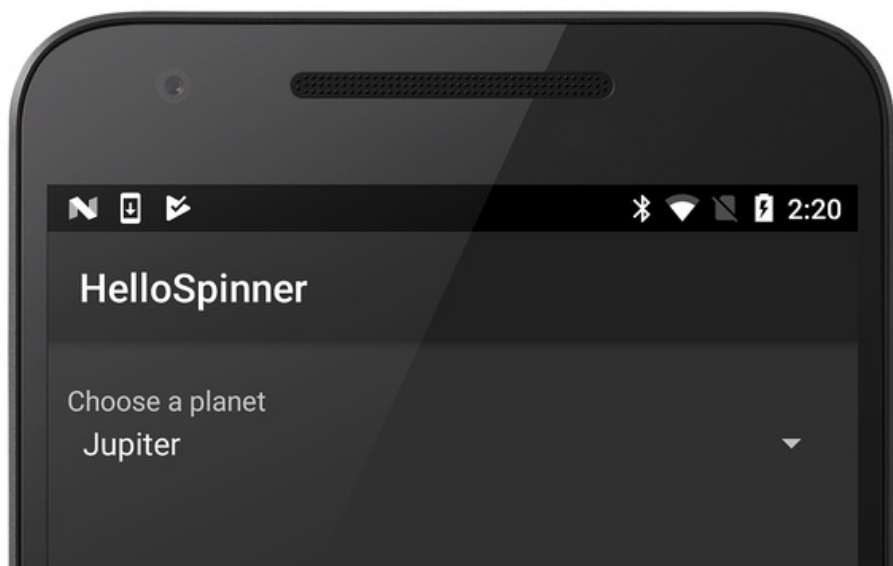
之后 `Main.xml` 布局设置为内容视图中, `Spinner` 小组件捕获从与布局 `FindViewById<(int)>` 的 `CreateFromResource()` 方法然后创建一个新 `ArrayAdapter`, 其中将每个项的字符串数组中绑定到的初始外观 `Spinner` (这是每个项中选择时旋转图标显示方式). `Resource.Array.planets_array` ID 引用 `string-array` 上面定义和 `Android.Resource.Layout.SimpleSpinnerItem` ID 引用由平台定义的标准微调控件外观的布局。 `SetDropDownViewResource` 调用以打开小组件时定义的每个项的外观。最后, `ArrayAdapter` 设置, 使关联的所有与项 `Spinner` 通过设置 `Adapter` 属性。

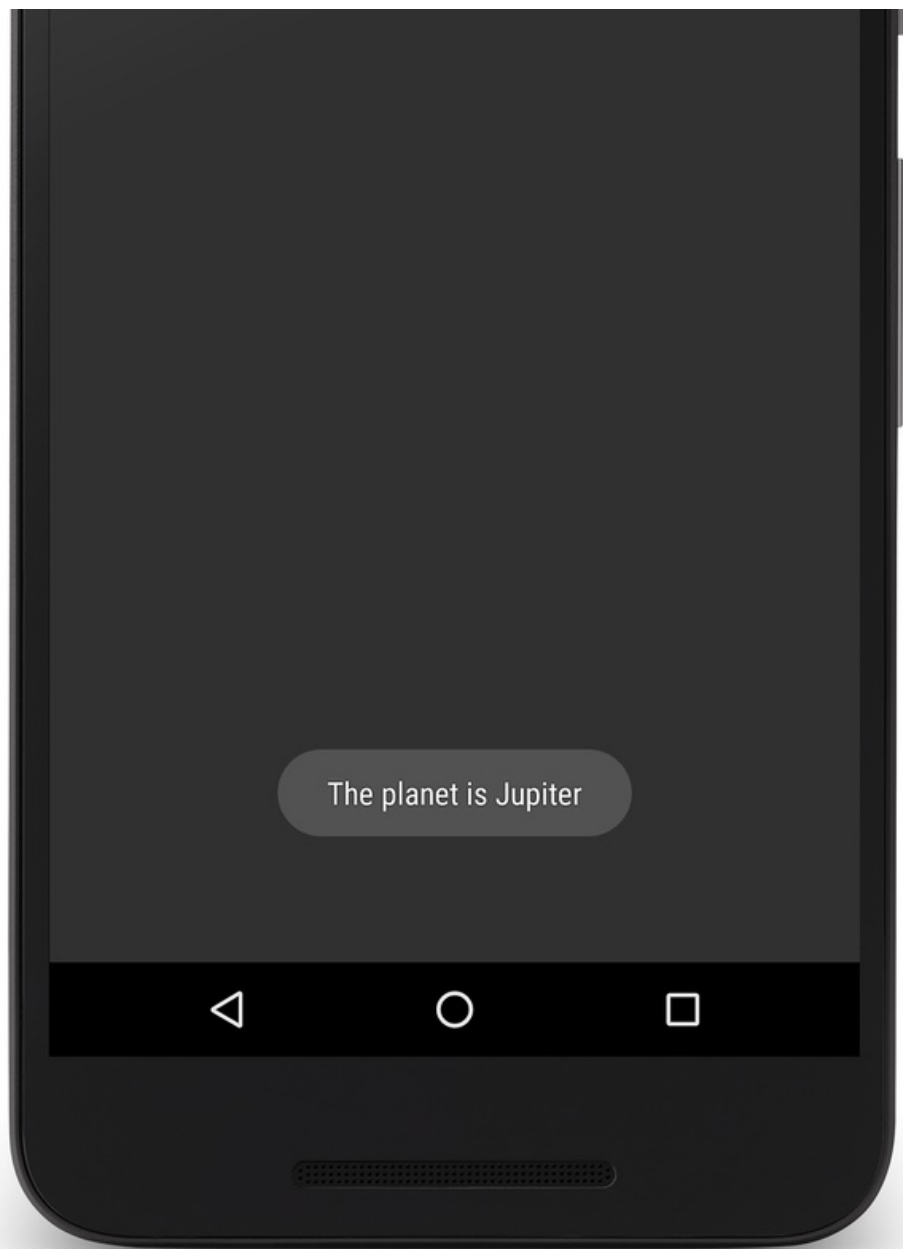
现在提供 `notifys` 应用程序中选择了某个项时的回调方法 `Spinner`。下面是此方法应如下所示:

```
private void spinner_ItemSelected (object sender, AdapterView.ItemSelectedEventArgs e)
{
    Spinner spinner = (Spinner)sender;
    string toast = string.Format ("The planet is {0}", spinner.GetItemAtPosition (e.Position));
    Toast.MakeText (this, toast, ToastLength.Long).Show ();
}
```

当选中某个项时, 发件人被强制转换为 `Spinner`, 以便可以访问项。使用 `Position` 上的属性 `ItemEventArgs`, 可以找出所选对象的文本并使用它来显示 `Toast`。

运行该应用程序;它应如下所示:





使用键/值对的微调框

通常很有必要将使用 `Spinner` 以显示与某种类型的应用使用的数据相关联的密钥值。因为 `Spinner` 不起作用，你必须直接与键/值对单独存储键/值对，填充 `Spinner` 与密钥值，然后使用在旋转图标中的所选密钥位置中查找关联的数据值。

在以下步骤中，**HelloSpinner**应用修改为显示所选的地球平均温度：

添加以下 `using` 语句**MainActivity.cs**:

```
using System.Collections.Generic;
```

添加以下实例变量 `MainActivity` 类。此列表将包含键/值对的行星和其平均温度：

```
private List<KeyValuePair<string, string>> planets;
```

在中 `OnCreate` 方法中, 添加以下代码之前 `adapter` 声明:

```
planets = new List<KeyValuePair<string, string>>
{
    new KeyValuePair<string, string>("Mercury", "167 degrees C"),
    new KeyValuePair<string, string>("Venus", "464 degrees C"),
    new KeyValuePair<string, string>("Earth", "15 degrees C"),
    new KeyValuePair<string, string>("Mars", "-65 degrees C"),
    new KeyValuePair<string, string>("Jupiter", "-110 degrees C"),
    new KeyValuePair<string, string>("Saturn", "-140 degrees C"),
    new KeyValuePair<string, string>("Uranus", "-195 degrees C"),
    new KeyValuePair<string, string>("Neptune", "-200 degrees C")
};
```

此代码创建简单的存储的行星和其关联的平均温度。(在实际应用中, 数据库通常用于存储密钥和其关联的数据。)

上述代码中后立即, 添加以下行, 以提取密钥, 并将它们放入列表 (按顺序):

```
List<string> planetNames = new List<string>();
foreach (var item in planets)
    planetNames.Add (item.Key);
```

传递到此列表 `ArrayAdapter` 构造函数 (而不是 `planets_array` 资源):

```
var adapter = new ArrayAdapter<string>(this,
    Android.Resource.Layout.SimpleSpinnerItem, planetNames);
```

修改 `spinner_ItemSelected`, 以便使用所选的位置以查找与所选的全球关联的值 (温度):

```
private void spinner_ItemSelected(object sender, AdapterView.ItemSelectedEventArgs e)
{
    Spinner spinner = (Spinner)sender;
    string toast = string.Format("The mean temperature for planet {0} is {1}",
        spinner.GetItemAtPosition(e.Position), planets[e.Position].Value);
    Toast.MakeText(this, toast, ToastLength.Long).Show();
}
```

运行该应用程序, toast 应如下所示:



资源

- [Resource.Layout](#)
- [ArrayAdapter](#)
- [Spinner](#)

此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution](#) 许可证.

开关

2018/10/26 • [Edit Online](#)

`Switch` 小组件（如下所示）允许用户以两个状态，例如在之间切换或禁用。`Switch` 默认值为 OFF。小组件是在其 ON 和 OFF 状态如下所示：

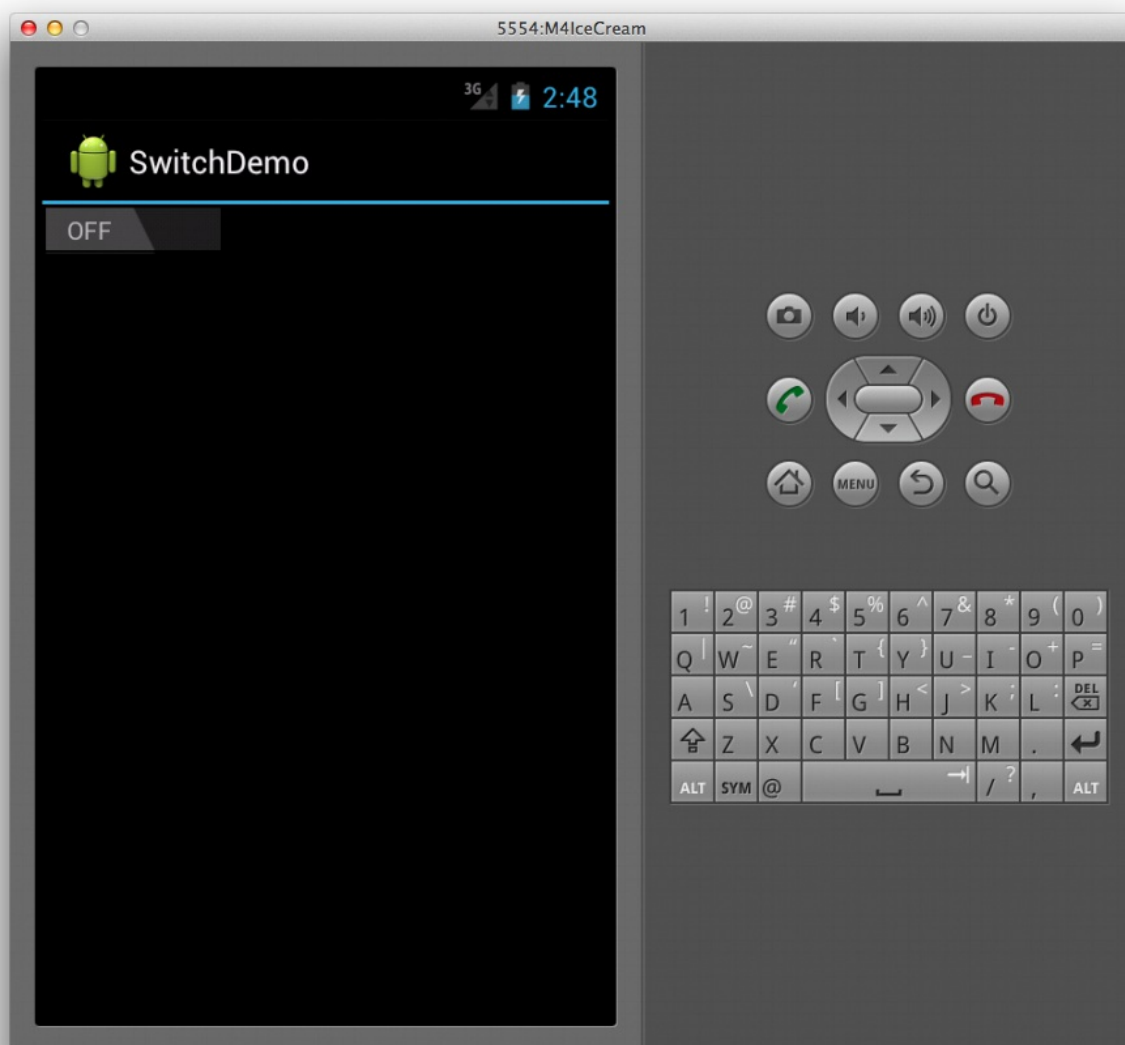


创建交换机

若要创建一个开关，只需声明 `Switch` XML 中的元素，如下所示：

```
<Switch android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

这会创建一个基本的开关，如下所示：



更改默认值

可配置该控件将显示为 ON 和 OFF 状态的文本和默认值。例如, 若要使开关默认为 ON 和读取而不是 OFF/ON 否 / 是, 我们可以设置 `checked`, `textOn`, 和 `textOff` 以下 XML 中的属性。

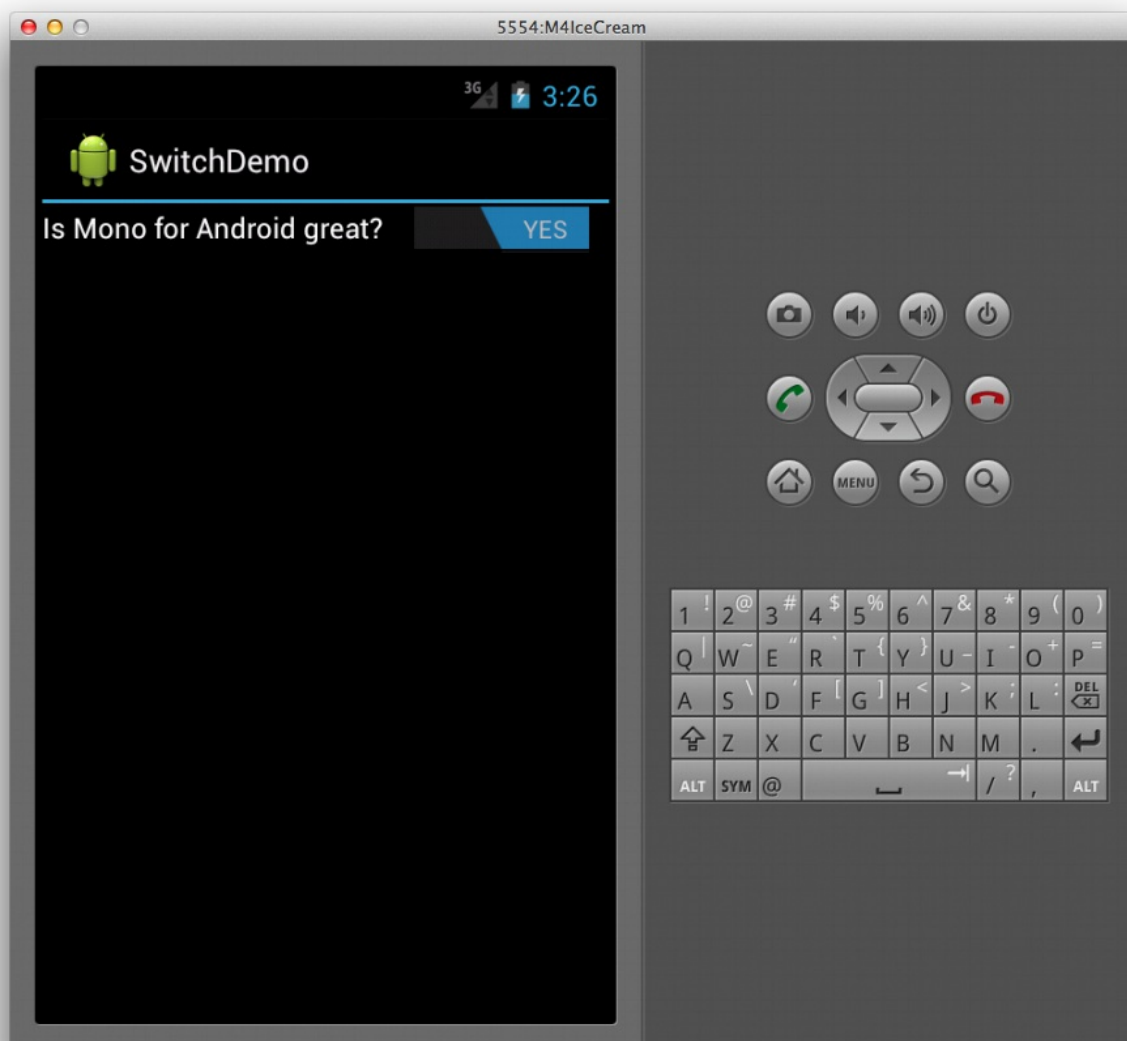
```
<Switch android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:textOn="YES"
        android:textOff="NO" />
```

提供一个标题

`Switch` 小组件还支持通过设置包括的文本标签 `text` 属性, 如下所示:

```
<Switch android:text="Is Xamarin.Android great?"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:textOn="YES"
        android:textOff="NO" />
```

此标记将生成下面的屏幕截图在运行时:



当 `Switch` 的值发生更改, 它会发出 `CheckedChange` 事件。例如, 下面的代码中我们捕获此事件并提供 `Toast` 使用一条消息的小组件基于 `isChecked` 的值 `Switch`, 它作为的一部分传递给事件处理程序 `CompoundButton.CheckedChangeEventArgs` 参数。

```
Switch s = FindViewById<Switch> (Resource.Id.monitored_switch);

s.CheckedChange += delegate(object sender, CompoundButton.CheckedChangeEventArgs e) {
    var toast = Toast.MakeText (this, "Your answer is " +
        (e.IsChecked ? "correct" : "incorrect"), ToastLength.Short);
    toast.Show ();
};
```

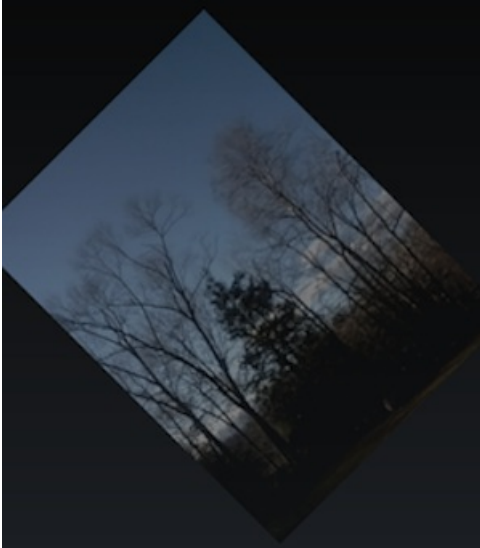
相关链接

- [SwitchDemo \(示例\)](#)
- [选项卡布局教程](#)

TextureView

2018/10/26 • [Edit Online](#)

`TextureView` 类是一个使用硬件加速 2D 呈现若要启用的视频或 OpenGL 内容流, 要显示的视图。例如, 下面的屏幕截图显示了 `TextureView` 显示设备的摄像机从实时源:



与不同 `SurfaceView` 类, 还可用于显示 OpenGL 或视频内容, `TextureView` 不呈现到一个单独的窗口。因此, `TextureView` 能够支持与任何其他视图一样视图转换。例如, 旋转 `TextureView` 可以通过只需设置来实现其 `Rotation` 属性中, 通过设置其透明度其 `Alpha` 属性中, 依次类推。

因此, 对于 `TextureView` 我们现在可以从照相机中执行操作, 例如显示实时流和转换, 如下面的代码中所示:

```

public class TextureViewActivity : Activity,
    TextureView.ISurfaceTextureListener
{
    Camera _camera;
    TextureView _textureView;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);
        _textureView = new TextureView (this);
        _textureView.SurfaceTextureListener = this;

        SetContentView (_textureView);
    }

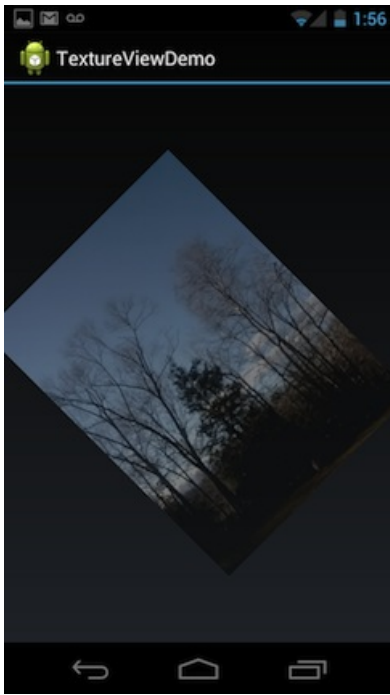
    public void OnSurfaceTextureAvailable (
        Android.Graphics.SurfaceTexture surface,
        int width, int height)
    {
        _camera = Camera.Open ();
        var previewSize = _camera.GetParameters ().PreviewSize;
        _textureView.LayoutParameters =
            new FrameLayout.LayoutParams (previewSize.Width,
                previewSize.Height, (int)GravityFlags.Center);

        try {
            _camera.SetPreviewTexture (surface);
            _camera.StartPreview ();
        } catch (Java.IO.IOException ex) {
            Console.WriteLine (ex.Message);
        }

        // this is the sort of thing TextureView enables
        _textureView.Rotation = 45.0f;
        _textureView.Alpha = 0.5f;
    }
    ...
}

```

上面的代码创建 `TextureView` 中的活动实例 `OnCreate` 方法，并设置为活动 `TextureView` 的 `SurfaceTextureListener`。要 `SurfaceTextureListener`，活动实现 `TextureView.ISurfaceTextureListener` 接口。系统将调用 `OnSurfaceTextAvailable` 方法时 `SurfaceTexture` 随时可供使用。在此方法中，我们需要 `SurfaceTexture`，将传递中，并将其设置为照相机的预览纹理。然后我们立即执行正常基于视图的操作，例如设置 `Rotation` 和 `Alpha`，如上面的示例。生成的应用程序，在设备上运行如下所示：



若要使用 `TextureView`，硬件加速必须启用，它将默认情况下，从 API 级别 14 开始。此外，因为此示例使用照相机，同时 `android.permission.CAMERA` 权限和 `android.hardware.camera` 必须在设置功能 **AndroidManifest.xml**。

相关链接

- [TextureViewDemo（示例）](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

Toolbar

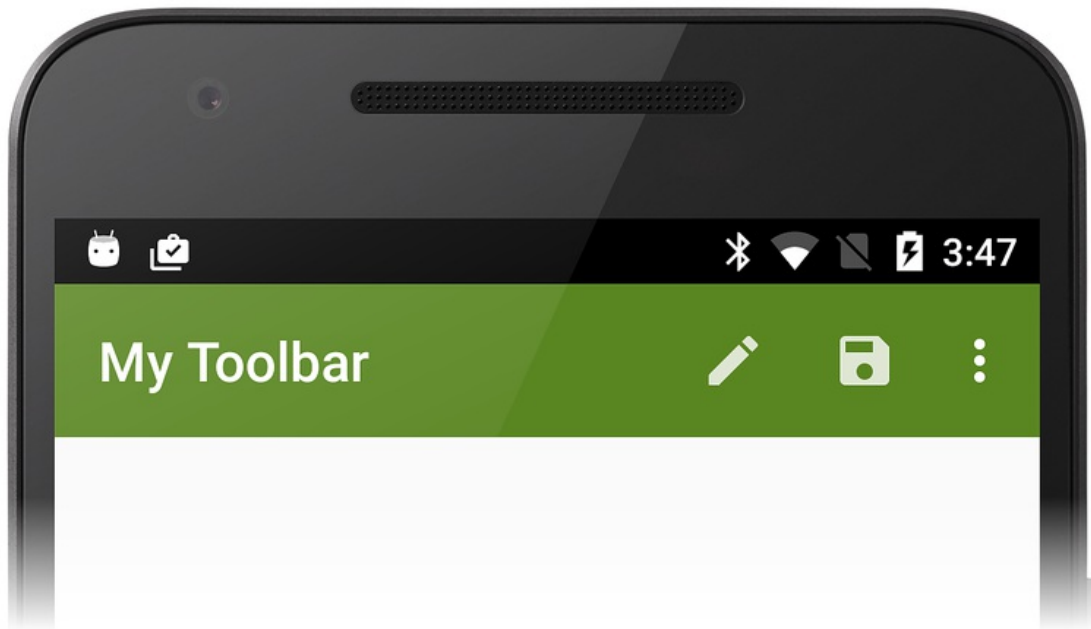
2018/10/26 • [Edit Online](#)

工具栏是提供了更大的灵活性比默认操作栏的操作栏组件：可以在应用中任意位置放置，可以更改其大小，并且它可以使用不同于应用的主题配色方案。此外，每个应用屏幕可以有多个工具栏。

概述

任何 Android 活动的一个重要的设计元素是操作栏。在操作栏是用于导航、搜索、菜单和品牌 Android 应用中的 UI 组件。在 Android 5.0 Lollipop，操作栏之前的 Android 版本 (也称为应用程序栏) 已提供此功能的建议的组件。

`Toolbar` (在 Android 5.0 Lollipop 中引入) 的小组件可以看作操作栏接口的泛化-它旨在替换操作栏。`Toolbar` 可以应用的布局中，在任何地方使用，它更自定义操作栏比。以下屏幕截图展示了自定义 `Toolbar` 本指南中创建的示例：



有一些重要差异 `Toolbar` 和操作栏：

- 一个 `Toolbar` 可以放置在用户界面中的任何地方。
- 可以在同一屏幕上显示多个工具栏。
- 如果使用片段，每个片段可以拥有其自身 `Toolbar`。
- 一个 `Toolbar` 可以配置为跨越仅屏幕的部分的宽度。
- 因为 `Toolbar` 未绑定到活动的窗口装潢的配色方案，它可以在视觉上不同的配色方案。
- 与操作栏中，不同 `Toolbar` 不包括在左侧的图标。在右侧其菜单使用较少的空间。
- `Toolbar` 高度是可调整。
- 其他视图可以包含在 `Toolbar`。

一个 `Toolbar` 可以包含一个或多个以下元素：

- [导航按钮](#)
- [品牌的徽标图像](#)
- [标题和副标题](#)
- [自定义视图](#)
- [操作菜单](#)
- [溢出菜单](#)

Google [Material Design 准则](#)、建议利用这些元素以不同的外观（而不是仅依赖于应用程序图标和标题）赋予应用程序。

本指南介绍了最常用的 `Toolbar` 方案：

- 替换与某个活动的默认操作栏 `Toolbar`。
- 添加另一个 `Toolbar` 向活动。
- 使用 **Android 支持库 v7 AppCompat** 库（称为 *AppCompat* 本指南的其余部分）来部署 `Toolbar` 早期版本的 Android 上。

要求

`Toolbar` Android 5.0 Lollipop (API 21) 及更高版本，提供。当以 Android 为目标低于 Android 5.0 发布后时，使用 [Android 支持库 v7 AppCompat](#)，其中提供了向后兼容 `Toolbar` 支持 NuGet 包中。[工具栏兼容性](#) 说明如何使用此库。

相关链接

- [棒棒糖形工具栏（示例）](#)
- [AppCompat 工具栏（示例）](#)

替换操作栏

2018/10/26 • [Edit Online](#)

概述

对于最常见的一个用途 `Toolbar` 是默认操作栏替换为自定义 `Toolbar` (创建新的 Android 项目后, 它使用默认操作栏)。因为 `Toolbar` 提供的功能将品牌的徽标、标题、菜单项、导航按钮和甚至是自定义视图添加到应用程序栏部分中的活动的 UI, 它提供重大升级通过默认操作栏。

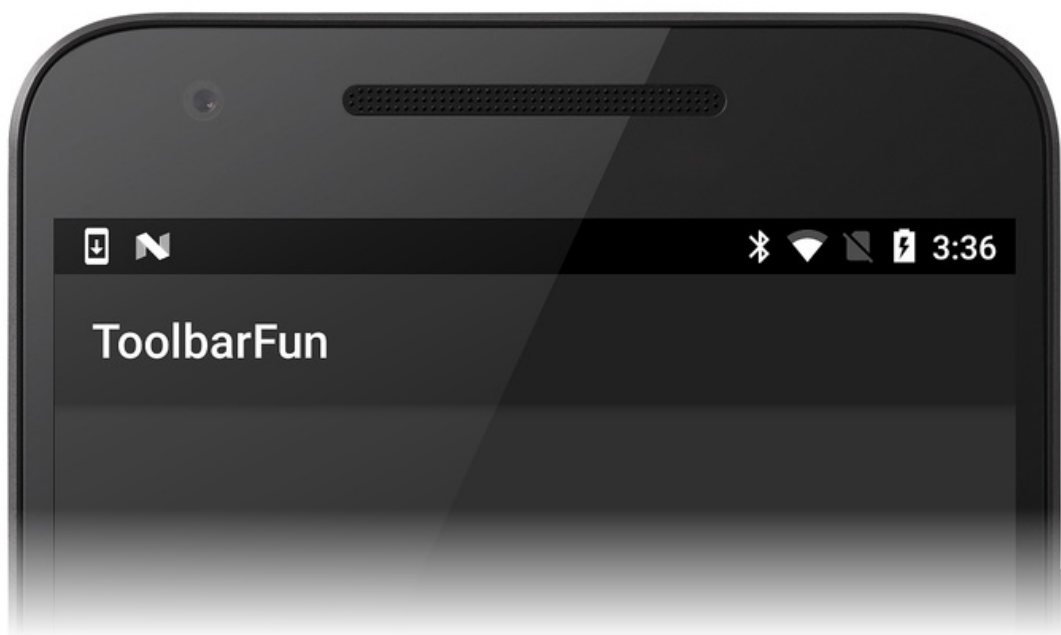
若要替换为与应用程序的默认操作栏 `Toolbar` :

1. 创建一个新的自定义主题, 并修改应用程序的属性, 使其使用此新主题。
2. 禁用 `windowActionBar` 属性中的自定义主题, 并启用 `windowNoTitle` 属性。
3. 定义布局 `Toolbar` 。
4. 包括 `Toolbar` 中的活动的布局 **Main.xml** 布局文件。
5. 将代码添加到活动的 `onCreate` 方法来查找 `Toolbar` , 并调用 `setSupportActionBar` 安装 `ToolBar` 作为操作栏。

以下部分介绍此过程的详细信息。创建一个简单的应用程序和其操作栏替换为自定义 `Toolbar` 。

启动应用程序项目

创建一个名为新的 Android 项目 **ToolbarFun** (请参阅[Hello, Android](#)有关创建新的 Android 项目的详细信息)。创建此项目后, 将目标和最低 Android API 级别设置为 **Android 5.0 (API 级别 21-棒棒糖)** 或更高版本。有关设置 Android 版本级别的详细信息, 请参阅[了解 Android API 级别](#)。当生成和运行应用程序时, 它会显示默认操作栏, 此屏幕截图中所示:



创建自定义主题

打开资源/**values**目录并创建一个新的文件称为**styles.xml**。其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
    <style name="MyTheme" parent="@android:style/Theme.Material.Light.DarkActionBar">
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowActionBar">false</item>
        <item name="android:colorPrimary">#5A8622</item>
    </style>
</resources>
```

此 XML 定义新的自定义主题名为**MyTheme**基于**Theme.Material.Light.DarkActionBar**棒棒糖形中的主题。

`windowNoTitle` 属性设置为 `true` 隐藏的标题栏：

```
<item name="android:windowNoTitle">true</item>
```

若要显示的自定义工具栏，默认值 `ActionBar` 必须禁用：

```
<item name="android:windowActionBar">false</item>
```

Olive-green `colorPrimary` 设置用于在工具栏的背景色：

```
<item name="android:colorPrimary">#5A8622</item>
```

应用自定义主题

编辑**properties/Androidmanifest.xml**并添加以下 `android:theme` 归于 `<application>` 元素，以便该应用使用 `MyTheme` 自定义主题：

```
<application android:label="@string/app_name" android:theme="@style/MyTheme"></application>
```

有关将自定义主题应用到应用程序的详细信息，请参阅[使用自定义主题](#)。

定义工具栏布局

在中资源/布局 目录中，创建名为的新文件**toolbar.xml**。其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?android:attr/actionBarSize"
    android:background="?android:attr/colorPrimary"
    android:theme="@android:style/ThemeOverlay.Material.Dark.ActionBar"/>
```

此 XML 定义自定义 `Toolbar` 替换默认操作栏。最小高度 `Toolbar` 设置它将替换操作栏的大小为：

```
android:minHeight="?android:attr/actionBarSize"
```

背景色 `Toolbar` 设置为前面定义的 olive-green 颜色**styles.xml**：

```
android:background="?android:attr/colorPrimary"
```

棒棒糖形，从开始 `android:theme` 属性可用于设置样式的单个视图。 `ThemeOverlay.Material` 棒棒糖形中引入的主题，让可以覆盖默认 `Theme.Material` 覆盖相关的属性，以使它们浅色或深色主题。在此示例中， `Toolbar` 使用深色主题，以便其内容是亮色：

```
android:theme="@android:style/ThemeOverlay.Material.Dark.ActionBar"
```

使用此设置，以便菜单项的较暗的背景色与之相反。

包括工具栏布局

编辑布局文件 **Resources/layout/Main.xml** 并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include
        android:id="@+id/toolbar"
        layout="@layout/toolbar" />
</RelativeLayout>
```

此布局包括 `Toolbar` 中定义 **toolbar.xml**，并使用 `RelativeLayout` 以指定 `Toolbar` 要放置在（上面显示的按钮）的用户界面的最顶部。

查找和激活工具栏

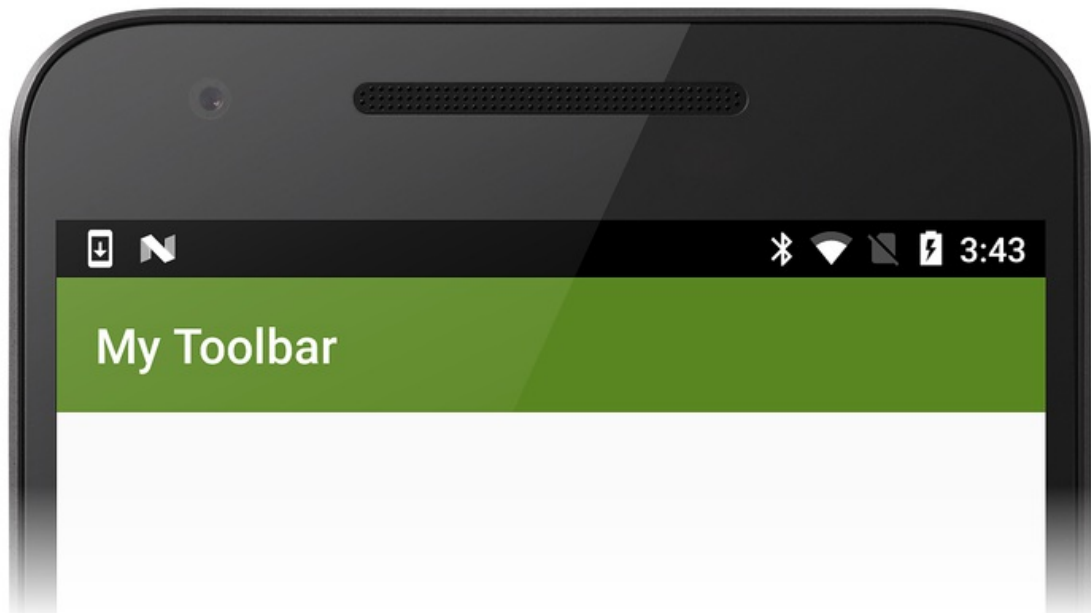
编辑 **MainActivity.cs** 并添加以下 using 语句：

```
using Android.Views;
```

此外，将以下代码行添加到末尾 `OnCreate` 方法：

```
var toolbar = FindViewById<Toolbar>(Resource.Id.toolbar);
SetSupportActionBar(toolbar);
ActionBar.Title = "My Toolbar";
```

此代码将查找 `Toolbar` 并调用 `SetSupportActionBar` 以便 `Toolbar` 将执行默认操作栏特征。工具栏的标题更改为**我工具栏**。在此代码示例中，所示 `ToolBar` 可以作为操作栏直接引用。编译并运行此应用-的自定义 `Toolbar` 代替默认操作栏显示：



请注意，`Toolbar` 独立于样式 `Theme.Material.Light.DarkActionBar` 应用于应用程序的其余部分的主题。

如果运行该应用程序时出现异常，请参阅[故障排除](#)下面一节。

添加菜单项

在本部分中，菜单将添加到 `Toolbar`。右上角区域 `ToolBar` 是保留的菜单项-每个菜单项 (也称为 *操作项*) 可以执行的当前活动中的操作或它可以执行代表整个应用程序的操作。

若要添加到菜单 `Toolbar`：

1. 添加到菜单图标 (如果需要) `mipmap-` 的应用程序项目的文件夹。Google 上提供了一套免费的菜单图标[材料图标](#)页。
2. 通过添加新的菜单资源文件下定义的菜单项的内容资源/菜单。
3. 实现 `onCreateOptionsMenu` 活动的方法-此方法增大的菜单项。
4. 实现 `onOptionsItemSelected` 活动的方法-点击的菜单项时，此方法执行的操作。

以下各节演示此过程的详细信息，通过添加编辑并保存的自定义的菜单项 `Toolbar`。

安装菜单图标

在继续进行 `ToolbarFun` 示例应用将菜单图标添加到应用程序项目。下载[工具栏图标](#)、解压缩，然后将所提取的内容复制 `mipmap-` 到项目文件夹 `mipmap-` 下的文件夹 `ToolbarFun` /资源和在项目中包含的每个添加的图标文件。

定义菜单资源

创建一个新菜单下的子目录资源。在中菜单子目录中，创建名为新的菜单资源文件 `top_menus.xml` 并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_edit"
        android:icon="@mipmap/ic_action_content_create"
        android:showAsAction="ifRoom"
        android:title="Edit" />
    <item
        android:id="@+id/menu_save"
        android:icon="@mipmap/ic_action_content_save"
        android:showAsAction="ifRoom"
        android:title="Save" />
    <item
        android:id="@+id/menu_preferences"
        android:showAsAction="never"
        android:title="Preferences" />
</menu>
```

此 XML 将创建三个菜单项：

- 编辑使用的菜单项 `ic_action_content_create.png` 图标（铅笔）。
- 一个保存使用的菜单项 `ic_action_content_save.png` 图标（磁盘）。
- 一个首选项没有图标的菜单项。

`showAsAction` 的特性编辑并保存菜单项设置为 `ifRoom` — 此设置会导致这些菜单项显示在 `Toolbar` 是否存在足够的空间，才能显示。首选项菜单项集 `showAsAction` 到 `never` — 这将导致首选项菜单中显示溢出菜单（三个垂直点）。

实现 `OnCreateOptionsMenu`

添加以下方法 `MainActivity.cs`：

```
public override bool OnCreateOptionsMenu(IMenu menu)
{
    MenuInflater.Inflate(Resource.Menu.top_menus, menu);
    return base.OnCreateOptionsMenu(menu);
}
```

Android 调用 `OnCreateOptionsMenu` 方法，以便应用程序可以指定活动的菜单资源。在此方法中，`top_menus.xml` 资源被放大到传递 `menu`。此代码会导致新编辑，保存，并首选项菜单项显示在 `Toolbar`。

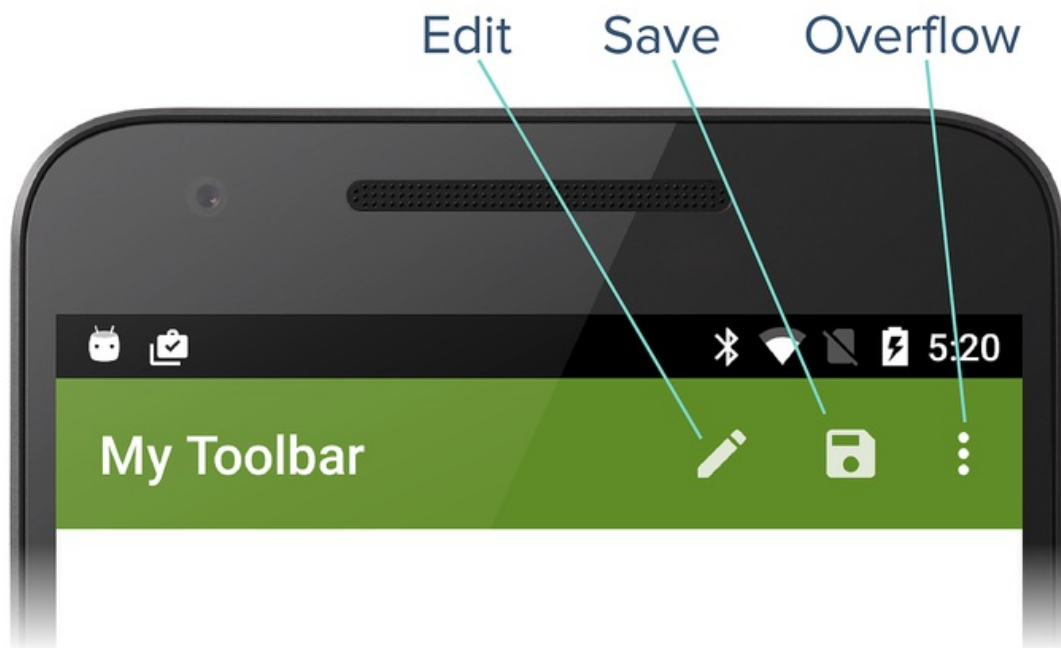
实现 `OnOptionsItemSelected`

添加以下方法 `MainActivity.cs`：

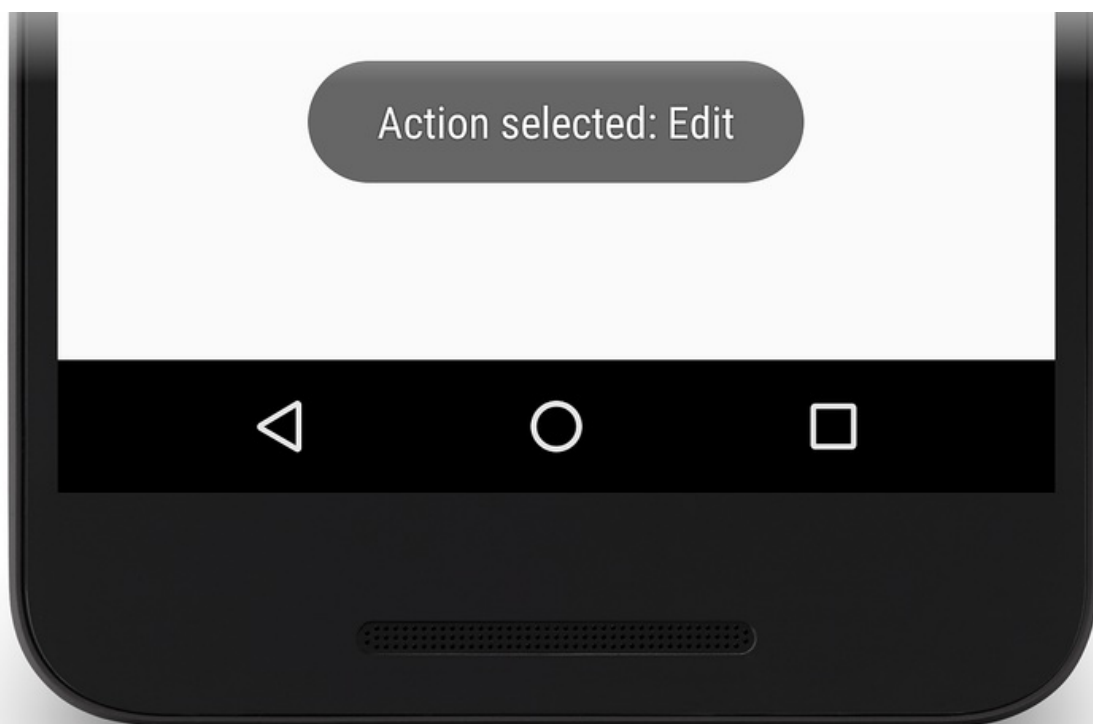
```
public override bool OnOptionsItemSelected(IMenuItem item)
{
    Toast.MakeText(this, "Action selected: " + item.TitleFormatted,
        ToastLength.Short).Show();
    return base.OnOptionsItemSelected(item);
}
```

当用户点击的菜单项时，Android 会调用 `OnOptionsItemSelected` 方法，并传递所选菜单项中的。在此示例中，实现只是显示 toast 通知来指示点击的菜单项的。

生成并运行 `ToolbarFun` 以查看在工具栏中的新菜单项。 `Toolbar` 现在显示三个菜单图标，如以下屏幕截图中所示：



当用户点击编辑显示菜单项, toast 通知, 指示 `onOptionsItemSelected` 方法被调用:



当用户点击溢出菜单中, 首选项显示菜单项。通常情况下, 不太常见的操作应放置在溢出菜单-此示例使用的溢出菜单首选项因为不经常使用它作为编辑和保存:



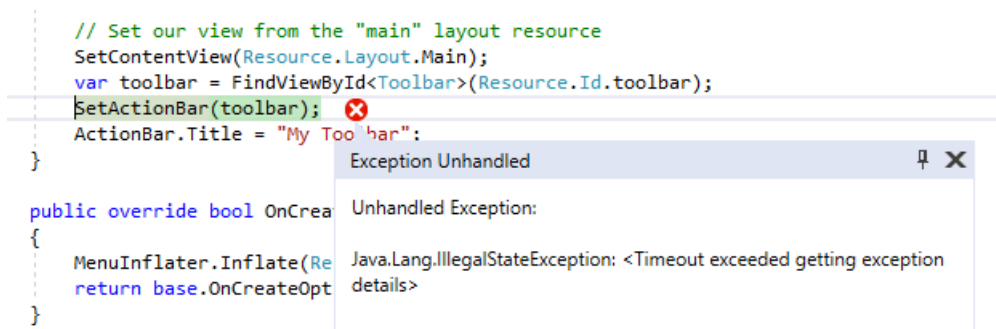
有关 Android 菜单的详细信息，请参阅 Android 开发人员[菜单](#)主题。

疑难解答

以下提示可以帮助调试在操作栏替换工具栏时可能发生的问题。

活动已具有操作栏

如果应用未正确配置为使用自定义主题中所述[应用自定义主题](#)，出现以下异常时运行该应用程序可能会发生：



此外，错误消息，因为这样以下可生成：*Java.Lang.IllegalStateException：此活动已由窗口装潢提供操作栏。*

若要更正此错误，请确认 `android:theme` 属性 (attribute) 的自定义主题添加到 `<application>` (在 `properties/AndroidManifest.xml`) 中所述[应用自定义主题](#)。如果此外，可能会导致此错误 `Toolbar` 布局或自定义主题未正确配置。

相关链接

- [棒棒糖形工具栏（示例）](#)
- [AppBarLayout 工具栏（示例）](#)

添加第二个工具栏

2018/11/13 • [Edit Online](#)

概述

`Toolbar` 可以执行多个替换操作栏—它可以在活动内使用多个时间可以放置任意位置在屏幕上，自定义，可以将它配置为跨越仅屏幕的部分的宽度。下面的示例演示了如何创建第二个 `Toolbar` 并将其放在屏幕的底部。这 `Toolbar` 实现副本，剪切，并且粘贴菜单项。

定义第二个工具栏

编辑布局文件 **Main.xml** 和包含其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include
        android:id="@+id/toolbar"
        layout="@layout/toolbar" />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/main_content"
        android:layout_below="@id/toolbar">
        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_weight="1" />
        <Toolbar
            android:id="@+id/edit_toolbar"
            android:minHeight="?android:attr/actionBarSize"
            android:background="?android:attr/colorAccent"
            android:theme="@android:style/ThemeOverlay.Material.Dark.ActionBar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </LinearLayout>
</RelativeLayout>
```

此 XML 将添加第二个 `Toolbar` 用空屏幕的底部到 `ImageView` 填充在屏幕中间。此高度 `Toolbar` 设置为操作栏的高度：

```
android:minHeight="?android:attr/actionBarSize"
```

背景色 `Toolbar` 设置为将在接下来定义以强调文字颜色：

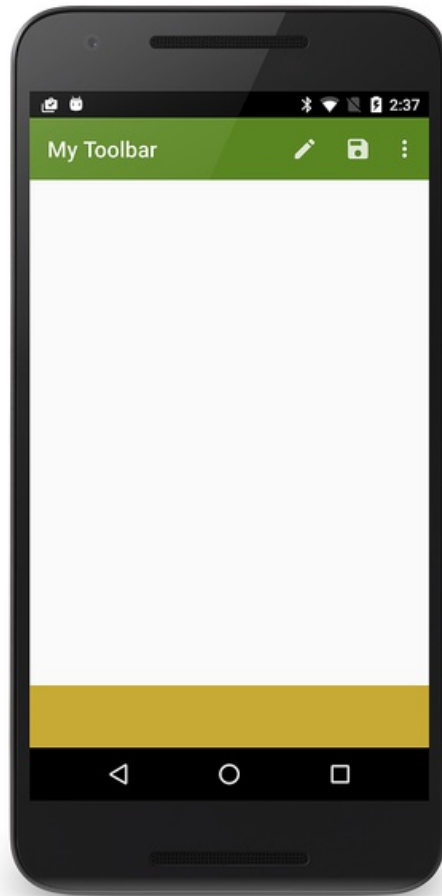
```
android:background="?android:attr/colorAccent"
```

请注意，此 `Toolbar` 基于不同的主题 (**ThemeOverlay.Material.Dark.ActionBar**) 比使用 `Toolbar` 中创建替换操作栏—它不绑定到活动的窗口装潢或使用在第一个主题 `Toolbar`。

编辑 **Resources/values/styles.xml** 并将以下的强调文字颜色添加到样式定义：


```
<item name="android:colorAccent">#C7A935</item>
```

这样，在底部工具栏深琥珀色颜色。生成和运行应用程序在屏幕的底部显示一个空白的第二个工具栏：



添加编辑菜单项

本部分介绍如何将编辑菜单项添加到底部 `Toolbar`。

若要将菜单项添加到辅助 `Toolbar`：

1. 添加到的菜单图标 `mipmap-` 文件夹的应用程序项目（如果需要）。
2. 通过添加到一个额外的菜单资源文件中定义的菜单项的内容资源/菜单。
3. 在活动的 `onCreate` 方法中，找到 `Toolbar`（通过调用 `findViewById`）和放大量 `Toolbar` 的菜单。
4. 实现单击处理程序中的 `onCreate` 的新菜单项。

以下各节演示此过程的详细信息：剪切，副本，并粘贴菜单项添加到底部 `Toolbar`。

定义编辑菜单资源

在中资源/菜单子目录中，创建名为的新 XML 文件 `edit_menus.xml` 并将内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_cut"
        android:icon="@mipmap/ic_menu_cut_holo_dark"
        android:showAsAction="ifRoom"
        android:title="Cut" />
    <item
        android:id="@+id/menu_copy"
        android:icon="@mipmap/ic_menu_copy_holo_dark"
        android:showAsAction="ifRoom"
        android:title="Copy" />
    <item
        android:id="@+id/menu_paste"
        android:icon="@mipmap/ic_menu_paste_holo_dark"
        android:showAsAction="ifRoom"
        android:title="Paste" />
</menu>
```

此 XML 创建剪切，副本，并粘贴菜单项 (使用已添加到的图标 `mipmap-` 中的文件夹 [替换操作栏](#)).

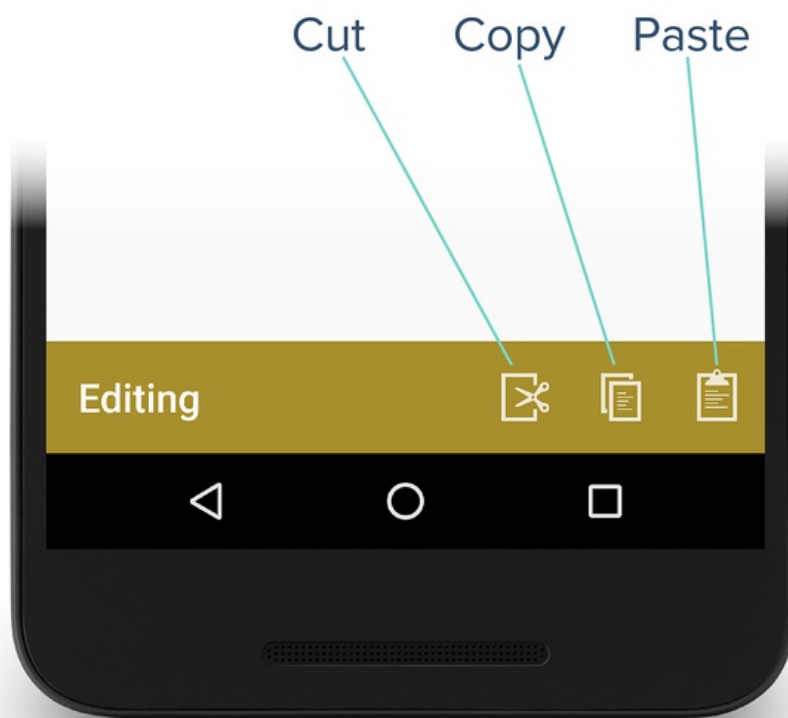
放大量菜单

在末尾 `OnCreate` 中的方法 **MainActivity.cs**, 添加以下代码行:

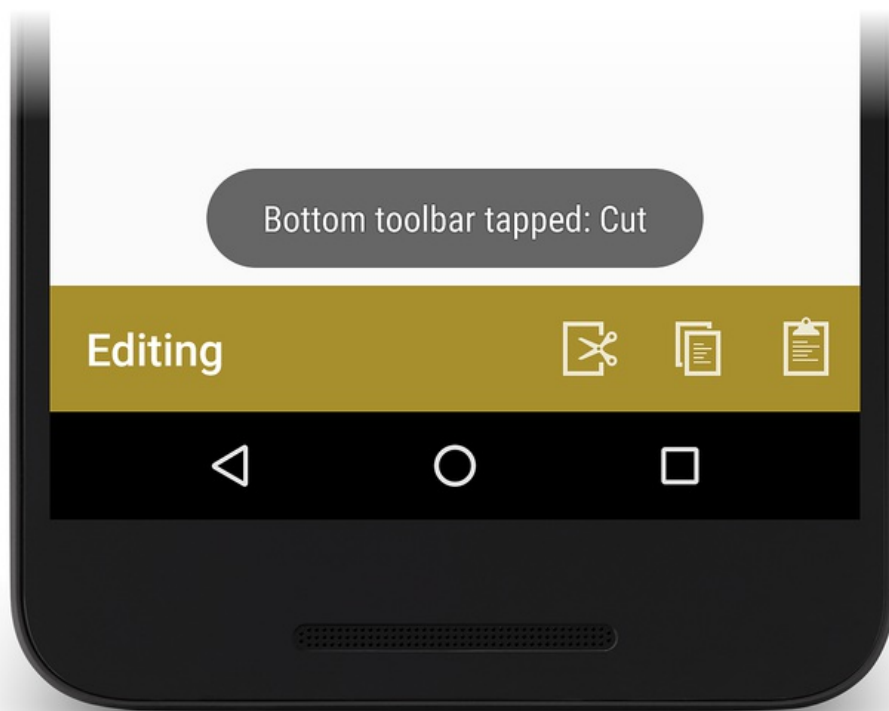
```
var editToolbar = FindViewById<Toolbar>(Resource.Id.edit_toolbar);
editToolbar.Title = "Editing";
editToolbar.InflateMenu (Resource.Menu.edit_menus);
editToolbar.MenuItemClick += (sender, e) => {
    Toast.MakeText(this, "Bottom toolbar tapped: " + e.Item.TitleFormatted, ToastLength.Short).Show();
};
```

此代码定位 `edit_toolbar` 视图中定义 **Main.axml**, 将其标题设置为编辑, 并增大其菜单项 (在中定义 **edit_menus.xml**). 它定义了一个菜单, 单击显示 toast 通知来指示点击的编辑图标的处理程序。

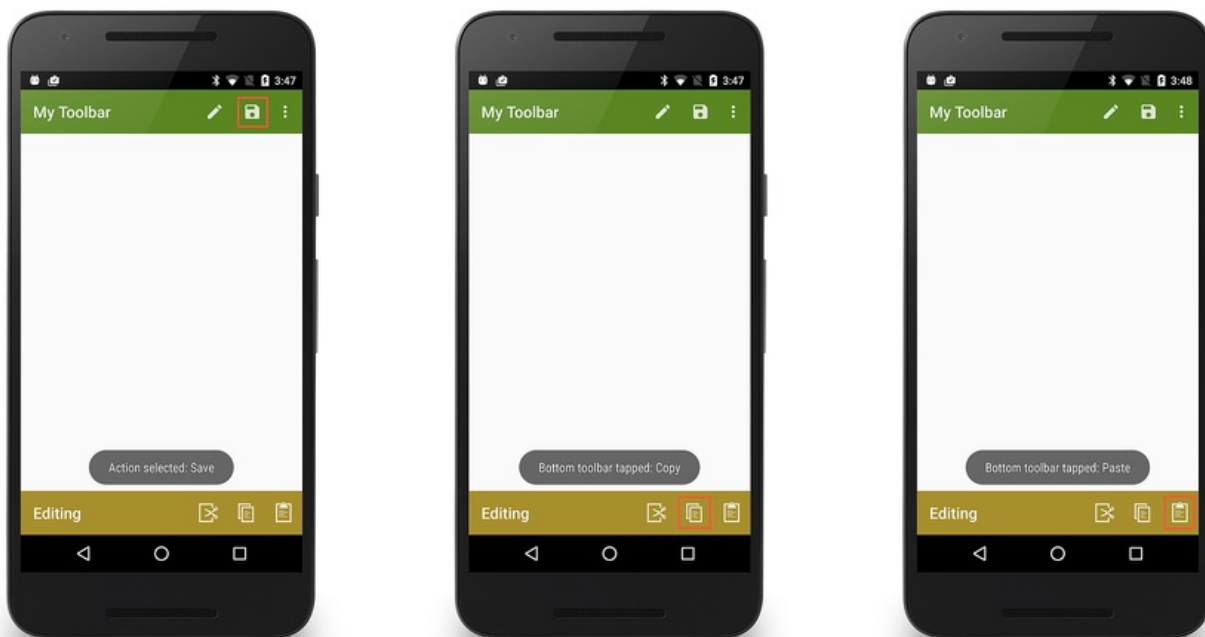
生成并运行应用。应用运行时, 则将显示的文本和前面添加的图标如下所示:



点击**剪切**菜单图标将导致显示以下 toast:



点击任一工具栏上的菜单项将显示生成 toast:



上移按钮

依赖于大多数 Android 应用回应用导航的按钮; 按回按钮以使用户可以在上一屏幕。但是, 您可能还想要提供向上便于用户在“已启动”到应用程序的主屏幕导航的按钮。当用户选择向上按钮, 用户会向上移到应用程序层次结构中较高级别-应用, 即从用户返回弹出 back 堆栈而非弹出回到以前访问的多个活动活动。

若要启用向上使用的第二个活动中的按钮 `Toolbar` 作为其操作栏中, 调用 `setDisplayHomeAsUpEnabled` 并 `setHomeButtonEnabled` 中的第二个活动的方法 `onCreate` 方法:

```

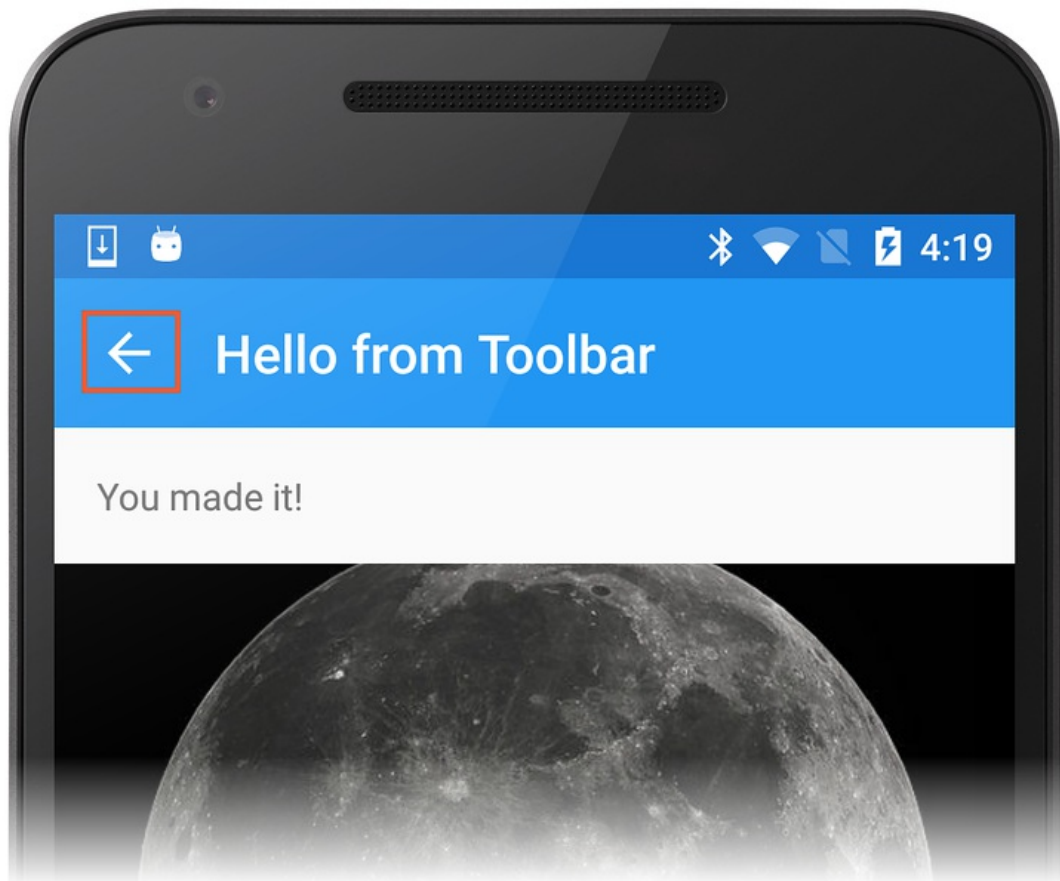
setSupportActionBar (toolbar);
...
ActionBar.setDisplayHomeAsUpEnabled (true);
ActionBar.setHomeButtonEnabled (true);

```

支持 v7 工具栏代码示例演示如何向上中操作的按钮。此示例（它使用下文所述的 AppCompatActivity 库）实现该第二个活动使用工具栏向上返回到上一个活动的分层导航的按钮。在此示例中，`DetailActivity` 主页按钮使向上通过进行以下按钮 `SupportActionBar` 方法调用：

```
SetSupportActionBar (toolbar);  
...  
SupportActionBar.SetDisplayHomeAsUpEnabled (true);  
SupportActionBar.SetHomeButtonEnabled (true);
```

当用户导航从 `MainActivity` 到 `DetailActivity`，则 `DetailActivity` 显示向上按钮（左箭头）的屏幕截图中所示：



点击这向上按钮将导致应用程序返回到 `MainActivity`。在具有多个级别的层次结构更复杂的应用，点击此按钮将返回用户的应用中的下一步最高级别，而不是上一屏幕。

相关链接

- [棒棒糖形工具栏（示例）](#)
- [AppCompatActivity 工具栏（示例）](#)

工具栏兼容性

2018/10/26 • [Edit Online](#)

概述

本部分介绍如何使用 `Toolbar` 早于 Android 5.0 Lollipop 版本的 Android 上。如果您的应用程序不支持早于 Android 5.0 的 Android 版本，则可以跳过本部分中。

因为 `Toolbar` 是一部分的 Android v7 支持库，它可以在设备上使用运行 Android 2.1 (API 级别 7) 和更高版本。但是，[Android 支持库 v7 AppCompat](#) 必须在安装 NuGet 和代码修改，使其使用 `Toolbar` 此库中提供的实现。本部分介绍如何安装此 NuGet 和修改 `ToolbarFun` 应用程序从[添加第二个工具栏](#)，以便它早于棒棒糖 5.0 版本的 Android 上运行。

若要修改应用以使用 AppCompat 版本的 Toolbar:

1. 设置应用程序的最小值和目标 Android 版本。
2. 安装 AppCompat NuGet 包。
3. 而不是内置的 Android 主题使用 AppCompat 主题。
4. 修改 `MainActivity`，以便它子类 `AppCompatActivity` 而非 `Activity`。

每个步骤是以下各节中详细介绍。

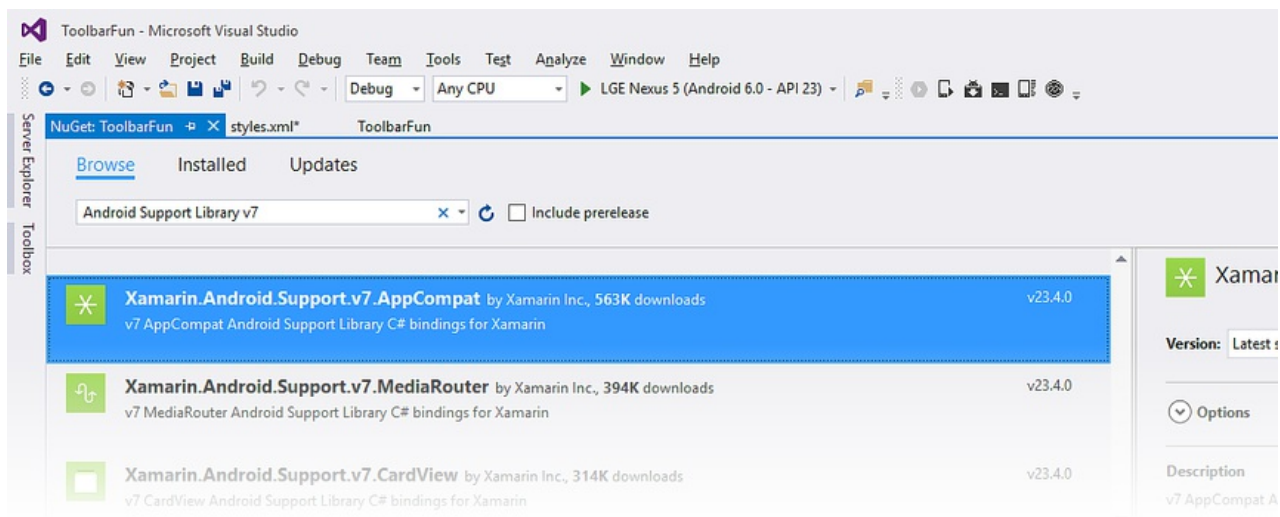
设置最小值和目标 Android 版本

为 API 级别 21 或更高版本，必须设置应用的目标框架或应用程序将不能正确部署。如果如错误没有资源标识符找到属性 `tileModeX` 包 `android` 中将被同时部署应用，这是因为目标框架未设置为 **Android 5.0 (API 级别 21-棒棒糖)** 或更高版本。

设置目标框架级别为 API 级别 21 或更高版本并将 Android API 级别的项目设置设置为应用程序是支持的最低 Android 版本。有关设置 Android API 级别的详细信息，请参阅[了解 Android API 级别](#)。在 `ToolbarFun` 示例中，最低 Android 版本设置为 KitKat (API 级别 4.4)。

安装 AppCompat NuGet 包

接下来，添加[Android 支持库 v7 AppCompat](#)到项目的包。在 Visual Studio 中，右键单击引用，然后选择**管理 NuGet 包...**单击浏览并搜索**Android 支持库 v7 AppCompat**。选择 `Xamarin.Android.Support.v7.AppCompat` 然后单击**安装**：



安装此 NuGet 时，多个其他 NuGet 包还会安装如果尚不存在 (如 **Xamarin.Android.Support.Animated.Vector.Drawable**，**Xamarin.Android.Support.v4**，并 **Xamarin.Android.Support.Vector.Drawable**)。有关安装 NuGet 包的详细信息，请参阅[演练：在项目中包括 NuGet](#)。

使用 AppCompat 主题和工具栏

AppCompat 库提供了几个 `Theme.AppCompat` Android AppCompat 库支持的任何版本可用的主题。`ToolbarFun` 示例应用主题派生自 `Theme.Material.Light.DarkActionBar`，其上不可用之前的 Android 版本比棒棒糖形。因此，`ToolbarFun` 必须进行适配化以便 AppCompat 对应用于此主题 `Theme.AppCompat.Light.DarkActionBar`。此外，由于 `Toolbar` 是不可用的 Android 版本早于棒棒糖形，我们必须使用的 AppCompat 版本 `Toolbar`。因此，必须使用布局 `android.support.v7.widget.Toolbar` 而不是 `Toolbar`。

更新布局

编辑 **Resources/layout/Main.xml**，并将为 `Toolbar` 具有下面的 XML 元素：

```
<android.support.v7.widget.Toolbar
    android:id="@+id/edit_toolbar"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorAccent"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

编辑 **Resources/layout/toolbar.xml** 并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

请注意，`?attr` 值不能再带有前缀 `android:` (请记住，`?` 表示法引用当前主题中的资源)。如果 `?android:attr` 仍使用了此处，Android 会引用属性值，从当前正在运行的平台，而不是从 AppCompat 库。因为此示例使用 `actionBarSize` AppCompat 库定义的 `android:` 删除前缀。同样，`@android:style` 更改为 `@style` 以便 `android:theme` AppCompat 库中将属性设置为主题 `ThemeOverlay.AppCompat.Dark.ActionBar` 此处使用主题而非 `ThemeOverlay.Material.Dark.ActionBar`。

更新样式

编辑 **Resources/values/styles.xml** 并将其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
    <style name="MyTheme" parent="MyTheme.Base"> </style>
    <style name="MyTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
        <item name="colorPrimary">#5A8622</item>
        <item name="colorAccent">#A88F2D</item>
    </style>
</resources>
```

项名称和在此示例中的父主题不再带有前缀 `android:`，因为我们使用 AppCompat 库。此外，父主题更改为 AppCompat 新版 `Light.DarkActionBar`。

更新菜单

为了支持早期版本的 Android，AppCompat 库使用镜像的属性的自定义特性 `android:` 命名空间。但是，某些属性（如 `showAsAction` 属性中使用 `<menu>` 标记）在较旧的设备上的 Android framework 中不存在— `showAsAction` Android API 11 中引入了，但在 Android API 7 中不可用。出于此原因，必须使用自定义的命名空间前缀的所有支持库定义的属性。在菜单资源文件，命名空间中调用 `local` 为前缀定义 `showAsAction` 属性。

编辑 **Resources/menu/top_menus.xml** 并将其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:local="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu_edit"
        android:icon="@mipmap/ic_action_content_create"
        local:showAsAction="ifRoom"
        android:title="Edit" />
    <item
        android:id="@+id/menu_save"
        android:icon="@mipmap/ic_action_content_save"
        local:showAsAction="ifRoom"
        android:title="Save" />
    <item
        android:id="@+id/menu_preferences"
        local:showAsAction="never"
        android:title="Preferences" />
</menu>
```

`local` 命名空间添加用下面这行:

```
xmlns:local="http://schemas.android.com/apk/res-auto">
```

`showAsAction` 属性开头，但这 `local:` 命名空间而非 `android:`

```
local:showAsAction="ifRoom"
```

同样，编辑 **Resources/menu/edit_menus.xml** 并将其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:local="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu_cut"
        android:icon="@mipmap/ic_menu_cut_holo_dark"
        local:showAsAction="ifRoom"
        android:title="Cut" />
    <item
        android:id="@+id/menu_copy"
        android:icon="@mipmap/ic_menu_copy_holo_dark"
        local:showAsAction="ifRoom"
        android:title="Copy" />
    <item
        android:id="@+id/menu_paste"
        android:icon="@mipmap/ic_menu_paste_holo_dark"
        local:showAsAction="ifRoom"
        android:title="Paste" />
</menu>
```

此命名空间开关如何提供对支持 `showAsAction` API 级别 11 之前的 Android 版本上的属性？自定义特性 `showAsAction`，并且所有可能的值包含在应用程序时 AppCompatActivity NuGet 是否已安装。

子类 AppCompatActivity

在转换过程中的最后一步是修改 `MainActivity`，以便它是一个的子类 `AppCompatActivity`。编辑 `MainActivity.cs` 并添加以下 `using` 语句：

```
using Android.Support.V7.App;
using Toolbar = Android.Support.V7.Widget.Toolbar;
```

这会将声明 `Toolbar` AppCompatActivity 版本的 `Toolbar`。接下来，更改的类定义 `MainActivity`：

```
public class MainActivity : AppCompatActivity
```

若要设置到 AppCompatActivity 版本的操作栏 `Toolbar`，替换为调用 `SetSupportActionBar` 与 `SetSupportActionBar`。在此示例中，标题也会更改以指示 AppCompatActivity 新版 `Toolbar` 正在使用：

```
SetSupportActionBar (toolbar);
SupportActionBar.Title = "My AppCompatActivity Toolbar";
```

最后，将最小 Android 级别更改为支持（例如 API 19）的预棒棒糖形值。

生成应用并在预棒棒糖形设备或 Android 仿真器上运行。以下屏幕截图显示了 AppCompatActivity 新版 `Toolbar` 上 Nexus 4 个正在运行 KitKat (API 19)：



当使用 AppCompat 库时，主题而不必切换根据 Android 版本-AppCompat 库可以跨所有受支持的 Android 版本提供一致的用户体验。

相关链接

- [棒棒糖形工具栏（示例）](#)
- [AppCompat 工具栏（示例）](#)

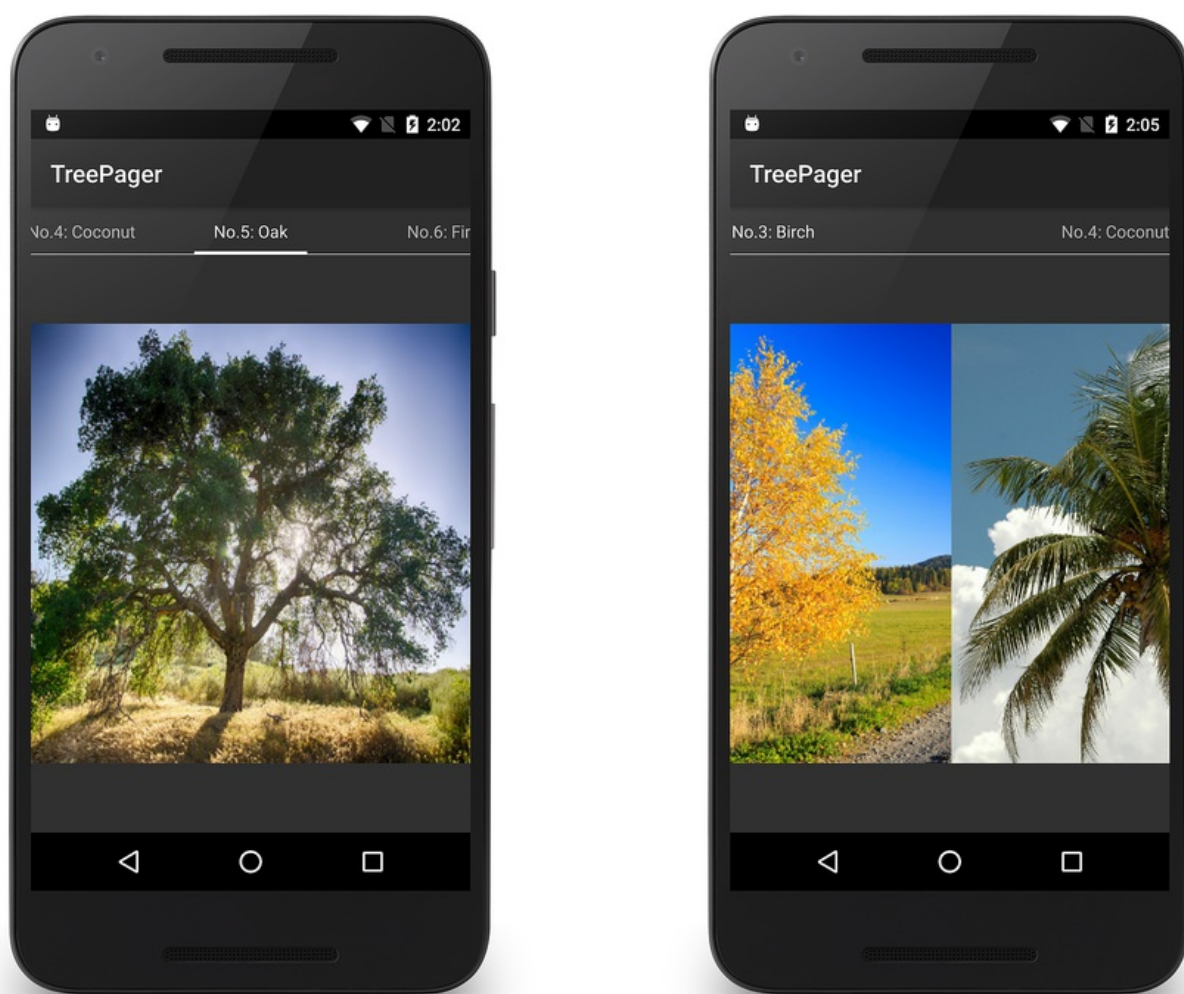
ViewPager

2018/10/26 • • [Edit Online](#)

ViewPager 是使您可以实现动作导航的布局管理器。动作导航允许用户轻扫，左侧和右侧到单步执行的数据页。本指南介绍如何实现动作导航使用 ViewPager，使用和不使用片段。它还介绍了如何添加使用 PagerTitleStrip 和 PagerTabStrip 页指示符。

概述

应用程序开发中的常见方案是需要向用户提供动作同级视图之间导航。在此方法中，在用户轻扫左或向右访问页的内容（例如，在安装向导或幻灯片放映）。可以使用来创建这些轻扫视图 `ViewPager` 小组件中可用 [Android 支持库 v4](#)。`ViewPager` 是布局小组件的多个子视图组成，其中每个子视图构成布局中的页：



通常情况下，`ViewPager` 结合使用 [片段](#)；但是，某些情况下，可能想要使用 `ViewPager` 而无需增加的复杂性 `Fragment` S。

`ViewPager` 使用适配器模式为其提供要显示的视图。此处使用的适配器不从概念上讲类似于由 `RecyclerView` 提供的实现 `PagerAdapter` 生成页面的 `ViewPager` 向用户显示。通过显示的页 `ViewPager` 可以是 `View` S 或 `Fragment` S。当 `View` 显示，适配器子类 Android 的 `PagerAdapter` 基类。如果 `Fragment` 显示，适配器子类 Android 的 `FragmentPagerAdapter`。Android 支持库还包括 `FragmentPagerAdapter` (的子类 `PagerAdapter`) 来帮助进行连接的详细信息 `Fragment` 的数据。

本指南演示了这两种方法：

- 在视图的 [ViewPager](#) 即 [TreePager](#) 开发出应用程序来演示如何使用 `ViewPager` 显示树目录（落叶和长期有效树的图像库）的视图。`PagerTabStrip` 和 `PagerTitleStrip` 用于显示帮助页面导航的标题。
- 在中片段的 [ViewPager](#)，稍微复杂 [FlashCardPager](#) 开发出应用程序来演示如何使用 `ViewPager` 与 `Fragment s` 构建的应用程序带来了作为数学问题闪存卡，并响应用户输入。

要求

若要使用 `ViewPager` 在应用程序项目中，必须安装 [Android 支持库 v4](#) 包。有关安装 NuGet 包的详细信息，请参阅 [演练：在项目中包括 NuGet](#)。

体系结构

用于三个组件实现使用动作导航 `ViewPager`：

- `ViewPager`
- 适配器
- 页导航指示器

下面概述了每个组件。

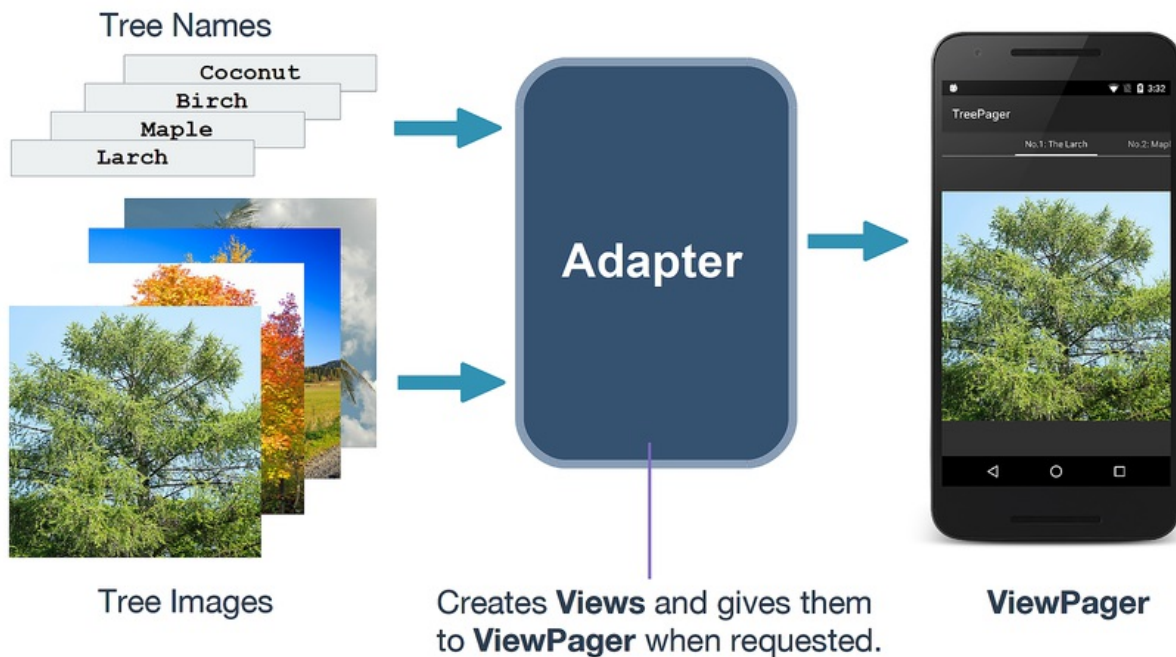
ViewPager

`ViewPager` 是显示的集合的布局管理器 `View s` 一次。其工作是检测用户的轻扫手势，并导航到相应的下一步或上一个视图。例如，下面的屏幕截图演示了 `ViewPager` 以响应用户手势进行到下一个图像中的转换：



适配器

`ViewPager` 从其数据中提取 *适配器*。适配器的作业将创建 `View` 情况下显示的 `s ViewPager`，根据需要提供它们。下图阐释了这一概念-适配器创建并填充 `View` s，并提供到 `ViewPager`。作为 `ViewPager` 检测到用户的轻扫手势，它会要求提供相应的适配器 `View` 显示：

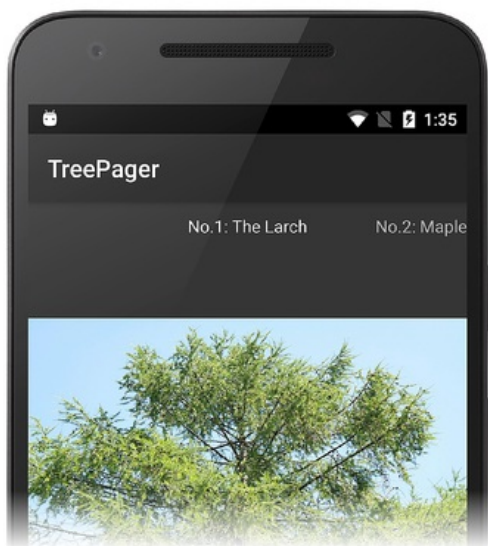


在此特定示例中，每个 `View` 之前将它传递到构造从树图像和树名称 `ViewPager`。

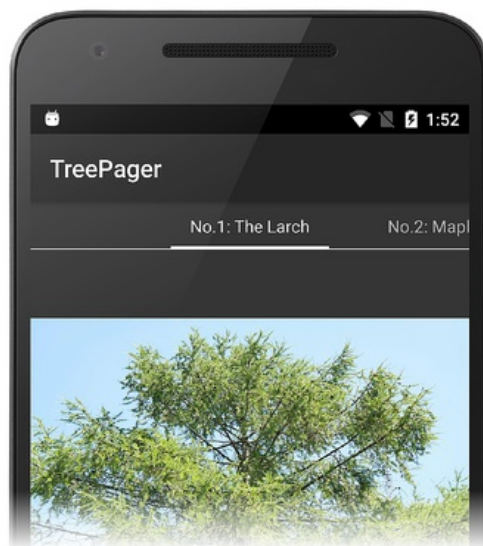
页导航指示器

`ViewPager` 用于显示大型数据集（例如，图像库可能包含数百个图像）。若要帮助用户导航大型数据集，`ViewPager` 通常伴随 `寻呼指示器` 显示的字符串。字符串可能是图像标题、标题或只是当前视图的数据集内的位置。

有两个视图可能会产生此导航信息：`PagerTabStrip` 和 `PagerTitleStrip`。每个顶部显示的字符串 `ViewPager`，和每个拉取从其数据 `ViewPager` 的适配器，使其始终保持与同步当前显示 `View`。它们之间的区别在于 `PagerTabStrip` 包括“当前”字符串时的可视指示器 `PagerTitleStrip` 不（根据这些屏幕截图所示）执行：



PagerTitleStrip



PagerTabStrip

本指南演示如何 implement `ViewPager`，适配器和指示器应用程序组件并将它们以支持动作导航集成。

相关链接

- [TreePager（示例）](#)
- [FlashCardPager（示例）](#)

视图的 ViewPager

2018/10/26 • [Edit Online](#)

ViewPager 是使您可以实现动作导航的布局管理器。动作导航允许用户轻扫，左侧和右侧到单步执行的数据页。本指南介绍如何实现使用 ViewPager 和 PagerTabStrip，为数据页使用视图可轻扫 UI（后续指南介绍如何使用页面片段）。

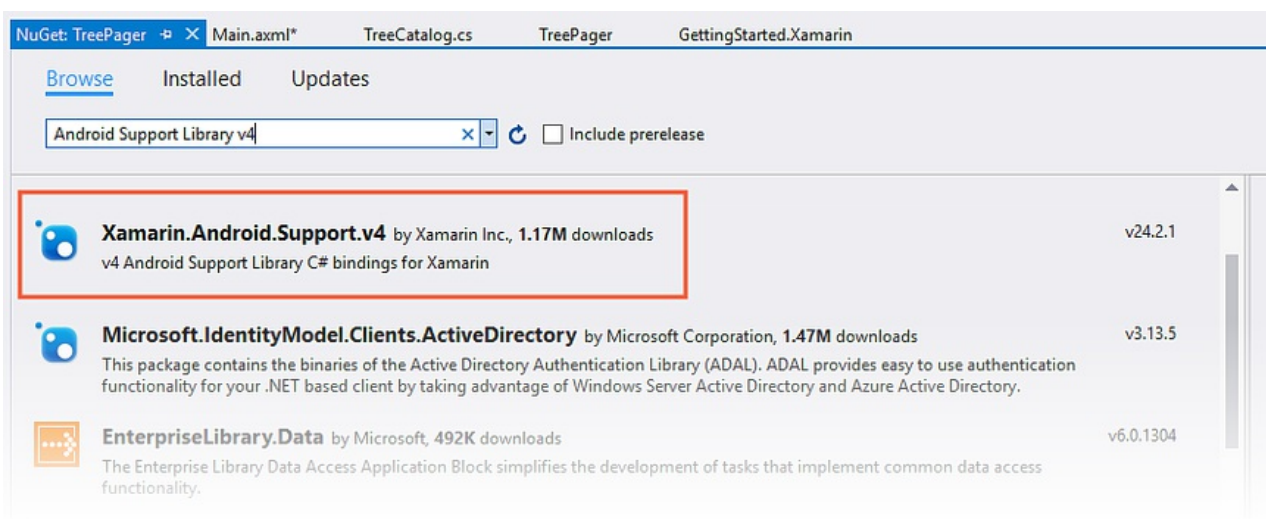
概述

本指南是一个演练，提供了分步演示如何使用 `ViewPager` 实现落叶和长期有效树的图像库。在用户轻左侧和右侧的“树目录”通过扫此应用中查看树映像。目录的每个页面顶部的树的名称被列入 `PagerTabStrip`，并在树的映像显示在 `ImageView`。使用适配器接口 `ViewPager` 到基础数据模型。此应用程序实现派生自的适配器 `PagerAdapter`。

尽管 `ViewPager` 基于应用程序通常实现与 `Fragment` s，有一些相对简单的用例，额外的复杂性 `Fragment` s 不是必要。例如，在本演练中所示的基本映像库应用不需要使用 `Fragment` s。因为内容是静态的且不同的映像之间来回用用户唯一扫，实现可以保持为更简单，使用标准 Android 视图和布局。

启动应用程序项目

创建一个名为的新的 Android 项目 **TreePager** (请参阅[Hello, Android](#)有关创建新的 Android 项目的详细信息)。接下来，启动 NuGet 包管理器。(有关安装 NuGet 包的详细信息，请参阅[演练：在项目中包括 NuGet](#))。查找和安装 **Android 支持库 v4**：



这还将安装的任何其他包 required **Android 支持库 v4**。

添加示例数据源

在此示例中，树目录数据源 (由 `TreeCatalog` 类) 提供 `ViewPager` 项内容。 `TreeCatalog` 包含一系列现成树映像和树标题，则适配器将使用用于创建 `View` s。 `TreeCatalog` 构造函数不需要任何参数：

```
TreeCatalog treeCatalog = new TreeCatalog();
```

中的映像集合 `TreeCatalog` 组织，因此可通过索引器访问的每个图像。例如，以下代码行检索集合中的第三个图像的图像资源 ID：

```
int imageId = treeCatalog[2].imageId;
```

因为的实现细节 `TreeCatalog` 了解与不相关 `ViewPager`，则 `TreeCatalog` 此处未列出的代码。向源代码 `TreeCatalog` 网址 [TreeCatalog.cs](https://github.com/TreeCatalog/TreeCatalog)。下载此源文件 (或复制并粘贴到一个新的代码 `TreeCatalog.cs` 文件) 并将其添加到你的项目。此外，下载并解压缩 [映像文件](#) 到你资源 `/drawable` 文件夹并将其包含在项目中。

创建 ViewPager 布局

打开 `Resources/layout/Main.axml` 并将其内容替换为以下 XML:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</android.support.v4.view.ViewPager>

```csharp
This XML defines a `ViewPager` that occupies the entire screen. Note that
you must use the fully-qualified name **android.support.v4.view.ViewPager**
because `ViewPager` is packaged in a support library. `ViewPager` is
available only from
[Android Support Library v4](https://www.nuget.org/packages/Xamarin.Android.Support.v4/);
it is not available in the Android SDK.

Set up ViewPager

Edit **MainActivity.cs** and add the following `using` statement:

```csharp
using Android.Support.V4.View;
```

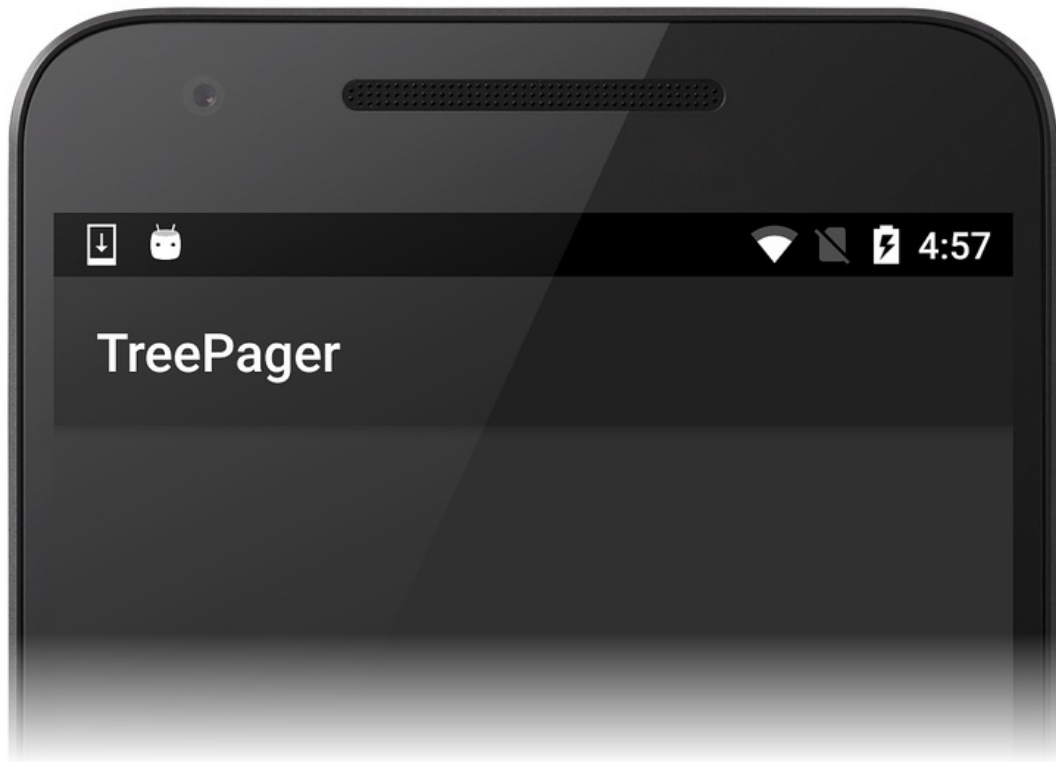
将 `OnCreate` 方法替换为以下代码:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    ViewPager viewPager = FindViewById<ViewPager>(Resource.Id.viewpager);
    TreeCatalog treeCatalog = new TreeCatalog();
}
```

此代码将执行以下操作:

1. 设置从视图 `Main.axml` 布局资源。
2. 检索到的引用 `ViewPager` 从布局。
3. 实例化新 `TreeCatalog` 作为数据源。

当生成并运行此代码时，您应看到类似于以下屏幕截图显示:



在此情况下，`ViewPager` 为空，因为它缺少适配器用于访问中的内容 **TreeCatalog**。在下一步部分中，**PagerAdapter** 创建连接 `ViewPager` 到 **TreeCatalog**。

创建适配器

`ViewPager` 使用适配器控制器对象位于之间 `ViewPager` 和数据源 (请参阅中的图示 [适配器](#))。若要访问此数据，`ViewPager` 要求你提供自定义适配器派生自 `PagerAdapter`。此适配器将填充每个 `ViewPager` 页中的数据源的内容。由于此数据源是特定于应用程序，自定义适配器将是一个知道如何访问数据的代码。作为通过页的用户扫描 `ViewPager`，该适配器从数据源中提取信息，并将其加载到页面 `ViewPager` 显示。

当您实现 `PagerAdapter`，必须重写以下：

- **InstantiateItem** – 创建页 (`View`) 的给定位置并将其添加到 `ViewPager` 的视图的集合。
- **DestroyItem** – 从给定位置移除一个页面。
- **计数** – 只读属性，该属性返回的视图 (页面) 可用。
- **IsViewFromObject** – 确定页是否与特定的密钥对象相关联。(此对象创建的 `InstantiateItem` 方法。)在此示例中，关键对象是 `TreeCatalog` 数据对象。

添加名为的新文件 **TreePagerAdapter.cs** 和其内容替换为以下代码：


```

using System;
using Android.App;
using Android.Runtime;
using Android.Content;
using Android.Views;
using Android.Widget;
using Android.Support.V4.View;
using Java.Lang;

namespace TreePager
{
    class TreePagerAdapter : PagerAdapter
    {
        public override int Count
        {
            get { throw new NotImplementedException(); }
        }

        public override bool IsViewFromObject(View view, Java.Lang.Object obj)
        {
            throw new NotImplementedException();
        }

        public override Java.Lang.Object InstantiateItem (View container, int position)
        {
            throw new NotImplementedException();
        }

        public override void DestroyItem(View container, int position, Java.Lang.Object view)
        {
            throw new NotImplementedException();
        }
    }
}

```

此代码存根不可或缺 `PagerAdapter` 实现。在以下部分中，每种方法将被替换的工作代码。

实现构造函数

当应用程序实例化 `TreePagerAdapter`，它会提供一个上下文（`MainActivity`）和实例化 `TreeCatalog`。将以下成员变量和构造函数添加到顶部 `TreePagerAdapter` 类中**`TreePagerAdapter.cs`**：

```

Context context;
TreeCatalog treeCatalog;

public TreePagerAdapter (Context context, TreeCatalog treeCatalog)
{
    this.context = context;
    this.treeCatalog = treeCatalog;
}

```

此构造函数的目的是为存储上下文和 `TreeCatalog` 实例的 `TreePagerAdapter` 将使用。

实现计数

`Count` 实现是相对简单：它返回树目录中的树数。将 `Count` 替换为以下代码：

```

public override int Count
{
    get { return treeCatalog.NumTrees; }
}

```

`NumTrees` 属性的 `TreeCatalog` 数据集中返回树（页数）数。

实现 `InstantiateItem`

`InstantiateItem` 方法创建的给定位置的页。它还必须添加到新创建的视图 `ViewPager` 的查看集合。为了实现这一点, `ViewPager` 将自身作为容器参数中传递。

将 `InstantiateItem` 方法替换为以下代码:

```
public override Java.Lang.Object InstantiateItem (View container, int position)
{
    var imageView = new ImageView (context);
    imageView.SetImageResource (treeCatalog[position].imageId);
    var viewPager = container.JavaCast<ViewPager>();
    viewPager.AddView (imageView);
    return imageView;
}
```

此代码将执行以下操作:

1. 实例化新 `ImageView` 在指定位置显示树图像。应用程序的 `MainActivity` 是将传递给上下文 `ImageView` 构造函数。
2. 集 `ImageView` 资源 `TreeCatalog` 映像中的指定位置的资源 ID。
3. 将强制转换传递的容器 `View` 到 `ViewPager` 引用。请注意, 必须使用 `JavaCast<ViewPager>()` 若要正确执行此转换 (这需要以便 Android 执行运行时检查类型转换)。
4. 添加了实例化 `ImageView` 到 `ViewPager`, 并返回 `ImageView` 给调用方。

当 `ViewPager` 显示在图像 `position`, 它会显示以下 `ImageView`。最初, `InstantiateItem` 调用两次以填充具有视图的前两个页面。当用户滚动时, 它再次被调用来维护视图只是后面和前面的当前显示的项。

实现 `DestroyItem`

`DestroyItem` 方法从给定位置移除一个页面。在任意给定位置处的视图可以更改其中的应用程序中 `ViewPager` 必须有办法其替换为新视图之前删除过时视图中的该位置。在 `TreeCatalog` 示例中, 每个位置处的视图不会更改, 因此视图删除通过 `DestroyItem` 只需进行重新添加时 `InstantiateItem` 为该位置调用。(一个可以为更好地提高效率, 实现池回收 `View` 将重新显示在相同的位置上的 s。)

将 `DestroyItem` 方法替换为以下代码:

```
public override void DestroyItem(View container, int position, Java.Lang.Object view)
{
    var viewPager = container.JavaCast<ViewPager>();
    viewPager.RemoveView(view as View);
}
```

此代码将执行以下操作:

1. 将强制转换传递的容器 `View` 到 `ViewPager` 引用。
2. 将转换所传递的 Java 对象 (`view`) 到 C# `View` (`view as View`);
3. 从视图中删除 `ViewPager`。

实现 `IsViewFromObject`

作为用户滑向左和右页的内容, 通过 `ViewPager` 调用 `IsViewFromObject` 若要验证的子 `View` 给定的位置是与该相同位置的适配器的对象相关联 (因此, 调用适配器的对象的对象键)。对于相对简单的应用程序, 关联是一个标识-在该实例上的适配器的对象键是之前返回到视图 `ViewPager` 通过 `InstantiateItem`。但是对于其他应用程序, 该对象键可能与关联的某些其他特定于适配器的类实例 (但不是完全相同) 子视图, `ViewPager` 在该位置显示。只有适配器知道是否不传递的视图和对象键相关联。

`IsViewFromObject` 必须为实现 `PagerAdapter` 才能正常工作。如果 `IsViewFromObject` 将返回 `false` 的给定位置 `ViewPager` 将不显示视图中的相应位置。在 `TreePager` 应用程序, 该对象返回键 `InstantiateItem` 是页面 `View` 的树, 因此该代码仅有以检查标识 (即, 该对象键和视图是同一个)。将 `IsViewFromObject` 替换为以下代码:

```
public override bool IsViewFromObject(View view, Java.Lang.Object obj)
{
    return view == obj;
}
```

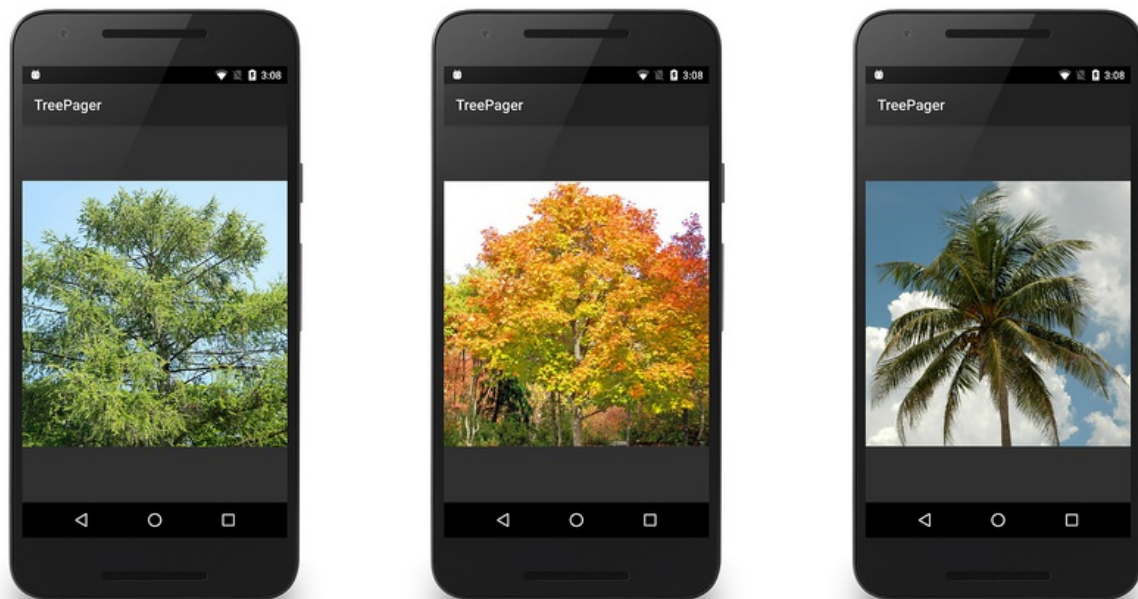
将适配器添加到 ViewPager

既然 `TreePagerAdapter` 是实现, 是时候将其添加到 `ViewPager`。在中 `MainActivity.cs`, 将以下代码行添加到末尾 `OnCreate` 方法:

```
viewPager.Adapter = new TreePagerAdapter(this, treeCatalog);
```

此代码实例化 `TreePagerAdapter`, 并传入 `MainActivity` 作为上下文 (`this`)。实例化 `TreeCatalog` 传递到构造函数的第二个参数。 `ViewPager` 的 `Adapter` 属性设置为实例化 `TreePagerAdapter` 对象; 此插入 `TreePagerAdapter` 到 `ViewPager`。

现已完成的核心实现-生成并运行应用。应会看到第一个图像的树目录显示在屏幕上, 在左侧的下一步的屏幕截图中所示。轻扫保留以查看更多的树视图, 然后右轻扫可向后移动树目录:



添加页导航指示器

此最小 `ViewPager` 实现显示树目录的映像, 但它提供了没有指明用户所在目录中。下一步是添加 `PagerTabStrip`。`PagerTabStrip` 通知哪一页显示, 并通过显示上一个和下一页面的提示来提供导航上下文作为到用户。`PagerTabStrip` 旨在用作当前页的指示符 `ViewPager`; 进行滚动和更新为每个页上用户扫。

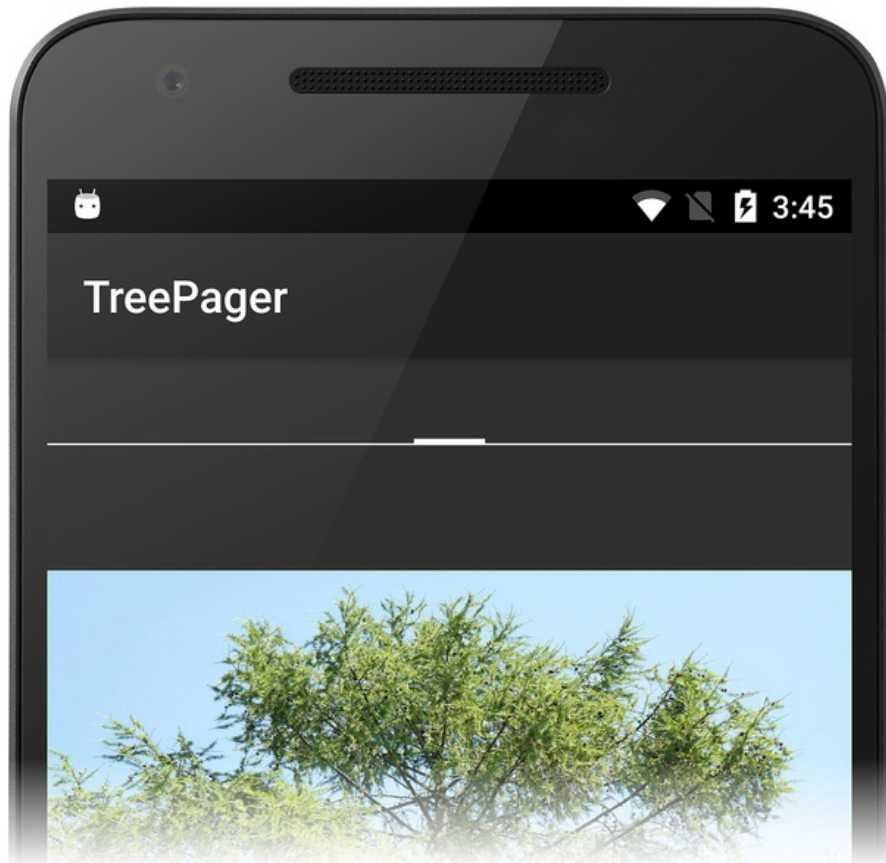
打开 `Resources/layout/Main.axml` 并添加 `PagerTabStrip` 到布局:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <android.support.v4.view.PagerTabStrip
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:paddingBottom="10dp"
        android:paddingTop="10dp"
        android:textColor="#fff" />

</android.support.v4.view.ViewPager>
```

`ViewPager` 和 `PagerTabStrip` 设计为协同工作。当你声明 `PagerTabStrip` 内 `ViewPager` 布局 `ViewPager` 将自动查找 `PagerTabStrip` 并将其连接到适配器。当生成并运行应用时，应该会看到空 `PagerTabStrip` 显示在每个屏幕的顶部：



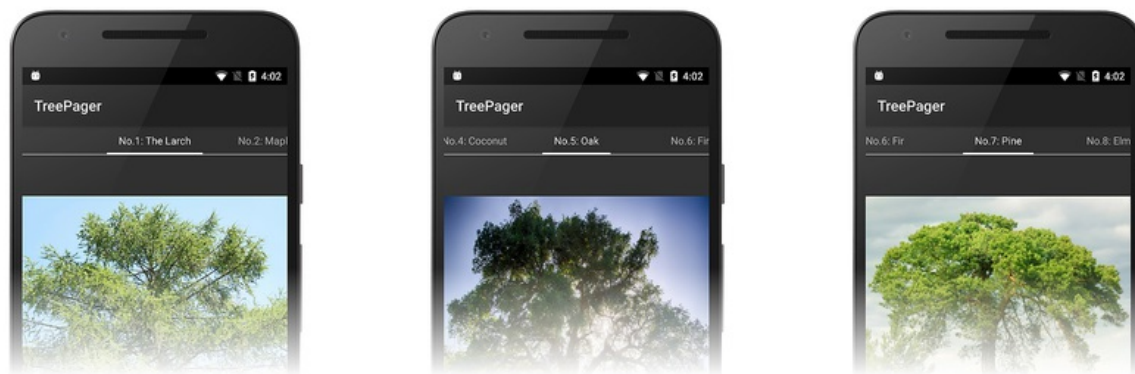
显示标题

若要添加到每个页选项卡标题，实现 `GetPageTitleFormatted` 中的方法 `PagerAdapter` 派生的类。 `ViewPager` 调用 `GetPageTitleFormatted` (如果已实现) 获取描述指定位置处的页的标题字符串。添加以下方法 `TreePagerAdapter` 类中 `TreePagerAdapter.cs`:

```
public override Java.Lang.ICharSequence GetPageTitleFormatted(int position)
{
    return new Java.Lang.String(treeCatalog[position].caption);
}
```

此代码从树目录中指定的页（位置）中检索树标题字符串，将其转换到 Java `String`，并返回到 `ViewPager`。每个页

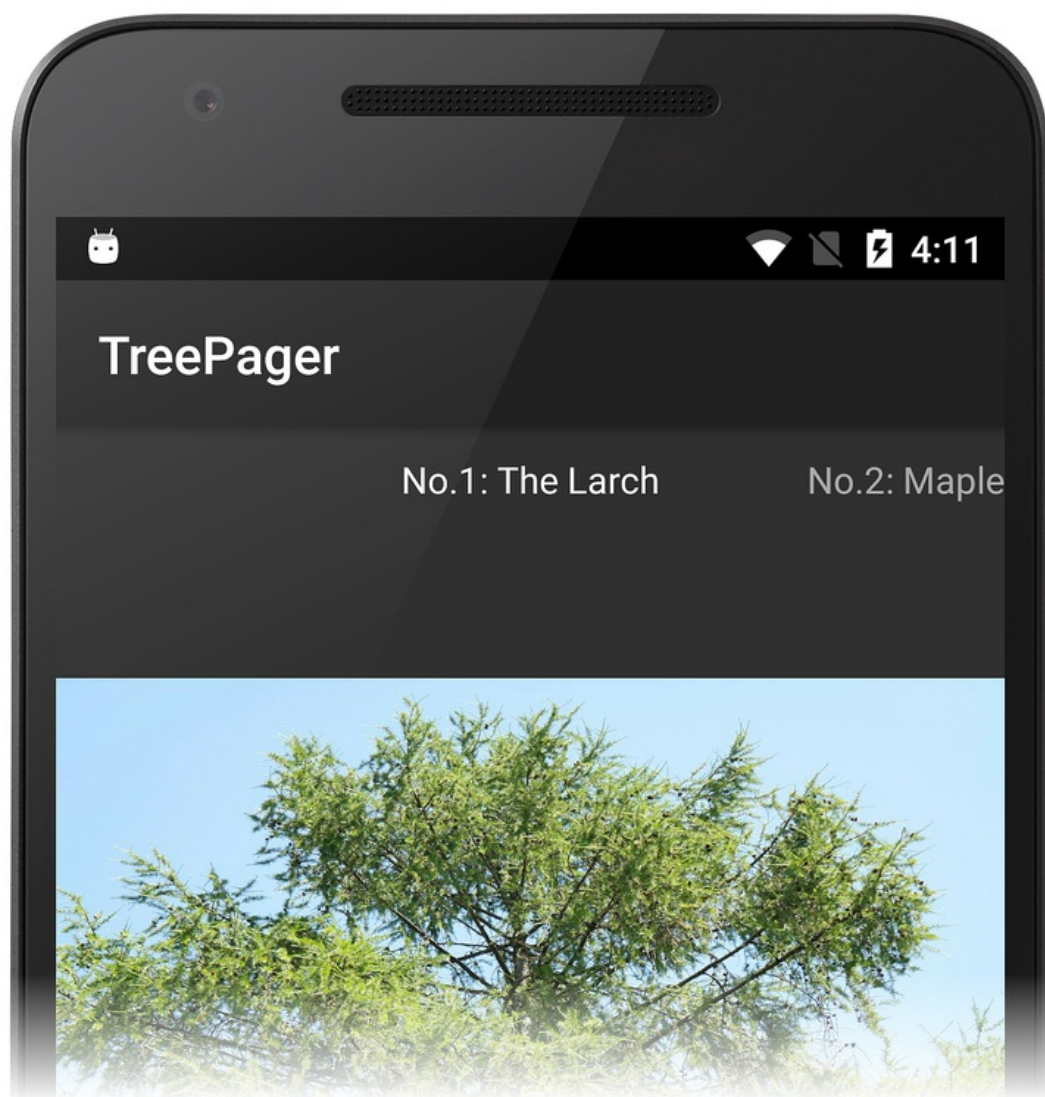
时使用此新方法运行该应用，显示的树标题 `PagerTabStrip`。应看到在不使用下划线屏幕顶部的树的名称：



轻扫可以来回若要查看目录中的每个没有标题树映像。

PagerTitleStrip 变体

`PagerTitleStrip` 非常类似于 `PagerTabStrip` 只不过 `PagerTabStrip` 添加当前所选的选项卡的下划线。您可以替换 `PagerTabStrip` 与 `PagerTitleStrip` 在上面的布局和运行应用以查看其外观与 `PagerTitleStrip`：



请注意，下划线删除时将转换为 `PagerTitleStrip`。

总结

本演练提供如何生成基本的分步示例 `ViewPager` - 基于应用程序而无需使用 `Fragment` s。显示包含图像和标题字符串的示例数据源 `ViewPager` 布局来显示图像, 和一个 `PagerAdapter` 连接的子类 `ViewPager` 到数据源。为了帮助用户可以浏览数据集, 说明已包含说明了如何添加 `PagerTabStrip` 或 `PagerTitleStrip` 若要在每个页面的顶部显示的图像标题。

相关链接

- [TreePager \(示例\)](#)

带片段的 ViewPager

2018/11/13 • [Edit Online](#)

ViewPager 是使您可以实现动作导航的布局管理器。动作导航允许用户轻扫，左侧和右侧到单步执行的数据页。本指南介绍如何实现使用 ViewPager，使用片段作为数据页的可轻扫 UI。

概述

ViewPager 通常使用与片段结合使用，因此它更轻松地管理中每个页面的生命周期 ViewPager。在此演练中，ViewPager 用于创建名为应用程序FlashCardPager的闪存卡上显示一系列数学问题。每个闪存卡实现为一个片段。用户通过闪存卡左右轻扫并点击以显示其答案对数学问题。此应用程序创建 Fragment 实例的每个适配器派生自的闪存卡并实现 FragmentPagerAdapter。在中ViewPager 和视图，执行大部分工作的 MainActivity 生命周期方法。在中FlashCardPager，将通过完成大部分工作 Fragment 在其生命周期方法之一。

本指南不涉及片段的基础知识-如果您尚不熟悉在 Xamarin.Android 中的片段，请参阅[片段](#)来帮助你开始使用片段。

启动应用程序项目

创建一个名为新的 Android 项目FlashCardPager。接下来，启动 NuGet 包管理器 (有关安装 NuGet 包的详细信息，请参阅[演练：在项目中包括 NuGet](#))。查找和安装Xamarin.Android.Support.v4包中所述ViewPager 和视图。

添加示例数据源

在FlashCardPager，数据源是由表示的 flash 一副纸牌 FlashCardDeck 类; 此数据源提供，数量 ViewPager 项内容。FlashCardDeck 包含一系列现成数学问题和答案。FlashCardDeck 构造函数不需要任何参数：

```
FlashCardDeck flashCards = new FlashCardDeck();
```

集合中的闪存卡 FlashCardDeck 组织，因此可以通过索引器访问每个闪存卡。例如，以下代码行检索一副纸牌中的第四个闪存卡问题：

```
string problem = flashCardDeck[3].Problem;
```

这行代码检索上一个问题的相应答案：

```
string answer = flashCardDeck[3].Answer;
```

因为的实现细节 FlashCardDeck 了解与不相关 ViewPager，则 FlashCardDeck 此处未列出的代码。向源代码 FlashCardDeck 网址[FlashCardDeck.cs](#)。下载此源文件 (或复制并粘贴到一个新的代码FlashCardDeck.cs文件) 并将其添加到你的项目。

创建 ViewPager 布局

打开Resources/layout/Main.axml并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</android.support.v4.view.ViewPager>
```

此 XML 定义 `ViewPager` 占据整个屏幕。请注意，必须使用完全限定名称 `android.support.v4.view.ViewPager` 因为 `ViewPager` 打包在支持库中。`ViewPager` 只有通过 [Android 支持库 v4](#)；它不是在 Android SDK 中提供。

设置 ViewPager

编辑 `MainActivity.cs` 并添加以下 `using` 语句：

```
using Android.Support.V4.View;
using Android.Support.V4.App;
```

更改 `MainActivity`，以便从派生类声明 `AppCompatActivity`：

```
public class MainActivity : AppCompatActivity
```

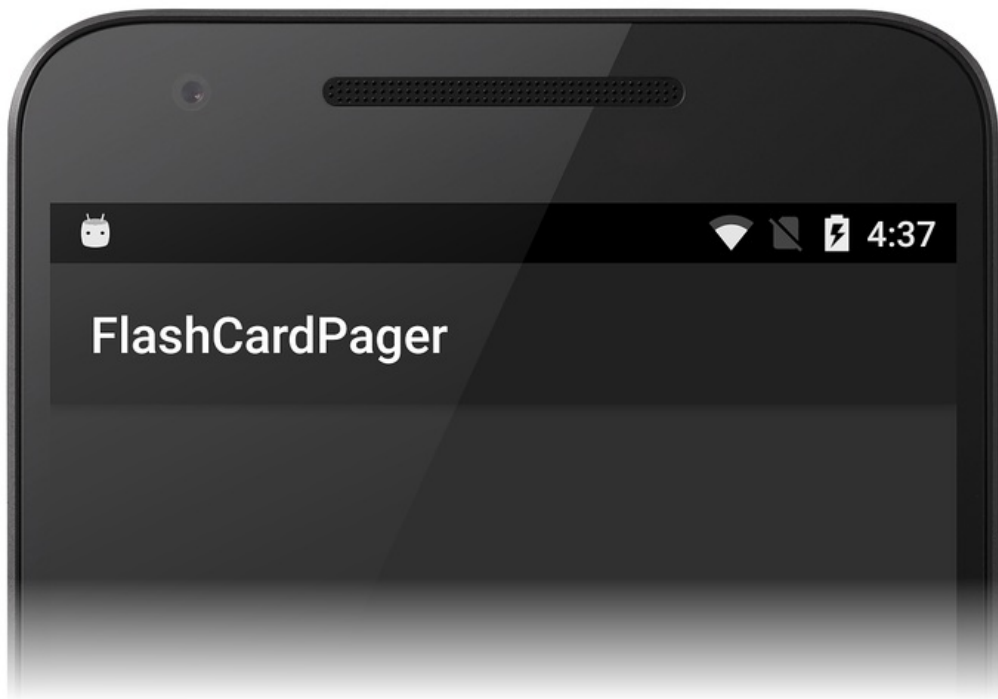
`MainActivity` 派生自 `FragmentActivity` (而非 `Activity`) 因为 `FragmentActivity` 知道如何管理的片段的支持。将 `OnCreate` 方法替换为以下代码：

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    ViewPager viewPager = FindViewById<ViewPager>(Resource.Id.viewpager);
    FlashCardDeck flashCards = new FlashCardDeck();
}
```

此代码将执行以下操作：

1. 设置从视图 `Main.axml` 布局资源。
2. 检索到的引用 `ViewPager` 从布局。
3. 实例化新 `FlashCardDeck` 作为数据源。

当生成并运行此代码时，您应看到类似于以下屏幕截图显示：



在此情况下，`ViewPager` 填充因为它缺少片段使用的是空 `ViewPager`，并且它用于从数据中创建这些片段缺乏适配器 **FlashCardDeck**。

在以下部分中，`FlashCardFragment` 是创建实现的功能的每个 flash 卡和一个 `FragmentPagerAdapter` 创建连接 `ViewPager` 到片段中创建 `FlashCardDeck`。

创建片段

每个闪存卡将由名为的 UI 片段 `FlashCardFragment`。`FlashCardFragment` 视图将显示与单个的闪存卡所包含的信息。每个实例 `FlashCardFragment` 将由承载 `ViewPager`。`FlashCardFragment` 视图将包含的 `TextView` 显示闪存卡问题文本。此视图将实现的事件处理程序使用 `Toast` 来在用户点击闪存卡问题时显示答案。

创建 `FlashCardFragment` 布局

之前 `FlashCardFragment` 可以是实现，就必须定义其布局。此布局是单个片段的片段容器布局。添加到新的 Android 布局资源/布局称为 `flashcard_layout.xml`。打开 `Resources/layout/flashcard_layout.xml` 和其内容替换为以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/flash_card_question"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textSize="100sp"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Question goes here" />
</RelativeLayout>
```

此布局定义的单个闪存卡片段;每个片段组成 `TextView` 显示数学问题使用较大 (100sp) 的字体。此文本垂直和水平

居中, 闪存卡上。

创建初始 FlashCardFragment 类

添加名为的新文件**FlashCardFragment.cs**和其内容替换为以下代码:

```
using System;
using Android.OS;
using Android.Views;
using Android.Widget;
using Android.Support.V4.App;

namespace FlashCardPager
{
    public class FlashCardFragment : Android.Support.V4.App.Fragment
    {
        public FlashCardFragment() { }

        public static FlashCardFragment newInstance(String question, String answer)
        {
            FlashCardFragment fragment = new FlashCardFragment();
            return fragment;
        }

        public override View onCreateView (
            LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
        {
            View view = inflater.Inflate (Resource.Layout.flashcard_layout, container, false);
            TextView questionBox = (TextView)view.FindViewById (Resource.Id.flash_card_question);
            return view;
        }
    }
}
```

此代码存根不可或缺 `Fragment` 定义将用于显示闪存卡。请注意, `FlashCardFragment` 派生自的支持库版本 `Fragment` 中定义 `Android.Support.V4.App.Fragment`。构造函数为空, 以便 `newInstance` 工厂方法用于创建一个新 `FlashCardFragment` 而不是一个构造函数。

`onCreateView` 生命周期方法创建并配置 `TextView`。增大的片段的布局 `TextView`, 并返回夸大 `TextView` 给调用方。`LayoutInflater` 并 `ViewGroup` 传递给 `onCreateView`, 以便它可以放大量布局。`savedInstanceState` 捆绑包包含数据的 `onCreateView` 用于重新创建 `TextView` 从已保存状态。

通过调用显式放大片段的视图 `inflater.Inflate`。`container` 自变量是视图的父级, 并 `false` 标志指示 `inflater` 不要夸大的视图添加到视图的父级 (将添加时 `ViewPager` 调用的适配器的 `getItem` 更高版本在此方法演练)。

将状态代码添加到 FlashCardFragment

类似于活动具有一个片段 `Bundle` 它用来保存和检索其状态。在中 `FlashCardPager`, 则此 `Bundle` 用来保存问题和回答有关关联的闪存卡的文本。在中 `FlashCardFragment.cs`, 添加以下 `Bundle` 顶部的关键 `FlashCardFragment` 类定义:

```
private static string FLASH_CARD_QUESTION = "card_question";
private static string FLASH_CARD_ANSWER = "card_answer";
```

修改 `newInstance` 工厂方法, 使其创建 `Bundle` 对象并使用上述密钥存储传递的问题和回答片段中的文本, 它实例化之后:

```
public static FlashCardFragment newInstance(String question, String answer)
{
    FlashCardFragment fragment = new FlashCardFragment();

    Bundle args = new Bundle();
    args.PutString(FLASH_CARD_QUESTION, question);
    args.PutString(FLASH_CARD_ANSWER, answer);
    fragment.Arguments = args;

    return fragment;
}
```

修改片段生命周期方法 `OnCreateView` 来检索此信息从传入的捆绑和加载到的问题文本 `TextBox` :

```
public override View OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
{
    string question = Arguments.GetString(FLASH_CARD_QUESTION, "");
    string answer = Arguments.GetString(FLASH_CARD_ANSWER, "");

    View view = inflater.Inflate(Resource.Layout.flashcard_layout, container, false);
    TextView questionBox = (TextView)view.FindViewById(Resource.Id.flash_card_question);
    questionBox.Text = question;

    return view;
}
```

`answer` 不在这里, 使用变量, 但它将在事件处理程序代码添加到此文件时更高版本使用。

创建适配器

`ViewPager` 使用适配器控制器对象位于之间 `ViewPager` 和数据源 (请参阅图中 `ViewPager` [适配器](#) 文章)。若要访问此数据, `ViewPager` 要求你提供自定义适配器派生自 `PagerAdapter`。此示例使用片段, 因为它使用 `FragmentPagerAdapter` – `FragmentPagerAdapter` 派生自 `PagerAdapter`。 `FragmentPagerAdapter` 表示为每个页 `Fragment` 的永久保留的片段管理器中, 只要用户可以返回到页。为浏览页用户扫 `ViewPager`, 则 `FragmentPagerAdapter` 从数据源中提取信息并使用它来创建 `Fragment` s 表示 `ViewPager` 显示。

当您实现 `FragmentPagerAdapter`, 必须重写以下:

- **计数** – 只读属性, 该属性返回的视图 (页面) 可用。
- **GetItem** – 返回要为指定的页显示的片段。

添加名为的新文件 **FlashCardDeckAdapter.cs** 和其内容替换为以下代码:

```

using System;
using Android.Views;
using Android.Widget;
using Android.Support.V4.App;

namespace FlashCardPager
{
    class FlashCardDeckAdapter : FragmentPagerAdapter
    {
        public FlashCardDeckAdapter (Android.Support.V4.App.FragmentManager fm, FlashCardDeck flashCards)
            : base(fm)
        {
        }

        public override int Count
        {
            get { throw new NotImplementedException(); }
        }

        public override Android.Support.V4.App.Fragment GetItem(int position)
        {
            throw new NotImplementedException();
        }
    }
}

```

此代码存根不可或缺 `FragmentPagerAdapter` 实现。在以下部分中，每种方法将被替换的工作代码。构造函数的目的是将传递到片段管理器 `FlashCardDeckAdapter` 的基类构造函数。

实现适配器构造函数

当应用程序实例化 `FlashCardDeckAdapter`，它会提供对片段管理器和实例化的引用 `FlashCardDeck`。将以下成员变量添加到顶部 `FlashCardDeckAdapter` 类中**FlashCardDeckAdapter.cs**:

```
public FlashCardDeck flashCardDeck;
```

添加以下代码行 `FlashCardDeckAdapter` 构造函数：

```
this.flashCardDeck = flashCards;
```

这行代码将存储 `FlashCardDeck` 实例的 `FlashCardDeckAdapter` 将使用。

实现计数

`Count` 实现是相对简单：它在闪存卡一副纸牌中返回的闪存卡数。将 `Count` 替换为以下代码：

```

public override int Count
{
    get { return flashCardDeck.NumCards; }
}

```

`NumCards` 属性的 `FlashCardDeck` 数据集中返回的闪存卡（的片段数）数。

实现 `GetItem`

`GetItem` 方法返回与给定的位置相关联的片段。当 `GetItem` 调用中 flash 纸牌的位置返回 `FlashCardFragment` 配置为在该位置显示闪存卡问题。将 `GetItem` 方法替换为以下代码：

```
public override Android.Support.V4.App.Fragment GetItem(int position)
{
    return (Android.Support.V4.App.Fragment)
        FlashCardFragment.newInstance (
            flashCardDeck[position].Problem, flashCardDeck[position].Answer);
}
```

此代码将执行以下操作：

1. 查找数学问题字符串 `FlashCardDeck` 指定位置的一副纸牌。
2. 查找答案字符串 `FlashCardDeck` 指定位置的一副纸牌。
3. 调用 `FlashCardFragment` 工厂方法 `newInstance`、传入的闪存卡问题和答案字符串。
4. 创建并返回新的闪存卡 `Fragment`，包含该位置的问题和答案的文本。

当 `ViewPager` 呈现 `Fragment` 在 `position`，它将显示 `TextBox` 包含驻留在数学问题字符串 `position` flash 卡片组中。

将适配器添加到 ViewPager

既然 `FlashCardDeckAdapter` 是实现，是时候将其添加到 `ViewPager`。在中 `MainActivity.cs`，将以下代码行添加到末尾 `OnCreate` 方法：

```
FlashCardDeckAdapter adapter =
    new FlashCardDeckAdapter(SupportFragmentManager, flashCards);
viewPager.Adapter = adapter;
```

此代码实例化 `FlashCardDeckAdapter`，并传入 `SupportFragmentManager` 中的第一个参数。（`SupportFragmentManager` `FragmentManager` 属性用于获取对引用 `FragmentManager` - 有关详细信息 `FragmentManager`，请参阅[管理片段](#)。）

现已完成的核心实现-生成并运行应用。应看到显示在屏幕上，在左侧的下一步的屏幕截图中所示的第一个图像的闪存卡一副纸牌。轻扫保留以查看更多的闪存卡，然后右轻扫可向后移动到闪存卡一副纸牌：



添加页导航指示器

此最小 `ViewPager` 实现显示每个闪存卡中的一副纸牌，但它可以提供没有指明用户所在的一副纸牌中。下一步是添

加 `PagerTabStrip`。 `PagerTabStrip` 通知的用户有关的问题数显示, 并提供导航上下文通过显示上一个和下一步闪存卡的提示。

打开 **Resources/layout/Main.xml** 并添加 `PagerTabStrip` 到布局:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <android.support.v4.view.PagerTabStrip
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:paddingBottom="10dp"
        android:paddingTop="10dp"
        android:textColor="#fff" />

</android.support.v4.view.ViewPager>
```

当生成并运行应用时, 应该会看到空 `PagerTabStrip` 显示在每个闪存卡的顶部:



显示标题

若要添加到每个页选项卡标题, 实现 `GetPageTitleFormatted` 适配器中的方法。 `ViewPager` 调用

`GetPageTitleFormatted` (如果已实现) 获取描述指定位置处的页的标题字符串。添加以下方法

`FlashCardDeckAdapter` 类中 **FlashCardDeckAdapter.cs**:

```
public override Java.Lang.ICharSequence GetPageTitleFormatted(int position)
{
    return new Java.Lang.String("Problem " + (position + 1));
}
```

此代码将闪存卡一副纸牌中的位置转换为问题数。生成的字符串转换为 Java `String` 返回到 `ViewPager`。当使用此新方法运行该应用时，每个页面将显示问题数量中的 `PagerTabStrip`：



您可以轻扫来回若要查看每个闪存卡顶部显示闪存卡一副纸牌中的问题编号。

处理用户输入

FlashCardPager提供了一系列中基于片段的闪存卡 `ViewPager`，但它还没有一种方法，以显示每个问题的答案。在本部分中，事件处理程序添加到 `FlashCardFragment` 来在用户点击闪存卡问题文本上时显示答案。

打开 **FlashCardFragment.cs** 并将以下代码添加到末尾 `OnCreateView` 方法视图返回给调用方之前：

```
questionBox.Click += delegate
{
    Toast.MakeText(Activity.ApplicationContext,
        "Answer: " + answer, ToastLength.Short).Show();
};
```

这 `Click` 事件处理程序中将显示当用户点击 toast 通知显示答案 `TextBox`。`answer` 变量已初始化之前从传递给绑定读取状态信息时 `OnCreateView`。生成和运行应用，然后点击上每个闪存卡以查看解答的问题文本：



FlashCardPager显示在本演练中使用 `MainActivity` 派生自 `FragmentActivity`，但也可以派生 `MainActivity` 从 `AppCompatActivity`（其中还提供对管理支持片段）。若要查看 `AppCompatActivity` 示例中，请参阅 [FlashCardPager](#) 示例库中。

总结

本演练提供如何生成基本的分步示例 `ViewPager` - 基于应用程序使用 `Fragment` s。显示包含闪存卡问题和答案，示例数据源 `ViewPager` 布局，以显示闪存卡中，和一个 `FragmentPagerAdapter` 连接的子类 `ViewPager` 到数据源。为了帮助用户可以浏览闪存卡，说明已包含说明了如何添加 `PagerTabStrip` 若要在每个页面的顶部显示的问题数量。最后，添加事件处理代码是为了在用户点击闪存卡问题时显示答案。

相关链接

- [FlashCardPager（示例）](#)

Web 视图

2018/10/26 • [Edit Online](#)

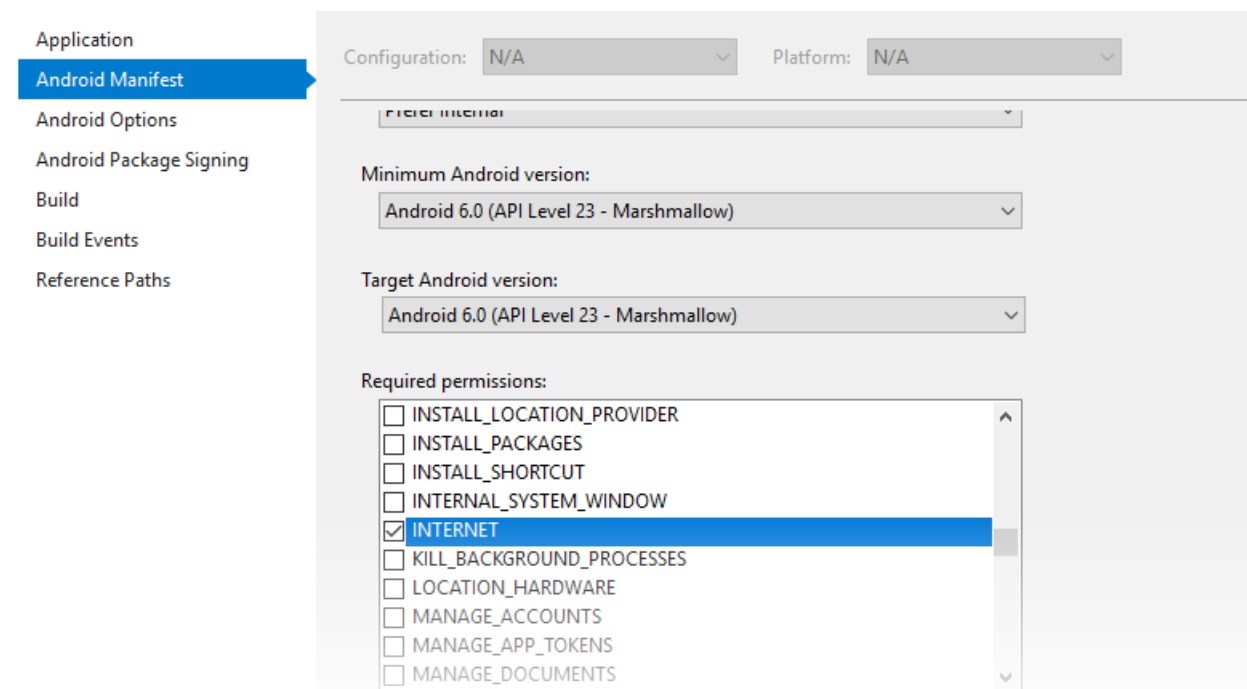
`WebView` 可以创建自己的窗口用于查看的网页（或甚至开发完整的浏览器）。在本教程中，你将创建一个简单 `Activity` 可以查看和导航网页。

创建一个名为的新项目 **HelloWebView**。

打开 **Resources/Layout/Main.xml** 并插入以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

由于此应用程序将访问 Internet, 必须添加相应的权限向 Android 清单文件。打开项目的属性, 以指定你的应用程序运行所需的权限。启用 `INTERNET` 权限如下所示：



现在, 打开 **MainActivity.cs** 和添加 using 指令的易于使用的功能：

```
using Android.Webkit;
```

在顶部 `MainActivity` 类中, 声明 `WebView` 对象：

```
WebView web_view;
```

当 **WebView** 是要求加载 URL 时, 它会将默认情况下委托对默认浏览器的请求。能够 **WebView** 加载 URL (而不是默认浏览器), 则必须子类 `Android.Webkit.WebViewClient` 并重写 `ShouldOverrideUrlLoading` 方法。此自定义的实例 `WebViewClient` 提供给 `WebView`。若要执行此操作, 添加以下嵌套 `HelloWebViewClient` 类 `MainActivity`：

```
public class HelloWebViewClient : WebViewClient
{
    public override bool ShouldOverrideUrlLoading (WebView view, string url)
    {
        view.LoadUrl(url);
        return false;
    }
}
```

当 `ShouldOverrideUrlLoading` 返回 `false`，向 Android 发出信号，它的当前 `WebView` 实例处理请求并不需要任何进一步的操作。

如果您的目标 API 级别 24 或更高版本，请使用的重载 `ShouldOverrideUrlLoading` 采用 `IWebResourceRequest` 的第二个参数而不是 `string`：

```
public class HelloWebViewClient : WebViewClient
{
    // For API level 24 and later
    public override bool ShouldOverrideUrlLoading (WebView view, IWebResourceRequest request)
    {
        view.LoadUrl(request.Url.ToString());
        return false;
    }
}
```

接下来，使用以下代码 `OnCreate()` 方法：

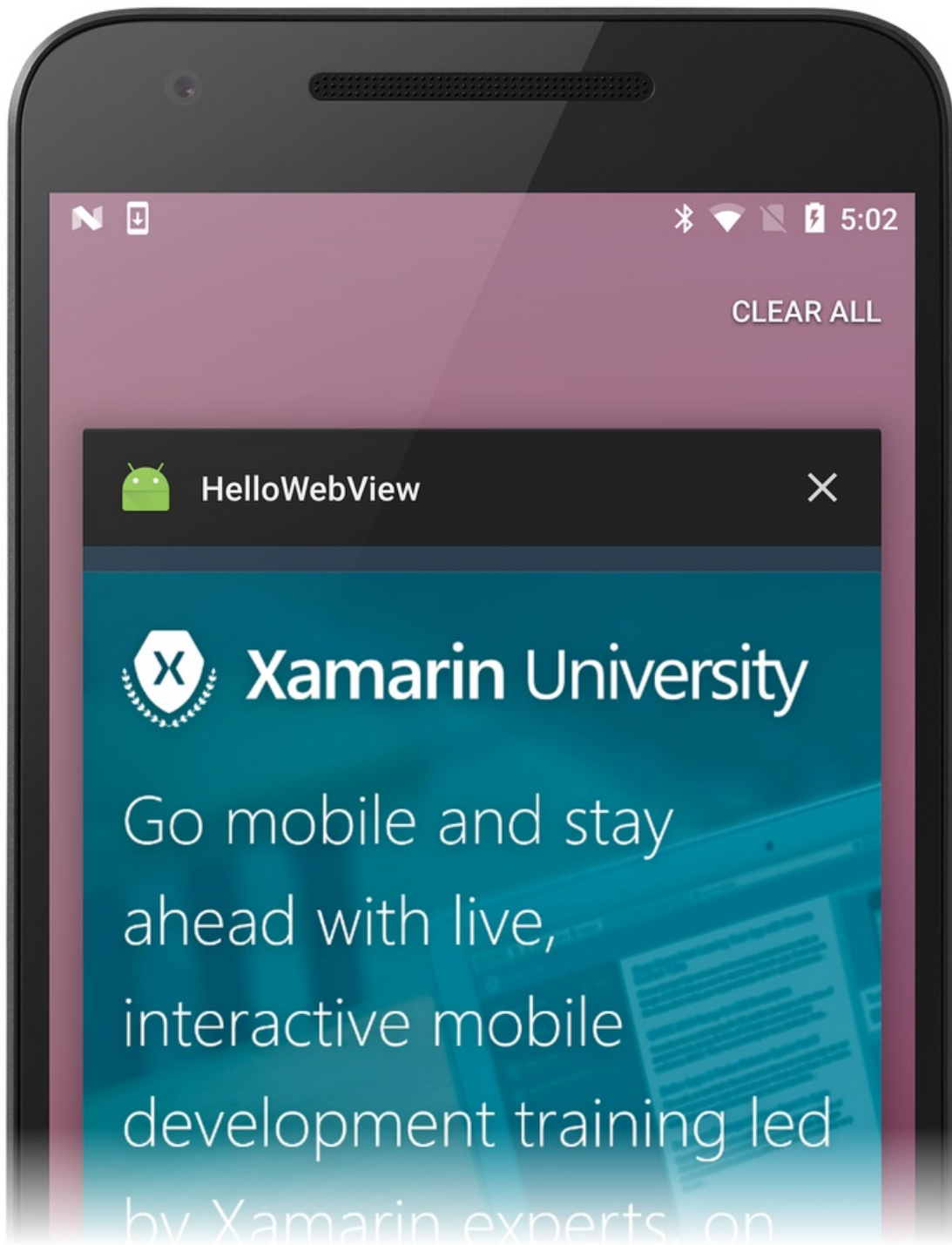
```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    // Set our view from the "main" layout resource
    SetContentView (Resource.Layout.Main);

    web_view = FindViewById<WebView> (Resource.Id.webview);
    web_view.Settings.JavaScriptEnabled = true;
    web_view.SetWebViewClient(new HelloWebViewClient());
    web_view.LoadUrl ("https://www.xamarin.com/university");
}
```

此值将初始化成员 `WebView` 中 `Activity` 布局，并启用适用于 JavaScript `WebView` 与 `JavaScriptEnabled = true` (请参阅[调用 C#从 JavaScript](#)方案了解如何调用 C#从 JavaScript 函数)。最后，与加载初始网页 `LoadUrl(String)`。

生成并运行应用。您应该看到一个简单的 web 页面查看器应用程序与以下屏幕截图所示：



若要处理回按钮按下，添加以下 using 语句：

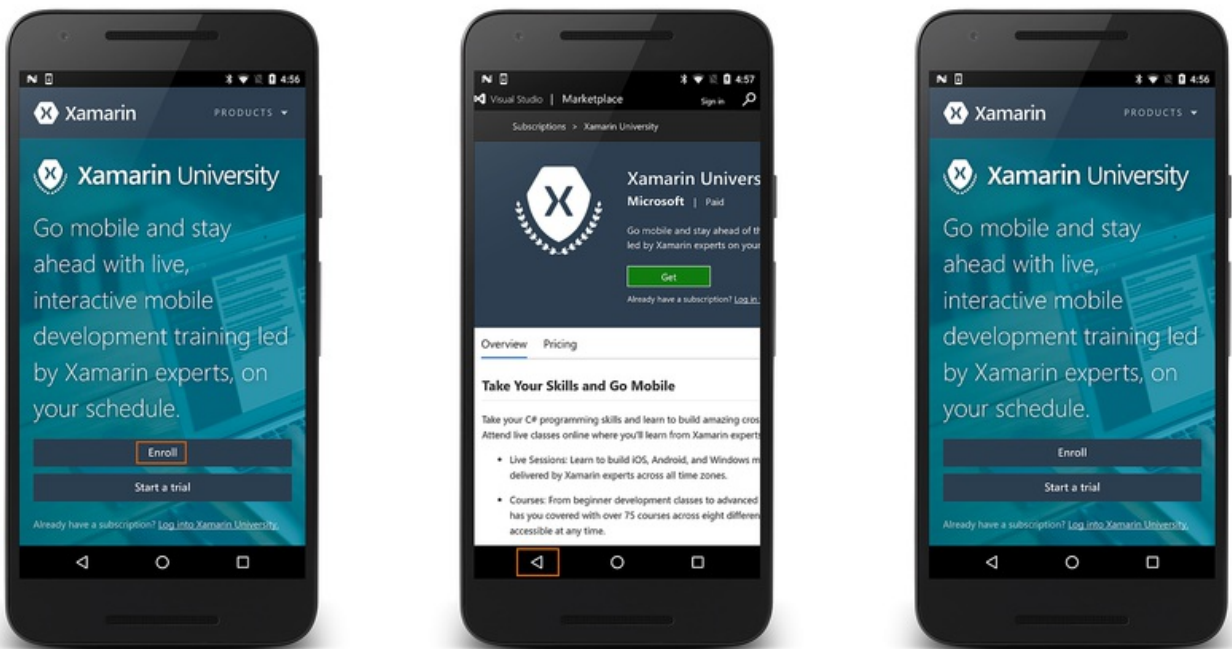
```
using Android.Views;
```

接下来，添加以下方法 `HelloWebView` 活动：

```
public override bool OnKeyDown (Android.Views.Keycode keyCode, Android.Views.KeyEvent e)
{
    if (keyCode == Keycode.Back && web_view.CanGoBack ())
    {
        web_view.GoBack ();
        return true;
    }
    return base.OnKeyDown (keyCode, e);
}
```

此 `OnKeyDown(int, KeyEvent)` 该活动正在运行并且按下按钮时，将调用回调方法。内部使用的条件 `KeyEvent` 若要检查是否按下了键回按钮以及是否 `WebView` 实际上能够导航回（如果它具有历史记录）。如果两者均为 `true`，则 `GoBack()` 调用方法时，这将导航中的后退一步 `WebView` 历史记录。返回 `true` 指示已处理该事件。如果不满足此条件，然后该事件发送到系统。

再次运行该应用程序。现在应能够单击的链接，并在向后导航页历史记录：



此页的部分是基于工作创建和共享通过 Android 的开放源项目和使用中所述的条款的修改 [Creative Commons 2.5 Attribution 许可证](#)。

相关链接

- [从 JavaScript 中调用 C#](#)
- [Android.Webkit.WebView](#)
- [KeyEvent](#)

平台功能

2018/10/26 • [Edit Online](#)

在本部分中的文档介绍特定于 Android 的功能。这里可以找到主题，如使用片段、使用映射和封装与内容提供商的数据。

Android 无线发送

Android 无线发送是使应用程序可以通过在邻近的 NFC 共享信息的 Android 4 中的新近场通信 (NFC) 技术。

使用文件

本部分讨论如何访问在 Xamarin.Android 中的文件。

指纹身份验证

本部分讨论如何使用 Android 6.0 中首次引入到 Xamarin.Android 应用程序的指纹身份验证。

Firebase 作业调度程序

本指南讨论了 Firebase 作业调度程序以及如何使用它来简化在 Xamarin.Android 应用程序中正在运行的后台作业。

片段

Android 3.0 引入了片段，演示如何在手机和平板电脑上找到的许多不同的屏幕大小支持更灵活的设计。本文将介绍如何使用片段开发 Xamarin.Android 应用程序，以及如何在预 Android 3.0 (API 级别 11) 设备上支持片段。

将应用链接

本指南介绍了如何支持 Android 6.0_将应用链接_，该技术允许移动应用来响应在网站上的 Url。它将讨论如何实现应用程序将 Android 6.0 应用程序中的链接以及如何配置网站以授予对移动应用的权限来处理应用程序链接的域。

Android 9 饼图

本文提供 Android 饼图中的新增功能的概述，介绍如何进行准备 Xamarin.Android 进行 Android 饼图开发，并提供了一个应用示例，演示了如何使用新 Android 饼图显示切除和通知中的功能Xamarin.Android 应用。

Android 8 Oreo

本文提供了 Android Oreo 中的新增功能的概述介绍了如何准备 Xamarin.Android 进行 Android Oreo 开发，并且提供了示例应用程序，展示了如何在 Xamarin.Android 应用中使用 Android Oreo 功能的链接。

Android 7 Nougat

本文提供了 Android 7.0 Nougat 中引入的新功能的高级概述。

Android 6 Marshmallow

本文提供了 Android 6.0 Marshmallow 中引入的新功能的高级概述。

Android 5 Lollipop

此指南概述了新的 Android 5.0 Lollipop 功能，例如材料主题、CardView、RecyclerView 和 Heads 注册通知，并将其链接供深入了解文章，帮助你在应用中使用这些新功能。

Android 4.4 KitKat

Android 4.4 (KitKat) 是加载了太多的用户和开发人员的功能。本指南重点介绍了这些功能的几个，并提供代码示例和实现详细信息，以帮助您充分利用 KitKat。

Android 4.1 Jelly Bean

本文档将 Android 4.1 中引入的开发人员提供新功能的高级的概述。这些功能包括：增强的通知，Android 无线发送共享大型文件、多媒体、对等网络发现、动画、新的权限的更新的更新。

Android 4.0 Ice Cream Sandwich

本指南介绍了几个应用程序开发者使用的新功能 *Android 4 API-Ice Cream Sandwich*。它介绍了几种新的用户界面技术，然后检查各种 Android 4 提供的数据之间的应用程序和设备之间共享的新功能。

使用 Android 清单

本文介绍了 AndroidManifest.xml 文件，以及如何将它可能用来控制功能和描述 Mono for Android 应用程序的要求。

内容提供程序简介

ContentProvider 封装的数据存储库，并提供一个 API 来访问它。提供程序还提供一个 UI 以便显示/管理的数据的 Android 应用程序的一部分存在。使用内容提供商的主要好处，让其他应用程序能够轻松地访问封装的数据使用提供程序客户端对象（称为 ContentResolver）。一起内容提供商和内容冲突解决程序提供一致的应用程序间 API 进行生成和使用简单的数据访问。本文档演示如何访问和生成使用 Xamarin.Android Contentprovider。

地图和位置

本部分讨论如何使用 Xamarin.Android 使用地图和位置。它涵盖了从利用内置地图应用程序使用的所有内容 [Google Maps Android API v2](#) 直接。此外，它说明了如何使用单个 API 来使用位置服务，使用移动电话三角测量以使应用程序来获取位置的修补程序、Wi-fi 位置和 GPS。

Android 语音

本部分讨论如何使用 Android 文本到语音和语音到文本功能。它还介绍了安装语言包和解释的到设备所说的文本。

绑定 Java 库

本指南介绍如何将 Java 库合并到 Xamarin.Android 应用，通过创建绑定库。

Java 集成

本文概述了开发人员可以重复使用 Xamarin.Android 应用中的现有 Java 组件的方式。

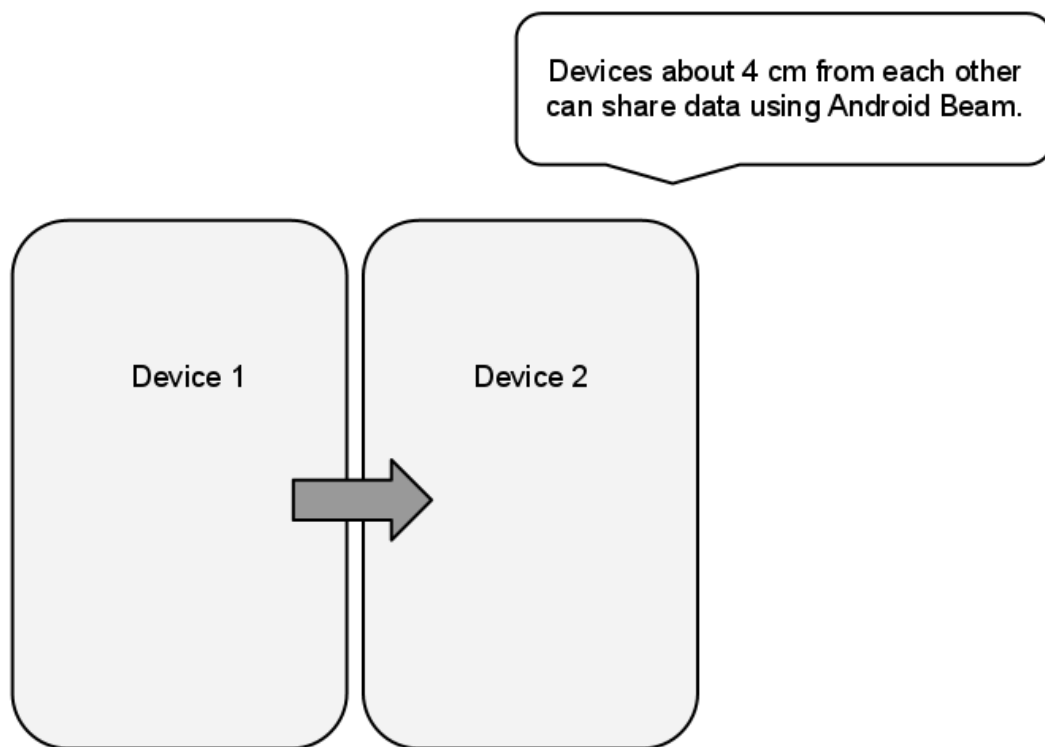
RenderScript

本指南介绍 RenderScript。

Android 无线发送

2018/10/26 • [Edit Online](#)

Android 无线发送是使应用程序可以通过在邻近的 NFC 共享信息的 Android 4.0 中引入的近场通信 (NFC) 技术。



Android 无线发送的工作原理是通过 NFC 推送消息，当两个设备在范围内时。设备占用约 4 厘米彼此可以共享使用 Android 无线发送的数据。一台设备上的活动创建一条消息，并指定一个活动（或活动），它可以处理将其推送。如果设备处于范围内指定的活动是在前台，Android 无线发送会将消息推送到另一台设备。接收设备上包含消息数据调用意向。

Android 支持两种方法的设置与 Android 无线发送的消息：

- `SetNdefPushMessage` - 启动 Android 无线发送之前，应用程序可以调用 `SetNdefPushMessage` 指定 `NdefMessage` 推送通过 NFC 和将其推送的活动。当一条消息不会更改应用程序正在使用中时，最好使用此机制。
- `SetNdefPushMessageCallback` - 当启动 Android 无线发送时，应用程序可以处理一个回调，以创建 `NdefMessage`。此机制允许为消息创建可延迟到设备处于范围。它支持基于应用程序中发生的情况，消息可能会有所不同的方案。

在任一情况下，将数据与 Android 无线发送，发送应用程序发送 `NdefMessage`，在多个将数据打包 `NdefRecords`。让我们看看我们可以触发 Android 无线发送之前必须解决的关键点。首先，我们将使用创建的回调样式 `NdefMessage`。

创建一条消息

我们可以注册使用回调 `NfcAdapter` 中的活动 `onCreate` 方法。例如，假设 `NfcAdapter` 名为 `mNfcAdapter` 声明为类变

量在活动中，我们可以编写以下代码以创建将构造消息的回调：

```
mNfcAdapter = NfcAdapter.getDefaultAdapter (this);  
mNfcAdapter.SetNdefPushMessageCallback (this, this);
```

实现的活动 `NfcAdapter.ICreateNdefMessageCallback`，传递给 `SetNdefPushMessageCallback` 上述方法。启动 Android 无线发送时，系统将调用 `CreateNdefMessage`，从该活动可以构造 `NdefMessage`，如下所示：

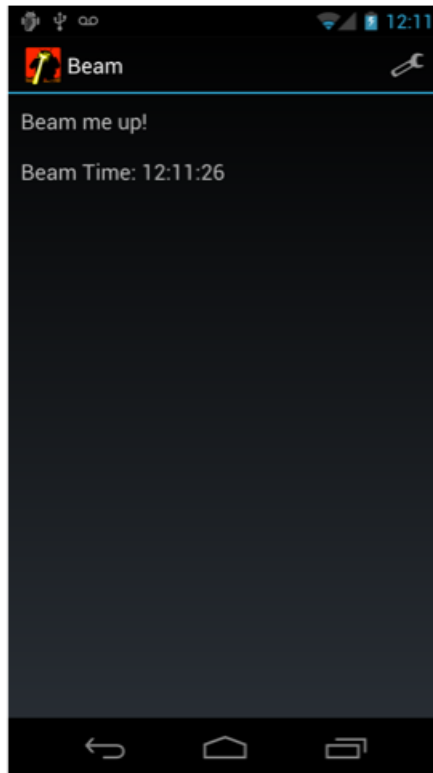
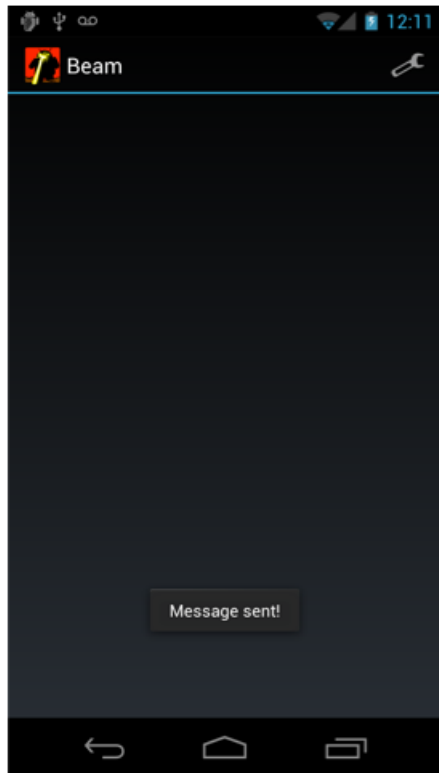
```
public NdefMessage CreateNdefMessage (NfcEvent evt)  
{  
    DateTime time = DateTime.Now;  
    var text = ("Beam me up!\n\n" + "Beam Time: " +  
        time.ToString ("HH:mm:ss"));  
    NdefMessage msg = new NdefMessage (  
        new NdefRecord[] { CreateMimeRecord (  
            "application/com.example.android.beam",  
            Encoding.UTF8.GetBytes (text)) } });  
    } };  
    return msg;  
}  
  
public NdefRecord CreateMimeRecord (String mimeType, byte [] payload)  
{  
    byte [] mimeBytes = Encoding.UTF8.GetBytes (mimeType);  
    NdefRecord mimeRecord = new NdefRecord (  
        NdefRecord.TnfMimeMedia, mimeBytes, new byte [0], payload);  
    return mimeRecord;  
}
```

接收消息

在接收端，系统会调用与意向 `ActionNdefDiscovered` 操作，从中我们可以提取 `NdefMessage`，如下所示：

```
IParcelable [] rawMsgs = intent.GetParcelableArrayExtra (NfcAdapter.ExtraNdefMessages);  
NdefMessage msg = (NdefMessage) rawMsgs [0];
```

使用 Android 无线发送，下面的屏幕截图中所示的完整代码示例，请参阅[Android 无线发送演示](#)示例库中。



相关链接

- [Android 无线发送演示（示例）](#)
- [引入 Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

使用 Android 清单

2018/10/26 • [Edit Online](#)

概述

AndroidManifest.xml是功能强大的文件，在 Android 平台，可用于描述的功能和到 Android 应用程序的要求。但是，使用它并不容易。Xamarin.Android 可帮助尽量减少此问题通过允许您将自定义属性添加到你的类，然后将用于自动生成的清单。我们的目标是 99%的用户应永远不需要手动修改**AndroidManifest.xml**。

AndroidManifest.xml作为生成过程中和找到的 XML 的一部分生成**properties/Androidmanifest.xml**与从自定义特性生成的 XML 合并。生成合并**AndroidManifest.xml**驻留在obj子目录;例如，它位于obj/Debug/android/**AndroidManifest.xml**对于调试版本。合并过程非常简单：它使用自定义特性的代码中生成 XML 元素和将插入到这些元素**AndroidManifest.xml**。

基本知识

在编译时，程序集进行扫描的非 `abstract` 派生的类活动并且有 `[Activity]` 在其上声明属性。它然后使用这些类和属性生成清单。例如，考虑以下代码：

```
namespace Demo
{
    public class MyActivity : Activity
    {
    }
}
```

这会导致执行任何操作中生成**AndroidManifest.xml**。如果你想 `<activity/>` 元素生成，你需要使用 `[Activity]` 自定义属性：

```
namespace Demo
{
    [Activity]
    public class MyActivity : Activity
    {
    }
}
```

此示例会导致下面的 xml 片段，若要添加到**AndroidManifest.xml**：

```
<activity android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity" />
```

`[Activity]` 属性不起任何作用 `abstract` 类型; `abstract` 类型将被忽略。

活动名称

从 Xamarin.Android 5.1 开始，活动的类型名称基于要导出的类型的程序集限定名称的 MD5SUM。这样，相同的完全限定名称，以提供从两个不同的程序集，并且不使打包错误。（在 Xamarin.Android 5.1 中之前，活动的默认类型名称已从创建小写的命名空间和类名。）

如果你想要覆盖此默认值并显式指定您的活动，使用名称 `Name` 属性：

```
[Activity (Name="awesome.demo.activity")]
public class MyActivity : Activity
{
}
```

此示例将生成以下 xml 片段：

```
<activity android:name="awesome.demo.activity" />
```

*请注意：*应使用 `Name` 仅出于向后兼容性原因，这种情况下重命名的属性可能会降低在运行时类型查找。如果有旧代码预期要基于的小写的命名空间的活动的默认类型名称和类名，请参阅[Android 可调用包装器命名](#)有关保持兼容性的提示。

活动标题栏

默认情况下，Android 提供你的应用程序标题栏将在运行时。使用此值是

`/manifest/application/activity/@android:label`。在大多数情况下，此值将不同于您的类名。若要指定应用的标签的标题栏上，使用 `Label` 属性。例如：

```
[Activity (Label="Awesome Demo App")]
public class MyActivity : Activity
{
}
```

此示例将生成以下 xml 片段：

```
<activity android:label="Awesome Demo App"
          android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity" />
```

可以从应用程序选择器

默认情况下，您的活动不会出现在 Android 的应用程序启动器屏幕中。这是因为可能有许多活动应用程序中，并且对于每个不想一个图标。若要指定哪一列应为可从应用程序启动程序启动，请使用 `MainLauncher` 属性。例如：

```
[Activity (Label="Awesome Demo App", MainLauncher=true)]
public class MyActivity : Activity
{
}
```

此示例将生成以下 xml 片段：

```
<activity android:label="Awesome Demo App"
          android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

活动图标

默认情况下，您的活动将提供由系统提供的默认启动器图标。若要使用自定义图标，请首先添加你 `.png` 到资源 `/drawable`，将其生成操作设置为 **AndroidResource**，然后使用 `Icon` 属性来指定要使用的图标。例如：

```
[Activity (Label="Awesome Demo App", MainLauncher=true, Icon="@drawable/myicon")]
public class MyActivity : Activity
{
}
```

此示例将生成以下 xml 片段：

```
<activity android:icon="@drawable/myicon" android:label="Awesome Demo App"
    android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

权限

当将权限添加到 Android 清单 (如中所述[将权限添加到 Android 清单](#))，这些权限记录在**properties/Androidmanifest.xml**。例如，如果您设置 `INTERNET` 权限，将以下元素添加到**properties/Androidmanifest.xml**：

```
<uses-permission android:name="android.permission.INTERNET" />
```

调试版本会自动设置某些权限以简化调试 (例如 `INTERNET` 并 `READ_EXTERNAL_STORAGE`)—配置这些设置仅在生成**obj/Debug/android/AndroidManifest.xml**，而不显示为已启用所需的权限设置。

例如，如果您检查在生成清单文件**obj/Debug/android/AndroidManifest.xml**，可能会看到以下添加的权限元素：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

发行版中的生成清单的版本 (在**obj/Debug/android/AndroidManifest.xml**)，这些权限是不自动配置。如果您发现，切换到发布版本会导致应用失去了调试版本中提供的权限，验证是否已显式设置此权限所需的权限应用 (请参阅的设置生成 > **Android** 应用程序在 Visual Studio for Mac; 请参阅属性 > **Android** 清单 Visual Studio 中)。

高级的功能

意向操作和功能

Android 清单提供了一种方法，以便描述您的活动的功能。这是通过意向和 `[IntentFilter]` 自定义属性。可以指定的操作是适用于您使用的活动 `IntentFilter` 构造函数，和使用适当的哪些类别 `Categories` 属性。至少一个活动必须提供 (这是在构造函数中提供的活动的原因)。`[IntentFilter]` 可以提供多个，且每次使用会导致单独

`<intent-filter/>` 元素内的 `<activity/>`。例如：

```
[Activity (Label="Awesome Demo App", MainLauncher=true, Icon="@drawable/myicon")]
[IntentFilter (new[]{Intent.ActionView},
    Categories=new[]{Intent.CategorySampleCode, "my.custom.category"})]
public class MyActivity : Activity
{
}
```

此示例将生成以下 xml 片段：

```
<activity android:icon="@drawable/myicon" android:label="Awesome Demo App"
    android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.SAMPLE_CODE" />
        <category android:name="my.custom.category" />
    </intent-filter>
</activity>
```

应用程序元素

Android 清单还提供了一种方法，以便声明在整个应用程序的属性。这是通过 `<application>` 元素和其对应[应用程序](#)自定义属性。请注意，这些是应用程序级（程序集范围）的设置，而不是每个活动的设置。通常情况下，声明 `<application>` 整个应用程序的属性和替代这些设置（如需要）在每个活动的基础上。

例如，以下 `Application` 属性添加到**AssemblyInfo.cs**以指示可以调试该应用程序，其用户可读名称是**My App**，并使用 `Theme.Light` 所有活动的默认主题样式：

```
[assembly: Application (Debuggable=true,
    Label="My App",
    Theme="@android:style/Theme.Light")]
```

此声明将导致下面的 XML 片段，若要在其中生成**obj/Debug/android/AndroidManifest.xml**：

```
<application android:label="My App"
    android:debuggable="true"
    android:theme="@android:style/Theme.Light"
    ... />
```

在此示例中，应用程序中的所有活动将都默认为 `Theme.Light` 样式。如果活动的主题设置为 `Theme.Dialog`，仅该活动将会使用 `Theme.Dialog` 风格的应用程序中的所有其他活动将默认为 `Theme.Light` 样式中设置 `<application>` 元素。

`Application` 元素不是唯一的方法来配置 `<application>` 属性。或者，可以插入特性直接插入 `<application>` 的元素 **properties/Androidmanifest.xml**。这些设置将合并成最终 `<application>` 驻留在的元数据 **obj/Debug/android/AndroidManifest.xml**。请注意的内容 **properties/Androidmanifest.xml** 一直重写提供自定义特性的数据。

还有很多应用程序范围属性，可以在中配置 `<application>` 元素；有关这些设置的详细信息，请参阅[公共属性](#)一部分 **ApplicationAttribute**。

自定义特性列表

- [Android.App.ActivityAttribute](#)：生成 **/manifest/application/activity** XML 片段
- [Android.App.ApplicationAttribute](#)：生成 **/清单/应用程序** XML 片段
- [Android.App.InstrumentationAttribute](#)：生成 **/清单/检测** XML 片段
- [Android.App.IntentFilterAttribute](#)：生成 **//intent-filter** XML 片段
- [Android.App.MetadataAttribute](#)：生成 **//meta-data** XML 片段
- [Android.App.PermissionAttribute](#)：生成 **//permission** XML 片段
- [Android.App.PermissionGroupAttribute](#)：生成 **//permission-group** XML 片段
- [Android.App.PermissionTreeAttribute](#)：生成 **//permission-tree** XML 片段

- [Android.App.ServiceAttribute](#) : 生成/manifest/application/service XML 片段
- [Android.App.UsesLibraryAttribute](#) : 生成/manifest/application/uses-library XML 片段
- [Android.App.UsesPermissionAttribute](#) : 生成/manifest/uses-permission XML 片段
- [Android.Content.BroadcastReceiverAttribute](#) : 生成/manifest/application/receiver XML 片段
- [Android.Content.ContentProviderAttribute](#) : 生成/manifest/application/provider XML 片段
- [Android.Content.GrantUriPermissionAttribute](#) : 生成/manifest/application/provider/grant-uri-permission XML 片段

文件存储和使用 Xamarin.Android 的访问

2018/10/26 • [Edit Online](#)

Android 应用的一个常见要求是操作文件—保存图片、下载的文档，或导出数据与其他程序共享。Android（这基于 Linux）通过提供针对文件存储空间来支持此模式。Android 到两个不同类型的存储组文件系统：

- **内部存储**—这是可以访问只能由应用程序或操作系统文件系统的一部分。
- **外部存储**—这是存储的文件访问的所有应用、用户和其他设备可能是一个分区。在某些设备上外部存储可能是可移动（如 SD 卡）。

这些分组才概念，并不一定是引用单个分区或设备上的目录。Android 设备将始终提供内部存储和外部存储的分区。很可能某些设备可能包含被视为外部存储的多个分区。而不考虑分区进行读取的 Api，编写，或创建的文件是相同的。有两个 Xamarin.Android 应用程序可用于文件访问的 Api 集：

1. **.NET Api**（提供的 **Mono** 和 **xamarin.android** 包装）—这些包含 [文件系统帮助程序](#) 提供 [Xamarin.Essentials](#)。
.NET Api 提供最佳的跨平台兼容性和本指南的重点将这种情况下是这些 Api。
2. **本机 Java 文件访问 Api**（提供的 **Java** 和 **xamarin.android** 包装）—Java 提供其自身 Api，可用于读取和写入文件。这些是对 .NET Api 中，完全可以接受的替代方案，但特定于 Android，并不适合用于跨平台的应用。

读取和写入文件，在 Xamarin.Android 中几乎完全相同，因为它是与任何其他 .NET 应用程序。Xamarin.Android 应用程序确定将进行操作，然后使用标准 .NET 习惯用语文件访问的文件的路径。由于内部和外部存储的实际路径可能不同设备的或从到 Android 版本的 Android 版本，建议不要进行硬编码文件的路径。相反，使用 Xamarin.Android Api 来确定文件的路径。这样一来，用于读取和写入文件的 .NET Api 公开的本机 Android Api，可帮助确定文件存储在内部和外部存储的路径。

在讨论之前具有文件访问权限相关的 Api，务必了解一些有关内部和外部存储的详细信息。这将在下一节中讨论。

内部与外部存储

从概念上讲，内部存储和外部存储都是非常类似—它们是一个 Xamarin.Android 应用程序可能会将文件保存的这两个位置。这种相似性可能令人困惑，因为它不是清除应用程序应使用内部存储 vs 外部存储时不熟悉 Android 开发人员。

内部存储是指 Android 分配到操作系统 Apk，并为单个应用的非易失性内存。此空间不可访问除操作系统或应用程序。Android 将为每个应用分配中内部存储分区的目录。卸载应用，都将在内部存储在该目录中的所有文件也将被删除。内部存储最适合用于文件才可以访问该应用程序，并将不会与其他应用共享或卸载应用后，将具有很少的值。Android 6.0 或更高版本，文件存储在内部存储可能会自动备份使用 Google [Android 6.0 中的自动备份功能](#)。内部存储具有以下缺点：

- 不能共享文件。
- 卸载应用程序时，将删除文件。
- 可能是有限的内部存储上的可用空间。

外部存储是指不是内部存储的文件存储和应用程序无法以独占方式访问。外部存储的主要用途是提供放置的文件，用于在应用之间共享或太大，无法在内部存储的地方。外部存储的优点是它通常具有比内部存储的文件的更多空间。但是，外部存储不能始终保证存在的设备上，可能需要从用户的特殊权限来访问它。

NOTE

Android 将适用于支持多个用户的设备，提供每个用户自身的目录，对内部和外部存储。此目录是在设备上的其他用户无法访问。只要他们做不到文件存储在内部或外部存储进行硬编码路径，这种分离是对应用程序不可见。

根据经验, Xamarin.Android 应用应更喜欢将其文件存储在内部存储保存时是合理的并依赖于外部存储或文件时需要与其他应用程序共享, 非常大, 即使在卸载应用程序应保留。例如, 配置文件是最适合用于内部存储, 因为它具有除到创建它的应用不重要。与此相反, 照片是外部存储的良好候选项。它们可能会非常大, 在许多情况下用户可能想要共享或访问它们, 即使卸载应用。

本指南将重点介绍内部存储。请参阅指南[外部存储](#)有关 Xamarin.Android 应用程序中使用外部存储的详细信息。

使用内部存储

应用程序的内部存储目录由操作系统, 并且向 Android 应用程序的公开 `Android.Content.Context.FileDir` 属性。这将返回 `Java.IO.File` 表示 Android 有专门的专用于应用程序的目录对象。例如, 包名称的应用 `com.companyname` 可能是内部存储目录:

```
/data/user/0/com.companyname/files
```

本文档将到内部存储的目录, 请参阅[_内部_存储_](#)。

IMPORTANT

到设备和 Android 的不同版本之间的内部存储目录的确切路径可以因设备。因此, 应用必须未硬代码内部文件存储目录的路径并改为使用 Xamarin.Android Api, 如 `System.Environment.GetFolderPath()`。

若要最大化代码共享, Xamarin.Android 应用程序 (或面向 Xamarin.Android 的 Xamarin.Forms 应用程序) 应使用 `System.Environment.GetFolderPath()` 方法。在 Xamarin.Android 中, 此方法将返回与相同的位置的目录的字符串 `Android.Content.Context.FileDir`。此方法采用枚举, `System.Environment.SpecialFolder`, 用于标识一组表示由操作系统使用的特殊文件夹的路径的枚举常量。不是所有 `System.Environment.SpecialFolder` 值将映射到 Xamarin.Android 上的有效目录。下表描述了有关在给定的值的预期路径是什么 `System.Environment.SpecialFolder`:

| SYSTEM.ENVIRONMENT.SPECIALFOLDER | 路径 |
|-----------------------------------|---------------------------|
| <code>ApplicationData</code> | <i>内部_存储/.config</i> |
| <code>Desktop</code> | <i>内部_存储 /桌面</i> |
| <code>LocalApplicationData</code> | <i>内部_存储/.local/share</i> |
| <code>MyComputer</code> | <i>内部_存储/.local/share</i> |
| <code>MyDocuments</code> | <i>内部_存储</i> |
| <code>MyMusic</code> | <i>内部_存储/Music</i> |
| <code>MyPictures</code> | <i>内部_存储/Music</i> |
| <code>MyVideos</code> | <i>内部_存储/Videos</i> |
| <code>Personal</code> | <i>内部_存储</i> |

读取或写入到文件存储在内部存储

任一 [C# Api](#), 可用于编写到一个文件是不够的; 只需是获取分配给应用程序的目录中的文件的路径。强烈建议具有文件访问权限阻止主线程将关联的异步版本的.NET Api 用于最小化可能是任何问题。

此代码片段是一个整数写入 utf-8 文本文件的内部存储目录的应用程序的一个示例：

```
public async Task SaveCountAsync(int count)
{
    var backingFile =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal), "count.txt");
    using (var writer = File.CreateText(backingFile))
    {
        await writer.WriteLineAsync(count.ToString());
    }
}
```

下一步的代码片段提供了一种方法来读取存储在文本文件中的一个整数值：

```
public async Task<int> ReadCountAsync()
{
    var backingFile =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal), "count.txt");

    if (backingFile == null || !File.Exists(backingFile))
    {
        return 0;
    }

    var count = 0;
    using (var reader = new StreamReader(backingFile, true))
    {
        string line;
        while ((line = await reader.ReadLineAsync()) != null)
        {
            if (int.TryParse(line, out var newcount))
            {
                count = newcount;
            }
        }
    }

    return count;
}
```

使用 Xamarin.Essentials—文件系统帮助程序

[Xamarin.Essentials](#)是一套 Api 用于编写跨平台兼容的代码。[文件系统帮助程序](#)是一个类, 包含一系列的帮助程序来简化查找应用程序的缓存和数据目录。此代码片段提供了如何查找应用的内部存储目录和缓存目录的示例：

```
// Get the path to a file on internal storage
var backingFile = Path.Combine(Xamarin.Essentials.FileSystem.AppDataDirectory, "count.txt");

// Get the path to a file in the cache directory
var cacheFile = Path.Combine(Xamarin.Essentials.FileSystem.CacheDirectory, "count.txt");
```

隐藏文件 MediaStore

MediaStore Android 设备上收集有关媒体文件（视频、音乐、图像）的元数据的 Android 组件。其目的是简化设备上的所有 Android 应用之间共享这些文件。

专用文件都不会显示为可共享的媒体中。例如，如果应用程序将图片保存到其专用的外部存储，然后该文件将不选取媒体扫描程序（MediaStore）。

将提取公共文件通过 `MediaStore`。具有零字节文件名称的目录。**NOMEDIA**不会通过扫描 `MediaStore`。

相关链接

- [外部存储](#)
- [保存文件存储在设备存储](#)
- [Xamarin.Essentials 文件系统帮助程序](#)
- [使用自动备份的备份用户数据](#)
- [采纳存储](#)

外部存储

2018/10/26 • [Edit Online](#)

外部存储是指不在内部存储且无法以独占方式访问到的应用程序负责将文件的文件存储。外部存储的主要用途是提供放置的文件，用于在应用之间共享或太大，无法在内部存储的地方。

从历史上来说，外部存储称为可移动媒体，如 SD 卡上的磁盘分区（也称为_便携式存储_）。这一区别不再的结果已演变成将 Android 设备和 Android 设备不再支持可移动存储。而某些设备将分配一些其内部的非易失性内存执行相同的函数可移动媒体的 Android。这称为_仿真_存储，并仍被视为是外部存储。或者，某些 Android 设备可能有多个外部存储分区。例如，在 Android 平板电脑（除了其内部存储）可能会模拟存储和一个或多个槽的 SD 卡。所有这些分区被视为 android 外部存储。

具有多个用户的设备，每个用户将能够在其外部存储主外部存储分区上的专用的目录。以某一用户运行的应用将无权访问文件从设备上的另一个用户。为所有用户文件是仍世界上可读和世界可写;但是，Android 将沙盒与其他每个用户配置文件。

读取和写入文件，在 Xamarin.Android 中几乎完全相同，因为它是与任何其他.NET 应用程序。Xamarin.Android 应用程序确定将进行操作，然后使用标准.NET 习惯用语文件访问的文件的路径。由于内部和外部存储的实际路径可能不同设备的或从到 Android 版本的 Android 版本，建议不要进行硬编码文件的路径。Xamarin.Android 公开的本机 Android Api，可帮助确定文件存储在内部和外部存储的路径。

本指南将讨论的概念和特定于外部存储在 Android 中的 Api。

公钥和私钥文件存储在外部存储

有两种不同类型的应用程序可能将放在外部存储的文件：

- **私有文件**—专用文件是特定于应用程序（但仍完全公开和全局可写）的文件。Android 需要以专用文件存储在外部存储上的特定目录。即使文件被称为“专用”，它们仍可见且可由其他应用在设备上访问，它们不会提供任何特殊保护 Android。
- **公共文件**—这些是不被视为是特定于应用程序，旨在自由地共享的文件。

这些文件之间的区别是主要概念。专用文件是专用的意义上说，它们被视为公共文件时存在于外部存储的任何其他文件是应用程序的一部分。Android 提供两个不同的 Api 来解析到私有和公共文件的路径，但否则使用相同的.NET Api 来读取和写入这些文件。这些是相同的 Api，上节中讨论[读取和写入](#)。

专用的外部文件

专用的外部文件被视为特定于应用程序（类似于内部文件），但保持在外部存储中以任意数量的原因（例如正在用于内部存储太大）。类似于内部文件，这些文件将被删除时由用户卸载应用。

通过调用该方法找到的专用外部文件的主要位置 `Android.Content.Context.GetExternalFilesDir(string type)`。此方法将返回 `Java.IO.File` 对象，表示应用程序的专用外部存储目录。传递 `null` 到此方法将返回路径到应用程序的用户存储目录。例如，应用程序的包名称 `com.companyname.app`，是专用的外部文件的“root”目录：

```
/storage/emulated/0/Android/data/com.companyname.app/files/
```

本文档将专用文件存储在为外部存储的存储目录引用_私有_外部_存储_。

为参数 `GetExternalFilesDir()` 是一个字符串，指定_应用程序目录_。这是用于提供文件的逻辑组织的标准位置的目录。字符串值是可通过在常量 `Android.OS.Environment` 类：

| | |
|------------------------|------------------------|
| ANDROID.OS.ENVIRONMENT | 目录 |
| DirectoryAlarms | 私有_外部_存储/警报 |
| DirectoryDcim | 私有_外部_存储/DCIM |
| DirectoryDownloads | 私有_外部_存储/下载 |
| DirectoryDocuments | 私有_外部_存储/文档 |
| DirectoryMovies | 私有_外部_存储/Movies |
| DirectoryMusic | 私有_外部_存储/Music |
| DirectoryNotifications | 私有_外部_存储/Notifications |
| DirectoryPodcasts | 私有_外部_存储/Podcasts |
| DirectoryRingtones | 私有_外部_存储/Ringtones |
| DirectoryPictures | 私有_外部_存储/图片 |

对于具有多个外部存储分区的设备，每个分区都适用于专用文件的目录。该方法

`Android.Content.Context.GetExternalFileDirs(string type)` 将返回一个数组 `Java.IO.File`。每个对象将表示专用的特定于应用程序目录在所有应用程序可以在其中放置文件的共享/外部存储设备上它所拥有。

IMPORTANT

到设备和 Android 的不同版本之间的专用 external 存储目录的确切路径可以因设备。正因为如此，应用必须未硬编码到该目录，路径并改为使用 Xamarin.Android Api，如 `Android.Content.Context.GetExternalFilesDir()`。

公共外部文件

公共文件是存在于外部存储不在 Android 分配的专用文件的目录中存储的文件。卸载应用程序时，将不会删除公共文件。Android 应用程序之前，必须授予权限他们可以读取或写入任何公共文件。很可能存在于外部存储的任何位置的公共文件，但按照惯例 Android 要求要由属性标识在目录中存在的公共文件

`Android.OS.Environment.ExternalStorageDirectory`。此属性将返回 `Java.IO.File` 表示主外部存储目录的对象。例如，`Android.OS.Environment.ExternalStorageDirectory` 可能指以下目录：

```
/storage/emulated/0/
```

本文档将引用公共文件存储在为外部存储的存储目录_公共_外部_存储_。

Android 上还支持的目录_公共_外部_存储_。这些目录是完全相同的目录_应用程序_

`_PRIVATE_EXTERNAL_STORAGE_` 和上一节中表所述。该方法

`Android.OS.Environment.GetExternalStoragePublicDirectory(string directoryType)` 将返回 `Java.IO.File` 公共应用程序目录所对应的对象。`directoryType` 参数是必需的参数，且不能为 `null`。

例如，调用 `Environment.GetExternalStoragePublicDirectory(Environment.DirectoryDocuments).AbsolutePath` 将返回一个字符串，它将类似于：

```
/storage/emulated/0/Documents
```

IMPORTANT

到设备和 Android 的不同版本之间的公共外部存储目录的确切路径可以因设备。正因为如此，应用必须未硬编码到该目录，路径并改为使用 Xamarin.Android Api, 如 `Android.OS.Environment.ExternalStorageDirectory`。

使用外部存储

后一个 Xamarin.Android 应用程序已获取到文件的完整路径，它应使用任何标准的.NET Api, 用于创建、读取、写入或删除文件。这可最大化跨平台兼容应用程序代码的量。但是，在尝试访问的文件之前 Xamarin.Android 应用程序必须确保这是它可以访问该文件。

1. 验证外部存储–具体取决于外部存储的性质，很可能不可能装入和可用的应用程序。所有应用程序应尝试使用它之前检查外部存储的状态。
2. 执行运行时权限检查 – Android 应用程序必须以访问外部存储从用户请求权限。这意味着，如果运行的时间应在任何文件访问权限之前执行权限请求。本指南[权限在 Xamarin.Android](#)包含 Android 权限的详细信息。

这两项任务的每个将如下所述。

验证外部存储可用

写入到外部存储之前，第一步是检查它可读或可写。`Android.OS.Environment.ExternalStorageState` 属性包含一个字符串，标识外部存储的状态。此属性将返回一个字符串，表示状态。此表是一系列 `ExternalStorageState` 可能返回的值 `Environment.ExternalStorageState`：

| EXTERNALSTORAGESTATE | 描述 |
|----------------------|--------------------------------|
| MediaBadRemoval | 媒体已突然删除不正确所造成。 |
| MediaChecking | 媒体存在，但正在执行磁盘检查。 |
| MediaEjecting | 媒体正在被卸载，弹出。 |
| MediaMounted | 媒体已装载，可读取或写入。 |
| MediaMountedReadOnly | 媒体已装载，但只能从读取。 |
| MediaNofs | 媒体存在，但不包含适用于 Android 的文件系统。 |
| MediaRemoved | 没有存在的媒体。 |
| MediaShared | 媒体存在，但未装入。它是正在通过 USB 与共享另一台设备。 |
| MediaUnknown | 媒体的状态不受 Android。 |
| MediaUnmountable | 媒体存在，但不能由 Android 装载。 |
| MediaUnmounted | 媒体存在但未装入。 |

大多数 Android 应用只需检查是否已装入外部存储。以下代码片段演示如何验证外部存储已装入的只读访问权限或读写访问权限：

```
bool isReadonly = Environment.MediaMountedReadOnly.Equals(Environment.ExternalStorageState);
bool isWriteable = Environment.MediaMounted.Equals(Environment.ExternalStorageState);
```

外部存储权限

Android 会考虑访问外部存储要_非常危险的权限_, 这通常要求用户授予其访问资源的权限。用户可以撤销此权限在任何时间。这意味着, 如果运行的时间应在任何文件访问权限之前执行权限请求。应用会自动授予权限以读取和写入其自身的专用文件。应用读取和写入后属于其他应用程序的专用文件可能[授予的权限](#)用户。

所有 Android 应用必须声明外部存储中的两个权限之一 **AndroidManifest.xml**。若要标识的权限, 以下两种状态之一 `uses-permission` 必须将元素添加到**AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

NOTE

如果用户授予 `WRITE_EXTERNAL_STORAGE`, 然后 `READ_EXTERNAL_STORAGE` 也是隐式授予。不需要请求中的这两个权限**AndroidManifest.xml**。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

此外可以使用添加的权限**Android** 清单选项卡解决方案属性:

LocalFiles.Native

Application

Android Manifest

Android Options

Android Package Signing

Build

Build Events

Reference Paths

Configuration: N/A Platform: N/A

Application name:

@string/app_name

Package name:

com.xamarin.android.samples.localfiles

Application icon:

@drawable/icon

Application theme:

@style/MainTheme

Version number:

3

Version name:

2

Install location:

Prefer Internal

Minimum Android version:

Android 6.0 (API Level 23 - Marshmallow)

Target Android version:

Android 8.1 (API Level 27 - Oreo)

Required permissions:

☐ WRITE_APN_SETTINGS

☐ WRITE_CALENDAR

☐ WRITE_CALL_LOG

☐ WRITE_CONTACTS

☒ WRITE_EXTERNAL_STORAGE

☐ WRITE_GSERVICES

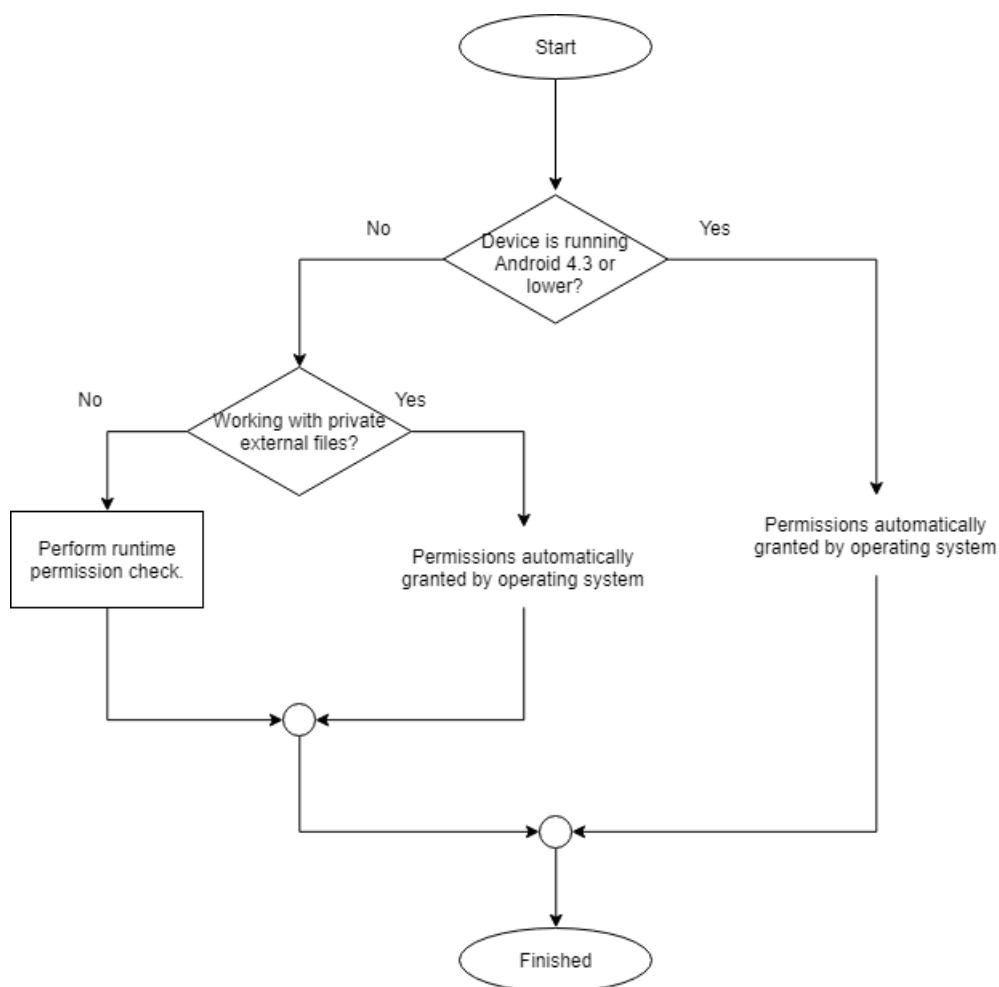
☐ WRITE_HISTORY_BOOKMARKS

☐ WRITE_PROFILE

☐ WRITE_SECURE_SETTINGS

☐ WRITE_SETTINGS

通常情况下，用户必须批准所有危险的权限。外部存储的权限，因为此规则，具体取决于应用程序运行版本的例外情况是 android 的异常：



有关执行运行时权限请求的详细信息，请参阅指南[权限在 Xamarin.Android](#)。**Monodroid** 示例 [LocalFiles](#)还演示了一种方法的执行运行时权限检查。

授予和撤销权限与 ADB

在过程中开发 Android 应用程序，它可能需要授权或撤销权限来测试各种工作流所涉及的运行时权限检查。就可以在使用 ADB 命令提示符下执行此操作。以下命令行代码片段演示了如何授予或撤销权限的 Android 应用包名称是使用 ADB **com.companyname.app**:

```
$ adb shell pm grant com.companyname.app android.permission.WRITE_EXTERNAL_STORAGE

$ adb shell pm revoke com.companyname.app android.permission.WRITE_EXTERNAL_STORAGE
```

正在删除文件

任何可以使用 C# Api 从外部存储，如删除文件的标准 `System.IO.File.Delete`。还有可能要使用 Java Api，但要牺牲代码可移植性。例如：

```
System.IO.File.Delete("/storage/emulated/0/Android/data/com.companyname.app/files/count.txt");
```

相关链接

- [Xamarin.Android 本地文件示例上monodroid 示例](#)
- [在 Xamarin.Android 中的权限](#)

指纹身份验证

2018/10/26 • [Edit Online](#)

本指南介绍如何将添加到 *Xamarin.Android* 应用程序在 *Android 6.0* 中引入的指纹身份验证。

指纹身份验证概述

在 *Android* 设备上的指纹扫描程序到达提供与传统用户名/密码方法的替代方法的用户身份验证的应用程序。使用指纹对用户进行身份验证使应用程序合并少受侵入比用户名和密码的安全。

`FingerprintManager` Api 使用指纹扫描程序的目标设备并运行 API 级别 23 (*Android 6.0*) 或更高版本。这些 Api 位于 `Android.Hardware.Fingerprints` 命名空间。*Android* 支持库 v4 提供指纹针对较旧版本的 *Android* Api 的版本。兼容性 Api 中找到 `Android.Support.v4.Hardware.Fingerprint` 命名空间中, 通过分发 [Xamarin.Android.Support.v4 NuGet 包](#)。

`FingerprintManager` (和其支持库对应 `FingerprintManagerCompat`) 是使用指纹扫描功能的硬件的主类。此类是围绕管理与硬件本身之间的交互的系统级别服务的 *Android* SDK 包装器。它负责启动指纹扫描程序并响应来自扫描程序的反馈。此类具有一个只有三个成员具有界面, 非常简单:

- `Authenticate` – 此方法将初始化硬件扫描程序, 并等待用户扫描其指纹在后台启动服务。
- `EnrolledFingerprints` – 此属性将返回 `true` 如果用户已注册设备的一个或多个指纹。
- `HardwareDetected` – 此属性用于确定设备是否支持指纹扫描。

`FingerprintManager.Authenticate` *Android* 应用程序使用方法启动指纹扫描程序。以下代码片段示范了如何使用支持库兼容性 Api 调用它:

```
// context is any Android.Content.Context instance, typically the Activity
FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.From(context);
fingerprintManager.Authenticate(FingerprintManager.CryptoObject crypto,
                                int flags,
                                CancellationSignal cancel,
                                FingerprintManagerCompat.AuthenticationCallback callback,
                                Handler handler
                                );
```

本指南介绍了如何使用 `FingerprintManager` Api 来增强具有指纹身份验证的 *Android* 应用程序。它将介绍如何实例化并创建 `CryptoObject` 来帮助保护指纹扫描程序的结果。我们将介绍如何应用程序应子类 `FingerprintManager.AuthenticationCallback` 并响应来自指纹扫描程序的反馈。最后, 我们将了解如何注册 *Android* 设备或仿真程序上的指纹以及如何使用 `adb` 来模拟指纹扫描。

要求

指纹身份验证需要 *Android 6.0* (API 级别 23) 或更高版本和设备与指纹扫描程序。

指纹必须已注册该设备的每个用户都是要进行身份验证。这涉及到使用密码、PIN、轻扫模式或面部识别屏幕锁定设置。就可以模拟某些 *Android* 仿真程序中的指纹身份验证功能。有关这两个主题的详细信息, 请参阅[注册指纹](#)部分。

相关链接

- [指纹指南示例应用](#)
- [指纹对话框示例](#)

- [在运行时的请求权限](#)
- [android.hardware.fingerprint](#)
- [android.support.v4.hardware.fingerprint](#)
- [Android.Content.Context](#)
- [指纹和付款 API \(视频\)](#)

开始使用指纹身份验证

2018/10/26 • [Edit Online](#)

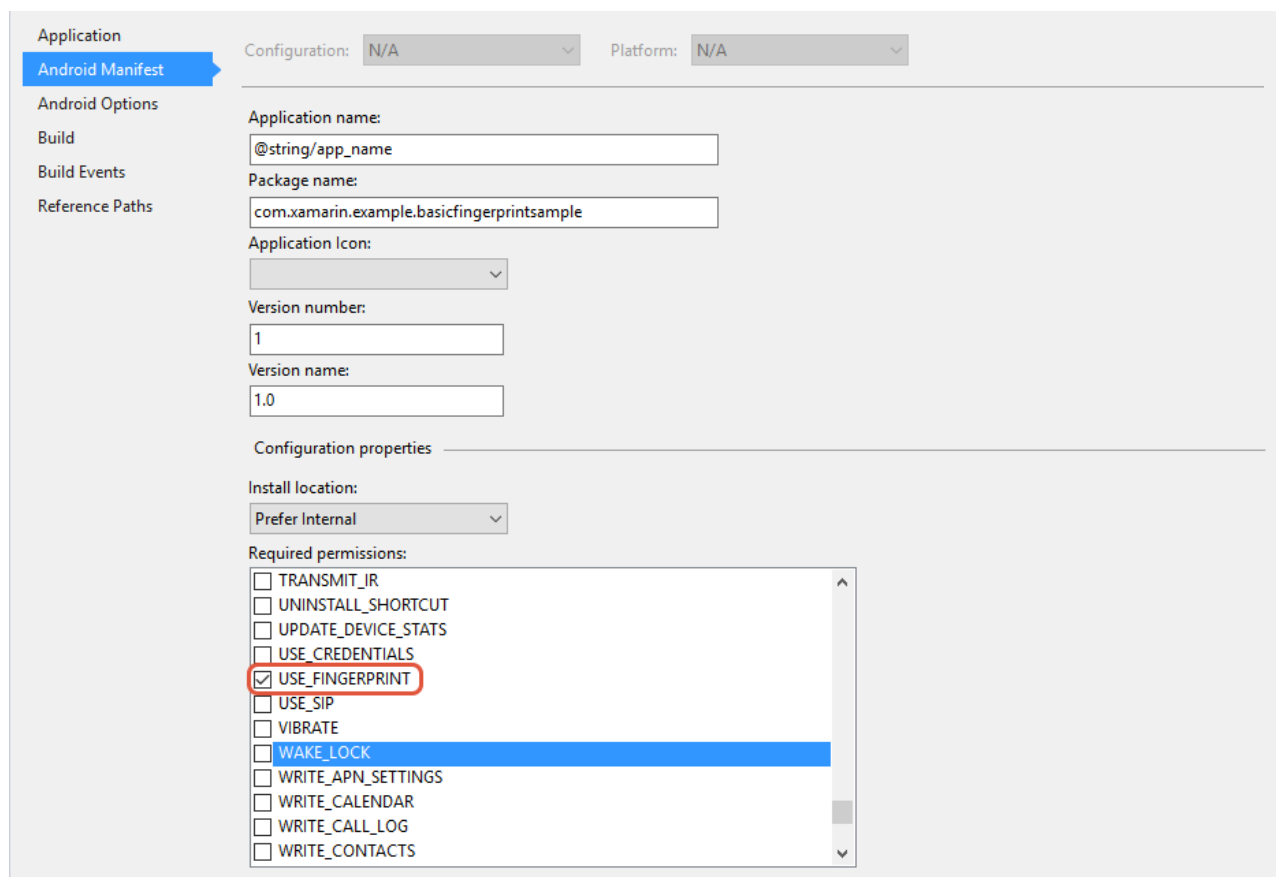
若要开始，让我们先了解如何配置 Xamarin.Android 项目，以便应用程序都可以使用指纹身份验证：

1. 更新**AndroidManifest.xml**声明指纹 Api 需要的权限。
2. 获取对引用 `FingerprintManager` 。
3. 检查设备能够指纹扫描。

在应用程序中的请求权限清单

- [Visual Studio](#)
- [Visual Studio for Mac](#)

Android 应用程序必须请求 `USE_FINGERPRINT` 清单中的权限。下面的屏幕截图显示了如何将此权限添加到 Visual Studio 2015 中的应用程序：



获取 FingerprintManager 的实例

接下来，应用程序必须获取的实例 `FingerprintManager` 或 `FingerprintManagerCompat` 类。若要使用较旧版本的 Android 兼容，Android 应用程序应使用的兼容性 API 的 Android 支持 v4 NuGet 包中找到。以下代码片段演示了如何从操作系统获取相应的对象：

```
// Using the Android Support Library v4
FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.From(context);

// Using API level 23:
FingerprintManager fingerprintManager = context.getSystemService(Context.FingerprintService) as
FingerprintManager;
```

在上一片段中，`context` 是任何 Android `Android.Content.Context`。这通常是执行身份验证的活动。

检查合格性

应用程序必须执行多个检查以确保其可以使用指纹身份验证。总共有五个应用程序使用检查有资格的条件：

API 级别 23 – 指纹 Api 需要 API 级别 23 或更高版本。`FingerprintManagerCompat` 类将包装对你的 API 级别检查。出于此原因，它是建议用于 **Android 支持库 v4** 和 `FingerprintManagerCompat`；这将为这些检查的一个帐户。

硬件 – 首次启动应用程序，应检查是否存在指纹扫描程序：

```
FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.From(context);
if (!fingerprintManager.IsHardwareDetected)
{
    // Code omitted
}
```

设备保护 – 用户必须具有与屏幕锁保护的设备。如果用户具有不安全的屏幕锁定的设备和安全性是对应用程序，然后用户应收到通知，必须配置屏幕锁定。下面的代码段演示了如何检查此先决条件：

```
KeyguardManager keyguardManager = (KeyguardManager) GetSystemService(KeyguardService);
if (!keyguardManager.IsKeyguardSecure)
{
}
```

注册指纹 – 用户必须具有至少一个操作系统系统中注册的指纹。在每次身份验证尝试之前应发生这种权限检查：

```
FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.From(context);
if (!fingerprintManager.HasEnrolledFingerprints)
{
    // Can't use fingerprint authentication - notify the user that they need to
    // enroll at least one fingerprint with the device.
}
```

权限 – 应用程序必须使用应用程序之前从用户中请求权限。Android 5.0 及更低，用户授予的权限，以安装该应用程序的状态。Android 6.0 引入了新的权限模型，用于检查在运行时权限。此代码片段示范了如何在 Android 6.0 上的权限检查：

```
// The context is typically a reference to the current activity.
Android.Content.PM.Permission permissionResult = ContextCompat.CheckSelfPermission(context,
Manifest.Permission.UseFingerprint);
if (permissionResult == Android.Content.PM.Permission.Granted)
{
    // Permission granted - go ahead and start the fingerprint scanner.
}
else
{
    // No permission. Go and ask for permissions and don't start the scanner. See
    // http://developer.android.com/training/permissions/requesting.html
}
```

检查所有这些条件，每次应用程序提供了身份验证选项可确保用户获取最佳用户体验。更改或对其设备或操作系统升级可能会影响指纹身份验证的可用性。如果您选择要缓存的任何这些检查的结果，请确保满足客户的升级方案。

有关如何在 Android 6.0 中的请求权限的详细信息，请参阅 Android 指南[在运行时请求权限](#)。

相关链接

- [上下文](#)
- [ContextCompat](#)
- [KeyguardManager](#)
- [FingerprintManager](#)
- [FingerprintManagerCompat](#)
- [在运行时的请求权限](#)

扫描指纹

2018/10/26 • [Edit Online](#)

既然我们已了解如何准备 Xamarin.Android 应用程序使用指纹身份验证，让我们返回到

`FingerprintManager.Authenticate` 方法，并讨论其位置在 Android 6.0 指纹身份验证。指纹身份验证的工作流的快速概述此列表所述：

1. 调用 `FingerprintManager.Authenticate`，并传入 `CryptoObject` 和一个 `FingerprintManager.AuthenticationCallback` 实例。`CryptoObject` 用于确保指纹身份验证结果被篡改。
2. 子类 `FingerprintManager.AuthenticationCallback` 类。此类的实例将提供给 `FingerprintManager` 时指纹身份验证入门。指纹扫描程序完成后，它将调用其中一个此类上的回调方法。
3. 编写代码以更新 UI，以让用户知道设备指纹扫描程序启动并正在等待用户交互。
4. Android 指纹扫描程序完成操作后，将通过在调用的方法向应用程序返回结果 `FingerprintManager.AuthenticationCallback` 在上一步中提供的实例。
5. 应用程序将通知指纹身份验证结果的用户，并将结果作为相应做出反应。

下面的代码段是将启动扫描指纹的活动中方法的示例：

```
protected void FingerPrintAuthenticationExample()
{
    const int flags = 0; /* always zero (0) */

    // CryptoObjectHelper is described in the previous section.
    CryptoObjectHelper cryptoHelper = new CryptoObjectHelper();

    // cancellationSignal can be used to manually stop the fingerprint scanner.
    cancellationSignal = new Android.Support.V4.OS.CancellationSignal();

    FingerprintManagerCompat fingerPrintManager = FingerprintManagerCompat.From(this);

    // AuthenticationCallback is a base class that will be covered later on in this guide.
    FingerprintManagerCompat.AuthenticationCallback authenticationCallback = new MyAuthCallbackSample(this);

    // Start the fingerprint scanner.
    fingerPrintManager.Authenticate(cryptoHelper.BuildCryptoObject(), flags, cancellationSignal,
    authenticationCallback, null);
}
```

让我们来讨论这些参数中的每个 `Authenticate` 中更多详细信息的方法：

- 第一个参数是 `_crypto_` 对象指纹扫描程序将用来帮助进行身份验证的指纹扫描结果。此对象可以为 `null`，盲目地信任不出现任何应用程序必须在这种情况下已篡改指纹结果。建议 `CryptoObject` 实例化并提供给 `FingerprintManager` 而不是 `null`。创建 `CryptoObject` 将详细说明如何实例化 `CryptoObject` 基于 `Cipher`。
- 第二个参数始终为零。Android 文档表明这是组标志，最有可能保留供将来使用。
- 第三个参数， `cancellationSignal` 是用于关闭指纹扫描程序，并取消当前请求的对象。这是 [Android CancellationSignal](#)，并不是从 .NET framework 类型。
- 第四个参数是必需的是一个类，该子类 `AuthenticationCallback` 抽象类。将调用此类的方法将信号发送到客户端时 `FingerprintManager` 已完成，结果将是什么。如有很多关于实现需要了解 `AuthenticationCallback`，将在介绍它是自己的部分。
- 第五个参数是可选的 `Handler` 实例。如果 `Handler` 提供对象，则 `FingerprintManager` 将使用 `Looper` 从该对象处理从指纹硬件消息时。通常情况下，一个不需要提供 `Handler`，将使用 `FingerprintManager Looper` 从应用程序。

正在取消指纹扫描

可能有必要的用户（或应用程序）来取消指纹扫描后已启动。在这种情况下，将调用 `IsCancelled` 方法 `CancellationSignal` 提供给 `FingerprintManager.Authenticate` 时调用它以启动指纹扫描。

现在，我们看到 `Authenticate` 方法，让我们看一些更多详细信息中的更重要的参数。首先，我们将介绍[响应身份验证回调](#)，其中将讨论如何创建子类 `FingerprintManager.AuthenticationCallback`，启用对做出反应的 Android 应用程序提供指纹扫描程序的结果。

相关链接

- [CancellationSignal](#)
- [FingerprintManager.AuthenticationCallback](#)
- [FingerprintManager.CryptoObject](#)
- [FingerprintManagerCompat.CryptoObject](#)
- [FingerprintManager](#)
- [FingerprintManagerCompat](#)

创建 CryptoObject

2018/10/26 • [Edit Online](#)

指纹身份验证结果的完整性非常重要的应用程序-它是在应用程序如何知道用户的标识。它是理论上的第三方恶意软件以截获和篡改指纹扫描程序返回的结果。本部分将讨论一种方法来保留的指纹结果的有效性。

`FingerprintManager.CryptoObject` 是 Java 加密 Api 的包装, 并由 `FingerprintManager` 以保护身份验证请求的完整性。通常情况下, `Javax.Crypto.Cipher` 对象是用于加密指纹扫描程序的结果的机制。`Cipher` 对象本身会使用通过使用 Android 密钥存储 Api 的应用程序创建的密钥。

若要了解所有这些类是如何协同工作, 让我们首先看一下下面的代码演示了如何创建 `CryptoObject`, 然后更详细地解释:

```
public class CryptoObjectHelper
{
    // This can be key name you want. Should be unique for the app.
    static readonly string KEY_NAME = "com.xamarin.android.sample.fingerprint_authentication_key";

    // We always use this keystore on Android.
    static readonly string KEYSTORE_NAME = "AndroidKeyStore";

    // Should be no need to change these values.
    static readonly string KEY_ALGORITHM = KeyProperties.KeyAlgorithmAes;
    static readonly string BLOCK_MODE = KeyProperties.BlockModeCbc;
    static readonly string ENCRYPTION_PADDING = KeyProperties.EncryptionPaddingPkcs7;
    static readonly string TRANSFORMATION = KEY_ALGORITHM + "/" +
                                           BLOCK_MODE + "/" +
                                           ENCRYPTION_PADDING;

    readonly KeyStore _keystore;

    public CryptoObjectHelper()
    {
        _keystore = KeyStore.GetInstance(KEYSTORE_NAME);
        _keystore.Load(null);
    }

    public FingerprintManagerCompat.CryptoObject BuildCryptoObject()
    {
        Cipher cipher = CreateCipher();
        return new FingerprintManagerCompat.CryptoObject(cipher);
    }

    Cipher CreateCipher(bool retry = true)
    {
        IKey key = GetKey();
        Cipher cipher = Cipher.GetInstance(TRANSFORMATION);
        try
        {
            cipher.Init(CipherMode.EncryptMode | CipherMode.DecryptMode, key);
        } catch (KeyPermanentlyInvalidatedException e)
        {
            _keystore.DeleteEntry(KEY_NAME);
            if (retry)
            {
                CreateCipher(false);
            } else
            {
                throw new Exception("Could not create the cipher for fingerprint authentication.", e);
            }
        }
    }

    return cipher;
}
```



```

    }

    IKey GetKey()
    {
        IKey secretKey;
        if(!_keystore.IsKeyEntry(KEY_NAME))
        {
            CreateKey();
        }

        secretKey = _keystore.GetKey(KEY_NAME, null);
        return secretKey;
    }

    void CreateKey()
    {
        KeyGenerator keyGen = KeyGenerator.GetInstance(KeyProperties.KeyAlgorithmAes, KEYSTORE_NAME);
        KeyGenParameterSpec keyGenSpec =
            new KeyGenParameterSpec.Builder(KEY_NAME, KeyStorePurpose.Encrypt | KeyStorePurpose.Decrypt)
                .SetBlockModes(BLOCK_MODE)
                .SetEncryptionPaddings(ENCRYPTION_PADDING)
                .SetUserAuthenticationRequired(true)
                .Build();
        keyGen.Init(keyGenSpec);
        keyGen.GenerateKey();
    }
}

```

示例代码将创建一个新 `Cipher` 为每个 `CryptoObject`，使用的应用程序创建的密钥。该密钥由 `KEY_NAME` 已设置在开头的变量 `CryptoObjectHelper` 类。该方法 `GetKey` 将尝试并检索使用 Android 密钥存储 Api 的密钥。如果键不存在，则该方法 `CreateKey` 将创建应用程序的新键。

密码通过调用实例化 `Cipher.GetInstance`，采用_转换_(一个字符串值，该值指示如何进行加密和解密数据的密码)。对调用 `Cipher.Init` 将完成的密码初始化通过提供从应用程序密码。

请务必认识到有某些情况下，Android 其中可能会使该密钥：

- 新的具有指纹具有已注册设备。
- 没有已注册设备的指纹。
- 用户已禁用屏幕锁定。
- 屏幕锁定 (screenlock 或使用的 PIN/模式的类型)，用户已更改。

在此情况下，`Cipher.Init` 将引发 `KeyPermanentlyInvalidatedException`。上面的示例代码将捕获该异常、删除密钥，然后创建一个新。

下一节将讨论如何创建密钥，并将其存储在设备上。

创建机密密钥

`CryptoObjectHelper` 类使用 Android `KeyGenerator` 创建密钥并将其存储在设备上。`KeyGenerator` 类可以创建密钥，但需要一些有关要创建的密钥类型的元数据。此信息由实例提供 `KeyGenParameterSpec` 类。

一个 `KeyGenerator` 实例化时使用 `GetInstance` 工厂方法。示例代码使用 [高级加密标准](#) (AES) 作为加密算法。AES 会将数据分解为固定大小的块并对每个这些块加密。

下一步，`KeyGenParameterSpec` 使用创建 `KeyGenParameterSpec.Builder`。`KeyGenParameterSpec.Builder` 包装有关是要创建的密钥的以下信息：

- 密钥名称。
- 密钥必须同时用于加密和解密。

- 在示例代码 `BLOCK_MODE` 设置为 `密码块链_` (`KeyProperties.BlockModeCbc`)，这意味着每个块与上一个块（创建每个块之间的依赖关系）位移。
- `CryptoObjectHelper` 使用 [公共密钥加密标准 #7](#) (PKCS7) 来生成出的块，以确保它们是所有相同的大小将填充的字节数。
- `SetUserAuthenticationRequired(true)` 意味着该用户身份验证是必需的然后才能使用该密钥。

一次 `KeyGenParameterSpec` 是创建的它用于初始化 `KeyGenerator`，以生成密钥并安全地将其存储在设备上。

使用 CryptoObjectHelper

现在，示例代码已封装创建的逻辑的大部分 `CryptoWrapper` 成 `CryptoObjectHelper` 类，让我们重新审视该代码从一开
始本指南并使用 `CryptoObjectHelper` 创建密码并启动指纹扫描程序：

```
protected void FingerPrintAuthenticationExample()
{
    const int flags = 0; /* always zero (0) */

    CryptoObjectHelper cryptoHelper = new CryptoObjectHelper();
    cancellationSignal = new Android.Support.V4.OS.CancellationSignal();

    // Using the Support Library classes for maximum reach
    FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.From(this);

    // AuthCallbacks is a C# class defined elsewhere in code.
    FingerprintManagerCompat.AuthenticationCallback authenticationCallback = new MyAuthCallbackSample(this);

    // Here is where the CryptoObjectHelper builds the CryptoObject.
    fingerprintManager.Authenticate(cryptoHelper.BuildCryptoObject(), flags, cancellationSignal,
    authenticationCallback, null);
}
```

既然我们已了解如何创建 `CryptoObject`，可让看看如何 `FingerprintManager.AuthenticationCallbacks` 用于将指纹扫描程序服务的结果传输到 Android 应用程序。

相关链接

- [密码](#)
- [FingerprintManager.CryptoObject](#)
- [FingerprintManagerCompat.CryptoObject](#)
- [KeyGenerator](#)
- [KeyGenParameterSpec](#)
- [KeyGenParameterSpec.Builder](#)
- [KeyPermanentlyInvalidatedException](#)
- [KeyProperties](#)
- [AES](#)
- [RFC 2315-PCKS #7](#)

响应身份验证回调

2018/10/26 • [Edit Online](#)

指纹扫描程序在后台运行在它自己的线程上和完成后，它将通过调用的一种方法报告的扫描结果

`FingerprintManager.AuthenticationCallback` UI 线程上。Android 应用程序必须提供自己的处理程序扩展了此实现以下所有方法的抽象类：

- `OnAuthenticationError(int errorCode, ICharSequence errString)` – 不可恢复的错误时调用。没有任何应用程序或用户可以如何更正这种情况只可能是重试。
- `OnAuthenticationFailed()` – 已检测到但无法识别该设备的指纹时，将调用此方法。
- `OnAuthenticationHelp(int helpMsgId, ICharSequence helpString)` – 可恢复的错误，如通过扫描程序快速实现到手指时调用。
- `OnAuthenticationSucceeded(FingerprintManagerCompat.AuthenticationResult result)` – 这称为时已识别指纹。

如果 `CryptoObject` 调用时使用 `Authenticate`，建议调用 `Cipher.DoFinal` 中 `OnAuthenticationSuccessful`。 `DoFinal` 如果密码已被篡改或未正确初始化，将引发异常，这表明，指纹扫描程序的结果可能已被篡改应用程序之外。

NOTE

建议保留回调类相对轻量 and 免费的应用程序特定逻辑。回调应充当“流量 cop”Android 应用程序和结果之间指纹扫描程序。

示例身份验证回调处理程序

下面的类是举例说明最小 `FingerprintManager.AuthenticationCallback` 实现：

```

class MyAuthCallbackSample : FingerprintManagerCompat.AuthenticationCallback
{
    // Can be any byte array, keep unique to application.
    static readonly byte[] SECRET_BYTES = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    // The TAG can be any string, this one is for demonstration.
    static readonly string TAG = "X:" + typeof (SimpleAuthCallbacks).Name;

    public MyAuthCallbackSample()
    {
    }

    public override void OnAuthenticationSucceeded(FingerprintManagerCompat.AuthenticationResult result)
    {
        if (result.CryptoObject.Cipher != null)
        {
            try
            {
                // Calling DoFinal on the Cipher ensures that the encryption worked.
                byte[] doFinalResult = result.CryptoObject.Cipher.DoFinal(SECRET_BYTES);

                // No errors occurred, trust the results.
            }
            catch (BadPaddingException bpe)
            {
                // Can't really trust the results.
                Log.Error(TAG, "Failed to encrypt the data with the generated key." + bpe);
            }
            catch (IllegalBlockSizeException ibse)
            {
                // Can't really trust the results.
                Log.Error(TAG, "Failed to encrypt the data with the generated key." + ibse);
            }
        }
        else
        {
            // No cipher used, assume that everything went well and trust the results.
        }
    }

    public override void OnAuthenticationError(int errMsgId, ICharSequence errString)
    {
        // Report the error to the user. Note that if the user canceled the scan,
        // this method will be called and the errMsgId will be FingerprintState.ErrorCanceled.
    }

    public override void OnAuthenticationFailed()
    {
        // Tell the user that the fingerprint was not recognized.
    }

    public override void OnAuthenticationHelp(int helpMsgId, ICharSequence helpString)
    {
        // Notify the user that the scan failed and display the provided hint.
    }
}

```

`OnAuthenticationSucceeded` 检查以查看是否 `Cipher` 提供给 `FingerprintManager` 时 `Authentication` 调用时。如果是这样，`DoFinal` 密码上调用方法。这会关闭 `Cipher`，将其还原到其原始状态。如果出现问题具有密码，然后 `DoFinal` 将引发异常，并且应考虑身份验证尝试已失败。

`OnAuthenticationError` 和 `OnAuthenticationHelp` 回调每个接收一个整数，指示出现了什么问题。以下部分介绍每个可能的帮助或错误代码。两个回调具有相似的用途—以通知应用程序的指纹身份验证失败。它们之间的区别是在严重级别。`OnAuthenticationHelp` 用户可恢复错误，如轻扫太快；指纹 `OnAuthenticationError` 是更严重错误，例如损坏的指纹扫描程序。

请注意, `OnAuthenticationError` 通过取消指纹扫描时, 将调用 `CancellationSignal.Cancel()` 消息。 `errMsgId` 参数将具有值为 5 (`FingerprintState.ErrorCanceled`)。根据要求的实现 `AuthenticationCallbacks` 可能不同于其他错误处理这种情况。

`OnAuthenticationFailed` 已成功扫描指纹, 但与任何已注册设备的指纹不匹配时调用。

帮助代码和错误消息 Id

中可能找到的列表和说明的错误代码和帮助代码[Android SDK 文档](#)`FingerprintManager` 类。Xamarin.Android 表示这些值用于 `Android.Hardware.Fingerprints.FingerprintState` 枚举:

- `AcquiredGood` – (值 0) 获取图像的很好。
- `AcquiredImagerDirty` – (值 3) 指纹图像的传感器上的可疑或检测到灰尘由于过多的干扰。例如, 它是合理地在多个后返回此 `AcquiredInsufficient` 或实际检测的传感器 (像素滞留、负责等) 上的灰尘。用户需要采取措施来清理传感器时, 返回此项。
- `AcquiredInsufficient` – (值 2) 指纹图像的过多的干扰处理由于检测到的条件 (即 dry 外观) 或可能是脏的传感器 (请参阅 `AcquiredImagerDirty`)。
- `AcquiredPartial` – (值 1) 检测到仅部分指纹映像。注册期间, 用户应通知上需要发生若要解决此问题, 例如, “用力按传感器。”
- `AcquiredTooFast` – (值为 5) 指纹映像是由于快速动作不完整。尽管主要适用于线性数组传感器, 还会发生此情况期间获取移动手指。应会要求用户移动手指速度较慢 (线性) 或将手指保留更长的传感器上。
- `AcquiredTooSlow` – (值 4) 指纹图像会由于缺乏动作不可读。这是最适合于需要进行轻扫移动的线性数组传感器。
- `ErrorCanceled` – (值为 5) 该操作已取消, 因为指纹传感器是不可用。例如, 这可能会发生时切换用户、设备锁定状态, 或另一个挂起操作会阻止或禁用它。
- `ErrorHwUnavailable` – (值 1) 硬件不可用。请稍后再试。
- `ErrorLockout` – (值 7) 该操作已取消, 因为 API 因尝试次数过多而被锁定。
- `ErrorNoSpace` – (值 4) 为操作, 例如注册; 返回的错误状态无法完成该操作, 因为没有要完成该操作的剩余足够的存储。
- `ErrorTimeout` – (值 3) 当前请求已运行太长时返回的错误状态。这是为了防止程序无限期地等待指纹传感器。超时是平台和特定于传感器的但通常大约 30 秒。
- `ErrorUnableToProcess` – (值 2) 传感器无法处理当前映像时, 返回的错误状态。

相关链接

- [密码](#)
- [AuthenticationCallback](#)
- [AuthenticationCallback](#)

指纹身份验证指南

2018/11/1 • [Edit Online](#)

指纹身份验证指南

现在，我们已了解概念和 Api 围绕 Android 6.0 指纹身份验证，让我们讨论的指纹 Api 使用一些常规建议。

1. **使用 Android 支持库 v4 兼容性 Api**—这将简化应用程序代码通过从代码中删除 API 检查，并允许应用程序以面向最可能的设备。
2. **提供指纹身份验证的替代方法**—指纹身份验证是一种很好、快速的方法，应用程序用户进行身份验证，但是，它不能假定将始终运行或可用。很可能，指纹扫描程序可能会失败、可重用功能区可能变脏，用户可能未配置设备使用指纹身份验证或指纹以来已丢失。还有可能用户可能不希望与您的应用程序使用指纹身份验证。出于这些原因，Android 应用程序应提供用户名和密码等一个备用身份验证过程。
3. **使用 Google 的指纹图标**—所有应用程序应使用 Google 提供的相同指纹图标。使用一个标准图标轻松 Android 用户，以便识别在应用中使用指纹身份验证的位置：



4. **通知用户**—应用程序应显示给用户的通知指纹扫描程序处于活动状态的某种类型和等待触摸或轻扫。

总结

指纹身份验证是允许在 Xamarin.Android 应用程序来快速验证用户，使用户更轻松地使用敏感功能进行交互，如应用内购买的好办法。本指南介绍的概念和合并到 Xamarin.Android 应用程序中的 API 的 Android 6.0 指纹所需的代码。

首先，我们讨论了 API 的本身，指纹 `FingerprintManager` (和 `FingerprintManagerCompat`)。我们探讨了如何将 `FingerprintManager.AuthenticationCallbacks` 必须由应用程序扩展抽象类，并将其用作指纹硬件和应用程序本身之间的中介。然后介绍了如何验证指纹扫描程序结果使用 Java 的完整性 `Cipher` 对象。最后，我们谈及有点测试通过描述如何注册设备上的指纹，并使用 `adb` 来模拟指纹轻扫的仿真程序上。

如果尚未这样做，则应查看[示例应用程序](#)随附本指南。[指纹对话框示例](#)已经通过 Java 移植到 Xamarin.Android 和提供有关如何将指纹身份验证添加到 Android 应用程序的另一个示例。

相关链接

- [指纹指南示例应用](#)
- [指纹对话框示例](#)
- [指纹图标]([https://developer.android.com](https://developer.android.com/drawables/gallery/0-fingerprints.html)
https://developer.xamarin.com/samples/FingerprintDialog/res/drawable-hdpi/ic_fp_40px.html)

注册指纹

2018/10/26 • [Edit Online](#)

注册指纹概述

才有可能利用指纹身份验证，如果已使用指纹身份验证配置该设备的 Android 应用程序。本指南介绍了如何注册 Android 设备或仿真程序上的指纹。仿真程序不具有实际的硬件执行指纹扫描，但可以模拟指纹扫描 Android Debug Bridge（如下所述）的帮助。本指南介绍了如何启用 Android 设备上的屏幕锁定和注册身份验证的指纹。

要求

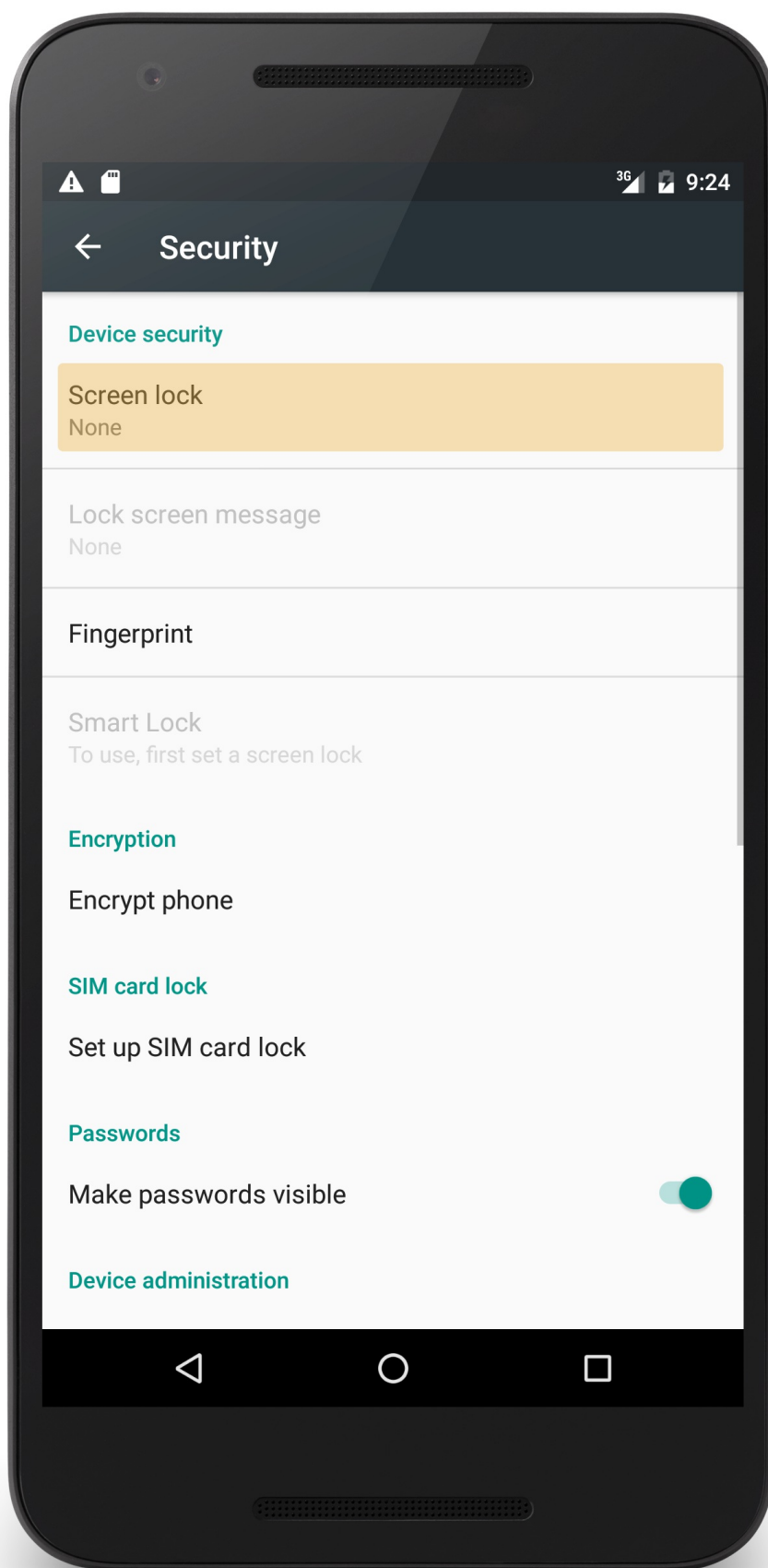
若要注册一个指纹，必须具有 Android 设备或仿真程序运行 API 级别 23 (Android 6.0)。

使用的 Android Debug Bridge (ADB)，都需要熟悉命令提示符处，并 **adb** 可执行文件必须是路径中的 Bash、PowerShell 或命令提示环境。

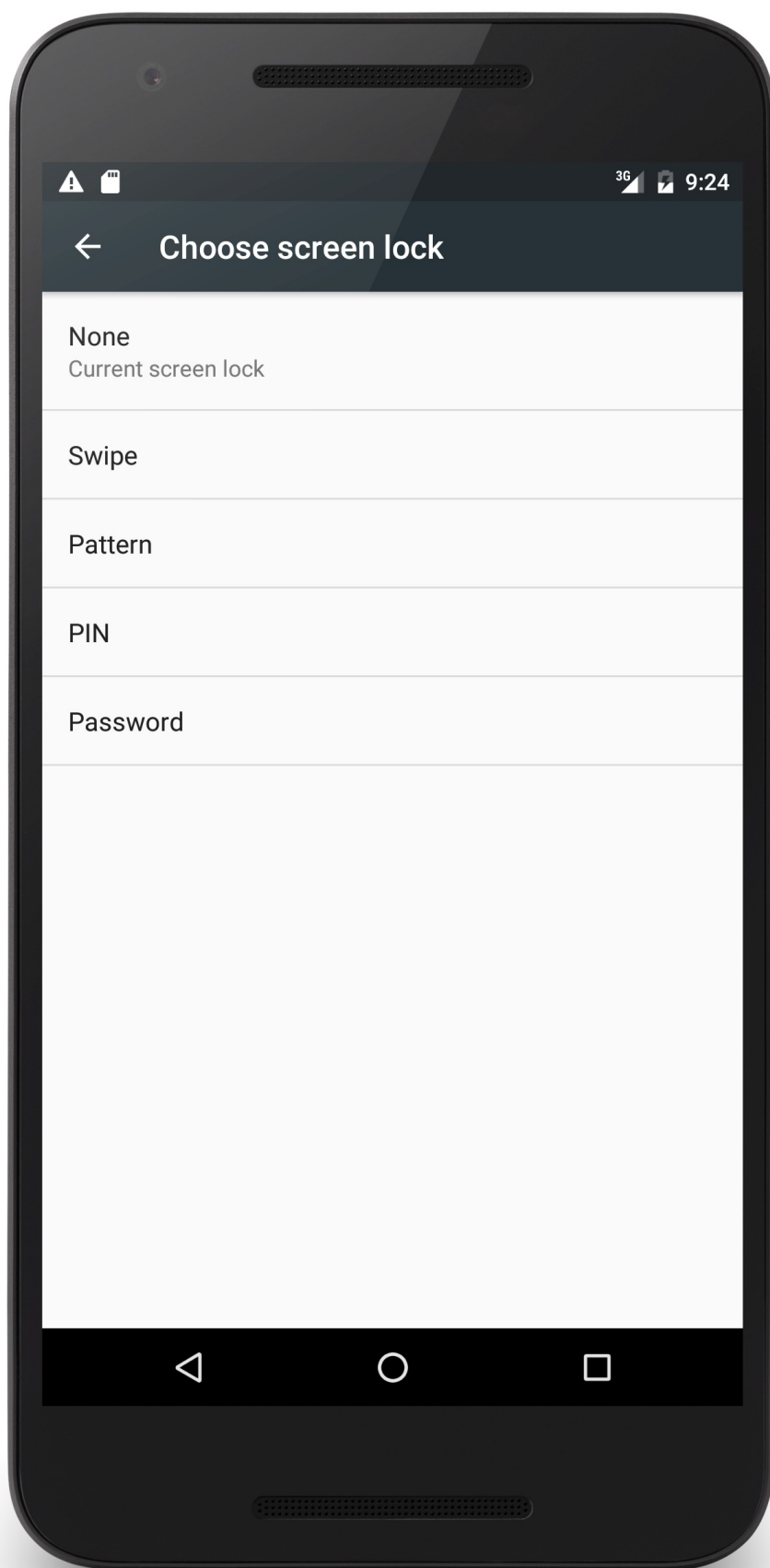
配置屏幕锁定和注册指纹

若要设置屏幕锁定，请执行以下步骤：

1. 转到 **设置 > 安全**，然后选择 **屏幕锁定**：

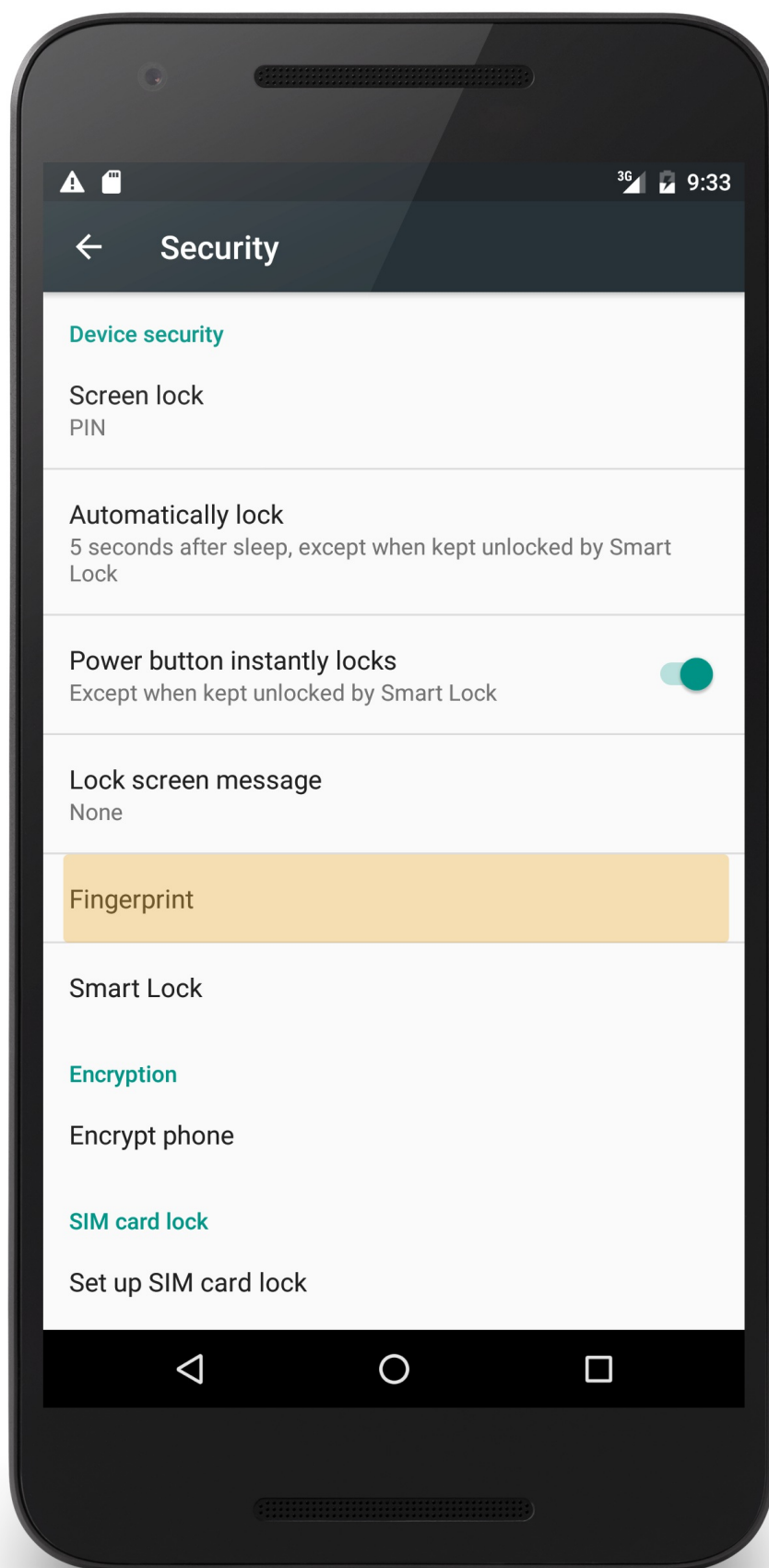


2. 将显示下一个屏幕将允许选择并配置屏幕锁定安全方法之一：

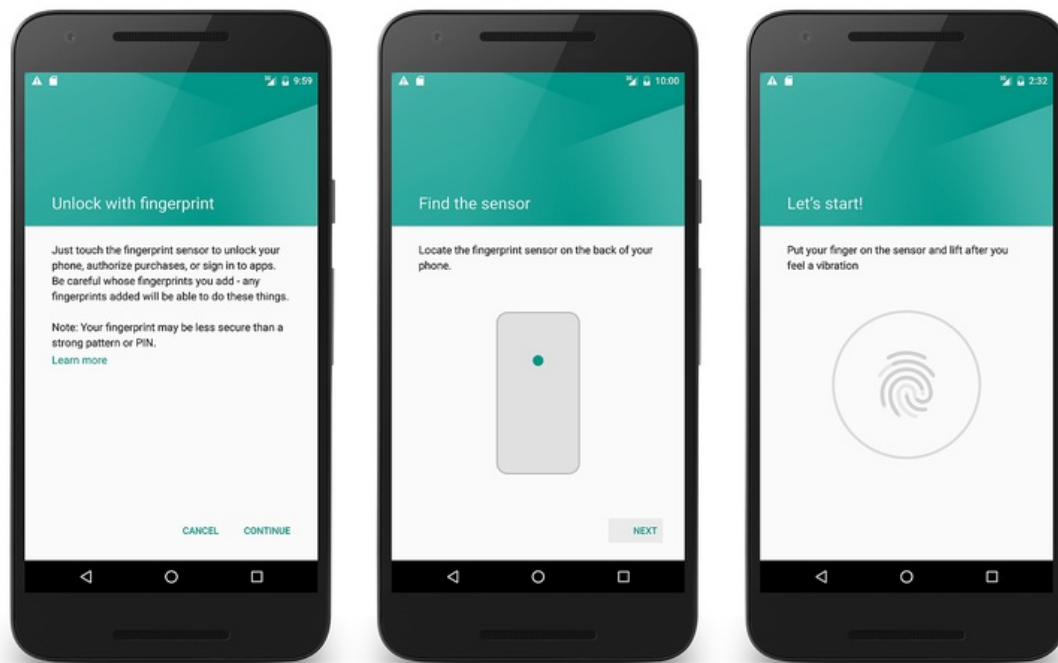


选择并完成其中一个可用的屏幕锁定方法。

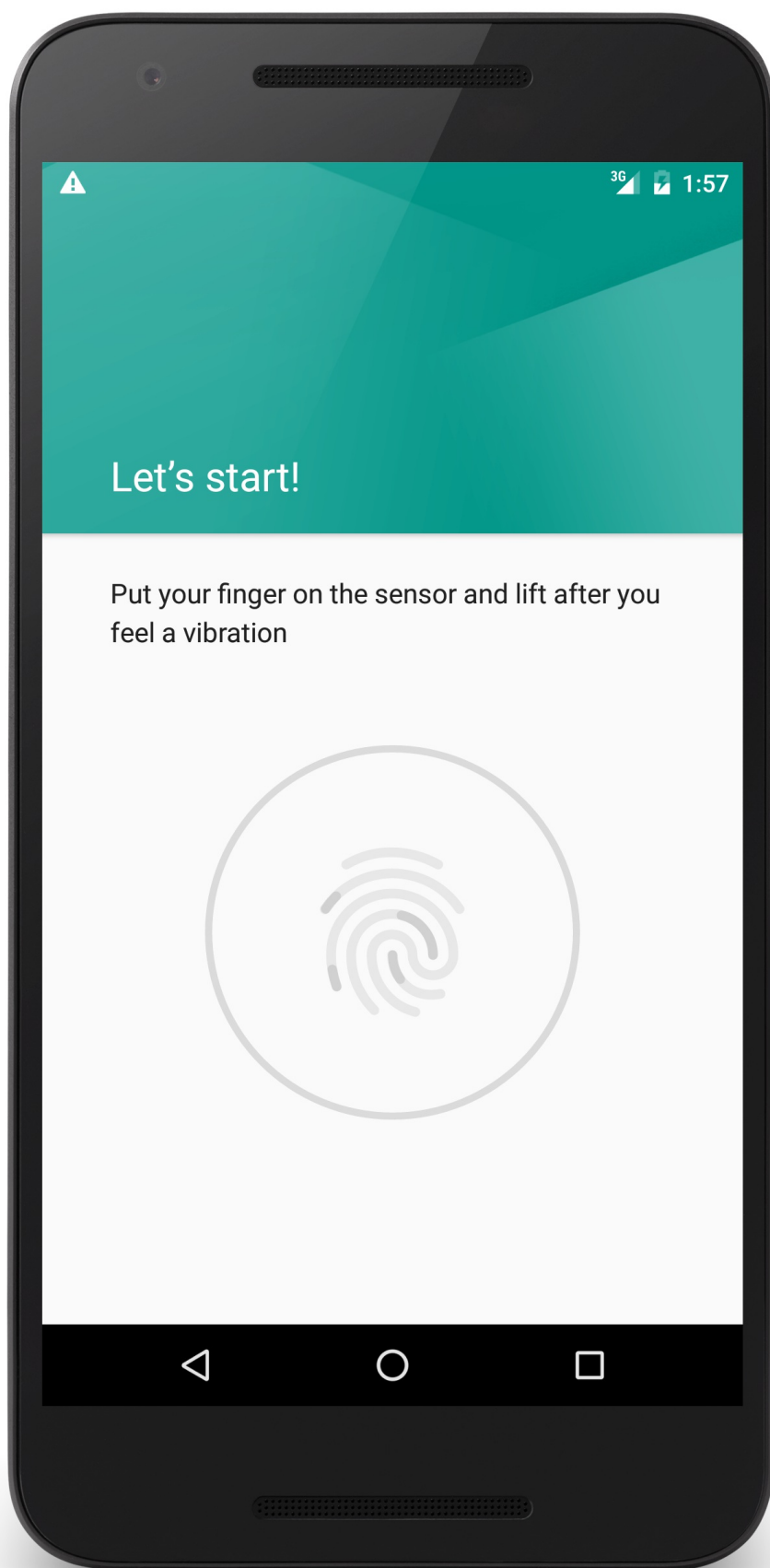
3. Screenlock 配置后, 返回到设置 > 安全 页, 然后选择指纹:



4. 在这里, 请执行用于将指纹添加到设备的序列:



5. 在最后一个屏幕会提示您若要将手指放在指纹扫描仪上:



如果使用 Android 设备, 通过触摸一根手指触扫描程序完成该过程。

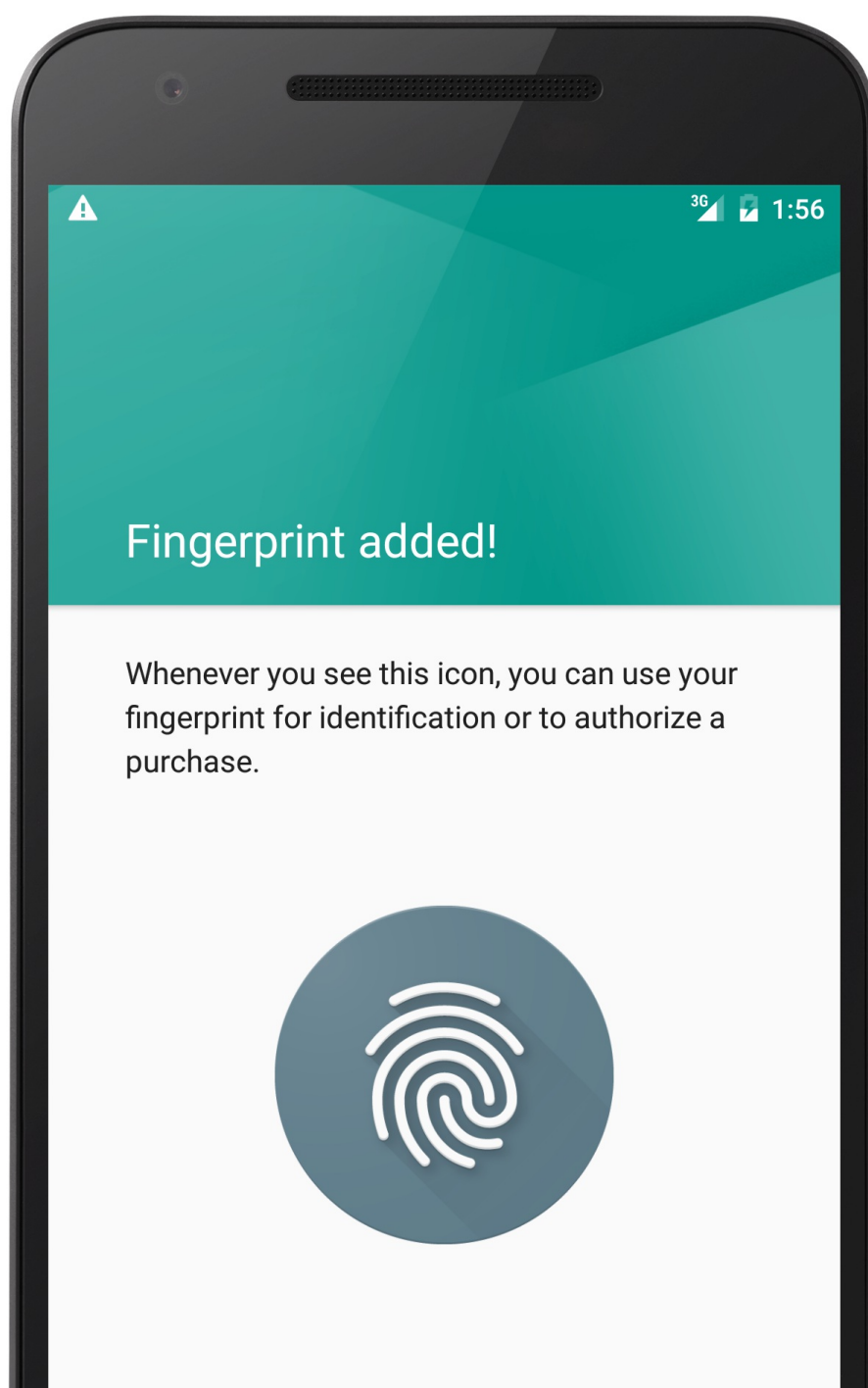
模拟指纹扫描在仿真程序

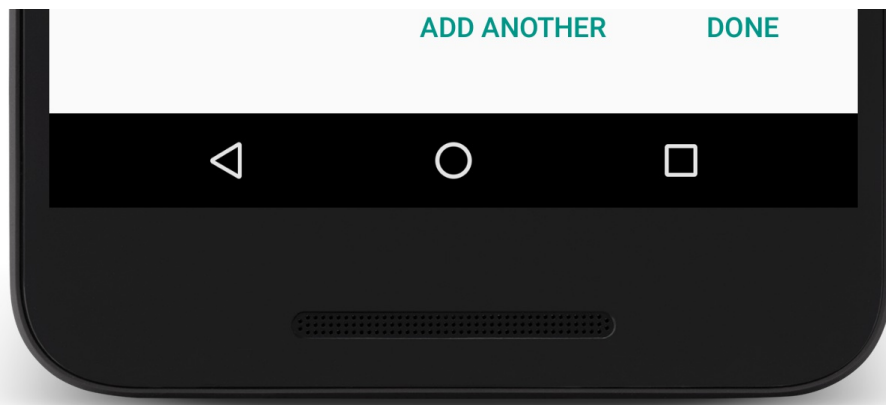
在 Android 模拟器, 则可以通过使用 Android 调试桥模拟指纹扫描。在 OS X 启动终端会话中的在 Windows 上启动命令提示符或 Powershell 会话并运行 `adb` :

```
$ adb -e emu finger touch 1
```

值1是_手指_id_的手指已"扫描"。它是为每个虚拟指纹分配的唯一整数。将来当你运行应用时可以运行此相同 ADB 命令每次在仿真程序将提示你对于指纹, 可以运行 `adb` 命令, 并将其传递_手指_id_来模拟指纹扫描。

指纹扫描完成后, Android 会通知您已添加指纹:





总结

本指南介绍了如何设置屏幕锁定和注册 Android 设备上或在 Android 模拟器中的指纹。

Android 作业计划程序

2018/11/13 • [Edit Online](#)

本指南介绍如何安排使用运行 Android 5.0 (API 级别 21) 的 Android 设备可用的 Android 作业计划程序 API 的后台工作和更高版本。

概述

若要使 Android 应用程序保持响应状态向用户的最佳方法之一是确保复杂或长时间运行的工作在后台执行。但是，务必与设备后台工作将带来负面影响的用户体验。

例如，后台作业可能每分钟轮询一次网站每三个或四个查询的特定的数据集的更改。这看起来良性的的但它会上的电池使用寿命带来灾难性影响。应用程序将重复唤醒设备，将 CPU 提升到更高版本的电源状态、接通电源的无线电收发器，使网络请求，然后处理结果。它获取更糟的是，因为该设备将不会立即关闭电源，并返回到低能耗空闲状态。效果不佳计划的后台工作可能会无意中保护设备的不必要和过多电源要求的状态中。此一项看似无害的活动（轮询网站）将导致设备相对较短的时间内不可用。

Android 提供了以下 Api，以帮助在后台执行工作，但本身不足够的智能作业计划。

- **意向服务** - 意向服务非常适合用于执行工作，但它们不提供任何方法来计划工作。
- **AlarmManager** - 这些 Api 仅允许以进行计划，但不提供任何方法来实际执行的工作的工作。此外，AlarmManager 仅允许基于时间的限制，这意味着在某个时间或经过一段时间后会引发警报。
- **广播接收器** - Android 应用程序可以设置广播的接收器来执行工作以响应系统级事件或意向。但是，广播的接收器不提供任何控制应何时运行该作业。此外限制 Android 操作系统中的更改将广播的接收器将起作用，或类型的响应的工作。

有两个有效地执行后台工作的关键功能 (有时称为_后台作业_或_作业_):

1. **以智能方式调度工作** - 非常重要的应用程序在后台的工作执行该操作时它会作为一名良好公民。理想情况下，应用程序不应要求，在运行作业。相反，应用程序应指定当作业可以运行，然后安排该作业与满足条件时将执行工作的操作系统必须满足的条件。这使 Android 可运行作业，以确保在设备上的最大效率。例如，网络请求可能会进行批处理运行全部在同一时间，以便与网络相关的开销最大值使用。
2. **封装工作** - 来执行后台工作的代码应封装在可运行独立于用户界面并将可以很轻松地重新计划如果无法完成工作的离散组件由于某种原因。

Android 作业计划程序是内置于 Android 操作系统，提供简化计划的后台工作的 fluent API 的框架。Android 作业计划程序包括以下类型：

- `Android.App.Job.JobScheduler` 是一种系统服务，用于计划、执行和根据需要取消，代表 Android 应用程序的作业。
- `Android.App.Job.JobService` 是一个抽象类，必须使用将在应用程序的主线程运行作业的逻辑扩展。这意味着，`JobService` 负责的工作方式是以异步方式执行。
- `Android.App.Job.JobInfo` 对象保留的条件来确定作业应该运行指导 Android。

若要计划使用 Android 作业计划程序的工作，Xamarin.Android 应用程序必须封装在扩展类中的代码 `JobService` 类。`JobService` 有三个作业的生存期内可以调用的生命周期方法：

- **bool (JobParameters 参数) OnStartJob** - 调用此方法 `JobScheduler` 来执行工作，并可在应用程序的主线程上运行。它负责 `JobService` 来以异步方式执行的工作并 `true` 剩余，如果没有工作或 `false` 如果完成工作。

当 `JobScheduler` 调用此方法，它将请求并通过 Android wakelock 保留作业的持续时间。完成作业后，它负责

`JobService` 告诉 `JobScheduler` 的调用这一事实 `JobFinished` 方法（接下来介绍）。

- **(`JobParameters` 参数, `bool needsReschedule`) `JobFinished`** – 必须调用此方法 `JobService` 告诉 `JobScheduler` 该操作完成。如果 `JobFinished` 未调用 `JobScheduler` 将不会删除 wakelock, 从而导致不必要的电池耗尽。
- **`bool` (`JobParameters` 参数) `OnStopJob`** – Android 过早停止该作业时调用。它应返回 `true` 如果应重新安排作业根据重试条件（下面详细讨论）。

可以指定_约束_或_触发器_控制可以或应该运行作业。例如, 就可以限制作业, 以便它仅将运行设备正在充电或启动作业时图片拍摄时。

本指南将详细地讨论如何实现 `JobService` 类并将其与安排 `JobScheduler` 。

要求

Android 作业计划程序需要 Android API 级别 21 (Android 5.0) 或更高版本。

使用 Android 作业计划程序

有三个步骤使用 Android `JobScheduler` API:

1. 实现 `JobService` 类型来封装工作。
2. 使用 `JobInfo.Builder` 对象来创建 `JobInfo` 将保存的条件对象 `JobScheduler` 运行作业。
3. 作业使用计划 `JobScheduler.Schedule` 。

实现 `JobService`

Android 作业计划程序库执行的所有工作必须都进行扩展的类型中 `Android.App.Job.JobService` 抽象类。创建 `JobService` 非常类似于创建 `Service` Android framework:

1. 扩展 `JobService` 类。
2. 修饰具有子类 `ServiceAttribute` 并设置 `Name` 参数为包名称和类的名称组成的字符串（请参阅下面的示例）。
3. 设置 `Permission` 上的属性 `ServiceAttribute` 到字符串 `android.permission.BIND_JOB_SERVICE` 。
4. 重写 `OnStartJob` 方法中, 添加代码来执行工作。Android 将调用要运行作业的应用程序的主线程上的此方法。需要较长时间, 应避免阻止应用程序的线程执行几毫秒的工作。
5. 完成工作后, `JobService` 必须调用 `JobFinished` 方法。此方法是如何 `JobService` 告知 `JobScheduler` 工作完成。调用失败 `JobFinished` 将导致 `JobService` 将不必要的需求放在设备上, 缩短电池寿命。
6. 它是一个好办法还重写 `OnStopJob` 方法。当作业正在关闭之前已完成, 并提供了, 调用此方法由 Android `JobService` 正确释放任何资源的机会。此方法应返回 `true` 如有必要, 重新计划作业, 或 `false` 如果不希望发生溢出, 若要重新运行该作业。

下面的代码是最简单的示例 `JobService` 应用程序, 使用 TPL 执行一些操作, 以异步方式:

```
[Service(Name = "com.xamarin.samples.downloadscheduler.DownloadJob",
    Permission = "android.permission.BIND_JOB_SERVICE")]
public class DownloadJob : JobService
{
    public override bool OnStartJob(JobParameters jobParams)
    {
        Task.Run(() =>
        {
            // Work is happening asynchronously

            // Have to tell the JobScheduler the work is done.
            JobFinished(jobParams, false);
        });

        // Return true because of the asynchronous work
        return true;
    }

    public override bool OnStopJob(JobParameters jobParams)
    {
        // we don't want to reschedule the job if it is stopped or cancelled.
        return false;
    }
}
```

创建对 **JobInfo** 安排作业计划

Xamarin.Android 应用程序不实例化 **JobService** 直接, 改为将传递 **JobInfo** 对象传递给 **JobScheduler**。

JobScheduler 将实例化请求 **JobService** 对象, 计划和运行 **JobService** 中的元数据根据 **JobInfo**。一个 **JobInfo** 对象必须包含以下信息:

- **JobId** –这是 **int** 值, 该值用于标识作业到 **JobScheduler**。重复使用此值将更新任何现有作业。值必须是唯一的应用程序。
- **JobService** –, 此参数才 **ComponentName** 的显式标识的类型的 **JobScheduler** 应该用于运行作业。

此扩展方法演示如何创建 **JobInfo.Builder** 与 Android **Context**, 例如活动:

```
public static class JobSchedulerHelpers
{
    public static JobInfo.Builder CreateJobBuilderUsingJobId<T>(this Context context, int jobId) where
    T:JobService
    {
        var javaClass = Java.Lang.Class.FromType(typeof(T));
        var componentName = new ComponentName(context, javaClass);
        return new JobInfo.Builder(jobId, componentName);
    }
}

// Sample usage - creates a JobBuilder for a DownloadJob and sets the Job ID to 1.
var jobBuilder = this.CreateJobBuilderUsingJobId<DownloadJob>(1);

var jobInfo = jobBuilder.Build(); // creates a JobInfo object.
```

Android 作业计划程序的一项强大功能是能够控制时的作业运行或在什么条件下一个作业可能会运行。下表描述的一些方法对 **JobInfo.Builder** 的允许的应用程序来影响一个作业的运行:

| 方法 | 描述 |
|--------------------------|----------------------------|
| SetMinimumLatency | 指定运行作业之前应观察到延迟时间 (以毫秒为单位)。 |

| 方法 | 描述 |
|---------------------------------------|--------------------------------|
| <code>SetOverridingDeadline</code> | 声明该作业必须运行, 然后经过此时间 (以毫秒为单位)。 |
| <code>SetRequiredNetworkType</code> | 指定作业的网络要求。 |
| <code>SetRequiresBatteryNotLow</code> | 设备不向用户显示"电池电量不足"警报时, 可能仅运行该作业。 |
| <code>SetRequiresCharging</code> | 当电池充电时, 可能仅运行该作业。 |
| <code>SetDeviceIdle</code> | 当设备繁忙时, 将运行作业。 |
| <code>SetPeriodic</code> | 指定作业应定期运行。 |
| <code>SetPersisted</code> | 作业在设备重新启动后应 persist。 |

`SetBackoffCriteria` 提供一些指导如何长 `JobScheduler` 尝试再次运行作业之前应等待。有两个部分的退避算法条件: 毫秒 (30 秒的默认值) 和类型的退让延迟, 应使用的 (有时称为_回退策略_或_重试策略_)。两个策略都封装在 `Android.App.Job.BackoffPolicy` 枚举:

- `BackoffPolicy.Exponential` – 指数退让策略将每个发生故障后呈指数级增加初始回退值。第一次的作业失败, 库将等待重新安排作业-示例 30 秒之前指定的初始间隔。第二次作业失败时, 库将等待至少 60 秒, 然后再尝试运行该作业。第三个尝试失败后, 库将等待 120 秒, 依此类推。这是默认值。
- `BackoffPolicy.Linear` – 此策略是作业应重新安排给线性退避算法 (直到成功) 按设置的间隔运行。线性退避算法最适合必须尽可能快地完成的工作或快速将自身解决的问题。

有关更多详细信息创建 `JobInfo` 对象, 请阅读[Google 的文档](#) `JobInfo.Builder` 类。

将参数传递给通过 `JobInfo` 作业

通过创建参数传递到作业 `PersistableBundle` 连同传递 `Job.Builder.SetExtras` 方法:

```
var jobParameters = new PersistableBundle();
jobParameters.PutInt("LoopCount", 11);

var jobBuilder = this.CreateJobBuilderUsingJobId<DownloadJob>(1)
    .SetExtras(jobParameters)
    .Build();
```

`PersistableBundle` 从访问 `Android.App.Job.JobParameters.Extras` 属性中的 `OnStartJob` 方法的 `JobService`:

```
public override bool OnStartJob(JobParameters jobParameters)
{
    var loopCount = jobParams.Extras.GetInt("LoopCount", 10);

    // rest of code omitted
}
```

计划的作业

若要计划作业, Xamarin.Android 应用程序将获取对的引用 `JobScheduler` 系统服务, 并调用 `JobScheduler.Schedule` 方法替换 `JobInfo` 上一步中创建的对象。 `JobScheduler.Schedule` 将立即返回两个整数值之一:

- **`JobScheduler.ResultSuccess`** – 成功计划作业。
- **`JobScheduler.ResultFailure`** – 无法计划作业。这通常由冲突引起 `JobInfo` 参数。

此代码是计划的作业和通知用户计划尝试的结果的示例：

```
var jobScheduler = (JobScheduler)GetSystemService(JobSchedulerService);
var scheduleResult = jobScheduler.Schedule(jobInfo);

if (JobScheduler.ResultSuccess == scheduleResult)
{
    Snackbar.Make(FindViewById(Android.Resource.Id.Content), Resource.String.jobscheduled_success,
        Snackbar.LengthShort);
}
else
{
    Snackbar.Make(FindViewById(Android.Resource.Id.Content), Resource.String.jobscheduled_failure,
        Snackbar.LengthShort);
}
```

正在取消作业

它是可以取消所有作业的计划，或只是单个作业使用 `JobsScheduler.CancelAll()` 方法或 `JobScheduler.Cancel(jobId)` 方法：

```
// Cancel all jobs
jobScheduler.CancelAll();

// to cancel a job with jobId = 1
jobScheduler.Cancel(1)
```

总结

本指南介绍如何使用 Android 作业计划程序以智能方式在后台执行工作。它介绍了如何封装为执行的工作 `JobService` 以及如何使用 `JobScheduler` 安排该工作，指定与条件 `JobTrigger` 方式与处理故障和 `RetryStrategy`。

相关链接

- [智能的作业计划](#)
- [JobScheduler API 参考](#)
- [计划与 JobScheduler 专业水准的作业](#)
- [Android 的电池和内存优化-Google I/O 2016 \(视频\)](#)
- [Android JobScheduler-René Ruppert-Xamarin 学院课程](#)

Firestore 作业调度程序

2018/11/13 • [Edit Online](#)

本指南讨论如何使用Google的Firestore作业调度程序库安排后台工作。

概述

若要使 Android 应用程序保持响应状态向用户的最佳方法之一是确保复杂或长时间运行的工作在后台执行。但是，务必与设备后台工作将带来负面影响的用户体验。

例如，后台作业可能分钟轮询一次网站每三个或四个查询的特定的数据集的更改。这看起来良性的的但它会上的电池使用寿命带来灾难性影响。应用程序将重复唤醒设备，将 CPU 提升到更高版本的电源状态、接通电源的无线电收发器，使网络请求，然后处理结果。它获取更糟的是，因为该设备将不会立即关闭电源，并返回到低能耗空闲状态。效果不佳计划的后台工作可能会无意中保护设备的不必要和过多电源要求的状态中。此一项看似无害的活动（轮询网站）将导致设备相对较短的时间内不可用。

Android 提供了以下 Api，以帮助在后台执行工作，但本身不足够的智能作业计划。

- **意向服务** – 意向服务非常适合用于执行工作，但它们不提供任何方法来计划工作。
- **AlarmManager** – 这些 Api 仅允许以进行计划，但不提供任何方法来实际执行的工作的工作。此外，AlarmManager 仅允许基于时间的限制，这意味着在某个时间或经过一段时间后会引发警报。
- **JobScheduler** – JobSchedule 是一个很好的 API，适用于计划作业的操作系统。但是，它是仅适用于 Android 应用程序的目标 API 级别 21 或更高版本。
- **广播接收器** – Android 应用程序可以设置广播的接收器来执行工作以响应系统级事件或意向。但是，广播的接收器不提供任何控制应何时运行该作业。此外限制 Android 操作系统中的更改将广播的接收器将起作用，或类型的响应的工作。

有两个有效地执行后台工作的关键功能（有时称为_后台作业_或_作业_）：

1. **以智能方式调度工作** – 非常重要的应用程序在后台的工作执行该操作时它会作为一名良好公民。理想情况下，应用程序不应要求，在运行作业。相反，应用程序应指定当作业可以运行，然后安排时满足条件时运行该工作必须满足的条件。这使 Android 可智能地执行工作。例如，网络请求可能会进行批处理运行全部在同一时间，以便与网络相关的开销最大值使用。
2. **封装工作** – 来执行后台工作的代码应封装在可运行独立于用户界面并将可以很轻松地重新计划如果无法完成工作的离散组件由于某种原因。

Firestore 作业调度程序是从 Google 提供了一个 fluent API 来简化计划的后台工作的库。它旨在替换为 Google 云管理器。Firestore 作业调度程序包括以下 Api：

- 一个 `Firestore.JobDispatcher.JobService` 是一个抽象类，必须使用将在后台作业中运行的逻辑扩展。
- 一个 `Firestore.JobDispatcher.JobTrigger` 声明时应启动的作业。这通常都表示为窗口的时间，例如，等待至少 30 秒，然后启动作业，但在 5 分钟内运行作业。
- 一个 `Firestore.JobDispatcher.RetryStrategy` 包含有关作业无法正确执行时应完成的操作的信息。重试策略指定尝试重新运行该作业之前等待多长时间。
- 一个 `Firestore.JobDispatcher.Constraint` 是一个可选值，描述一个条件，必须满足才能运行该作业，例如设备已打开 `unmetered` 网络或计费。
- `Firestore.JobDispatcher.Job` 是一个 API，统一了中的工作单元，可以通过计划到以前的 Api `JobDispatcher`。`Job.Builder` 类用于实例化 `Job`。
- 一个 `Firestore.JobDispatcher.JobDispatcher` 使用以前的三个 Api 来计划随操作系统一起工作，并提供一种方法可以取消作业，如有必要。

若要计划使用 Firebase 作业调度程序的工作，Xamarin.Android 应用程序必须封装扩展的类型中的代码

`JobService` 类。`JobService` 有三个作业的生存期内可以调用的生命周期方法：

- `bool OnStartJob(IJobParameters parameters)` – 此方法是在其中工作将发生，并且应始终实现。它在主线程上运行。此方法将返回 `true` 剩余，如果没有工作或 `false` 如果完成工作。
- `bool OnStopJob(IJobParameters parameters)` – 这称为时出于某种原因停止作业。它应返回 `true` 如果应供稍后重新安排作业。
- `JobFinished(IJobParameters parameters, bool needsReschedule)` – 将调用此方法 `JobService` 完成任何异步工作。

若要安排作业计划，应用程序将实例化 `JobDispatcher` 对象。然后，`Job.Builder` 用于创建 `Job` 对象，它提供给 `JobDispatcher` 以尝试，并计划要运行的作业。

本指南介绍了如何将 Firebase 作业调度程序添加到 Xamarin.Android 应用程序并使用它来计划后台工作。

要求

Firebase 作业调度程序需要 Android API 级别 9 或更高版本。Firebase 作业调度程序库依赖于某些组件提供的 Google Play Services;设备必须已安装的 Google Play Services。

使用在 Xamarin.Android 中的 Firebase 作业调度程序库

若要开始使用 Firebase 作业调度程序，首先添加[Xamarin.Firebase.JobDispatcher NuGet 包](#)到 Xamarin.Android 项目。搜索 NuGet 包管理器**Xamarin.Firebase.JobDispatcher**包（其中仍处于预发布）。

添加后 Firebase 作业调度程序库，创建 `JobService` 类，然后安排其运行的实例 `FirebaseJobDispatcher`。

创建 `JobService`

Firebase 作业调度程序库执行的所有工作必须都进行扩展的类型中 `Firebase.JobDispatcher.JobService` 抽象类。创建 `JobService` 非常类似于创建 `Service` Android framework:

1. 扩展 `JobService` 类
2. 修饰子类的 `ServiceAttribute`。虽然不是严格要求，但建议显式设置 `Name` 参数，以帮助进行调试 `JobService`。
3. 添加 `IntentFilter` 声明 `JobService` 中 **AndroidManifest.xml**。这将帮助您查找和调用的 Firebase 作业调度程序库 `JobService`。

下面的代码是最简单的示例 `JobService` 应用程序，使用 TPL 执行一些操作，以异步方式：

```
[Service(Name = "com.xamarin.fjdtestapp.DemoJob")]
[IntentFilter(new[] {FirebaseJobServiceIntent.Action})]
public class DemoJob : JobService
{
    static readonly string TAG = "X:DemoService";

    public override bool OnStartJob(IJobParameters jobParameters)
    {
        Task.Run(() =>
        {
            // Work is happening asynchronously (code omitted)

        });

        // Return true because of the asynchronous work
        return true;
    }

    public override bool OnStopJob(IJobParameters jobParameters)
    {
        Log.Debug(TAG, "DemoJob::OnStartJob");
        // nothing to do.
        return false;
    }
}
```

创建 `FirebaseJobDispatcher`

可以安排任何工作之前，有必要创建 `Firebase.JobDispatcher.FirebaseJobDispatcher` 对象。`FirebaseJobDispatcher` 负责计划 `JobService`。下面的代码段是一种方法创建的实例 `FirebaseJobDispatcher`：

```
// This is the "Java" way to create a FirebaseJobDispatcher object
IDriver driver = new GooglePlayDriver(context);
FirebaseJobDispatcher dispatcher = new FirebaseJobDispatcher(driver);
```

在上一代码片段中，`GooglePlayDriver` 是类，可帮助 `FirebaseJobDispatcher` 与 Google Play Services 中的计划 Api 的一些设备上交互。将参数 `context` 是任何 Android `Context`，例如活动。目前 `GooglePlayDriver` 是唯一 `IDriver` `Firebase` 作业调度程序库中的实现。

`Firebase` 作业调度程序的 `Xamarin.Android` 绑定提供了一个扩展方法来创建 `FirebaseJobDispatcher` 从 `Context`：

```
FirebaseJobDispatcher dispatcher = context.CreateJobDispatcher();
```

一次 `FirebaseJobDispatcher` 已被实例化，则可以创建 `Job` 运行中的代码和 `JobService` 类。`Job` 创建的 `Job.Builder` 对象，并将在下一节中讨论。

使用 `Job.Builder` 创建 `Firebase.JobDispatcher.Job`

`Firebase.JobDispatcher.Job` 类负责封装元数据运行所需 `JobService`。一个 `Job` 包含信息如之前必须满足作业可以运行，如果任何约束 `Job` 重复进行的或将使作业在运行任何触发器。作为最低限度 `Job` 必须具有 `_标记_` (标识到作业的唯一字符串 `FirebaseJobDispatcher`) 和类型的 `JobService` 应运行的。`Firebase` 作业调度程序将实例化 `JobService` 时的时间来运行作业。一个 `Job` 使用的实例创建 `Firebase.JobDispatcher.Job.JobBuilder` 类。

下面的代码段是如何创建的最简单的示例 `Job` 使用 `Xamarin.Android` 绑定：

```
Job myJob = dispatcher.NewJobBuilder()
    .SetService<DemoJob>("demo-job-tag")
    .Build();
```


`Job.Builder` 作业将执行一些基本验证检查, 对输入值。如果它不可能, 将引发异常 `Job.Builder` 若要创建 `Job`。

`Job.Builder` 将创建 `Job` 使用以下默认值:

- 一个 `Job` 的_生存期_(多长时间它将安排运行) 是仅在设备重启之前-设备重启后 `Job` 都将丢失。
- 一个 `Job` 不重复进行-它将只运行一次。
- 一个 `Job` 将按计划运行越早越好。
- 默认重试策略 `Job` 是使用_指数退避算法_(在部分下面更详细地讨论 [设置 RetryStrategy](#))

计划的作业

在创建后 `Job`, 它需要与计划 `FirebaseJobDispatcher` 在运行之前。有两种方法来计划 `Job`:

```
// This will throw an exception if there was a problem scheduling the job
dispatcher.MustSchedule(myJob);

// This method will not throw an exception; an integer result value is returned
int scheduleResult = dispatcher.Schedule(myJob);
```

返回的值 `FirebaseJobDispatcher.Schedule` 将是以下整数值之一:

- `FirebaseJobDispatcher.ScheduleResultSuccess` - `Job` 已成功安排。
- `FirebaseJobDispatcher.ScheduleResultUnknownError` - 出现某种未知的错误导致 `Job` 进行调度。
- `FirebaseJobDispatcher.ScheduleResultNoDriverAvailable` - 无效 `IDriver` 已使用或 `IDriver` 已由于某种原因不可用。
- `FirebaseJobDispatcher.ScheduleResultUnsupportedTrigger` - `Trigger` 不受支持。
- `FirebaseJobDispatcher.ScheduleResultBadService` - 该服务配置不正确或不可用。

配置作业

它是可以自定义作业。作业可以自定义的示例包括:

- [将参数传递到作业](#) - A `Job` 可能需要其他值以执行其工作, 例如下载文件。
- [设置约束](#)-可能有必要时满足某些条件时才运行作业。例如, 仅运行 `Job` 设备正在充电时。
- [指定何时 Job 应运行](#) - Firebase 作业调度程序允许应用程序指定运行作业的时间。
- [声明失败的作业的重试策略](#) - A_重试策略_提供的指导来 `FirebaseJobDispatcher` 上要执行的操作与 `Jobs`, 无法完成。

每个主题将讨论更以下各节中。

将参数传递到作业

通过创建参数传递到作业 `Bundle` 连同传递 `Job.Builder.SetExtras` 方法:

```
Bundle jobParameters = new Bundle();
jobParameters.PutInt(FibonacciCalculatorJob.FibonacciPositionKey, 25);

Job myJob = dispatcher.NewJobBuilder()
    .SetService<DemoJob>("demo-job-tag")
    .SetExtras(jobParameters)
    .Build();
```

`Bundle` 从访问 `IJobParameters.Extras` 属性上的 `OnStartJob` 方法:

```
public override bool OnStartJob(IJobParameters jobParameters)
{
    int position = jobParameters.Extras.GetInt(FibonacciPositionKey, DEFAULT_VALUE);

    // rest of code omitted
}
```

设置约束

约束可以帮助减少在设备上的成本或电池消耗。`Firebase.JobDispatcher.Constraint` 类将这些约束定义为整数值：

- `Constraint.OnUnmeteredNetwork` – 在设备连接到 unmetered 网络时，仅运行该作业。这可用于防止用户不会产生数据费用。
- `Constraint.OnAnyNetwork` – 设备连接到任何网络上运行作业。如果指定与 `Constraint.OnUnmeteredNetwork`，此值将优先。
- `Constraint.DeviceCharging` – 仅当设备正在充电时，请运行该作业。

使用设置约束 `Job.Builder.SetConstraint` 方法：

```
Job myJob = dispatcher.NewJobBuilder()
    .SetService<DemoJob>("demo-job-tag")
    .SetConstraint(Constraint.DeviceCharging)
    .Build();
```

`JobTrigger` 指导有关作业应该启动操作系统。一个 `JobTrigger` 已_执行窗口_定义计划的时间，何时 `Job` 应运行。执行窗口已_启动窗口_值和一个_结束时段_值。开始时段是设备应在运行作业之前等待的秒数和结束窗口值是在运行前等待的秒的最大数 `Job`。

一个 `JobTrigger` 可用于创建 `Firebase.Jobdispatcher.Trigger.ExecutionWindow` 方法。例如

`Trigger.ExecutionWindow(15,60)` 意味着，作业应运行在计划运行时从 15 到 60 秒之间。`Job.Builder.SetTrigger` 方法是使用

```
JobTrigger myTrigger = Trigger.ExecutionWindow(15,60);
Job myJob = dispatcher.NewJobBuilder()
    .SetService<DemoJob>("demo-job-tag")
    .SetTrigger(myTrigger)
    .Build();
```

默认值 `JobTrigger` 的值表示作业 `Trigger.Now`，它指定安排后尽快运行作业...

设置 `RetryStrategy`

`Firebase.JobDispatcher.RetryStrategy` 用来指定多少设备延迟应使用之前尝试重新运行失败的作业。一个 `RetryStrategy` 已_策略_，用于定义哪些时间基准算法将用于重新计划失败的作业，并指定应将作业安排在其中一个时段执行窗口。这_重新计划窗口_由两个值定义。第一个值是在重新安排作业之前等待的秒数（_初始退避算法_值），并且第二个数字的最大之前运行该作业必须等待的秒数（_最大回退_值）。

由这些整数值标识两种类型的重试策略：

- `RetryStrategy.RetryPolicyExponential` – _指数退避算法_策略将初始回退值指数级增长后增加每个失败。第一次的作业失败，库将等待重新安排作业之前指定的（_i_）间隔–示例 30 秒。第二次作业失败时，库将等待至少 60 秒，然后再尝试运行该作业。第三个尝试失败后，库将等待 120 秒，依此类推。默认值 `RetryStrategy` `Firebase` 作业调度程序库由 `RetryStrategy.DefaultExponential` 对象。它有 30 秒初始退避算法和 3600 秒的最大回退。
- `RetryStrategy.RetryPolicyLinear` – 此策略是_线性退让_（直到成功），应重新安排作业，若要在运行设置的时间间隔。线性退避算法最适合必须尽可能快地完成的工作或快速将自身解决的问题。`Firebase` 作业调度程序库定义 `RetryStrategy.DefaultLinear` 具有至少 30 秒重新计划窗口和高达 3600 秒。

可以定义一个自定义 `RetryStrategy` 与 `FirebaseJobDispatcher.NewRetryStrategy` 方法。它采用三个参数：

1. `int policy` – 策略_是上一个之一 `RetryStrategy` 值, `RetryStrategy.RetryPolicyLinear`, 或 `RetryStrategy.RetryPolicyExponential`。
2. `int initialBackoffSeconds` – 初始退避算法_延迟, 以秒为单位, 尝试重新运行该作业之前需。此默认值为 30 秒。
3. `int maximumBackoffSeconds` – 最大回退_值声明的最大尝试重新运行该作业之前的延迟秒数。默认值为 3600 秒。

```
RetryStrategy retry = dispatcher.NewRetryStrategy(RetryStrategy.RetryPolicyLinear, initialBackoffSeconds,
maximumBackoffSet);

// Create a Job and set the RetryStrategy via the Job.Builder
Job myJob = dispatcher.NewJobBuilder()
    .SetService<DemoJob>("demo-job-tag")
    .SetRetryStrategy(retry)
    .Build();
```

正在取消作业

它是可以取消所有作业的计划, 或只是单个作业使用 `FirebaseJobDispatcher.CancelAll()` 方法或 `FirebaseJobDispatcher.Cancel(string)` 方法：

```
int cancelResult = dispatcher.CancelAll();

// to cancel a single job:

int cancelResult = dispatcher.Cancel("unique-tag-for-job");
```

这两种方法将返回一个整数值：

- `FirebaseJobDispatcher.CancelResultSuccess` – 已成功取消该作业。
- `FirebaseJobDispatcher.CancelResultUnknownError` – 错误阻止, 正在取消作业。
- `FirebaseJobDispatcher.CancelResult.NoDriverAvailable` – `FirebaseJobDispatcher` 无法取消该作业, 因为没有有效 `IDriver` 可用。

总结

本指南介绍如何使用 Firebase 作业调度程序以智能方式在后台执行工作。它介绍了如何封装为执行的工作 `JobService` 以及如何使用 `FirebaseJobDispatcher` 安排该工作, 指定与条件 `JobTrigger` 方式与处理故障和 `RetryStrategy`。

相关链接

- [NuGet 上 Xamarin.Firebase.JobDispatcher](#)
- [firebase 作业-dispatcher GitHub 上](#)
- [Xamarin.Firebase.JobDispatcher 绑定](#)
- [智能的作业计划](#)
- [Android 的电池和内存优化-Google I/O 2016 \(视频\)](#)

片段

2018/10/26 • [Edit Online](#)

Android 3.0 引入了片段, 演示如何在手机和平板电脑上找到的许多不同的屏幕大小支持更灵活的设计。本文将介绍如何使用片段开发 Xamarin.Android 应用程序, 以及如何在预 Android 3.0 (API 级别 11) 设备上支持片段。

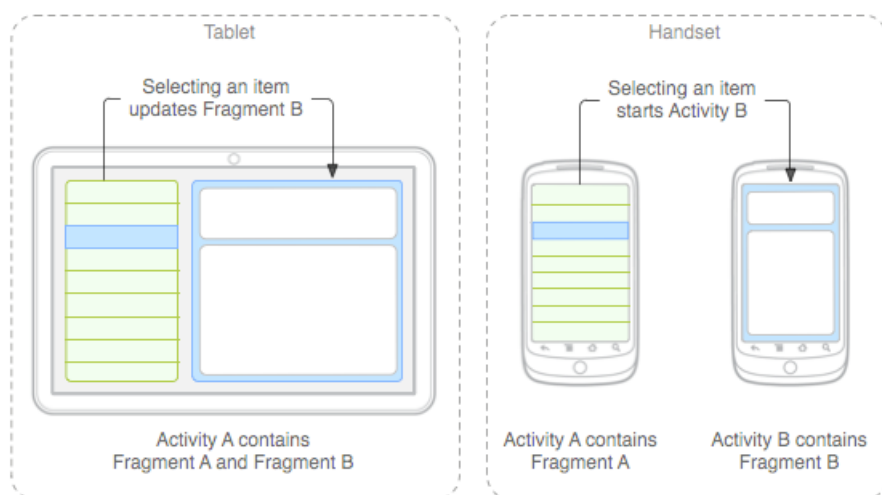
片段概述

在大多数平板电脑上找到更大的屏幕大小添加到 Android 开发的一层额外的复杂性, 此演示适合对于在小屏幕不一定适合也较大屏幕或进行相反转换的布局。若要减少的复杂情况, 这就产生了, Android 3.0 添加了两项新功能, 片段并支持包。

片段可视为用户界面模块。它们使开发人员分割成独立的、可重用部分, 可以在单独的活动中运行的用户界面。在运行时, 活动本身将决定要使用的片段。

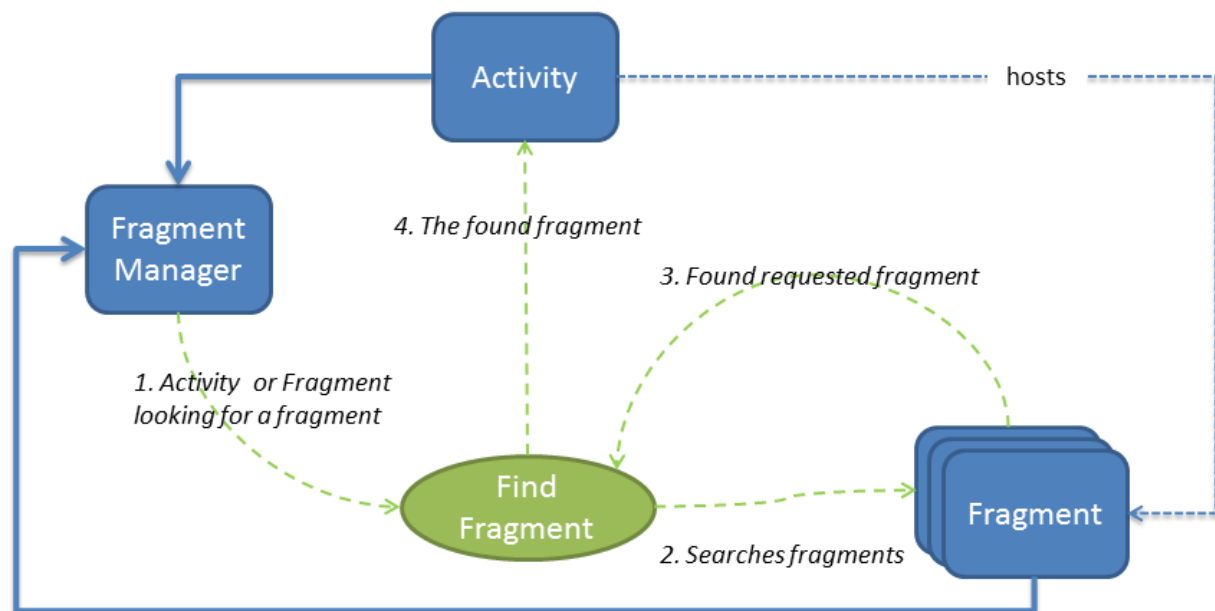
支持包最初称为兼容性库和允许的片段以在运行版本的 Android 3.0 (API 级别 11) 之前的 Android 设备上使用。

例如下, 图说明了单个应用程序如何使用跨不同设备外观造型的片段。



一个片段包含列表, 而片段 B 包含该列表中选择项的详细信息。平板电脑上运行应用程序时, 它可以在同一个活动上显示两个片段。(使用其较小的屏幕大小) 手机上运行同一应用程序时, 这些片段托管在两个单独的活动。片段 A 和片段 B 将这两个机型上相同, 但托管它们的活动不同。

若要帮助协调和管理所有这些片段的活动, Android 引入了一个名为的新类 `FragmentManager`。每个活动具有自己的实例 `FragmentManager` 用于添加、删除和查找托管片段。下图说明了片段和活动之间的关系:



在某些方面，片段可以认为是作为复合控件或微型活动。它们捆绑到可以然后独立使用由开发人员在活动的可重用模块的部分 UI。片段具有视图层次结构，就像活动 — 但不同于活动，它可以在共享跨屏幕。视图不同于片段，片段具有其自己的生命周期;视图不适用。

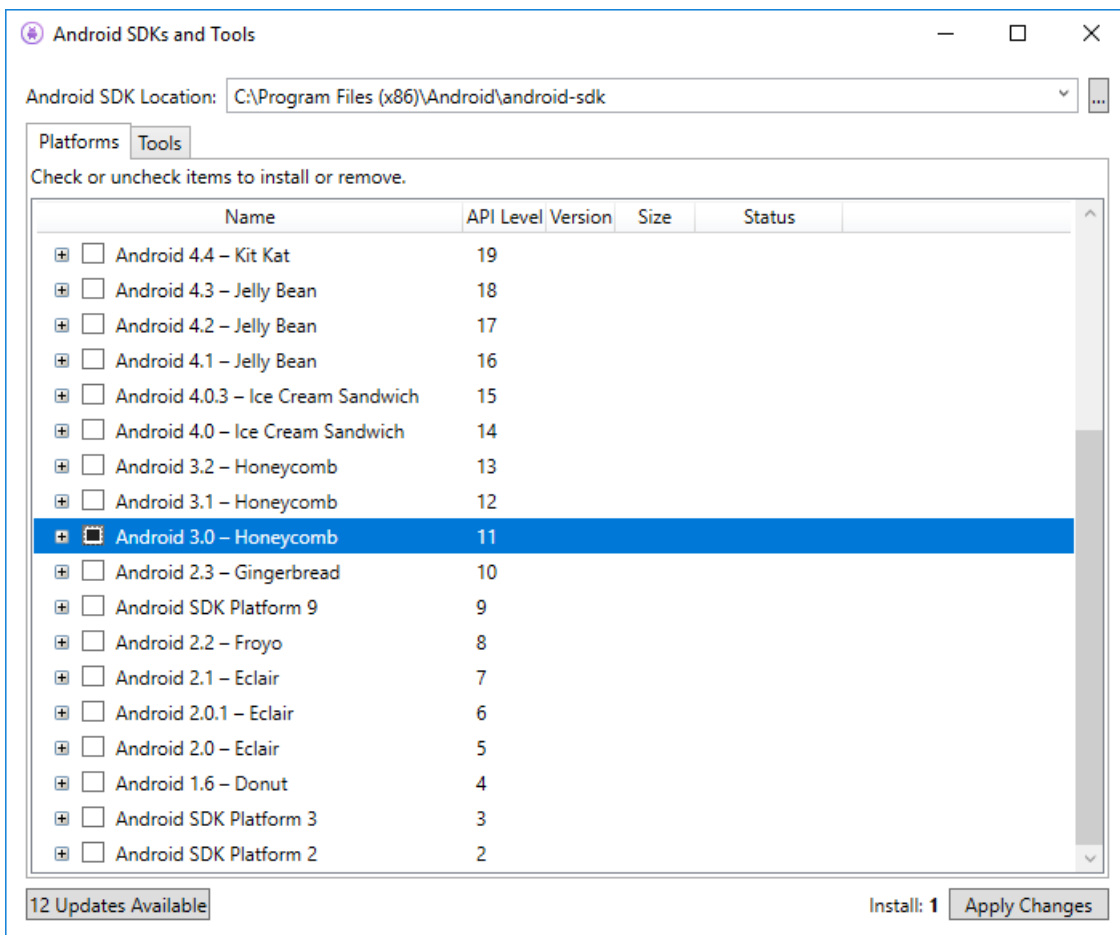
活动是一个主机到一个或多个片段，而并不直接知道自己的片段。同样，片段是不直接感知到的其他片段中托管的活动。但是，片段和活动都认识 `FragmentManager` 其活动中。通过使用 `FragmentManager`，可以为活动或片段以获取对片段的特定实例的引用，然后对该实例调用方法。这样一来，可以进行通信的活动或片段并将其与其他片段交互。

本指南包含有关如何使用片段，其中包括全面介绍：

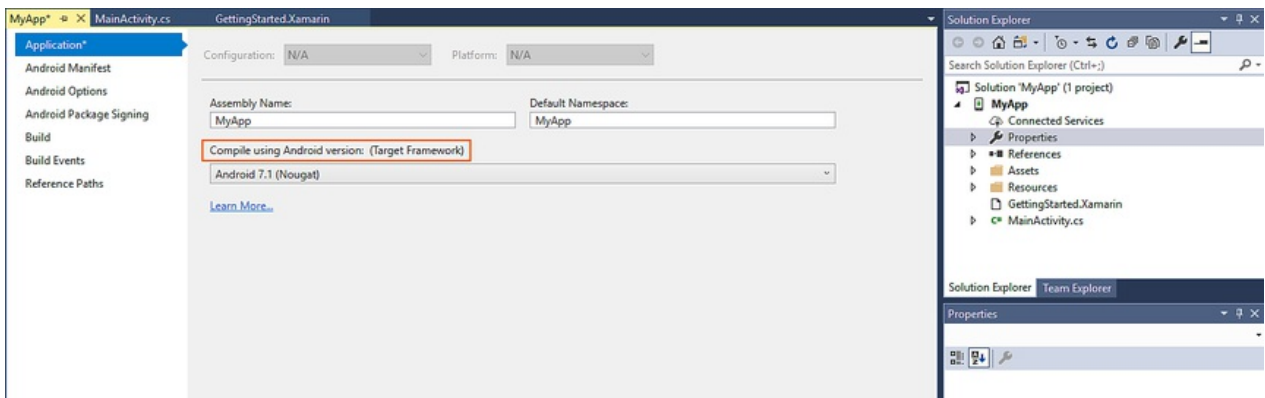
- **创建片段**– 如何创建基本片段和必须实现的主要方法。
- **管理和事务片段**– 如何在运行时操作片段。
- **Android 支持包**-如何使用库，使片段较旧版本的 Android 上使用。

要求

片段是在使用 API 级别 (Android 3.0) 11 启动 Android SDK 中提供，如以下屏幕截图中所示：



片段是可在 Xamarin.Android 4.0 和更高版本。Xamarin.Android 应用程序必须至少为目标 API 级别 11 (Android 3.0) 或更高版本才能使用片段。可能会在项目属性, 如下所示设置目标框架:



它是可以在较旧版本的 Android 使用 Android 支持包和 Xamarin.Android 4.2 或更高版本中使用片段。在本部分的文档中的更详细地介绍了如何执行此操作。

相关链接

- [Honeycomb 库 \(示例\)](#)
- [片段](#)
- [支持包](#)
- [MOTODEV 网络研讨会: 引入片段](#)

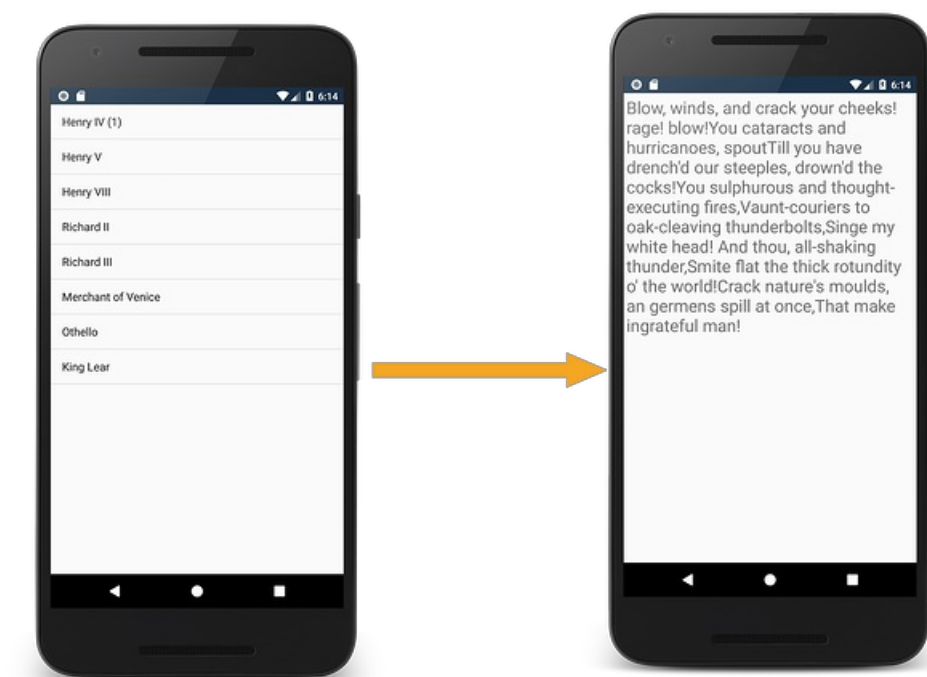
实现片段的演练

2018/10/26 • [Edit Online](#)

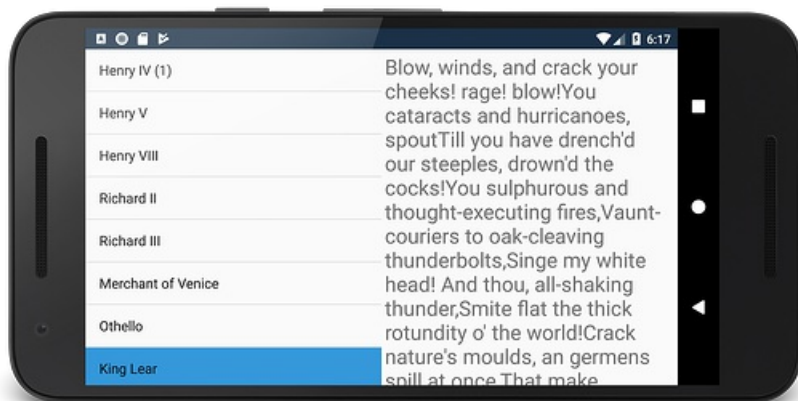
片段是自包含的模块化组件，可帮助您解决有多种不同的屏幕大小的目标设备的 Android 应用的复杂性。本文介绍如何创建和开发 Xamarin.Android 应用程序时使用片段。

概述

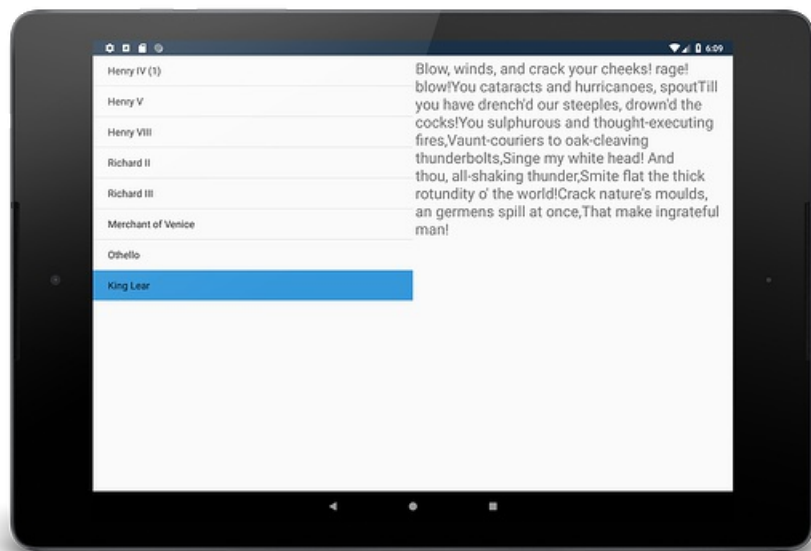
在本部分中，您将了解如何创建和使用 Xamarin.Android 应用程序中的片段。此应用程序将通过 William 莎士比亚显示在列表中的多个播放的标题。当用户点击播放的标题时，应用会在单独的活动显示该播放的引文：



时电话旋转为横向模式下，将更改应用的外观：列表播放和引号将出现在同一个活动。选择播放时，引号将显示在同一活动中：



最后，如果在平板电脑上运行该应用程序：



此示例应用程序可以轻松地适应不同外观造型和方向的最少的代码更改使用片段并[备用布局](#)。

应用程序的数据将作为应用程序中硬编码的两个字符串数组中存在C#的字符串数组。每个数组将充当一个段的数据源。一个数组将保存的莎士比亚，某些播放的名称和其他数组将包含该播放的引文。当应用启动时，它将显示中的 play 名称 `ListFragment`。当用户单击中发挥作用 `ListFragment`，应用将启动另一个活动将显示引号。

应用程序的用户界面将包含两个布局、一个用于纵向和横向模式下的一个。在运行时，Android 将确定要加载哪些布局基于设备的方向，并将提供给要呈现的活动的布局。片段中，将包含所有响应用户的单击操作和显示数据的逻辑。仅为将承载片段的容器存在的应用中的活动。

本演练分为两个指南。[第一部分](#)将专注于应用程序的核心部分。将创建一组布局（已优化为纵向模式），以及两个片段和两个活动：

1. `MainActivity` 这是应用程序的启动活动。
2. `TitlesFragment` 此代码段将显示的编写的 William 莎士比亚剧本的标题的列表。它将由托管 `MainActivity`。
3. `PlayQuoteActivity` `TitlesFragment` 将启动 `PlayQuoteActivity` 以响应用户选择在 play `TitlesFragment`。
4. `PlayQuoteFragment` 此代码段将显示通过 William 莎士比亚播放的引文。它将由托管 `PlayQuoteActivity`。

[本演练中的第二个](#)将讨论添加备用布局（已优化的横向模式下）这将在屏幕上显示两个片段。此外，一些细微的代码将进行更改的代码，以便该应用将调整其行为与在屏幕同时显示的片段数。

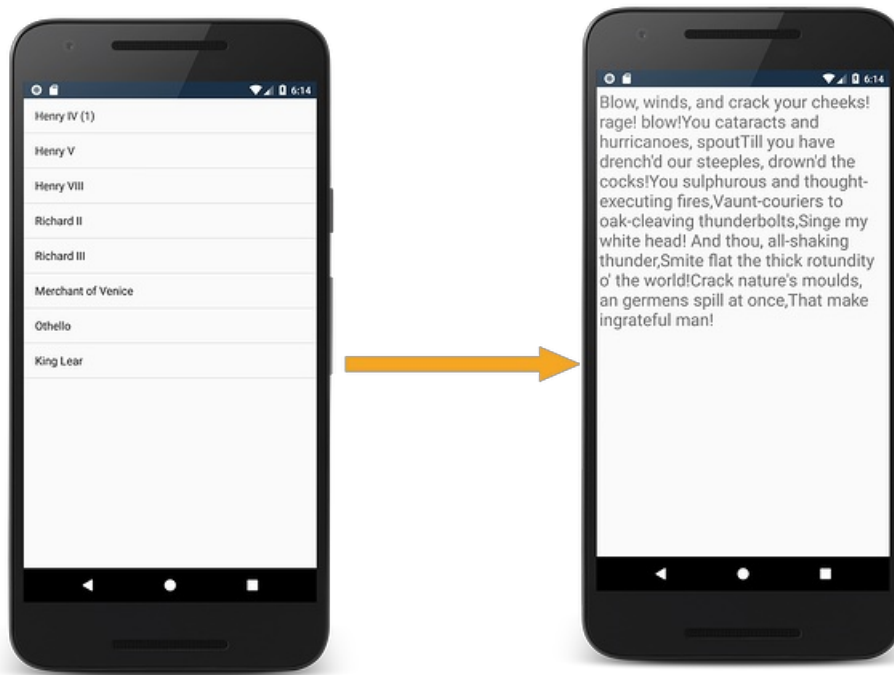
相关链接

- [FragmentsWalkthrough（示例）](#)
- [设计器概述](#)
- [实现片段](#)
- [支持包](#)

片段演练-电话

2018/10/26 • [Edit Online](#)

这是演练的将创建面向 Android 设备中纵向方向的 Xamarin.Android 应用的第一个部分。本演练中将讨论如何在 Xamarin.Android 中创建片段以及如何将它们添加到一个示例。



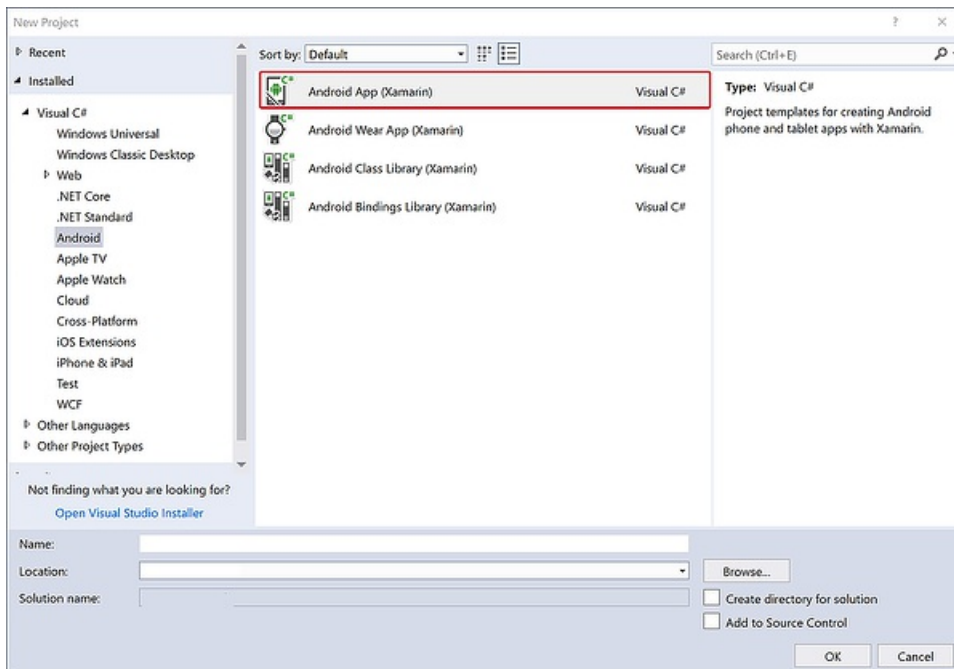
为此应用, 将创建以下类:

1. `PlayQuoteFragment` 此代码段将显示通过 William 莎士比亚播放的引文。它将由托管 `PlayQuoteActivity`。
2. `Shakespeare` 此类将作为属性保存两个硬编码的数组。
3. `TitlesFragment` 此代码段将显示的编写的 William 莎士比亚剧本的标题的列表。它将由托管 `MainActivity`。
4. `PlayQuoteActivity` `TitlesFragment` 将启动 `PlayQuoteActivity` 以响应用户选择在 play `TitlesFragment`。

1.创建 Android 项目

创建一个名为新的 Xamarin.Android 项目 **FragmentSample**。

- [Visual Studio](#)
- [Visual Studio for Mac](#)



2.添加数据

为此应用程序的数据将存储在两个类名称的属性的硬编码字符串数组 `Shakespeare` :

- `Shakespeare.Titles` 此数组将用于保存从 William 莎士比亚剧本的列表。这是为数据源 `TitlesFragment` 。
- `Shakespeare.Dialogue` 此数组将包含一组从一个在播放中包含引号 `Shakespeare.Titles` 。这是为数据源 `PlayQuoteFragment` 。

添加一个新C#类来**FragmentSample**项目, 并命名**Shakespeare.cs**。在此文件中, 创建一个新C#类调用 `Shakespeare` 包含以下内容

```
class Shakespeare
{
    public static string[] Titles = {
        "Henry IV (1)",
        "Henry V",
        "Henry VIII",
        "Richard II",
        "Richard III",
        "Merchant of Venice",
        "Othello",
        "King Lear"
    };

    public static string[] Dialogue = {
        "So shaken as we are, so wan with care, Find we a time for frightened
        peace to pant, And breathe short-winded accents of new broils To be commenced in strands afar remote. No more
        the thirsty entrance of this soil Shall daub her lips with her own children's blood; Nor more shall trenching
        war channel her fields, Nor bruise her flowerets with the armed hoofs Of hostile paces: those opposed eyes,
        Which, like the meteors of a troubled heaven, All of one nature, of one substance bred, Did lately meet in the
        intestine shock And furious close of civil butchery Shall now, in mutual well-beseeming ranks, March all one
        way and be no more opposed Against acquaintance, kindred and allies: The edge of war, like an ill-sheathed
        knife, No more shall cut his master. Therefore, friends, As far as to the sepulchre of Christ, Whose soldier
        now, under whose blessed cross We are impressed and engaged to fight, Forthwith a power of English shall we
        levy; Whose arms were moulded in their mothers' womb To chase these pagans in those holy fields Over whose
        acres walk'd those blessed feet Which fourteen hundred years ago were nail'd For our advantage on the bitter
        cross. But this our purpose now is twelve month old, And bootless 'tis to tell you we will go: Therefore we
        meet not now. Then let me hear Of you, my gentle cousin Westmoreland, What yesternight our council did decree
        In forwarding this dear expedience.",
        "Hear him but reason in divinity, And all-admiring with an inward wish
        You would desire the king were made a prelate: Hear him debate of commonwealth affairs, You would say it hath
        been all in all his study: List his discourse of war, and you shall hear A fearful battle render'd you in
```

music: Turn him to any cause of policy, The Gordian knot of it he will unloose, Familiar as his garter: that, when he speaks, The air, a charter'd libertine, is still, And the mute wonder lurketh in men's ears, To steal his sweet and honey'd sentences; So that the art and practis'd part of life Must be the mistress to this theoretic: Which is a wonder how his grace should glean it, Since his addiction was to courses vain, His companies unletter'd, rude and shallow, His hours fill'd up with riots, banquets, sports, And never noted in him any study, Any retirement, any sequestration From open haunts and popularity."

"I come no more to make you laugh: things now, That bear a weighty and a serious brow, Sad, high, and working, full of state and woe, Such noble scenes as draw the eye to flow, We now present. Those that can pity, here May, if they think it well, let fall a tear; The subject will deserve it. Such as give Their money out of hope they may believe, May here find truth too. Those that come to see Only a show or two, and so agree The play may pass, if they be still and willing, I'll undertake may see away their shilling Richly in two short hours. Only they That come to hear a merry bawdy play, A noise of targets, or to see a fellow In a long motley coat guarded with yellow, Will be deceived; for, gentle hearers, know, To rank our chosen truth with such a show As fool and fight is, beside forfeiting Our own brains, and the opinion that we bring, To make that only true we now intend, Will leave us never an understanding friend. Therefore, for goodness' sake, and as you are known The first and happiest hearers of the town, Be sad, as we would make ye: think ye see The very persons of our noble story As they were living; think you see them great, And follow'd with the general throng and sweat Of thousand friends; then in a moment, see How soon this mightiness meets misery: And, if you can be merry then, I'll say A man may weep upon his wedding-day."

"First, heaven be the record to my speech! In the devotion of a subject's love, Tendering the precious safety of my prince, And free from other misbegotten hate, Come I appellant to this princely presence. Now, Thomas Mowbray, do I turn to thee, And mark my greeting well; for what I speak My body shall make good upon this earth, Or my divine soul answer it in heaven. Thou art a traitor and a miscreant, Too good to be so and too bad to live, Since the more fair and crystal is the sky, The uglier seem the clouds that in it fly. Once more, the more to aggravate the note, With a foul traitor's name stuff I thy throat; And wish, so please my sovereign, ere I move, What my tongue speaks my right drawn sword may prove."

"Now is the winter of our discontent Made glorious summer by this sun of York; And all the clouds that lour'd upon our house In the deep bosom of the ocean buried. Now are our brows bound with victorious wreaths; Our bruised arms hung up for monuments; Our stern alarums changed to merry meetings, Our dreadful marches to delightful measures. Grim-visaged war hath smooth'd his wrinkled front; And now, instead of mounting barded steeds To fright the souls of fearful adversaries, He capers nimbly in a lady's chamber To the lascivious pleasing of a lute. But I, that am not shaped for sportive tricks, Nor made to court an amorous looking-glass; I, that am rudely stamp'd, and want love's majesty To strut before a wanton ambling nymph; I, that am curtail'd of this fair proportion, Cheated of feature by dissembling nature, Deform'd, unfinish'd, sent before my time Into this breathing world, scarce half made up, And that so lamely and unfashionable That dogs bark at me as I halt by them; Why, I, in this weak piping time of peace, Have no delight to pass away the time, Unless to spy my shadow in the sun And descant on mine own deformity: And therefore, since I cannot prove a lover, To entertain these fair well-spoken days, I am determin'd to prove a villain And hate the idle pleasures of these days. Plots have I laid, inductions dangerous, By drunken prophecies, libels and dreams, To set my brother Clarence and the king In deadly hate the one against the other: And if King Edward be as true and just As I am subtle, false and treacherous, This day should Clarence closely be mew'd up, About a prophecy, which says that 'G' Of Edward's heirs the murderer shall be. Dive, thoughts, down to my soul: here Clarence comes."

"To bait fish withal: if it will feed nothing else, it will feed my revenge. He hath disgraced me, and hindered me half a million; laughed at my losses, mocked at my gains, scorned my nation, thwarted my bargains, cooled my friends, heated mine enemies; and what's his reason? I am a Jew. Hath not a Jew eyes? hath not a Jew hands, organs, dimensions, senses, affections, passions? fed with the same food, hurt with the same weapons, subject to the same diseases, healed by the same means, warmed and cooled by the same winter and summer, as a Christian is? If you prick us, do we not bleed? if you tickle us, do we not laugh? if you poison us, do we not die? and if you wrong us, shall we not revenge? If we are like you in the rest, we will resemble you in that. If a Jew wrong a Christian, what is his humility? Revenge. If a Christian wrong a Jew, what should his sufferance be by Christian example? Why, revenge. The villainy you teach me, I will execute, and it shall go hard but I will better the instruction."

"Virtue! a fig! 'tis in ourselves that we are thus or thus. Our bodies are our gardens, to the which our wills are gardeners: so that if we will plant nettles, or sow lettuce, set hyssop and weed up thyme, supply it with one gender of herbs, or distract it with many, either to have it sterile with idleness, or manured with industry, why, the power and corrigible authority of this lies in our wills. If the balance of our lives had not one scale of reason to poise another of sensuality, the blood and baseness of our natures would conduct us to most preposterous conclusions: but we have reason to cool our raging motions, our carnal stings, our unbitted lusts, whereof I take this that you call love to be a sect or scion."

"Blow, winds, and crack your cheeks! rage! blow! You cataracts and hurricanoes, spout Till you have drench'd our steeples, drown'd the cocks! You sulphurous and thought-executing fires, Vault-couriers to oak-cleaving thunderbolts, Singe my white head! And thou, all-shaking thunder, Smite flat the thick rotundity o' the world! Crack nature's moulds, an germens spill at once, That make ingrateful man!"

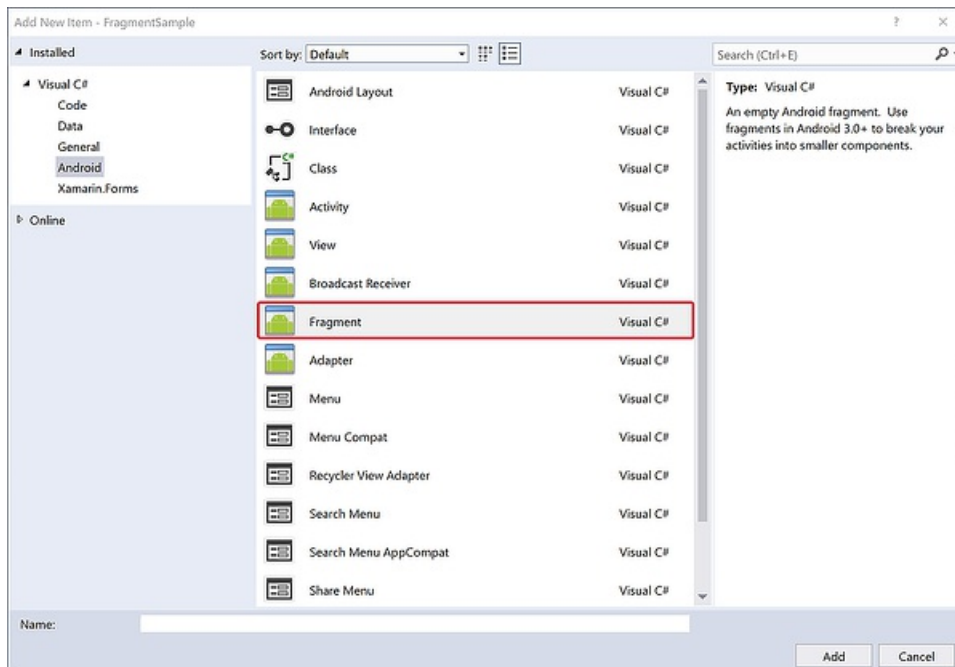
};

}

3.创建 PlayQuoteFragment

PlayQuoteFragment 是将显示一个由先前在应用程序, 用户选择了莎士比亚 play 的 Android 片段此片段不会使用一个 Android 布局文件; 相反, 它将动态创建其用户界面。添加一个新 `Fragment` 类名为 `PlayQuoteFragment` 到项目:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



然后, 将更改为类似于此代码片段的片段的代码:

```

public class PlayQuoteFragment : Fragment
{
    public int PlayId => Arguments.GetInt("current_play_id", 0);

    public static PlayQuoteFragment NewInstance(int playId)
    {
        var bundle = new Bundle();
        bundle.PutInt("current_play_id", playId);
        return new PlayQuoteFragment {Arguments = bundle};
    }

    public override View OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    {
        if (container == null)
        {
            return null;
        }

        var textView = new TextView(Activity);
        var padding = Convert.ToInt32(TypedValue.ApplyDimension(ComplexUnitType.Dip, 4,
Activity.Resources.DisplayMetrics));
        textView.SetPadding(padding, padding, padding, padding);
        textView.TextSize = 24;
        textView.Text = Shakespeare.Dialogue[PlayId];

        var scroller = new ScrollView(Activity);
        scroller.AddView(textView);

        return scroller;
    }
}

```

它是在 Android 应用提供将实例化一个片段的工厂方法中的常见模式。这可确保将具有正常工作所需的参数创建的片段。在此演练中，应用应使用 `PlayQuoteFragment.NewInstance` 方法来创建新的碎片引号所选的每个时段。

`NewInstance` 方法将采用单个参数—报价后，若要显示的索引。

`OnCreateView` 时就可以呈现在屏幕上的片段时，将由 Android 调用方法。它将返回 Android `View` 是片段的对象。此代码段不使用的布局文件以创建视图。相反，它以编程方式将通过实例化来创建视图 **TextView** 来保存报价，并将显示在该小组件 **ScrollView**。

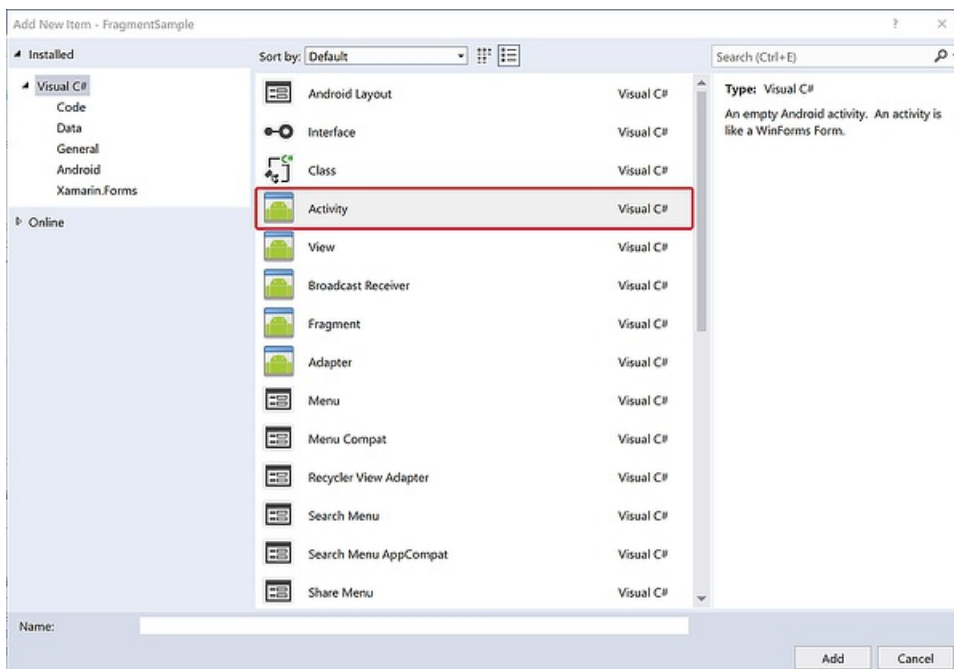
NOTE

片段类必须具有公共默认构造函数没有参数。

4.创建 PlayQuoteActivity

必须在活动内承载片段，因此此应用所需的活动，将承载 `PlayQuoteFragment`。该活动将动态将片段添加到其布局在运行时。将新活动添加到应用程序并将其命名 `PlayQuoteActivity`：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



编辑中的代码 `PlayQuoteActivity` :

```
[Activity(Label = "PlayQuoteActivity")]
public class PlayQuoteActivity : Activity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        var playId = Intent.Extras.GetInt("current_play_id", 0);

        var detailsFrag = PlayQuoteFragment.NewInstance(playId);
        FragmentManager.BeginTransaction()
            .Add(Android.Resource.Id.Content, detailsFrag)
            .Commit();
    }
}
```

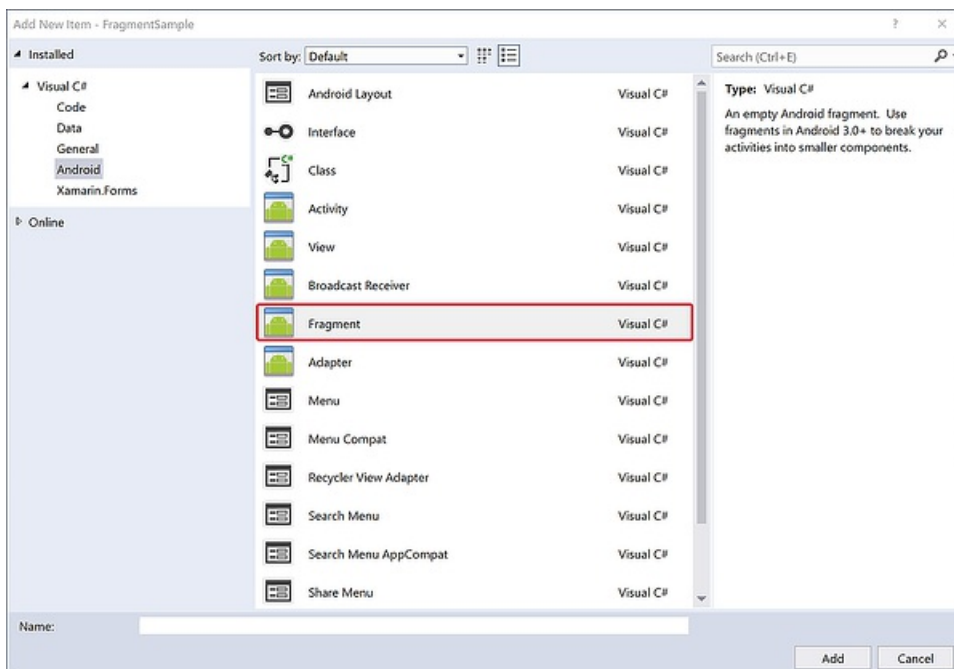
当 `PlayQuoteActivity` 是创建的它将实例化一个新 `PlayQuoteFragment` 并加载在其根视图中的上下文中该片段 `FragmentManager`。请注意，此活动不会加载其用户界面的 Android 布局文件。相反，新 `PlayQuoteFragment` 添加到应用程序的根视图。资源标识符 `Android.Resource.Id.Content` 用于无需知道其特定的标识符引用活动的根视图。

5.创建 TitlesFragment

`TitlesFragment` 将名为专用的片段的子类 `ListFragment` 封装用于显示逻辑 `ListView` 片段中。一个 `ListFragment` 公开 `ListAdapter` 属性(由 `ListView` 以显示其内容)和名为一个事件处理程序 `OnListItemClick` 这样的片段响应的情况下将显示的行上单击 `ListView`。

若要开始，将新片段添加到项目并将其命名 **TitlesFragment**:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



编辑的代码片段：

```
public class TitlesFragment : ListFragment
{
    int selectedPlayId;

    public TitlesFragment()
    {
        // Being explicit about the requirement for a default constructor.
    }

    public override void OnActivityCreated(Bundle savedInstanceState)
    {
        base.OnActivityCreated(savedInstanceState);
        ListAdapter = new ArrayAdapter<String>(Activity, Android.Resource.Layout.SimpleListItemActivated1,
        Shakespeare.Titles);

        if (savedInstanceState != null)
        {
            selectedPlayId = savedInstanceState.GetInt("current_play_id", 0);
        }
    }

    public override void OnSaveInstanceState(Bundle outState)
    {
        base.OnSaveInstanceState(outState);
        outState.PutInt("current_play_id", selectedPlayId);
    }

    public override void OnListItemClick(ListView l, View v, int position, long id)
    {
        ShowPlayQuote(position);
    }

    void ShowPlayQuote(int playId)
    {
        var intent = new Intent(Activity, typeof(PlayQuoteActivity));
        intent.PutExtra("current_play_id", playId);
        StartActivity(intent);
    }
}
```

Android 创建活动时将调用 `OnActivityCreated` 方法的片段; 这就是用于列表适配器 `ListView` 创建。

ShowQuoteFromPlay 方法将启动的实例 PlayQuoteActivity 以显示所选播放的报价。

在 MainActivity 中显示 TitlesFragment

最后一步是显示 TitlesFragment 内 MainActivity。活动不会动态加载片段。而是该片段将以静态方式加载的活动使用的布局文件中声明 fragment 元素。通过设置标识要加载的片段 android:name 片段类（包括类型的命名空间）的属性。例如，若要使用 TitlesFragment，然后 android:name 应设置为 FragmentSample.TitlesFragment。

编辑布局文件 activity_mail.axml，现有的 XML 替换为以下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment
        android:name="FragmentSample.TitlesFragment"
        android:id="@+id/titles"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

NOTE

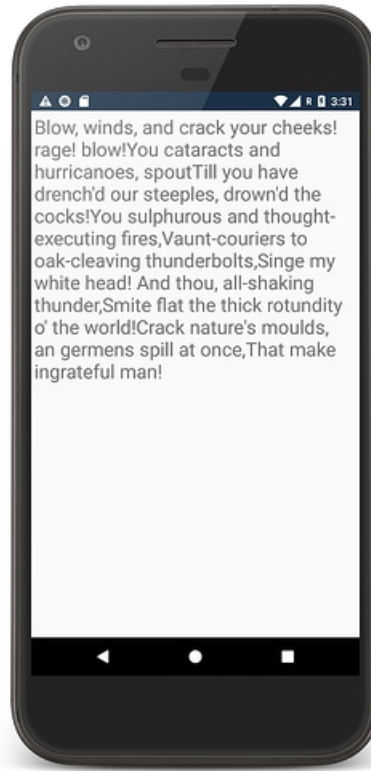
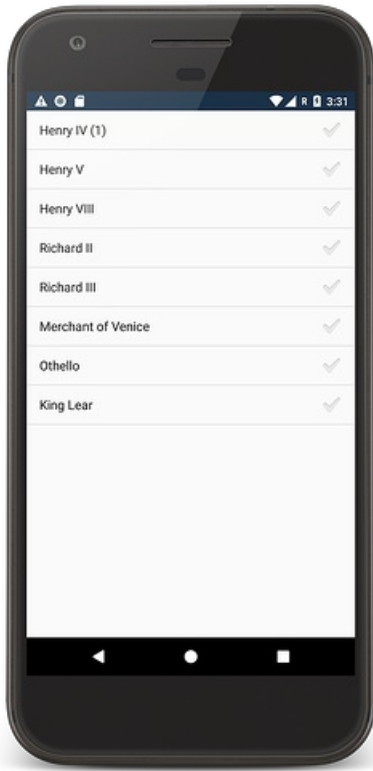
class 属性是一个有效替代 android:name。没有正式的指南在哪种形式是首选，有许多将使用的基本代码的示例 class 互换使用 android:name。

没有所需的 MainActivity 的代码更改。在该类中的代码应非常类似于此代码片段：

```
[Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
public class MainActivity : Activity
{
    protected override void onCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.activity_main);
    }
}
```

运行应用

现在，代码已完成，若要在操作中看到它在设备上运行该应用程序。

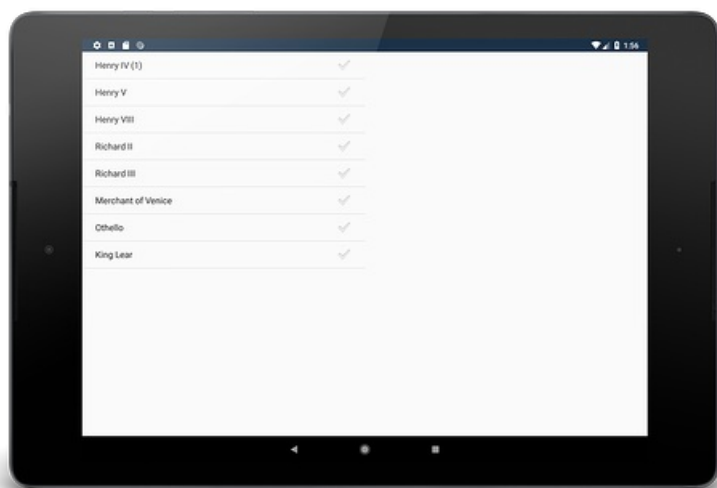


本演练的第 2 部分将 optimize 设备在横向模式下运行此应用程序。

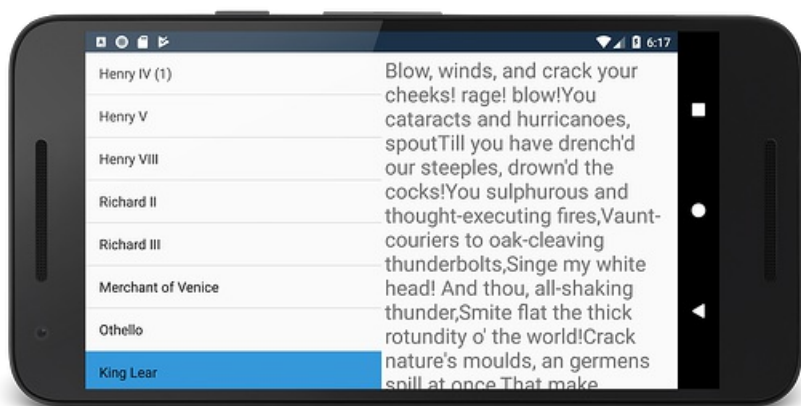
片段演练-横向

2018/10/26 • [Edit Online](#)

片段演练-第 1 部分演示了如何创建和使用面向较小屏幕手机上的 Android 应用中的片段。本演练的下一步是修改应用程序以充分利用平板电脑上的额外水平间距-将有一个活动，它将始终为播放列表 (`TitlesFragment`) 和 `PlayQuoteFragment` 将动态添加到用户所做的选择内容的响应中的活动：



在横向模式下运行的电话也将受益于此增强功能：



更新应用程序来处理横向方向

以下修改之处将生成执行的操作为基础 [片段演练-电话](#)

1. 创建备用布局，同时显示这两 `TitlesFragment` 和 `PlayQuoteFragment`。
2. 更新 `TitlesFragment` 来检测设备如果同时显示这两个片段，并相应地更改行为。

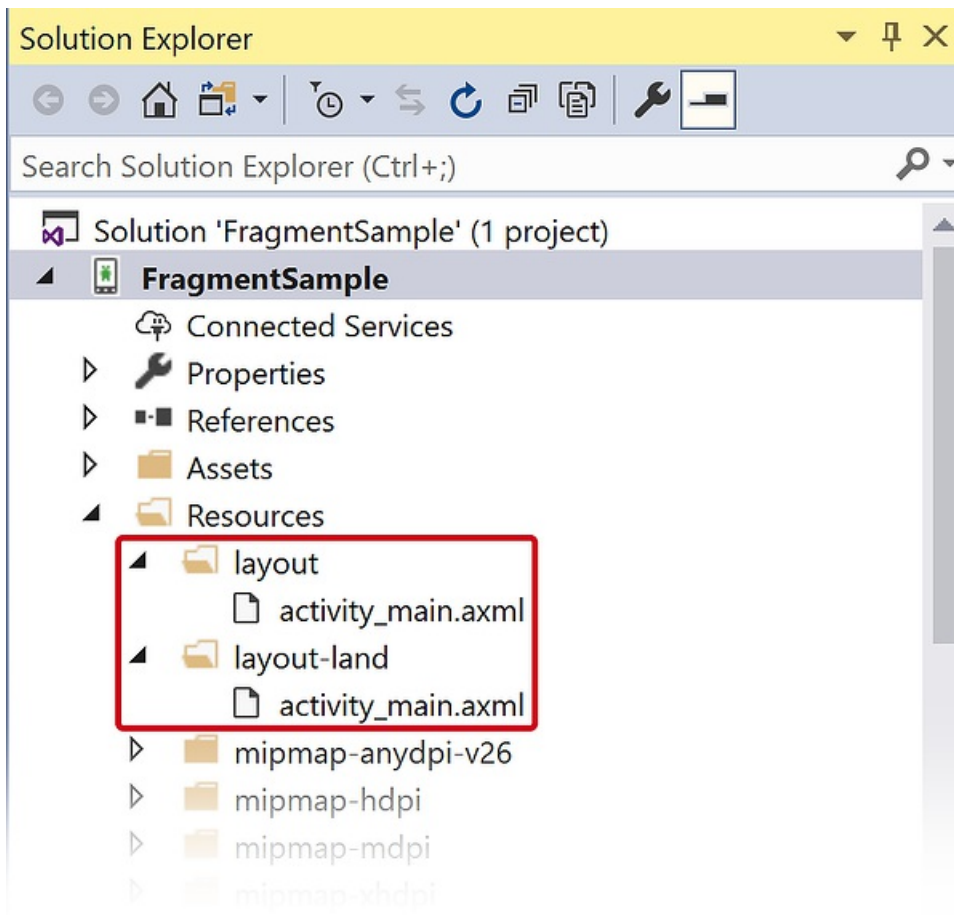
3. 更新 `PlayQuoteActivity` 关闭时在设备处于横向模式。

1.创建备用布局

Android 设备上创建主活动后, Android 将决定要加载的布局基于设备的方向。默认情况下, Android 将提供 **Resources/layout/activity_main.xml** 布局文件。对于在横向模式下加载的设备将提供 Android **Resources/layout-land/activity_main.xml** 布局文件。上的参考线[Android 资源](#)包含 Android 如何决定要为应用程序加载哪些资源文件的详细信息。

创建面向的备用布局横向中所述的步骤的方向 [备用布局](#) 指南。这应将新的布局资源文件添加到项目中, **Resources/layout/activity_main.xml**:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



在创建备用布局之后, 编辑该文件的源 **Resources/layout-land/activity_main.xml**, 使其匹配此 XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/two_fragments_layout"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="FragmentSample.TitlesFragment"
        android:id="@+id/titles"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />

    <FrameLayout android:id="@+id/playquote_container"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent"
        />
</LinearLayout>
```

活动的根视图提供的资源 ID `two_fragments_layout` 并且具有两个子视图 `fragment` 和一个 `FrameLayout`。虽然 `fragment` 静态加载 `FrameLayout` 充当将被替换为在运行时通过的“占位符” `PlayQuoteFragment`。每次在中选择新 play `TitlesFragment`，则 `playquote_container` 将使用的新实例更新 `PlayQuoteFragment`。

子视图的每个将占用其父级的最大高度。由控制每个子视图的宽度 `android:layout_weight` 和 `android:layout_width` 属性。在此示例中，每个子视图将占用 50% 的宽度由父提供。请参阅 [Google 的文档上 LinearLayout](#) 有关详细信息 `_布局权重_`。

2.对 TitlesFragment 的更改

一旦创建备用布局后，有必要更新 `TitlesFragment`。当应用显示的两个片段上一个活动，然后 `TitlesFragment` 应加载 `PlayQuoteFragment` 父活动中。否则为 `TitlesFragment` 应会启动 `PlayQuoteActivity` 哪台主机 `PlayQuoteFragment`。一个布尔型标志将帮助 `TitlesFragment` 确定应使用哪种行为。此标志将在初始化 `onActivityCreated` 方法。

首先，在顶部添加一个实例变量 `TitlesFragment` 类：

```
bool showingTwoFragments;
```

然后，添加以下代码片段 `onActivityCreated` 初始化该变量：

```
var quoteContainer = Activity.findViewById(Resource.Id.playquote_container);
showingTwoFragments = quoteContainer != null &&
    quoteContainer.Visibility == ViewStates.Visible;
if (showingTwoFragments)
{
    ListView.ChoiceMode = ChoiceMode.Single;
    ShowPlayQuote(selectedPlayId);
}
```

如果设备运行在横向模式下，则 `FrameLayout` 使用的资源 ID `playquote_container` 将显示在屏幕上，因此 `showingTwoFragments` 将初始化为 `true`。如果设备运行在纵向模式下，然后 `playquote_container` 不会在屏幕上，因此 `showingTwoFragments` 将 `false`。

`ShowPlayQuote` 方法将需要更改它如何显示引号-片段或启动新的活动。更新 `ShowPlayQuote` 方法以加载片段时显示两个片段，否则应启动活动：

```

void ShowPlayQuote(int playId)
{
    selectedPlayId = playId;
    if (showingTwoFragments)
    {
        ListView.SetItemChecked(selectedPlayId, true);

        var playQuoteFragment = FragmentManager.FindFragmentById(Resource.Id.playquote_container) as
        PlayQuoteFragment;

        if (playQuoteFragment == null || playQuoteFragment.PlayId != playId)
        {
            var container = Activity.FindViewById(Resource.Id.playquote_container);
            var quoteFrag = PlayQuoteFragment.NewInstance(selectedPlayId);

            FragmentTransaction ft = FragmentManager.BeginTransaction();
            ft.Replace(Resource.Id.playquote_container, quoteFrag);
            ft.Commit();
        }
    }
    else
    {
        var intent = new Intent(Activity, typeof(PlayQuoteActivity));
        intent.PutExtra("current_play_id", playId);
        StartActivity(intent);
    }
}

```

如果用户已经选择了不同于当前显示在播放 `PlayQuoteFragment`，然后一个新 `PlayQuoteFragment` 创建并将内容替换为 `playquote_container` 的上下文中 `FragmentTransaction`。

TitlesFragment 的完整代码

在完成到以前的所有更改后 `TitlesFragment`，完整的类应与此代码相匹配：

```

public class TitlesFragment : ListFragment
{
    int selectedPlayId;
    bool showingTwoFragments;

    public override void OnActivityCreated(Bundle savedInstanceState)
    {
        base.OnActivityCreated(savedInstanceState);
        ListAdapter = new ArrayAdapter<string>(Activity, Android.Resource.Layout.SimpleListItemActivated1,
        Shakespeare.Titles);

        if (savedInstanceState != null)
        {
            selectedPlayId = savedInstanceState.GetInt("current_play_id", 0);
        }

        var quoteContainer = Activity.FindViewById(Resource.Id.playquote_container);
        showingTwoFragments = quoteContainer != null &&
            quoteContainer.Visibility == ViewStates.Visible;
        if (showingTwoFragments)
        {
            ListView.ChoiceMode = ChoiceMode.Single;
            ShowPlayQuote(selectedPlayId);
        }
    }

    public override void OnSaveInstanceState(Bundle outState)
    {
        base.OnSaveInstanceState(outState);
        outState.PutInt("current_play_id", selectedPlayId);
    }
}

```

```

    }

    public override void OnListItemClick(ListView l, View v, int position, long id)
    {
        ShowPlayQuote(position);
    }

    void ShowPlayQuote(int playId)
    {
        selectedPlayId = playId;
        if (showingTwoFragments)
        {
            ListView.SetItemChecked(selectedPlayId, true);

            var playQuoteFragment = FragmentManager.FindFragmentById(Resource.Id.playquote_container) as
            PlayQuoteFragment;

            if (playQuoteFragment == null || playQuoteFragment.PlayId != playId)
            {
                var container = Activity.FindViewById(Resource.Id.playquote_container);
                var quoteFrag = PlayQuoteFragment.NewInstance(selectedPlayId);

                FragmentTransaction ft = FragmentManager.BeginTransaction();
                ft.Replace(Resource.Id.playquote_container, quoteFrag);
                ft.AddToBackStack(null);
                ft.SetTransition(FragmentTransit.FragmentFade);
                ft.Commit();
            }
        }
        else
        {
            var intent = new Intent(Activity, typeof(PlayQuoteActivity));
            intent.PutExtra("current_play_id", playId);
            StartActivity(intent);
        }
    }
}

```

3.对 PlayQuoteActivity 的更改

没有一个最终的详细信息，需要注意：`PlayQuoteActivity` 时在设备处于横向模式下不需要。如果设备在横向模式下 `PlayQuoteActivity` 应该是不可见。更新 `OnCreate` 方法的 `PlayQuoteActivity`，以便它将关闭本身。此代码是最终版本 `PlayQuoteActivity.OnCreate`：

```

protected override void OnCreate(Bundle savedInstanceState)
{
    base.OnCreate(savedInstanceState);

    if (Resources.Configuration.Orientation == Android.Content.Res.Orientation.Landscape)
    {
        Finish();
    }

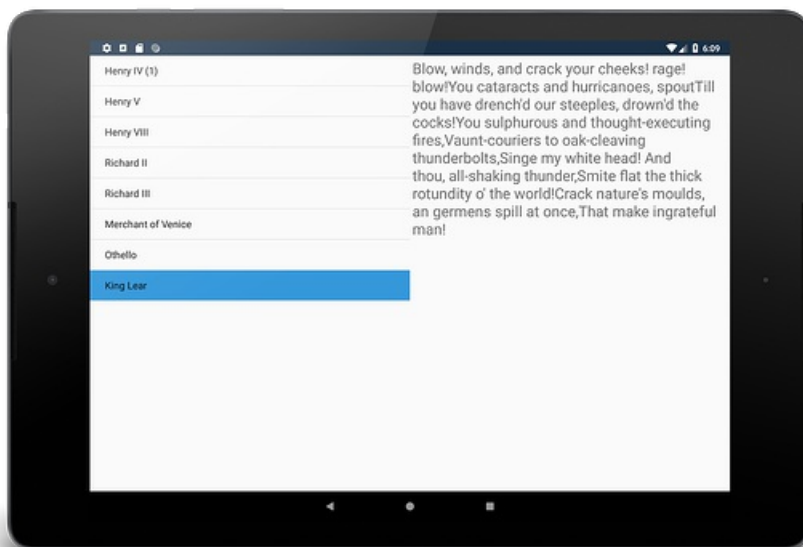
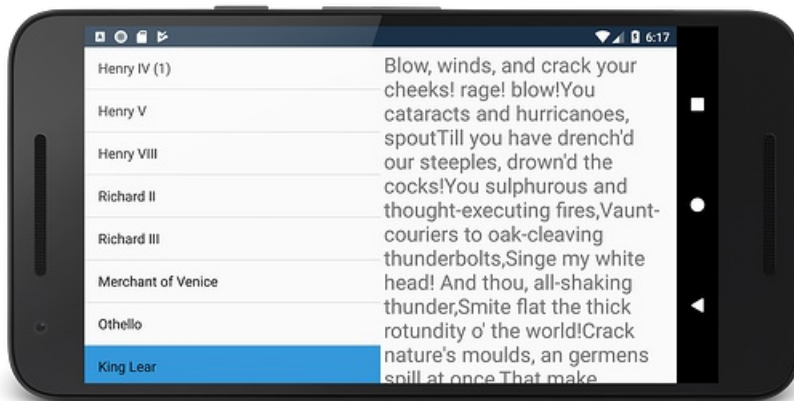
    var playId = Intent.Extras.GetInt("current_play_id", 0);
    var playQuoteFrag = PlayQuoteFragment.NewInstance(playId);
    FragmentManager.BeginTransaction()
        .Add(Android.Resource.Id.Content, playQuoteFrag)
        .Commit();
}

```

此修改将添加对设备方向的检查。如果是，在横向模式下则 `PlayQuoteActivity` 关闭本身。

4.运行此应用程序

完成这些更改后，运行应用时，旋转为横向模式下（如有必要），该设备，然后选择播放。应得的播放列表在同一屏幕上显示引号：



创建片段

2018/10/26 • [Edit Online](#)

若要创建一个片段，类必须继承 `Android.App.Fragment`，然后重写 `OnCreateView` 方法。`OnCreateView` 它是时候将片段放在屏幕上，并将返回时将调用由托管活动 `View`。典型 `OnCreateView` 将创建此 `View` 由以下布局文件，然后将其附加到父容器。容器的特征是父的非常重要，因为 Android 将应用于该片段的 UI 布局参数。下面的示例阐释了这一点：

```
public override View OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
{
    return inflater.Inflate(Resource.Layout.Example_Fragment, container, false);
}
```

上面的代码将放在视图大量 `Resource.Layout.Example_Fragment`，并将其添加到的子视图作为 `ViewGroup` 容器。

NOTE

片段类必须具有公共默认无参数构造函数。

将片段添加到活动

有两种方法可以托管一个片段活动内部的：

- **以声明方式**—可以以声明方式中使用片段 `.axml` 使用的布局文件 `<Fragment>` 标记。
- **以编程方式**—片段可以还实例化动态使用 `FragmentManager` 类的 API。

通过以编程方式使用 `FragmentManager` 类将本指南后面所述。

以声明方式使用段

添加布局内的一个片段需要使用 `<fragment>` 标记，然后通过提供可以确定片段 `class` 属性或 `android:name` 属性。以下代码片段演示如何使用 `class` 属性来声明 `fragment`：

```
<?xml version="1.0" encoding="utf-8"?>
<fragment class="com.xamarin.sample.fragments.TitlesFragment"
    android:id="@+id/titles_fragment"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

此下一个代码段演示了如何声明 `fragment` 通过使用 `android:name` 属性来标识片段类：

```
<?xml version="1.0" encoding="utf-8"?>
<fragment android:name="com.xamarin.sample.fragments.TitlesFragment"
    android:id="@+id/titles_fragment"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

Android 创建活动后，将实例化布局文件中指定每个片段和插入从创建该视图 `OnCreateView` 来代替 `Fragment` 元素。以声明方式添加到活动的片段是静态的将保留在活动中，直到销毁；不能动态替换或删除附加到的活动的生存期内的片段。

每个片段必须分配有唯一标识符：

- **android:id** –因为使用布局文件中的其他 UI 元素，这是唯一的 id。
- **android:tag** –此属性是唯一的字符串。

如果上述两种方法都不使用时，该片段将假定容器视图的 ID。在下面的示例其中既不 `android:id` 也不 `android:tag` 提供 Android 将向分配 ID `fragment_container` 到段：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment class="com.example.android.apis.app.TitlesFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

包名称大小写

Android 不允许在包名称; 中的大写字符尝试放大量视图，如果包名称包含大写字符时，它将引发异常。但是，Xamarin.Android 内更能容忍，并且将容忍在命名空间中的大写字符。

例如，这两个以下的代码片段将使用 Xamarin.Android。但是，第二个代码段将导致 `android.view.InflateException` 要引发的纯基于 Java 的 Android 应用程序。

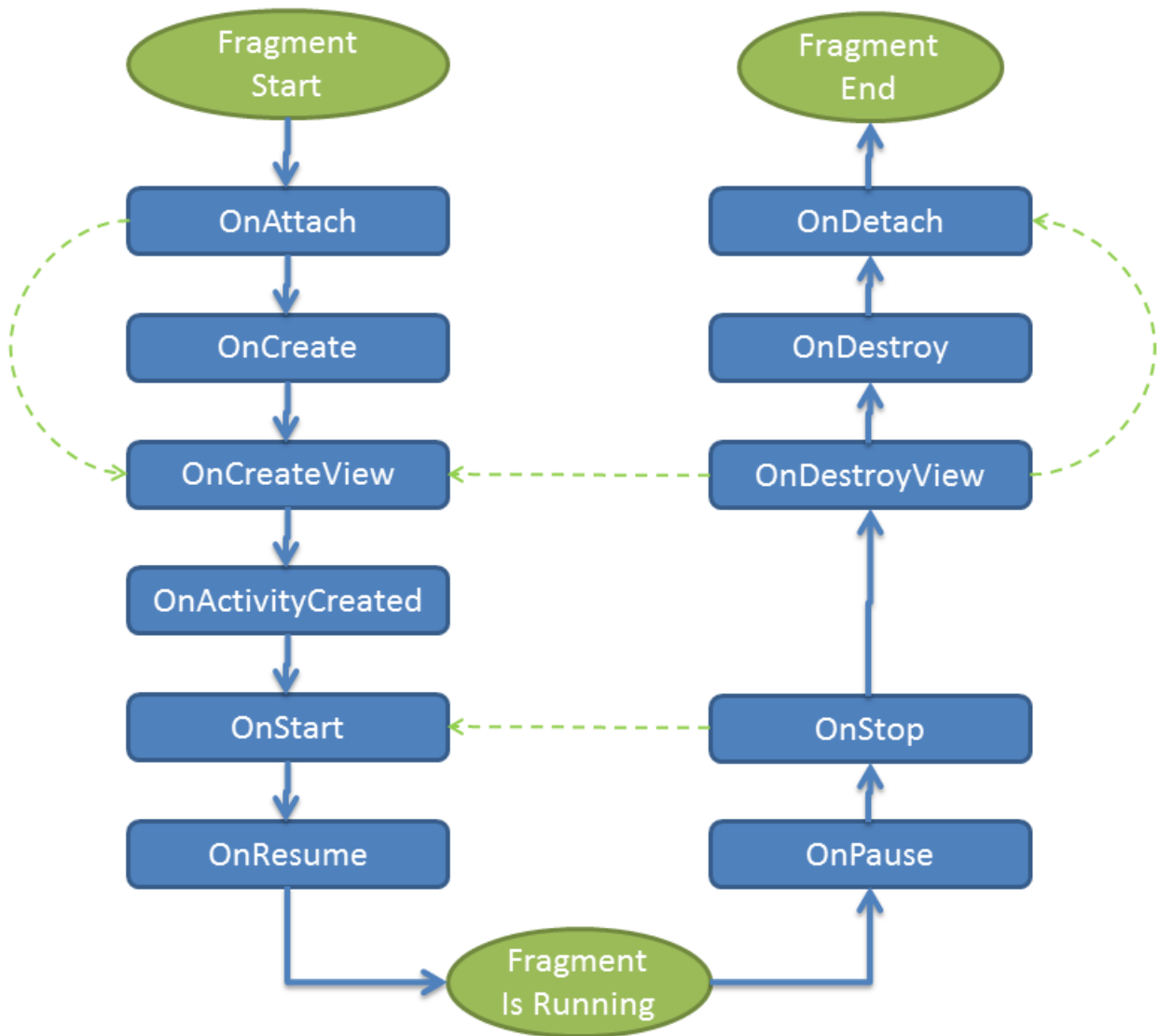
```
<fragment class="com.example.DetailsFragment" android:id="@+id/fragment_content"
    android:layout_width="match_parent" android:layout_height="match_parent" />
```

或

```
<fragment class="Com.Example.DetailsFragment" android:id="@+id/fragment_content"
    android:layout_width="match_parent" android:layout_height="match_parent" />
```

片段生命周期

片段具有一定程度上独立的但仍受其自身生命周期[托管的活动的生命周期](#)。例如，当活动暂停，所有其关联的片段将暂停。下图概述了片段的生命周期。



片段创建生命周期方法

下面列出了各种回调的流中片段的生命周期，在创建时：

- `OnInflate()` – 片段正在创建的视图布局的一部分时调用。这不能从 XML 布局文件以声明方式创建该片段后立即调用。片段不是其活动与相关联，但活动，捆绑，并 `AttributeSet` 从视图层次结构中作为参数传递。此方法最适合用于分析 `AttributeSet` 和保存属性可能会使用更高版本的片段。
- `OnAttach()` – 片段是与活动关联后调用。这是该片段是可供使用时要运行的第一个方法。一般情况下，片段不应实现一个构造函数或重写默认构造函数。应在此方法中初始化所需的片段的任何组件。
- `OnCreate()` – 要创建片段的活动由调用。调用此方法时，托管活动的视图层次结构可能不是完全实例化，因此该片段不应依赖直到更高版本上的片段的生命周期中的活动的视图层次结构的任何部分。例如，不要使用此方法以执行任何调整或应用程序用户界面调整。这是片段可能会开始收集数据所需的最早时间。片段现在运行在 UI 线程中，因此避免任何耗时较长的处理，或在后台线程上执行该处理。此方法可能会跳过，如果 `SetRetainInstance(true)` 调用。将下面更详细地介绍此替代方法。
- `OnCreateView()` – 创建片段的视图。此方法调用一次活动的 `OnCreate()` 方法已完成。此时，它是安全地与该活动的视图层次结构进行交互。此方法应返回将由该片段的视图。
- `OnActivityCreated()` – 之后调用 `Activity.OnCreate` 由托管活动已完成。此时应执行最终用户界面的调整。
- `OnStart()` – 包含活动已恢复后调用。这使得该片段对用户可见。在许多情况下，片段将包含代码，否则会采用 `onstart()` 活动的方法。
- `OnResume()` – 这是最后一个方法调用之前用户可以进行交互的片段。应在此方法中执行的代码的类的一个示例将启用功能的设备的用户可能会与进行交互，如照相机的位置服务。不过，这些服务可能会导致过多的

电池耗尽，并应用程序应尽可能少地使用以保持电池寿命。

片段析构生命周期方法

下一步的列表说明了调用正在销毁片段的生命周期方法：

- `OnPause()` – 用户不再能够与片段进行交互。这种情况存在由于某个其他片段操作修改此代码段，或托管活动已暂停。就可以托管此片段的活动可能仍是可见、焦点中的活动，即部分透明的或未占据整个屏幕。当此方法变为活动状态时，它是用户将离开该片段的第一个指示。片段应保存任何更改。
- `OnStop()` – 片段不再可见。主机活动可能会停止，或片段操作在活动中修改它。此回调提供相同的目的的**Activity.OnStop**。
- `OnDestroyView()` – 调用此方法来清理与视图关联的资源。与片断关联的视图已被销毁时调用。
- `OnDestroy()` – 不再使用该片段时，调用此方法。仍然与活动相关联，但该片段不再起作用。此方法应释放任何资源使用的片段，如 **SurfaceView** 可能用于相机。此方法可能会跳过，如果**SetRetainInstance(true)** 调用。将下面更详细地介绍此替代方法。
- `OnDetach()` – 该片段将不再与活动关联之前，调用此方法。片段的视图层次结构不再存在，并且应在此时释放片断所使用的所有资源。

使用 **SetRetainInstance**

很可能要指定，它不应完全销毁活动正被重新创建的片段。`Fragment` 类提供了方法 `SetRetainInstance` 实现此目的。如果 `true` 传递给此方法，则重新启动该活动时，将使用该片段的同一个实例。如果发生这种情况，则所有回叫方法将调用除外 `OnCreate` 和 `OnDestroy` 生命周期的回调。在上面所示（以绿色的点线）生命周期关系图中阐释了此流程。

片断状态管理

片段可能保存和片段生命周期内使用的实例还原其状态 `Bundle`。此捆绑包允许将数据保存为键/值对的片段和适用于的简单数据，无需太多内存。片段可以保存其状态，通过调用 `OnSaveInstanceState`：

```
public override void OnSaveInstanceState(Bundle outState)
{
    base.OnSaveInstanceState(outState);
    outState.PutInt("current_choice", _currentCheckPosition);
}
```

当创建片段的新实例，在保存的状态 `Bundle` 将变为可用的新实例通过 `OnCreate`，`OnCreateView`，和 `OnActivityCreated` 方法的新实例。下面的示例演示如何检索的值 `current_choice` 从 `Bundle`：

```
public override void OnActivityCreated(Bundle savedInstanceState)
{
    base.OnActivityCreated(savedInstanceState);
    if (savedInstanceState != null)
    {
        _currentCheckPosition = savedInstanceState.GetInt("current_choice", 0);
    }
}
```

重写 `OnSaveInstanceState` 是用于保存暂时性数据片段中跨方向更改，如适当机制 `current_choice` 上述示例中的值。但是的默认实现 `OnSaveInstanceState` 负责的暂时性数据保存在分配的 id 的每个视图的 UI。例如，查看应用程序具有 `EditText` 元素在 XML 中定义，如下所示：

```
<EditText android:id="@+id/myText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

由于 `EditText` 控件具有 `id` 分配，片段会自动将数据保存在小组件时 `onSaveInstanceState` 调用。

捆绑包限制

尽管使用 `onSaveInstanceState` 使我们可以很容易保存暂时性数据，使用此方法具有一些限制：

- 如果该片段不会添加到 back 堆栈中，则当用户按下时，其状态将不会还原回按钮。
- 当绑定用于保存数据时，该数据序列化。这可能会导致处理延迟。

参与到菜单

片段可能会导致其托管活动的菜单项。活动首次处理菜单项。如果活动没有处理程序，然后该事件将传递到段，然后将处理。

若要添加到活动的菜单项，片段必须做两件事。首先，该片段必须实现的方法 `onCreateOptionsMenu` 并将其项放置到菜单中，如下面的代码中所示：

```
public override void onCreateOptionsMenu(IMenu menu, MenuInflater menuInflater)
{
    menuInflater.inflate(Resource.Menu.menu_fragment_vehicle_list, menu);
    base.onCreateOptionsMenu(menu, menuInflater);
}
```

在前面的代码片段菜单项从位于文件中的以下 XML `menu_fragment_vehicle_list.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/add_vehicle"
        android:icon="@drawable/ic_menu_add_data"
        android:title="@string/add_vehicle" />
</menu>
```

接下来，必须调用片段 `setHasOptionsMenu(true)`。调用此方法向 Android 通知片段具有可用于参与选项菜单的菜单项。除非对此方法的调用，该片段的菜单项将不添加到活动的选项菜单。这通常是在生命周期方法 `onCreate()` 下，一步的代码片段中所示：

```
public override void onCreate(Bundle savedInstanceState)
{
    base.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}
```

以下屏幕显示了此菜单的外观：



6:30

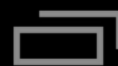


My Trips



Export To CSV

Add Vehicle



管理片段

2018/10/26 • [Edit Online](#)

为了帮助管理片段, Android 提供 `FragmentManager` 类。每个活动具有实例 `Android.App.FragmentManager`, 将查找或动态更改其片段。这些更改的每个组称为事务, 并使用该类中包含的 Api 之一执行 `Android.App.FragmentTransation`, 它由托管 `FragmentManager`。活动可能会启动此类事务:

```
FragmentTransaction fragmentTx = this.FragmentManager.BeginTransaction();
```

在执行这些更改到片段 `FragmentTransaction` 实例使用的方法类似于 `Add()`, `Remove()`, 并 `Replace()`。所做的更改使用, 然后应用 `Commit()`。在事务中的更改不会立即执行。相反, 它们被计划尽可能快地运行活动的 UI 线程上。

下面的示例演示如何将片段添加到现有的容器:

```
// Create a new fragment and a transaction.
FragmentTransaction fragmentTx = this.FragmentManager.BeginTransaction();
DetailsFragment aDifferentDetailsFrag = new DetailsFragment();

// The fragment will have the ID of Resource.Id.fragment_container.
fragmentTx.Add(Resource.Id.fragment_container, aDifferentDetailsFrag);

// Commit the transaction.
fragmentTx.Commit();
```

如果一个事务被提交之后 `Activity.OnSaveInstanceState()` 是调用, 就会引发异常。这是因为当该活动将保存其状态, Android 会保存任何托管片段的状态。如果任何片段的事务提交此点后, 这些事务的状态将丢失时还原该活动中。

可以将片段事务保存到活动的 **back 堆栈** 由调用 `FragmentTransaction.AddToBackStack()`。这允许用户通过向后导航片段更改何时回按下按钮。而无需对此方法的调用, 将删除的片段将被销毁和将用户导航返回到活动的情况下不可用。

下面的示例演示如何使用 `AddToBackStack` 方法的 `FragmentTransaction` 替换一个片段中, 同时保留后退堆栈上的第一个片段的状态:

```
// Create a new fragment and a transaction.
FragmentTransaction fragmentTx = this.FragmentManager.BeginTransaction();
DetailsFragment aDifferentDetailsFrag = new DetailsFragment();

// Replace the fragment that is in the View fragment_container (if applicable).
fragmentTx.Replace(Resource.Id.fragment_container, aDifferentDetailsFrag);

// Add the transaction to the back stack.
fragmentTx.AddToBackStack(null);

// Commit the transaction.
fragmentTx.Commit();
```

使用片段进行通信

`FragmentManager` 知道有关所有附加到活动的片段, 并提供两种方法来帮助查找这些片段:

- **FindFragmentById** - 此方法将查找一个片段作为事务的一部分添加的片段时使用的布局文件中指定的 ID

或容器 ID。

- **FindFragmentByTag** –使用此方法来查找已标记的布局文件中提供或在事务中已添加的片段。

片段和活动引用 `FragmentManager`，因此相同的方法使用了它们之间的往返通信。应用程序可能会使用这两种方法之一查找片段的引用、强制转换为适当的类型，该引用，然后在片段上直接调用方法。以下代码片段提供了一个示例：

还有可能要使用的活动的 `FragmentManager` 查找片段：

```
var emailList = FragmentManager.FindFragmentById<EmailListFragment>(Resource.Id.email_list_fragment);
emailList.SomeCustomMethod(parameter1, parameter2);
```

与活动通信

很可能要使用的片段 `Fragment.Activity` 属性来引用其主机。通过强制转换到更具体的类型的活动，它是调用方法和属性在其主机上，活动可能，如下面的示例中所示：

```
var myActivity = (MyActivity) this.Activity;
myActivity.SomeCustomMethod();
```

专用的片段类

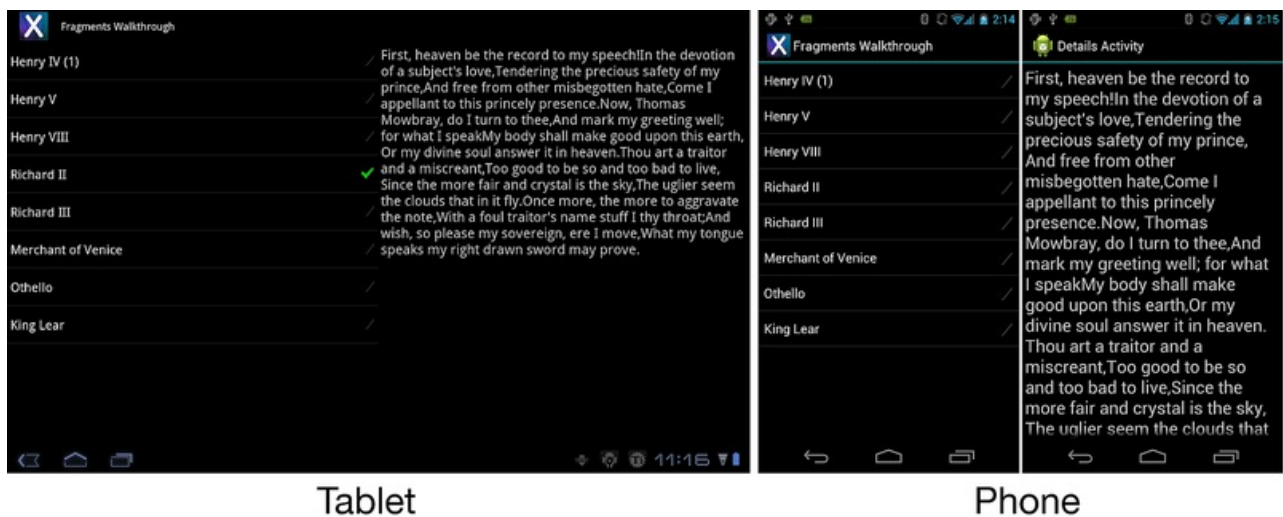
2018/10/26 • [Edit Online](#)

片段 API 提供了封装应用程序中的更常见功能的一些其他子类。这些子类是：

- **ListFragment** –此片段用于显示的项绑定到如数组或游标的数据源的列表。
- **DialogFragment** –作为包装一个对话框，使用此片段。该片段将显示在其活动之上对话框。
- **PreferenceFragment** –此片段用于显示作为列表的首选项对象。

ListFragment

`ListFragment` 的概念和功能提供给非常类似 `ListActivity`；它是托管的包装器 `ListView` 片段中。下的图显示 `ListFragment` 平板电脑和手机上运行：



Tablet

Phone

绑定数据和 ListAdapter

`ListFragment` 类已经提供了默认的布局，因此不需要重写 `OnCreateView` 若要显示的内容 `ListFragment`。 `ListView` 通过使用绑定到数据 `ListAdapter` 实现。下面的示例演示如何这也可以由通过将简单的字符串数组：

```
public override void OnActivityCreated(Bundle savedInstanceState)
{
    base.OnActivityCreated(savedInstanceState);
    string[] values = new[] { "Android", "iPhone", "WindowsMobile",
        "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
        "Linux", "OS/2" };
    this.ListAdapter = new ArrayAdapter<string>(Activity, Android.Resource.Layout.SimpleExpandableListItem1,
        values);
}
```

设置时 `ListAdapter`，务必要使用 `ListFragment.ListAdapter` 属性，而不 `ListView.ListAdapter` 属性。使用 `ListView.ListAdapter` 将导致重要的初始化代码要跳过。

响应用户选择

若要响应用户选择，应用程序必须重写 `OnListItemClick` 方法。下面的示例显示了一个这种可能性：


```

public override void OnListItemClick(ListView l, View v, int index, long id)
{
    // We can display everything in place with fragments.
    // Have the list highlight this item and show the data.
    ListView.SetItemChecked(index, true);

    // Check what fragment is shown, replace if needed.
    var details = FragmentManager.FindFragmentById<DetailsFragment>(Resource.Id.details);
    if (details == null || details.ShownIndex != index)
    {
        // Make new fragment to show this selection.
        details = DetailsFragment.NewInstance(index);

        // Execute a transaction, replacing any existing
        // fragment with this one inside the frame.
        var ft = FragmentManager.BeginTransaction();
        ft.Replace(Resource.Id.details, details);
        ft.SetTransition(FragmentTransit.FragmentFade);
        ft.Commit();
    }
}

```

在上面的代码，当用户选择中的项 `ListFragment`，托管活动，其中显示有关所选的项的更多详细信息中显示新片段。

DialogFragment

DialogFragment 是用于显示一个对话框对象内的活动窗口的顶部的浮动的片段的片段。它旨在替换为托管对话框中（从开始在 Android 3.0 中）的 `Api`。以下屏幕截图显示的示例 `DialogFragment`：



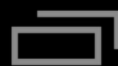
My Trips



Add New Vehicle

RAV 4

Save



一个 `DialogFragment` 可确保该片段和对话框之间的状态保持一致。所有的交互和对话框对象的控件应执行的操作通过 `DialogFragment` API, 并不会执行与直接调用对话框对象上。 `DialogFragment` API 提供了与每个实例 `Show()` 方法, 用来显示一个片段。有两种方法, 若要消除的片段:

- 调用 `DialogFragment.Dismiss()` 上 `DialogFragment` 实例。
- 显示另一个 `DialogFragment` 。

若要创建 `DialogFragment`, 一个类继承自 `Android.App.DialogFragment`, 和设置将覆盖以下两种方法之一:

- **OnCreateView** -这将创建并返回的视图。
- **OnCreateDialog** -这将创建一个自定义对话框。它通常用于显示 `AlertDialog`。当重写此方法, 不需要重写 `OnCreateView` 。

简单 `DialogFragment`

以下屏幕截图显示了一个简单 `DialogFragment`, 其 `TextView` 并将两个 `Button` s:



AndroidDialogFragment

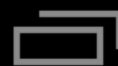
Text

Hello World, Click Me!

This has been displayed 1 times.
You clicked the button 8 times.

Click Me

Dismiss Dialog



`TextView` 将显示的用户已单击了其中的一个按钮的次数 `DialogFragment`，而单击其他按钮将关闭该片段。有关代码 `DialogFragment` 是：

```
public class MyDialogFragment : DialogFragment
{
    private int _clickCount;
    public override void OnCreate(Bundle savedInstanceState)
    {
        _clickCount = 0;
    }

    public override View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState)

        var view = inflater.Inflate(Resource.Layout.dialog_fragment_layout, container, false);
        var textView = view.FindViewById<TextView>(Resource.Id.dialog_text_view);

        view.FindViewById<Button>(Resource.Id.dialog_button).Click += delegate
        {
            textView.Text = "You clicked the button " + _clickCount++ + " times.";
        };

        // Set up a handler to dismiss this DialogFragment when this button is clicked.
        view.FindViewById<Button>(Resource.Id.dismiss_dialog_button).Click += (sender, args) => Dismiss();
        return view;
    }
}
```

显示片段

像所有片段 `DialogFragment` 显示的上下文中 `FragmentManager`。

`Show()` 方法 `DialogFragment` 采用 `FragmentManager` 和一个 `string` 作为输入。对话将添加到活动和 `FragmentManager` 提交。

以下代码演示了一个活动可能会使用一种 `Show()` 方法来显示 `DialogFragment`：

```
public void ShowDialog()
{
    var transaction = FragmentManager.BeginTransaction();
    var dialogFragment = new MyDialogFragment();
    dialogFragment.Show(transaction, "dialog_fragment");
}
```

正在取消片段

调用 `Dismiss()` 的实例上 `DialogFragment` 导致要从活动中删除的片段并提交该事务。将调用标准的片段生命周期方法涉及使用片段的析构。

警报对话框

而不是替代 `onCreateView`，一个 `DialogFragment` 可能会改为重写 `onCreateDialog`。这允许应用程序来创建 `AlertDialog` 由一个片段。下面的代码是使用的示例， `AlertDialog.Builder` 若要创建 `Dialog`：

```
public class AlertDialogFragment : DialogFragment
{
    public override Dialog OnCreateDialog(Bundle savedInstanceState)
    {
        EventHandler<DialogClickEventArgs> okhandler;
        var builder = new AlertDialog.Builder(Activity)
            .SetMessage("This is my dialog.")
            .SetPositiveButton("Ok", (sender, args) =>
            {
                // Do something when this button is clicked.
            })
            .SetTitle("Custom Dialog");
        return builder.Create();
    }
}
```

PreferenceFragment

为了帮助管理首选项，该片段 API 提供了 `PreferenceFragment` 子类。`PreferenceFragment` 类似于 [PreferenceActivity](#) – 它会在片段中显示给用户的首选项的层次结构。当用户首选项与交互时，它们将自动保存到 [SharedPreferences](#)。在 Android 3.0 或更高版本的应用程序中，使用 `PreferenceFragment` 处理应用程序中的首选项。下图显示的示例

`PreferenceFragment`：



PreferenceFragmentSample

INLINE PREFERENCES

Checkbox Preference Title

Checkbox Preference Summary



DIALOG BASED PREFERENCES

EditText Preference Title

EditText Preference Summary

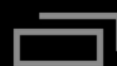
LAUNCH PREFERENCES

Title Screen Preferences

Summary Screen Preferences

Intent Preference Title

Intent Preference Summary



从资源中创建首选项片段

片段可能由使用放大从 XML 资源文件的首选项 [PreferenceFragment.AddPreferencesFromResource](#) 方法。逻辑起点片段的生命周期中调用此方法会采用 `OnCreate` 方法。

`PreferenceFragment` 图所示更高版本已通过从 XML 加载资源。资源文件是：

```
<?xml version="1.0" encoding="utf-8"?>

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory android:title="Inline Preferences">
        <CheckBoxPreference android:key="checkbox_preference"
            android:title="Checkbox Preference Title"
            android:summary="Checkbox Preference Summary" />
    </PreferenceCategory>

    <PreferenceCategory android:title="Dialog Based Preferences">

        <EditTextPreference android:key="edittext_preference"
            android:title="EditText Preference Title"
            android:summary="EditText Preference Summary"
            android:dialogTitle="Edit Text Preference Dialog Title" />

    </PreferenceCategory>

    <PreferenceCategory android:title="Launch Preferences">

        <!-- This PreferenceScreen tag serves as a screen break (similar to page break
            in word processing). Like for other preference types, we assign a key
            here so it is able to save and restore its instance state. -->
        <PreferenceScreen android:key="screen_preference"
            android:title="Title Screen Preferences"
            android:summary="Summary Screen Preferences">

            <!-- You can place more preferences here that will be shown on the next screen. -->

            <CheckBoxPreference android:key="next_screen_checkbox_preference"
                android:title="Next Screen Toggle Preference Title"
                android:summary="Next Screen Toggle Preference Summary" />

        </PreferenceScreen>

        <PreferenceScreen android:title="Intent Preference Title"
            android:summary="Intent Preference Summary">

            <intent android:action="android.intent.action.VIEW"
                android:data="http://www.android.com" />

        </PreferenceScreen>

    </PreferenceCategory>

</PreferenceScreen>
```

片段的首选项的代码如下所示：


```
public class PrefFragment : PreferenceFragment
{
    public override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        AddPreferencesFromResource(Resource.Xml.preferences);
    }
}
```

查询活动创建的首选项片段

另一种方法创建 `PreferenceFragment` 涉及查询活动。每个活动可以使用 [元数据_密钥_首选项](#) 将指向的 XML 资源文件的属性。在 Xamarin.Android 中, 这是通过装饰的活动 `MetaDataAttribute`, 然后指定要使用的资源文件。

`PreferenceFragment` 类提供了方法 [AddPreferenceFromIntent](#) 可用于查询用于查找此 XML 资源和放大量有关它的首选项层次结构的活动。

在以下代码片段中, 它使用提供了此过程的一个示例 `AddPreferencesFromIntent` 若要创建 `PreferenceFragment`:

```
public class MyPreferenceFragment : PreferenceFragment
{
    public override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        var intent = new Intent(this.Activity, typeof (MyActivityWithPreferences));
        AddPreferencesFromIntent(intent);
    }
}
```

Android 将看看的类 `MyActivityWithPreference`。类必须标有 `MetaDataAttribute`, 如下面的代码段中所示:

```
[Activity(Label = "My Activity with Preferences")]
[MetaData(PreferenceManager.MetadataKeyPreferences, Resource = "@xml/preference_from_intent")]
public class MyActivityWithPreferences : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        // This is deliberately blank
    }
}
```

`MetaDataAttribute` 声明的 XML 资源文件 `PreferenceFragment` 将使用放大量首选项层次结构。如果

`MetatDataAttribute` 未提供, 则在运行时将引发异常。此代码运行时, `PreferenceFragment` 显示如以下屏幕截图中所示:



My Activity

INLINE PREFERENCES

Checkbox Preference Title

Checkbox Preference Summary



DIALOG BASED PREFERENCES

EditText Preference Title

EditText Preference Summary

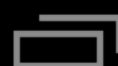
LAUNCH PREFERENCES

Title Screen Preferences

Summary Screen Preferences

Intent Preference Title

Intent Preference Summary



提供向后兼容性与 Android 支持包

2018/10/26 • [Edit Online](#)

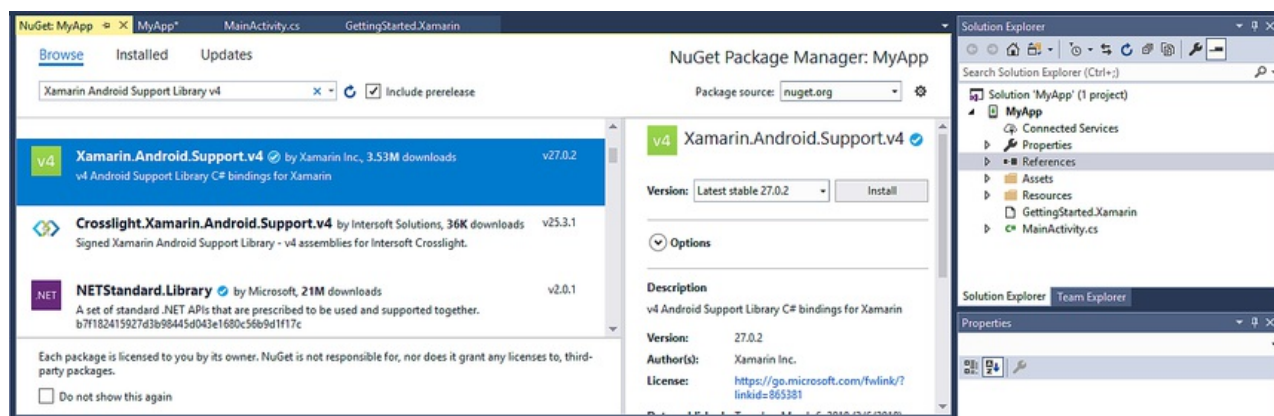
片段的有效性是有限而无需向后兼容性与预 Android 3.0 (API 级别 11) 设备。若要提供此功能, Google 引入了[支持库](#)(最初称为*Android Compatibility Library*发布时) 的 backports 的一些从较新版本的 Api 与旧版本的 Android android。它是 Android 支持包, 它使运行 Android 2.3.3 Android 1.6 (API 级别 4) 的设备。(API 级别 10)。

NOTE

仅 `ListFragment` 和 `DialogFragment` 通过 Android 支持包。没有其他片段子类, 如 `PreferenceFragment`, 所支持的 Android 支持包。它们不会预先 Android 3.0 应用程序中。

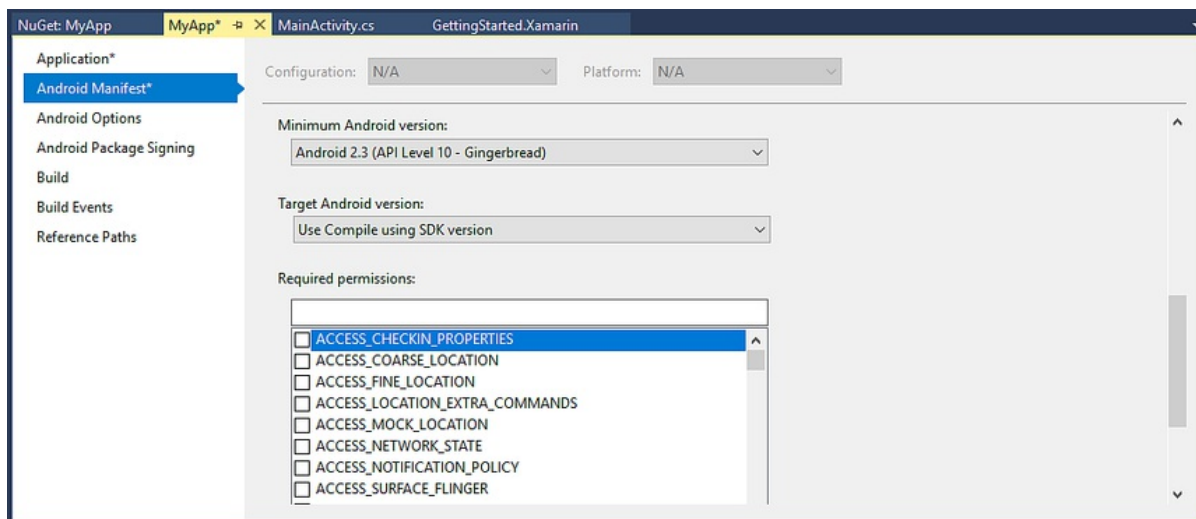
添加支持包

Android 支持包不会自动添加到 Xamarin.Android 应用程序。Xamarin 提供了[Android 支持库 v4 NuGet 包](#)来简化支持库添加到 Xamarin.Android 应用程序。将支持包包含到你的 Xamarin.Android 应用程序包含[Android 支持库 v4](#)组件到 Xamarin.Android 项目中, 如下屏幕截图中所示:



在执行这些步骤后, 就可以使用在早期版本的 Android 片段。片段 Api 将工作相同现在在这些早期版本, 但存在以下例外:

- **更改最低 Android 版本**–应用程序不再需要面向 Android 3.0 或更高版本, 如下所示:



- **扩展 FragmentActivity**–承载片段的活动必须现在继承自 `Android.Support.V4.App.FragmentActivity`, 而不是从 `Android.App.Activity`。

- **更新命名空间**—类继承自 `Android.App.Fragment` 现在必须继承自 `Android.Support.V4.App.Fragment`。删除正在使用语句" `using Android.App;` "顶部的源代码文件, 并将其替换为" `using Android.Support.V4.App` "。
- **使用 `SupportFragmentManager`** — `Android.Support.V4.App.FragmentActivity` 公开 `SupportFragmentManager` 属性, 必须用来实现的引用 `FragmentManager`。例如:

```
FragmentTransaction fragmentTx = this.SupportingFragmentManager.BeginTransaction();
DetailsFragment detailsFrag = new DetailsFragment();
fragmentTx.Add(Resource.Id.fragment_container, detailsFrag);
fragmentTx.Commit();
```

这些更改后, 它将可以在运行基于片段的应用程序, 以及 Honeycomb 和 Ice Cream Sandwich 上 Android 1.6 或 2.x。

相关链接

- [Android 支持库 v4 NuGet](#)

在 Android 应用链接

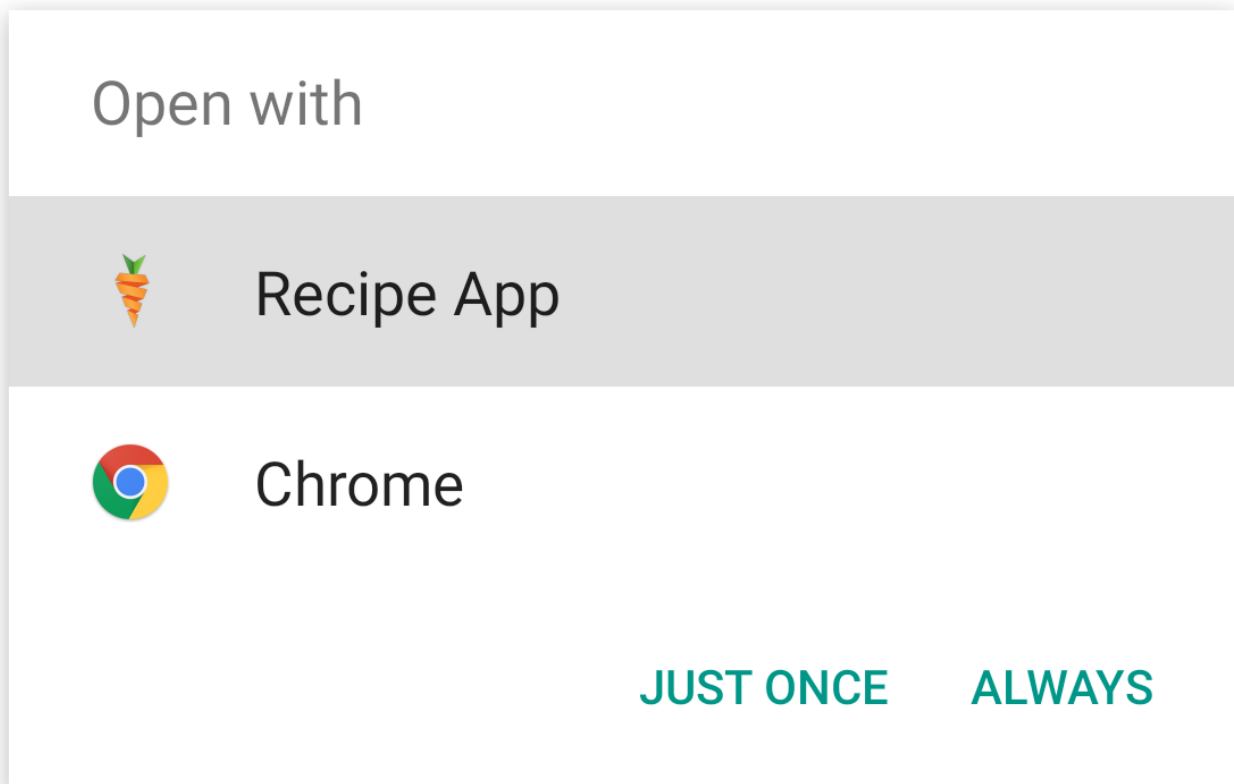
2018/11/2 • [Edit Online](#)

本指南介绍了 Android 6.0 如何支持应用的链接，该技术允许移动应用来响应在网站上的 URL。它将讨论哪些应用链接、如何实现应用链接在 Android 6.0 应用程序，以及如何配置网站以授予对域的移动应用的权限。

应用链接的概述

移动应用程序不再位于接收器—在许多情况下它们是其业务，以及他们的网站的重要组成部分。最好让企业能够将其网络影响力和移动应用程序，无缝连接上启动移动应用程序和移动应用中显示相关内容的网站的链接。将应用链接(也称为深度链接)是一种方法，允许移动设备，以响应一个 URI，并启动与该 URI 相对应的移动应用程序。

Android 处理通过将应用链接意向系统—当用户单击移动浏览器中的链接，在移动浏览器将调度 Android 将委托给已注册的应用程序意向。例如，单击烹饪与网站上的链接可打开该网站与关联的移动应用，并向用户显示特定的 recipe。如果有多个应用程序注册用于处理该意向，则 Android 将引发所谓消除二义性对话框，会询问用户要选择应如何处理目的，应用程序的应用程序示例：



通过使用自动链接处理情况下，android 6.0 提高此状态。适用于 Android 的 uri，用作默认处理程序会自动注册应用程序可以—应用将自动启动，并直接导航到相关活动。决定如何处理 URI 单击 Android 6.0 取决于以下条件：

1. 现有应用程序已与 **URI** 相关联—用户可能已经关联现有应用程序 URI。在这种情况下，Android 将继续使用该应用程序。
2. 没有任何现有应用程序与 **URI**，但安装支持的应用程序—在此方案中，用户未指定现有应用程序，因此 Android 将使用已安装支持的应用程序来处理该请求。
3. 没有任何现有应用程序与 **URI**，但许多支持的应用程序安装—因为有多多个应用程序支持的 URI，将显示消除二义性对话框并且用户必须选择将哪个应用处理 URI。

如果用户已安装的任何应用支持 URI，并且随后安装一个，则 Android 将设置该应用程序用作默认处理程序对 uri 验证利用与 URI 相关联的网站关联后。

本指南将讨论如何配置 Android 6.0 应用程序以及如何创建和发布数字资产链接文件以支持 Android 6.0 中的应用链接。

要求

本指南需要 Xamarin.Android 6.1 和面向 Android 6.0 (API 级别 23) 的应用程序或更高版本。

通过使用应用程序链接就可以在早期版本的 Android 铆钉 NuGet 包从 Xamarin 组件商店。铆钉包不兼容使用 Android 6.0; 中的应用链接它不支持 Android 6.0 应用程序链接。

在 Android 6.0 中配置应用链接

设置 Android 6.0 中的应用链接过程包括两个主要步骤：

1. 添加一个或多个目的的筛选器的网站 **URI** – 意向筛选器指导如何处理移动浏览器中的 URL, 请单击 Android。
2. 发布 **数字资产的链接 JSON** 网站上的文件 – 这是一个文件上传到网站和 Android 用于验证的移动应用和网站的域之间的关系。如果没有, Android 不能安装应用程序作为默认句柄的 URI; 用户必须手动执行此操作。

配置意向筛选器

它是所需配置一个意向筛选器, 将从网站的 URI (或可能的一组 Uri) 映射到 Android 应用程序中的活动。在 Xamarin.Android 中, 此关系的建立方式装饰的活动 `IntentFilterAttribute`。意向筛选器必须声明以下信息：

- `Intent.ActionView` – 此操作将注册意向筛选器可以响应请求来查看信息
- `Categories` – 意向的筛选器应注册同时 `Intent.CategoryBrowsable` 和 `Intent.CategoryDefault` 为了能够正确处理 web URI。
- `DataScheme` – 意向筛选器必须声明 `http` 和/或 `https`。这些是仅有两个有效的方案。
- `DataHost` – 这是 Uri 将源于的域。
- `DataPathPrefix` – 这是在网站上的资源的可选路径。
- `AutoVerify` – `autoVerify` 属性告知 Android 在验证该应用程序和网站之间的关系。这将下面详细讨论。

下面的示例演示如何使用 `IntentFilterAttribute` 来处理来自链接 `https://www.recipe-app.com/recipes` 并从 `http://www.recipe-app.com/recipes` :

```
[IntentFilter(new [] { Intent.ActionView },
    Categories = new[] { Intent.CategoryBrowsable, Intent.CategoryDefault },
    DataScheme = "http",
    DataHost = "recipe-app.com",
    DataPathPrefix = "/recipe",
    AutoVerify=true)]
public class RecipeActivity : Activity
{
    // Code for the activity omitted
}
```

Android 将验证针对该网站上的数字资产文件的意向筛选器之前应用程序的默认处理程序的注册的 uri, 标识每个主机。Android 可以建立应用程序的默认处理程序之前, 所有意向筛选器必须通过验证。

创建数字资产链接文件

将应用链接的 android 6.0 需要 Android 验证 URI, 该应用程序设置为默认处理程序之前应用程序和网站之间的关联。首次安装应用程序时, 将发生此验证。数字资产链接文件是由相关 webdomain(s) 承载一个 JSON 文件。

NOTE

`android:autoVerify` 属性必须由意向筛选器设置 – 否则 Android 不会执行验证。

文件将位于的位置的域的网络管理员通过<https://domain/.well-known/assetlinks.json>。

数字资产文件包含元数据所必需的 Android 验证关联。**Assetlinks.json**文件具有以下键 / 值对：

- `namespace` – Android 应用程序的命名空间。
- `package_name` – (在应用程序清单中声明) 的 Android 应用程序的包名称。
- `sha256_cert_fingerprints` – 已签名的应用程序的 SHA256 指纹。请参阅指南[查找密钥存储的 MD5 或 SHA1 签名](#)有关如何获取应用程序的 SHA1 指纹的详细信息。

以下代码片段示范了**assetlinks.json**了一个应用程序列出：

```
[
  {
    "relation": [
      "delegate_permission/common.handle_all_urls"
    ],
    "target": {
      "namespace": "android_app",
      "package_name": "com.example",
      "sha256_cert_fingerprints": [
        "14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:3F:CF:44:E5"
      ]
    }
  }
]
```

它可以注册多个 SHA256 指纹, 以支持不同版本或版本的应用程序。此下一步**assetlinks.json**文件是注册多个应用程序的示例：

```
[
  {
    "relation": [
      "delegate_permission/common.handle_all_urls"
    ],
    "target": {
      "namespace": "android_app",
      "package_name": "example.com.puppies.app",
      "sha256_cert_fingerprints": [
        "14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:3F:CF:44:E5"
      ]
    }
  },
  {
    "relation": [
      "delegate_permission/common.handle_all_urls"
    ],
    "target": {
      "namespace": "android_app",
      "package_name": "example.com.monkeys.app",
      "sha256_cert_fingerprints": [
        "14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:3F:CF:44:E5"
      ]
    }
  }
]
```

[Google 数字资产链接网站](#)有可能帮助进行创建和测试的数字资产文件的联机工具。

测试应用链接

实现应用链接之后, 应测试的各个部分, 以确保它们按预期方式工作。

则可以确认数字资产文件正确格式并托管通过使用 Google 的数字资产的链接 API, 在此示例中所示：

```
https://digitalassetlinks.googleapis.com/v1/statements:list?source.web.site=
https://<WEB SITE ADDRESS>:&relation=delegate_permission/common.handle_all_urls
```

有两个都可以用来确保已正确配置的意向筛选器和的 uri, 该应用程序设置为默认处理程序的测试:

1. 如上文所述正确托管数字资产文件。第一个测试将调度该 Android 应重定向到移动应用程序意向。Android 应用程序应启动并显示为 URL 注册的活动。在命令提示符下键入:

```
$ adb shell am start -a android.intent.action.VIEW \
-c android.intent.category.BROWSABLE \
-d "http://<domain1>/recipe/scalloped-potato"
```

2. 显示处理策略在给定设备上安装的应用程序的现有链接。以下命令将转储每个用户在设备上使用以下信息的链接策略的列表。在命令提示符处, 键入下列命令:

```
$ adb shell dumpsys package domain-preferred-apps
```

- **Package** – 应用程序的包名称。
- **Domain** – 应用程序将处理其 web 链接域 (由空格分隔)
- **Status** – 这是应用程序的当前链接处理状态。值为**始终**意味着该应用程序具有 `android:autoVerify=true` 声明, 并通过了系统验证。它是后接十六进制数表示的首选的 Android 系统记录。

例如:

```
$ adb shell dumpsys package domain-preferred-apps

App linkages for user 0:
Package: com.android.vending
Domains: play.google.com market.android.com
Status: always : 200000002
```

总结

本指南介绍如何将应用链接 Android 6.0 中的工作原理。然后介绍了如何配置 Android 6.0 应用程序以支持和响应的应用程序的链接。它还介绍了如何测试应用程序将 Android 应用程序中的链接。

相关链接

- [查找密钥存储的 MD5 或 SHA1 签名](#)
- [活动和意向](#)
- [AppLinks](#)
- [Google 数字资产的链接](#)
- [语句列表生成器和测试人员](#)

饼图的 android 功能

2018/10/26 • [Edit Online](#)

如何开始开发应用程序使用 Xamarin.Android 的 Android 9 饼图。

Android 9 饼图现在也可通过 Google。许多新功能和 Api 将在此版本中，提供和其中的许多所需的最新的 Android 设备中充分利用新的硬件能力。



本文旨在帮助您开始开发 Xamarin.Android 应用程序 Android 饼图。它说明了如何安装所需更新、配置 SDK，并准备好仿真器或设备进行测试。它还提供了 Android 饼图中的新增功能的概述，并提供说明了如何使用某些关键的 Android 饼图功能的示例源代码。



Xamarin.Android 9.0 提供对 Android 饼图的预览支持。有关 Android 饼图的 Xamarin.Android 支持的详细信息，请参阅[Android P 开发人员预览版 3](#)发行说明。

要求

以下列表是所需的基于 Xamarin 的应用中使用 Android 饼图功能：

- **Visual Studio** –如果使用的 Windows，更新到 Visual Studio 2017 15.8 或更高版本。如果使用的是 Mac，更新到 Visual Studio 2017 for Mac 7.6 或更高版本。
- **Xamarin.Android** – Xamarin.Android 9.0.0.17 或更高版本必须与 Visual Studio 一起安装 (作为的一部分自动安装 Xamarin.Android使用.NET 的移动开发工作负荷)。
- **Java 开发人员工具包** – Xamarin Android 9.0 开发需要JDK 8 (或可以试用 Microsoft 的分发的预览版OpenJDK)。作为的一部分自动安装 JDK8使用.NET 的移动开发工作负荷。
- **Android SDK** –必须通过 Android SDK 管理器安装 Android SDK API 28 或更高版本。

入门

若要开始开发使用 Xamarin.Android 的 Android 饼图应用，必须下载并安装最新工具和 SDK 包，然后才能创建第一个 Android 饼图项目：

1. 更新到[Visual Studio 2017 版本 15.8](#)或更高版本。如果你使用 Visual Studio for Mac，更新至[Visual Studio 2017 for Mac 版本 7.6](#)或更高版本。
2. 安装**Android 饼图 (API 28)** 包和工具通过 SDK 管理器。
3. 创建新的 Xamarin.Android 项目面向**Android 9.0**。
4. 配置仿真器或设备测试 Android 饼图应用。

以下部分解释了每个步骤：

更新 Visual Studio

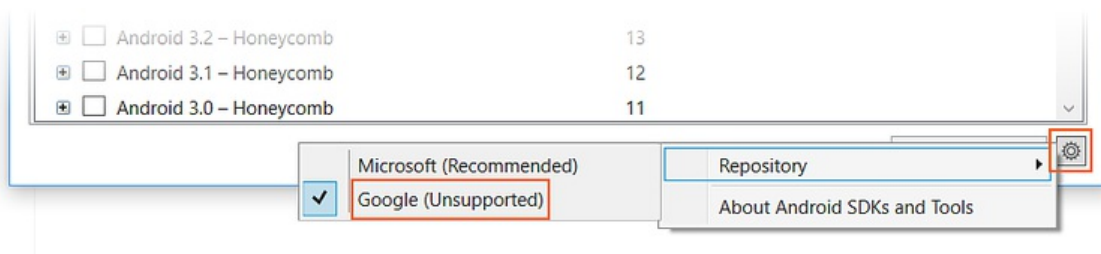
若要添加到 Visual Studio Android 饼图的支持，请更新到 Visual Studio 2017 15.8 或更高版本 (有关说明，请参阅[到最新版本更新 Visual Studio 2017](#))。

若要添加到 Visual Studio for Mac Android 饼图支持，请更新到 Mac 7.6 或更高版本的 Visual Studio 2017 (有关说明，请参阅[安装程序并安装 Visual Studio for Mac](#))。

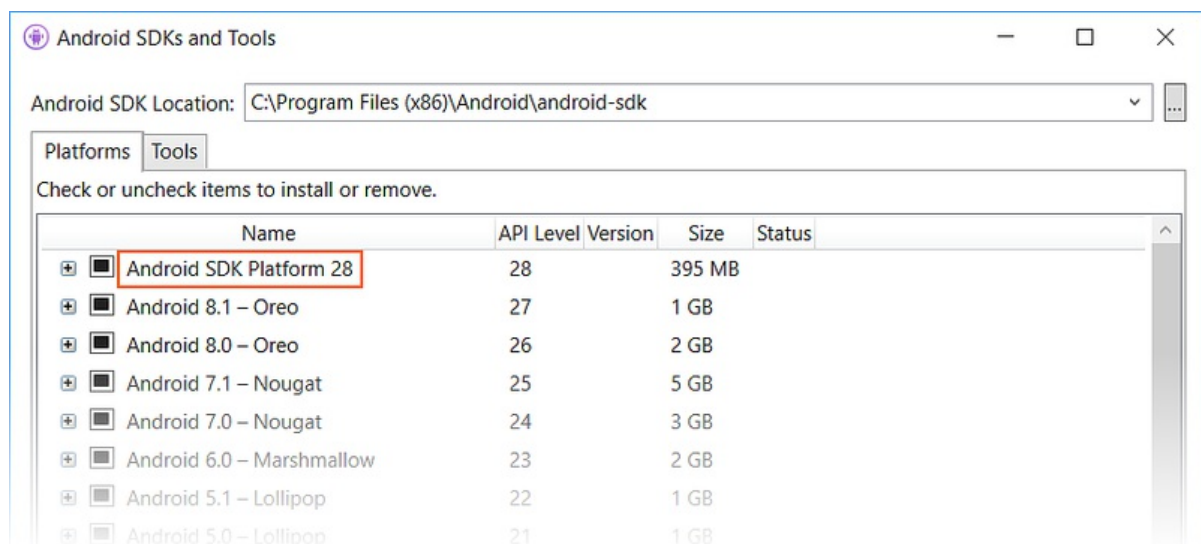
安装 Android SDK

若要使用 Xamarin.Android 9.0 创建项目，您必须首先使用 Android SDK 管理器安装的 SDK 平台**Android 饼图 (API 级别 28)** 或更高版本。

1. 启动 SDK 管理器。在 Visual Studio 中，单击**工具 > Android > Android SDK 管理器**。在 Visual Studio for Mac 中，单击**工具 > SDK 管理器**。
2. 在右下角，单击齿轮图标并选择**存储库 > Google (不受支持)**：



3. 安装**Android 饼图** SDK 包，被列为**Android SDK 平台 28**中平台选项卡 (有关使用 SDK 管理器的详细信息，请参阅[Android SDK 安装程序](#))：



4. 如果使用仿真程序, 创建一个虚拟设备, 支持**API 级别 28**。有关创建虚拟设备的详细信息, 请参阅[管理虚拟设备使用 Android 设备管理器](#)。

启动 Xamarin.Android 项目

创建新的 Xamarin.Android 项目。如果您不熟悉如何使用 Xamarin 进行 Android 开发, 请参阅[Hello, Android](#)若要了解有关创建 Xamarin.Android 项目。

创建 Android 项目时, 必须配置的版本设置应用到目标 Android 9.0 或更高版本。例如, 若要为 Android 饼图针对你的项目, 必须配置您的项目的目标 Android API 级别**Android 9.0** (API 28)。建议还目标框架级别设置为 API 28 或更高版本。有关配置的 Android API 级别的详细信息, 请参阅[了解 Android API 级别](#)。

配置设备或仿真程序

如果使用如 Nexus 或像素的物理设备, 你可以将设备更新到 Android 饼图中的说明[Nexus 和像素设备的出厂映像](#)。

如果使用仿真程序, 创建虚拟设备的 API 级别 28, 并选择基于 x86 的图像。有关使用 Android 设备管理器来创建和管理虚拟设备的信息, 请参阅[管理虚拟设备使用 Android 设备管理器](#)。有关使用 Android 仿真程序进行测试和调试的信息, 请参阅[Android 仿真器上调试](#)。

新增功能

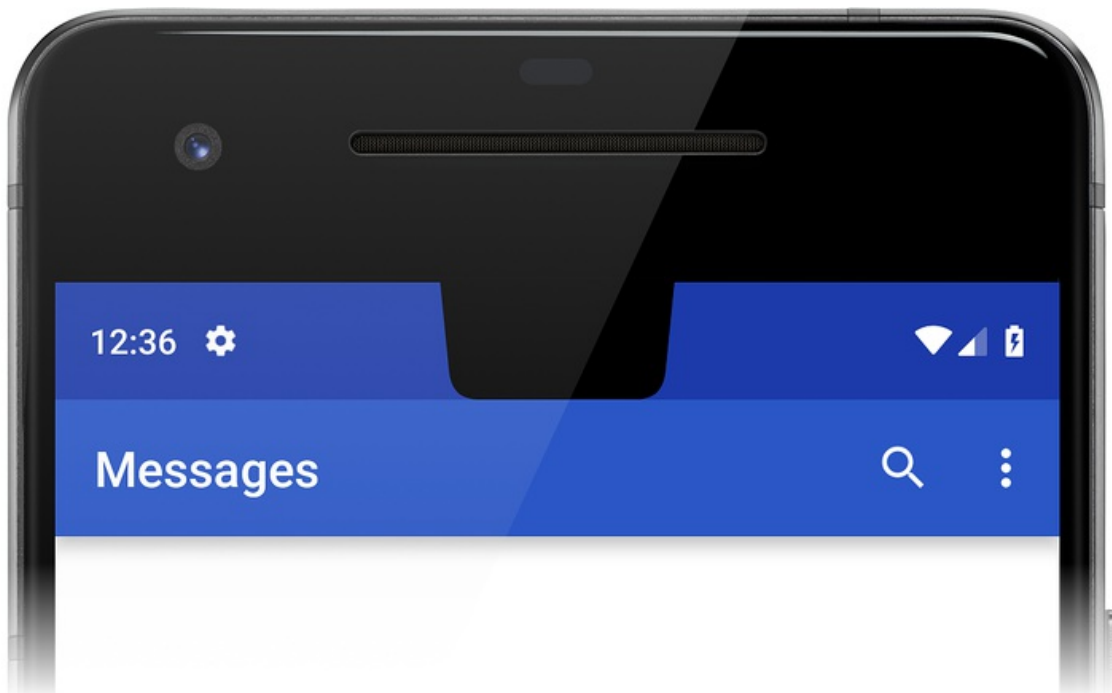
Android 饼图引入了各种新功能。其中一些新功能旨在利用最新的 Android 设备, 尽管其他人能够进一步增强 Android 用户体验所提供的新硬件功能:

- **显示切除支持**—提供了 `Api`, 若要查找的位置和形状_切除_在较新的 Android 设备上屏幕的顶部。
- **通知增强功能**—通知消息现在可以显示图像和一个新 `Person` 类用于简化对话参与者。
- **室内定位** – WiFi Round 往返时间协议, 从而使应用程序以使用在室内设置中进行导航的 WiFi 设备的平台支持。
- **支持多照相机**—提供的功能的访问的流同时从多个物理照相机 (例如双正面和双背面摄像头)。

以下部分重点介绍了这些特性, 并提供简短代码示例来帮助你开始在应用中使用它们。

显示切除支持

有许多较新的 Android 设备使用边到边屏幕**显示切除**(或“提高”) 顶部的照相机和演讲者的显示。下面的屏幕截图提供切除的仿真程序示例:



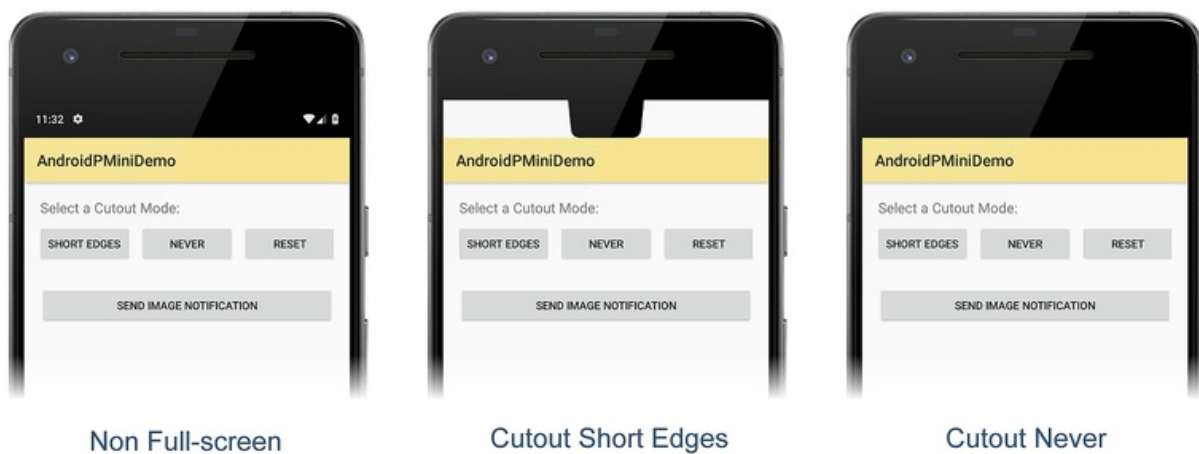
若要管理您的应用程序窗口具有显示切除设备上显示其内容的方式，Android 饼图添加了一个新 `LayoutInDisplayCutoutMode` 窗口布局属性。此属性可以设置为以下值之一：

- `LayoutInDisplayCutoutModeNever` – 永远不允许在窗口与切除区域重叠。
- `LayoutInDisplayCutoutModeShortEdges` – 窗口允许扩展到切除区域，但仅在屏幕的短边缘上。
- `LayoutInDisplayCutoutModeDefault` – 窗口允许将剪切块包含在系统栏的情况下扩展到切除区域。

例如，若要防止应用窗口与切除区域重叠，布局切除将模式设置为 *永远不会*。

```
Window.Attributes.LayoutInDisplayCutoutMode =  
    Android.Views.LayoutInDisplayCutoutMode.Never;
```

以下示例提供这些切除模式的示例。在左侧的第一个屏幕截图是在非全屏模式下的应用。在中心屏幕截图中，使用应用程序的全屏与 `LayoutInDisplayCutoutMode` 设置为 `LayoutInDisplayCutoutModeShortEdges`。请注意，应用程序的白色背景扩展到显示切除区域：



中的最终屏幕快照（上面在右侧），`LayoutInDisplayCutoutMode` 设置为 `LayoutInDisplayCutoutModeShortNever` 之后变为全屏。请注意，不允许使用应用程序的白色背景将扩展到显示切除区域。

如果需要更多详细的信息剪切块区域在设备上，你可以使用新 `DisplayCutout` 类。`DisplayCutout` 表示不能用于显示内容的显示区域。此信息可用于检索位置和形状的裁剪，使您的应用程序不会尝试在此非功能区域中显示的内容。

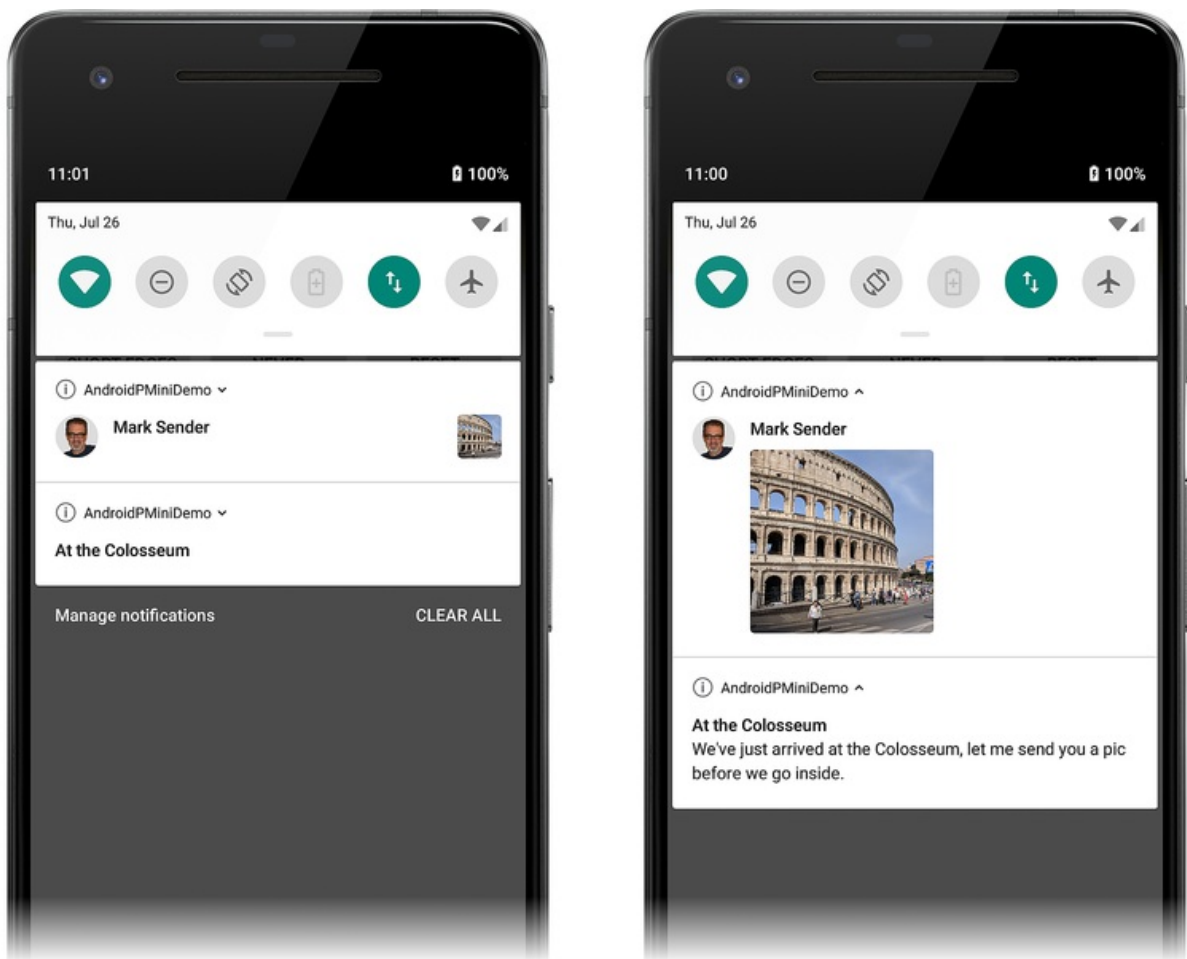
有关 Android P 中的新切除功能的详细信息，请参阅 [显示切除支持](#)。

通知增强功能

Android 饼图引入了以下增强功能以提高消息传递体验：

- 通知通道（在中引入 [Android Oreo](#)）现在支持通道组的阻止。
- 通知系统具有三个新 Do Not 打扰的类别（优先级的警报、系统声音和媒体源）。此外，有七个新 Do Not 打扰模式可用来取消显示 visual 中断（如徽章、通知光源、状态条的外观，和启动的全屏幕活动）。
- 一个新 `Person` 添加类来表示消息的发件人。使用此类有助于通过标识（包括其虚拟形象和 Uri）的会话中涉及的人员来优化每个通知的呈现。
- 通知现在可以显示图像。

下面的示例演示如何使用新的 Api 来生成包含图像的通知。在下面的屏幕截图，一个文本通知发布并后跟包含嵌入图像的通知。通知进行扩展（如右侧所示），会显示第一次通知的文本和图像中嵌入时扩大，第二个通知：



下面的示例演示如何在 Android 饼图通知中包含图像和它演示如何使用新 `Person` 类：

1. 创建 `Person` 表示发件人的对象。例如，在包含发件人的名称和图标 `fromPerson`：

```
Icon senderIcon = Icon.CreateWithResource(this, Resource.Drawable.sender_icon);
Person fromPerson = new Person.Builder()
    .SetIcon(senderIcon)
    .SetName("Mark Sender")
    .Build();
```

2. 创建 `Notification.MessagingStyle.Message`，其中包含要发送，图像将传递到新的映像 `Notification.MessagingStyle.Message.SetData` 方法。例如：

```
Uri imageUri = Uri.Parse("android.resource://com.xamarin.pminidemo/drawable/example_image");
Notification.MessagingStyle.Message message = new Notification.MessagingStyle
    .Message("Here's a picture of where I'm currently standing", 0, fromPerson)
    .SetData("image/", imageUri);
```

3. 向其中添加消息 `Notification.MessagingStyle` 对象。例如：

```
Notification.MessagingStyle style = new Notification.MessagingStyle(fromPerson)
    .AddMessage(message);
```

4. 此样式插入通知生成器。例如：


```
builder = new Notification.Builder(this, MY_CHANNEL)
    .setContentTitle("Tour of the Colosseum")
    .setContentText("I'm standing right here!")
    .setSmallIcon(Resource.Mipmap.ic_notification)
    .setStyle(style)
    .setChannelId(MY_CHANNEL);
```

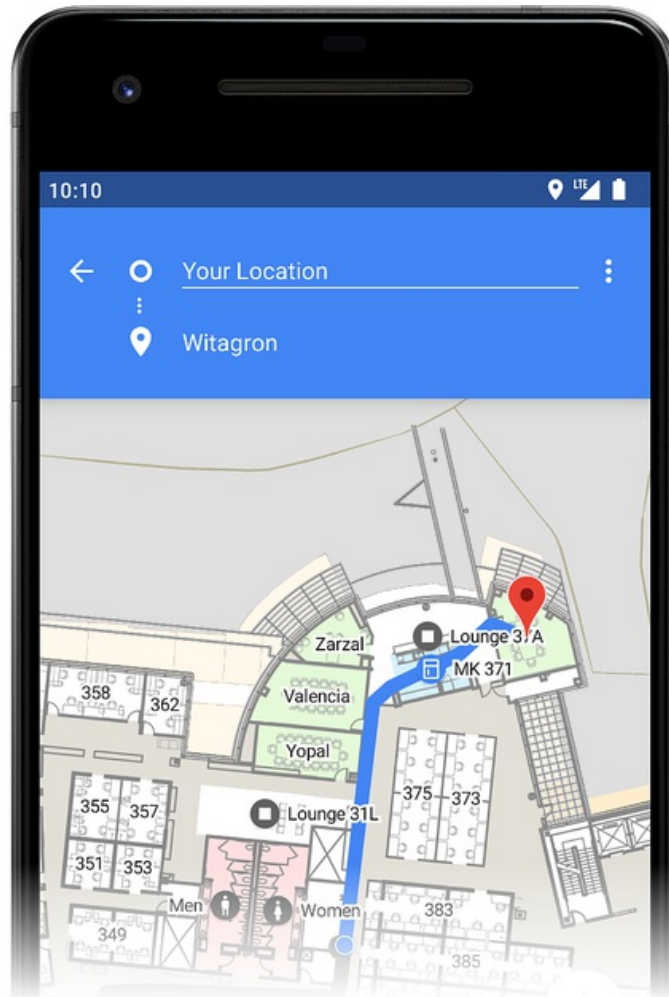
5. 发布通知。例如：

```
const int notificationId = 1000;
notificationManager.Notify(notificationId, builder.Build());
```

有关创建通知的详细信息，请参阅[本地通知](#)。

室内定位

Android 饼图支持 IEEE 802.11mc (也称为_WiFi Round 往返时间_或_WiFi RTT_)，这使得应用程序检测到一个距离或更多的 Wi-fi 访问点。使用此信息，很可能你的应用以充分利用室内定位一到两个计量的精度。Android 设备上提供的 IEEE 801.11mc 硬件支持，您的应用程序可以提供导航功能，例如基于位置的控件的智能设备或通过应用商店启用通过打开说明：



新[WifiRttManager](#)类和几个帮助程序类提供进行测量到的 Wi-fi 设备距离的手段。有关 Android P 中引入的室内定位 Api 的详细信息，请参阅[Android.Net.Wifi.Rtt](#)。

多照相机支持

许多较新的 Android 设备都具有双前面和/或双后可用于诸如立体声视觉、增强视觉效果和改进的缩放功能等功能的摄像头。Android P 引入了一个新[多照相机](#)API，它使应用程序以使用[逻辑照相机](#)(或[逻辑多照相机](#))支持的两个或多个物理相机。若要确定是否设备支持逻辑多照相机，则可以查看的每个设备上的照相机功能，以查看它是否支

持[RequestAvailableCapabilitiesLogicalMultiCamera](#)。

Android 饼图还包括一个新[SessionConfiguration](#)可用于帮助在初始捕获过程中减少延迟和无需启动并启动相机流的类。

详细了解多照相机中 Android P 支持，请参阅[多照相机支持和照相机更新](#)。

其他功能

此外，Android 饼图支持多个其他新功能：

- 新[AnimatedImageDrawable](#)类，该类可用于进行绘制和显示动画的图像。
- 一个新[ImageDecoder](#)类，用于替换 `BitmapFactory`。 `ImageDecoder` 可用于解码 `AnimatedImageDrawable`。
- 支持 HDR（高动态范围）视频和 HEIF（高效率图像文件格式）图像。
- [JobScheduler](#)已增强，以更智能地处理与网络相关的作业。新[GetNetwork](#)方法[JobParameters](#)类返回最佳网络用于为给定的作业执行的任何网络请求。

有关最新的 Android 饼图功能的详细信息，请参阅[Android 9 功能和 Api](#)。

行为更改

当目标 Android 版本设置为 API 级别 28 时，有可能会影响应用程序的行为，即使未实现上面所述的新功能的多个平台更改。以下列表是这些更改的简短摘要：

- 应用程序现在必须请求前景色的权限，才使用前景服务。
- 如果您的应用程序具有多个进程，它不能共享单个[WebView](#)跨进程的数据目录。
- 不再允许直接访问另一个应用的数据目录的路径。

有关面向 Android P 的应用程序的行为更改的详细信息，请参阅[的行为更改](#)。

示例代码

[AndroidPMiniDemo](#)演示如何设置显示切除模式的 Android 饼图是 Xamarin.Android 示例应用程序如何使用新 `Person` 类，以及如何将发送一条通知，包括图像。

总结

本文引入 Android 饼图，并介绍了如何使用安装和配置最新的工具和包 Xamarin.Android 开发 Android 饼图。它提供有关这些功能的几个示例源代码 Android 饼图中提供的关键功能的概述。它包含 API 文档的链接和 Android 开发人员主题可帮助您入门中创建适用于 Android 的饼图应用。它还突出显示可能会影响现有应用程序的最重要 Android 饼图行为更改。

相关链接

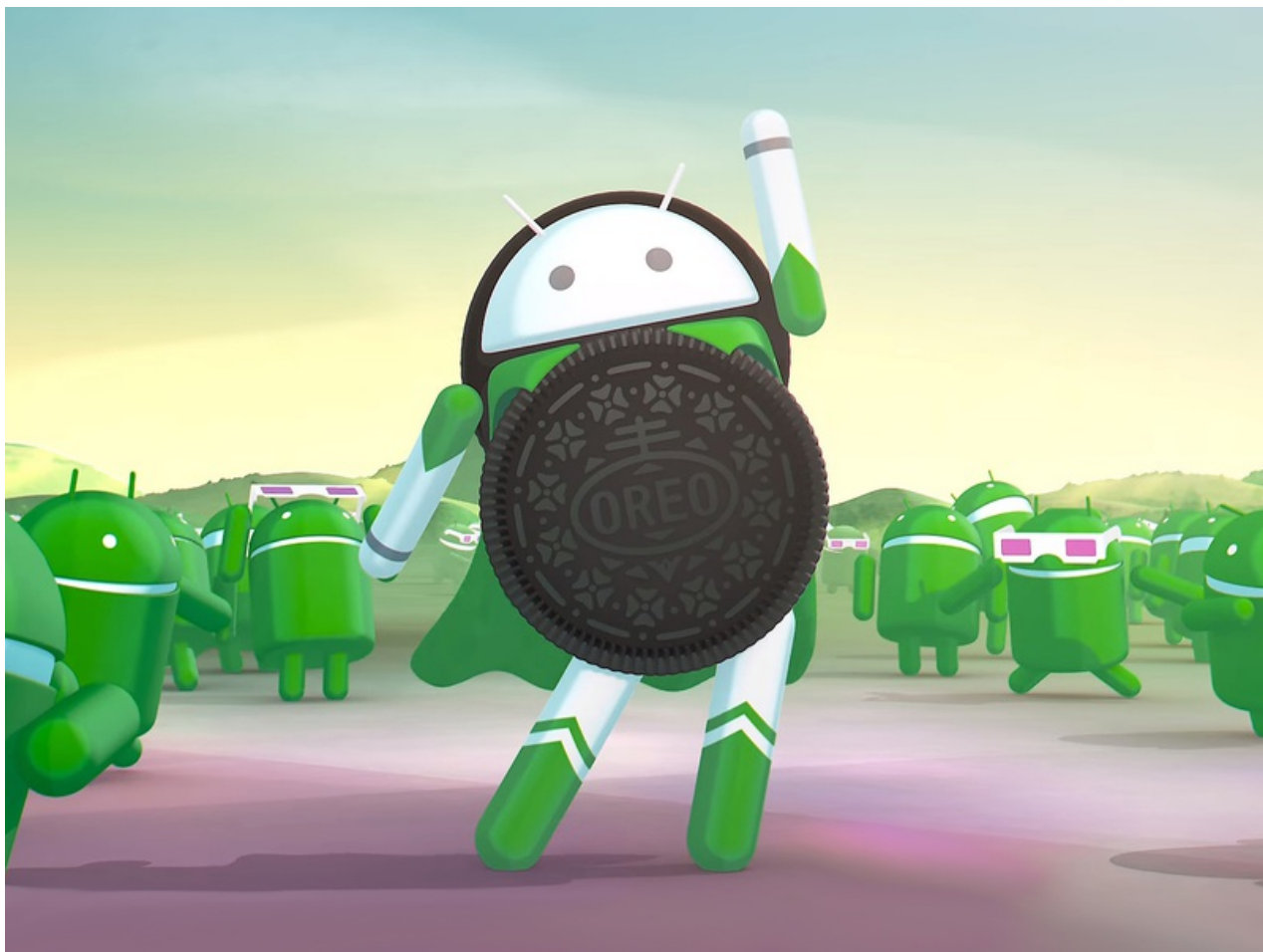
- [Android 9 饼图](#)

Oreo 功能

2018/11/13 • [Edit Online](#)

如何开始使用 Xamarin.Android 开发适用于 Android 的最新版本的应用。

Android 8.0 Oreo 是 Android 的最新版本可从 Google。Android Oreo 为 Xamarin.Android 开发人员提供所需的许多新功能。这些功能包括 XML、可下载的字体系列、自动填充和图片 (PIP) 中的图片中的通知通道、通知徽章、自定义字体。Android Oreo 包括这些新功能的新 API 和这些 API 是适用于 Xamarin.Android 应用时使用 Xamarin.Android 8.0 及更高版本。



本文旨在帮助您开始针对 Android 8.0 Oreo 中开发 Xamarin.Android 应用程序。其中介绍了如何为安装必要的更新、配置 SDK，并为测试创建模拟器（或设备）。它还提供 Android 8.0 Oreo 中的新增功能的概述，其中包含指向示例应用程序，展示了如何在 Xamarin.Android 应用中使用 Android Oreo 功能。

要求

以下是所需的基于 Xamarin 的应用中使用 Android Oreo 功能：

- **Visual Studio** – 如果使用的 Windows，则需要版本 15.5 或更高版本的 Visual Studio。如果使用的是 Mac，则需要 Visual Studio for Mac 版本 7.2.0。
- **Xamarin.Android** – Xamarin.Android 8.0 或更高版本必须安装并配置了 Visual Studio。
- **Android SDK** – Android SDK 8.0 (API 26) 或更高版本必须安装通过 Android SDK 管理器。

入门

若要开始使用 Xamarin.Android 的 Android Oreo, 必须下载并安装最新工具和 SDK 包, 然后才能创建 Android Oreo 项目:

1. 更新到最新版本的 Visual Studio。
2. 安装**Android 8.0.0 (API 26)** 或更高版本的包和工具通过 SDK 管理器。
3. 创建新的 Xamarin.Android 项目面向 Android Oreo (API 26)。
4. 配置仿真器或设备测试 Android Oreo 应用。

以下部分解释了每个步骤:

更新 Visual Studio 和 Xamarin.Android

若要添加到 Visual Studio Android Oreo 支持, 请执行以下操作:

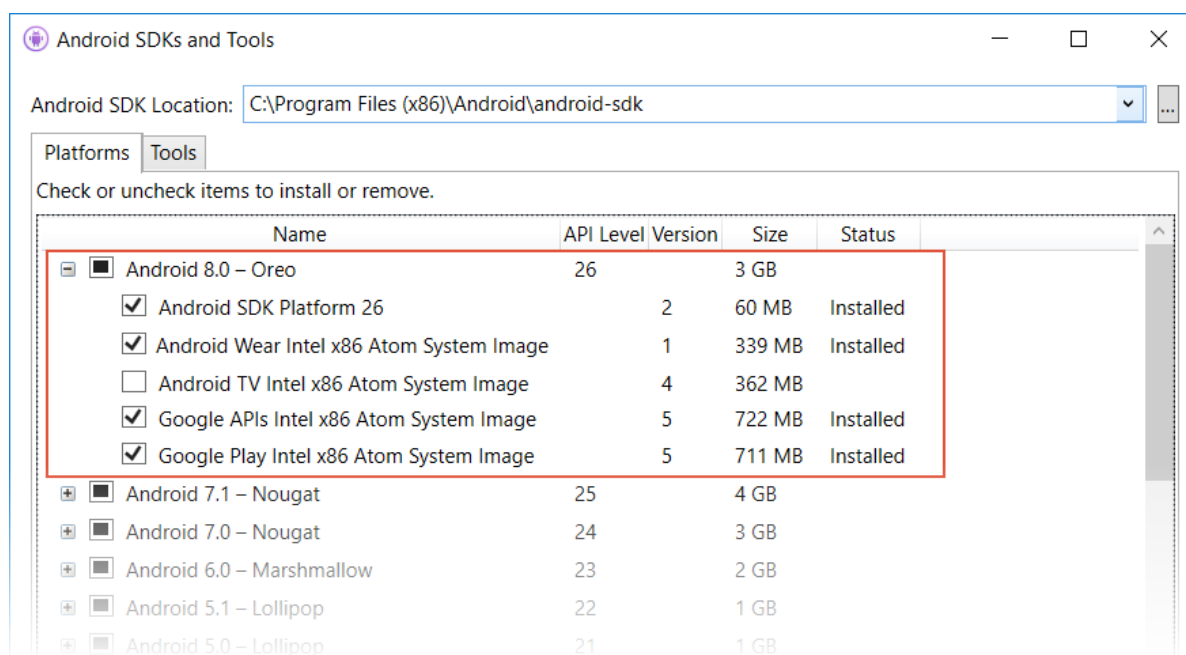
- [Visual Studio](#)
- [Visual Studio for Mac](#)
- 如果使用 Visual Studio 2017:
 1. 更新到 Visual Studio 2017 版本 15.7 或更高版本 (请参阅[Visual Studio 2017 更新](#))。
 2. 使用[SDK 管理器](#)安装 26.0 或更高版本的 API 级别。
- 如果使用 Visual Studio 2015, 建议降级 SDK Tools 为 25, 然后使用旧 Google 仿真器管理器 GUI。仍可以与 API 26, 27 日, 和更高版本, 结合使用的 SDK 工具 25 和不会影响开发新的平台。这将提供一个接口用于管理较旧版本的 VS Android SDK。

有关 Xamarin 支持 Android oreo 的详细信息, 请参阅[Xamarin.Android 8.0 发行说明](#)。

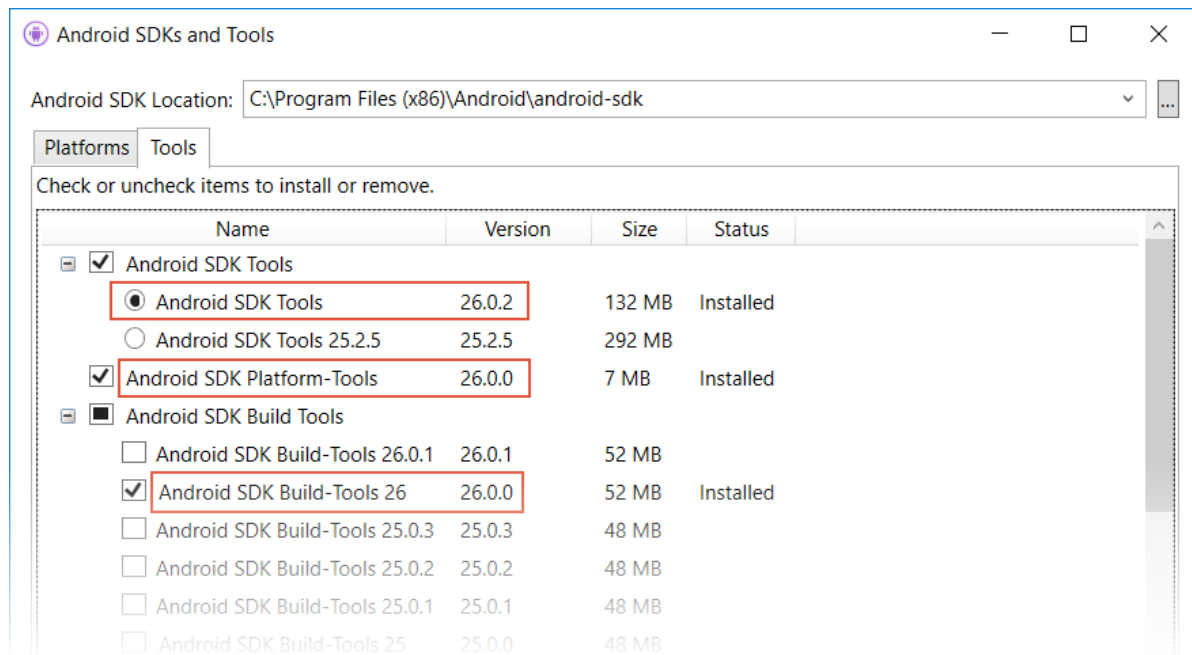
安装 Android SDK

若要创建项目时使用 Xamarin.Android 8.0, 您必须首先使用 Xamarin Android SDK 管理器安装的 SDK 平台 **Android 8.0-Oreo** 或更高版本。此外必须安装 Android SDK Tools 26.0 或更高版本。

- [Visual Studio](#)
 - [Visual Studio for Mac](#)
1. 启动 SDK 管理器 (在 Visual Studio 中, 单击 **工具 > Android > Android SDK 管理器**)。
 2. 安装**Android 8.0-Oreo**包。如果使用 Android SDK 仿真器, 请务必包括**x86**将需要的系统映像:



3. 安装**Android SDK Tools 26.0.2**或更高版本, **Android 的 SDK 平台工具 26.0.0**或更高版本, 并且**Android 的 SDK 生成工具 26.0.0** (或更高版本):



启动 Xamarin.Android 项目

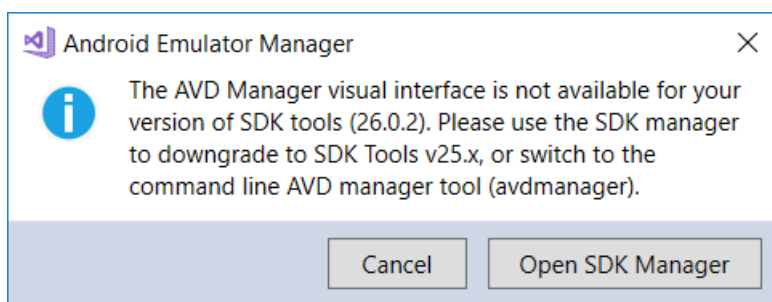
创建新的 Xamarin.Android 项目。如果您不熟悉如何使用 Xamarin 进行 Android 开发, 请参阅[Hello, Android](#)若要了解有关创建 Xamarin.Android 项目。

创建 Android 项目时, 必须配置的版本设置应用到目标 Android 8.0 或更高版本。例如, 若要为 Android 8.0 针对你的项目, 必须配置您的项目的目标 Android API 级别**Android 8.0 (API 26)**。建议, 还将目标框架设置为 API 26 或更高版本。有关配置 Android API 级别级别的详细信息, 请参阅[了解 Android API 级别](#)。

配置仿真器或设备

如果您尝试启动默认基于 Google GUI 的 AVD 管理器安装 Android SDK Tools 26.0 后或更高版本, 可能会收到以下错误对话框, 这会指示您可以使用命令行 AVD 管理器工具**avdmanager**改为:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



显示此消息, 因为 Google 不再提供独立支持 API 26.0 及更高版本的 GUI AVD 管理器。对于 Android 8.0 Oreo, 则必须使用 Xamarin Android 仿真器管理器或命令行 `avdmanager` 工具, 用于为 Android Oreo 中创建的虚拟设备。

若要使用 Android 设备管理器创建和管理虚拟设备, 请参阅[管理虚拟设备使用 Android 设备管理器](#)。若要创建虚拟设备而无需 Android 设备管理器, 请执行下一节中的步骤。

创建虚拟设备使用 avdmanager

若要使用**avdmanager**若要创建新的虚拟设备, 请执行以下步骤:

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 打开命令提示符窗口，并设置 `JAVA_HOME` 到您的计算机上的 Java SDK 的位置。对于典型的 Xamarin 安装，可以使用以下命令：

```
setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_131"
```

2. 将 Android SDK 的位置添加 `bin` 文件夹到你 `PATH`。对于典型的 Xamarin 安装，可以使用以下命令：

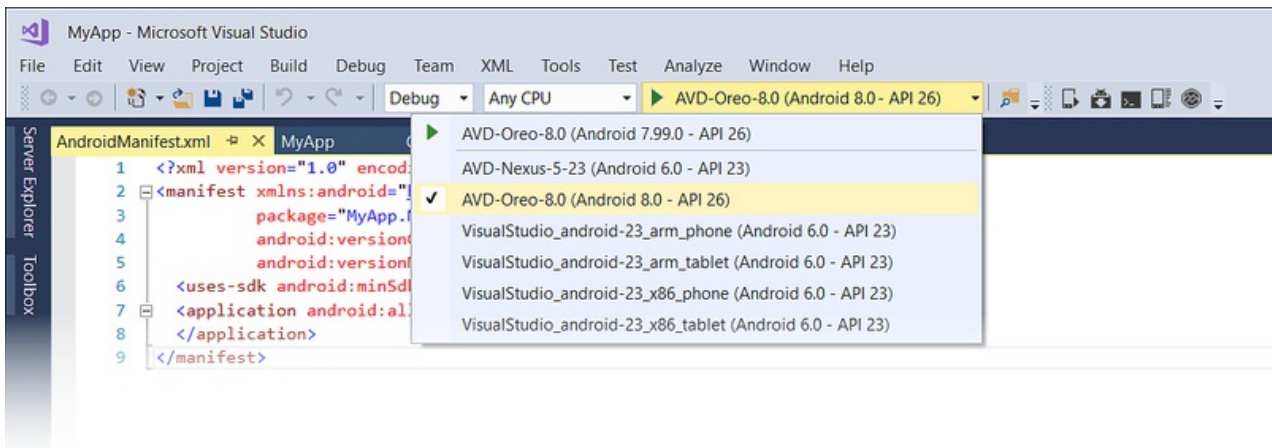
```
setx PATH "%PATH%;C:\Program Files (x86)\Android\android-sdk\tools\bin"
```

3. 关闭命令提示符窗口并打开新的命令提示符窗口。使用创建新的虚拟设备 `avdmanager` 命令。例如，若要创建 AVD 名为 **AVD Oreo 8.0** 使用 x86 系统映像的 API 级别 26，使用以下命令：

```
avdmanager create avd -n AVD-Oreo-8.0 -k "system-images;android-26;google_apis;x86"
```

4. 您使用的提示时你想要创建自定义硬件配置文件 [无] 您可以输入没有并接受默认硬件配置文件。如果您说是，`avdmanager` 将提示你提供一系列用于自定义硬件配置文件的问题。

检查完 `avdmanager` 若要创建虚拟设备，它将包含在设备下拉菜单中：



有关配置 Android 仿真器以测试和调试的详细信息，请参阅 [Android 仿真器上调试](#)。

如果使用如 Nexus 或像素的物理设备，可以通过无线 (OTA) 更新通过自动更新你的设备或下载的系统映像并直接刷新你的设备。有关手动将设备更新到 Android Oreo 的详细信息，请参阅 [Nexus 和像素设备的出厂映像](#)。

新增功能

Android Oreo 引入了各种新功能和功能，例如通知通道、通知徽章、在 XML 中的自定义字体、可下载的字体、自动填充和画中画。以下部分重点介绍了这些特性，并提供链接，以帮助你开始在应用中使用它们。

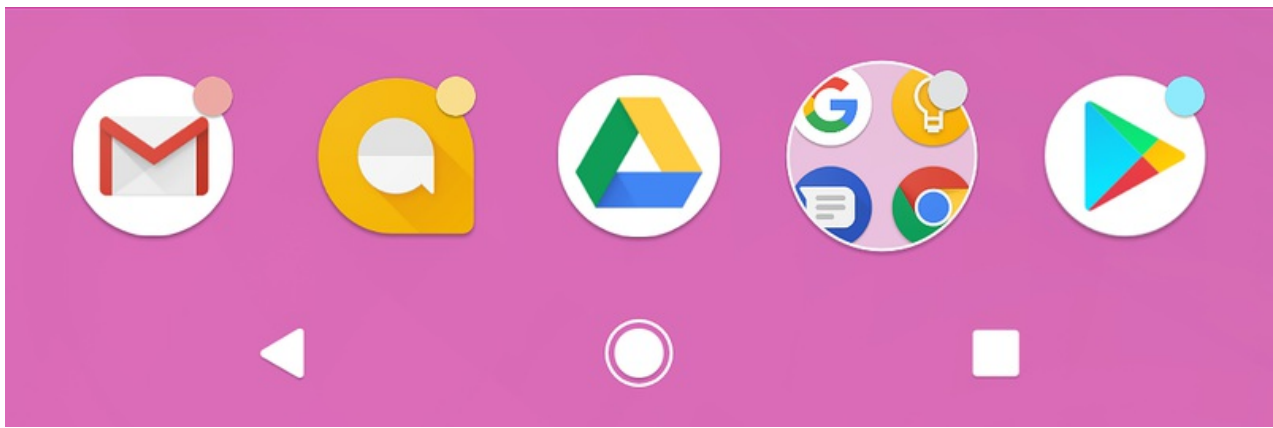
通知通道

通知通道是针对通知应用程序定义类别。可以创建通知通道的每种类型的通知，您需要发送，并且可以创建通知通道，以反映您的应用程序的用户进行选择。新的通知通道功能使您可以授予对不同类型的通知的用户进行精细控制。例如，如果要实现的消息传送应用程序，可以创建由用户创建每个会话组的单独通知通道。

通知通道介绍了如何创建通知通道并将其用于发布本地通知。实际的代码示例，请参阅 [NotificationChannels](#) 示例；此示例应用程序管理两个通道，并设置其他通知选项。

通知徽章

通知徽章是显示在应用图标上，如以下屏幕截图中所示的小圆点：



这些点表示有一个或多个通知通道，与该应用程序图标关联的应用中的新通知—这些是用户尚未取消或操作的通知。用户可以长时间的按键的图标可关闭或作用于从长时间按菜单的通知该 appears 观察一下通知徽章，与关联的通知。

有关通知徽章的详细信息，请参阅 Android 开发人员[通知徽章](#)主题。

在 XML 中的自定义字体

引入了 android Oreo XML 中的字体，从而有可能，从而将合并作为资源的自定义字体。OpenType (.otf) 和 TrueType (.ttf) 支持字体格式。若要将字体作为资源添加，请执行以下操作：

1. 创建资源/字体文件夹。
2. 将字体文件复制 (示例中，.ttf 并 .otf 文件) 到资源/字体。
3. 如有必要，重命名每个字体文件，以便它符合的 Android 文件命名约定 (即，使用仅小写 a 到 z，0-9，并在文件名中的下划线)。例如，字体文件 `Pacifico-Regular.ttf` 无法重命名为类似于 `pacifico.ttf`。
4. 使用新的应用自定义字体 `android:fontFamily` 在布局 XML 中的属性。例如，以下 `TextView` 声明使用添加 `pacifico.ttf` 字体资源：

```
<TextView
    android:text="Example Text in Pacifico Regular"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="@font/pacifico" />
```

此外可以创建字体系列 XML 文件，描述多个字体以及样式和粗细的详细信息。有关详细信息，请参阅 Android 开发人员[XML 中的字体](#)主题。

可下载的字体系列

从 Android Oreo 开始，应用程序可以请求字体，从提供程序，而无需将它们捆绑到 APK。仅在需要时，将从网络下载字体。此功能可以减少 APK 大小，节省电话内存和移动电话网络数据使用情况。您还可以通过安装 Android 支持库 26 包 Android API 版本 14 及更高版本上使用此功能。

当您的应用程序需要一种字体时，创建 `FontRequest` 对象 (指定要下载的字体) 然后将其传递给 `FontContract` 方法下载字体。以下步骤描述字体下载过程中更多详细信息：

1. 实例化 `FontRequest` 对象。
2. 子类并实例化 `FontContract.FontRequestCallback`。
3. 实现 `FontRequestCallback.OnTypeFaceRetrieved` 方法，用于处理完成字体请求。
4. 实现 `FontRequestCallback.OnTypeFaceRequestFailed` 方法，用来通知您的应用程序的字体请求过程中发生任何错误。

5. 调用[FontsContract.RequestFonts](#)方法以从字体提供程序检索字体的字体。

当您调用 `RequestFonts` 方法，它首先检查是否本地缓存字体 (从上一个调用 `RequestFont`)。如果未缓存，它调用字体提供程序，以异步方式检索字体，然后将结果返回到您的应用程序传递通过调用你 `OnTypeFaceRetrieved` 方法。

[可下载字体](#) 示例演示如何使用 Android Oreo 中引入的可下载字体功能。

有关下载字体的详细信息，请参阅 Android 开发人员[可下载字体](#)主题。

自动填充

新_自动填充_Android Oreo 中的框架，使用户更轻松的处理重复任务，例如登录名、帐户创建和信用卡交易。用户花费更少的时间重新键入信息（这可能会导致输入错误）。您的应用程序能够使用自动填充框架之前，必须在系统设置（用户可以启用或禁用自动填充）中启用自动填充服务。

[AutofillFramework](#) 示例演示如何使用自动填充框架。它包括客户端活动与应 autofilled，并可以提供给客户端活动的自动填充数据的服务的视图的实现。

有关新的自动填充功能以及如何优化自动填充你的应用的详细信息，请参阅 Android 开发人员[Autofill Framework](#)主题。

图片 (PIP) 中的图片

Android Oreo 就可以用于在画中画 (PIP) 模式下，启动的活动覆盖另一个活动的屏幕。此功能适用于视频播放。

若要指定应用程序的活动可以使用 PIP 模式，请为 true Android 清单中设置以下标记：

```
android:supportsPictureInPicture
```

若要指定您的活动应在 PIP 模式下时的行为方式，你使用新[PictureInPictureParams](#)对象。 `PictureInPictureParams` 表示一组用于初始化和更新 PIP 模式（例如，活动的首选纵横比）中的活动的参数。以下新的 PIP 方法已添加到 `Activity` Android Oreo 中：

- [EnterPictureInPictureMode](#) – 将活动放入 PIP 模式。活动放在屏幕上，角和屏幕的其余部分均显示在屏幕上的上一个活动。
- [SetPictureInPictureParams](#) – 更新活动的 PIP 配置设置（例如，更改纵横比）。

[PictureInPicture](#) 示例演示如何在 Oreo 中引入的手持设备的画中画 (PiP) 模式的基本用法。该示例播放不间断地同时显示模式或其他活动之间来回切换的视频，它会继续。

其他功能

Android Oreo 包含很多其他新功能，如表情符号支持库中，位置 API，后台限制、应用、新的音频编解码器、web 视图增强功能、改进了的键盘导航支持和适用于的新 `AAudio` (pro 音频) API 的范围内所有颜色高性能低滞后时间的音频，详细了解这些功能，请参阅 Android 开发人员[Android Oreo 功能和 Api](#)主题。

行为更改

Android Oreo 包括各种系统和 API 可能会影响的现有应用的功能的行为更改。按如下所示介绍了这些更改。

后台执行限制

为了改善用户体验，Android Oreo 规定应用程序可执行的限制时在后台运行。例如，如果用户是观看视频或玩游戏时，在后台运行的应用无法降低在前台运行需要进行大量视频的应用程序的性能。因此，Android Oreo 置于不直接与用户交互的应用以下限制：

1. **后台服务限制** – 时在后台运行应用，它具有一个窗口，在其中的几分钟的它仍可创建和使用服务。在该窗口结束时，Android 停止应用程序的后台服务并将其视为正在_空闲_。
2. **限制广播** – Android 7.0 (API 25) 放置在应用注册以接收广播上的限制。Android Oreo 使这些限制更严格。

例如, Android Oreo 应用不再可以在其清单中注册为隐式广播的广播的接收器。

有关新的后台执行限制的详细信息, 请参阅 Android 开发人员[后台执行限制](#)主题。

重大更改

应用程序面向 Android Oreo 或更高版本必须修改其应用程序以支持以下更改, 在适用的情况:

- Android Oreo 弃用了设置单个通知的优先级的能力。相反, 创建通知通道时设置建议的重要性级别。你将分配给通知通道的重要性级别适用于所有通知消息, 你向其发布到它。
- 为应用程序面向的 Android Oreo `PendingIntent.GetService()` 不工作, 因为加上在后台启动的服务的新限制。如果你面向的 Android Oreo, 则应使用[PendingIntent.GetBroadcast](#)相反。

代码示例

多个 Xamarin.Android 示例可以用来说明如何利用 Android Oreo 功能:

- [NotificationsChannels](#)演示了如何使用 Android Oreo 中引入了新的通知通道系统。此示例管理两个通知通道: 一个具有默认重要性和另一种具有高重要性。
- [PictureInPicture](#)演示 Oreo 中引入的手持设备的画中画 (PiP) 模式的基本用法。该示例播放不间断地同时显示模式或其他活动之间来回切换的视频, 它会继续。
- [AutofillFramework](#)演示如何使用自动填充框架。它包括客户端活动与应 autofilled, 并可以提供给客户端活动的自动填充数据的的服务的视图的实现。
- [可下载字体](#)举例说明如何使用前面所述的可下载字体功能。
- [EmojiCompat](#)演示 EmojiCompat 支持库的使用情况。可以使用此库以防止显示缺少的表情符号字符, 而"豆腐"字符作为你的应用。
- [位置更新挂起意向](#)说明了使用位置 API 来获取有关设备的定位更新使用 `PendingIntent`。
- [位置更新前景服务](#)演示了如何使用位置 API 来获取有关设备的位置使用绑定和启动前景服务更新。

视频

使用 C# 的 android 8.0 Oreo 开发

总结

本文引入了 Android Oreo, 并介绍了如何安装和配置 Android Oreo 上的最新工具和 Xamarin.Android 开发的包。它提供了 Android Oreo 中可用的主要功能的概述, 多项新功能的示例源代码的链接。它包含 API 文档的链接和 Android 开发人员主题可帮助您入门中创建适用于 Android Oreo 应用。它还突出显示可能会影响现有应用程序的最重要的 Android Oreo 行为更改。

相关链接

- [Android 8.0 Oreo](#)

Nougat 功能

2018/10/26 • [Edit Online](#)

如何开始使用 Xamarin.Android 开发适用于 Android Nougat 应用。

本文提供了 Android Nougat 中引入的功能的概述介绍了如何准备 Xamarin.Android 进行 Android Nougat 开发, 并且提供了示例应用程序, 展示了如何使用 Android Nougat 功能中的链接 Xamarin.Android 应用。

概述

[Android Nougat](#)是到 Android 6.0 Marshmallow Google 的跟进。Xamarin.Android 提供的支持**Android 7.x** 绑定 Xamarin Android 7.0 及更高版本。Android Nougat 添加 Nougat 功能; 如下所述的许多新 Api使用 Xamarin.Android 7.0 时, 这些 Api 是适用于 Xamarin.Android 应用。



有关 Android 7.x Api 的详细信息, 请参阅[面向开发人员的 Android 7.1](#)。有关 Xamarin.Android 7.0 的已知问题的列表, 请参阅[发行说明](#)。

Android Nougat Xamarin.Android 开发人员提供所需的许多新功能。这些功能包括:

- **多窗口支持**-此增强功能使用户可以同时打开两个应用程序在屏幕上的。
- **通知增强功能**-经过重新设计的通知系统中 Android Nougat 包括 *直接回复* 功能, 使用户能够快速响应直接从通知短信用户界面。此外, 如果您的应用程序创建的通知接收到消息, 则新 *捆绑通知* 功能可以捆绑在通知一起作为一个组时收到多条消息。
- **数据保护程序**-此功能是一种新的系统服务, 从而降低应用手机网络数据使用; 它给予用户对应用程序如何使用移动电话网络数据的控制。

此外, Android Nougat 带来了许多其他增强功能等新的网络安全的配置功能的应用程序开发人员感兴趣的 Doze 随时随地、密钥证明, 新快速设置 Api, ICU4J Api 的多区域设置支持 web 视图增强功能, 对 Java 8 语言功能的访问、作用域内的目录访问权限、自定义指针 API、平台 VR 支持、虚拟文件和后台处理优化。

本文介绍如何开始使用 Android Nougat 来试用新功能和规划迁移或功能的工作, 以面向新的 Android Nougat 平台构建应用程序。

要求

以下是所需的基于 Xamarin 的应用中使用新的 Android Nougat 功能:

- **Visual Studio 或 Visual Studio for Mac** –如果使用的 Visual Studio, 版本 4.2.0.628 或更高版本的 Visual Studio Tools for Xamarin 是必需的。如果你使用 Visual Studio for Mac, 版本 6.1.0 或更高版本的 Visual Studio for Mac 所需。
- **Xamarin.Android** – Xamarin.Android 7.0 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for Mac。
- **Android SDK** -Android SDK 7.0 (API 24) 或更高版本必须安装通过 Android SDK 管理器。
- **Java 开发人员工具包** – Xamarin Android 7.0 开发需要JDK 8或更高, 如果你正在开发的 API 级别 24 或更高版本 (JDK 8 还支持 API 级别低于 24)。如果使用的自定义控件或窗体预览程序, 则需要 JDK 8 的 64 位版本。

IMPORTANT

Xamarin.Android 不支持 JDK 9。

请注意, 应用必须重新生成 Xamarin C6SR4 或更高版本才能可靠地使用 Android Nougat。因为可以仅为链接 Android Nougat [NDK 提供本机库](#), 如使用库的现有应用程序Mono.Data.Sqlite.dll Android Nougat 上运行, 如果它们不正确时可能会崩溃重新生成。

入门

若要开始使用 Xamarin.Android 使用 Android Nougat, 必须下载并安装最新工具和 SDK 包, 然后才能创建 Android Nougat 项目:

1. 从 Xamarin 安装最新的 Xamarin.Android 更新。
2. 安装**Android 7.0 (API 24)** 包和工具或更高版本。
3. 创建新的 Xamarin.Android 项目面向 Android Nougat。
4. 为 Android Nougat 配置仿真器或设备。

以下部分解释了每个步骤:

安装 Xamarin 更新

若要添加对 Android Nougat Xamarin 支持, 将更新通道在 Visual Studio 或 Visual Studio for Mac 更改为稳定通道并应用最新的更新。如果您还需要仅在 Alpha 或 Beta 通道中当前可用的功能, 您可以切换到 Alpha 或 Beta 通道 (Alpha 和 Beta 通道还提供支持用于 Android 7.x)。有关如何更改更新 (版本) 通道的信息, 请参阅[更改更新通道](#)。


安装 Android SDK

若要使用 Xamarin Android 7.0 创建项目, 您必须首先使用 Android SDK 管理器来安装**SDK 平台 Android N (API 24)** 或更高版本。你还必须安装最新**Android SDK Tools**:

1. 启动 Android SDK 管理器 (在 Visual Studio for Mac 中, 使用工具 > 打开 **Android SDK 管理器...**; 在 Visual Studio 中, 使用工具 > **Android > Android SDK 管理器**)。
2. 安装**Android 7.0 (API 24)** 或更高版本:

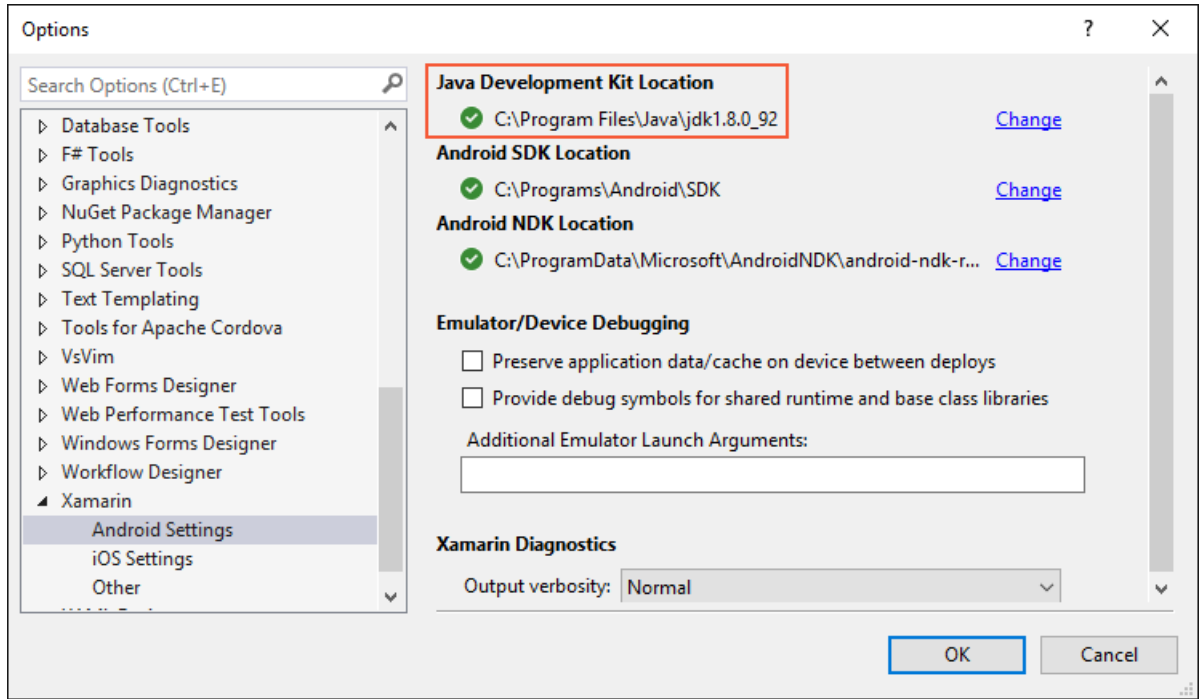
| | | | |
|---|----|---|---|
| Android 7.0 (API 24) | | | |
| <input type="checkbox"/> Documentation for Android SDK | 24 | 1 | <input type="checkbox"/> Not installed |
| <input type="checkbox"/> SDK Platform | 24 | 2 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Android TV Intel x86 Atom System Image | 24 | 6 | <input type="checkbox"/> Not installed |
| <input type="checkbox"/> Android Wear ARM EABI v7a System Image | 24 | 1 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Android Wear Intel x86 Atom System Image | 24 | 1 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> ARM 64 v8a System Image | 24 | 7 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> ARM EABI v7a System Image | 24 | 7 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Intel x86 Atom_64 System Image | 24 | 7 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Intel x86 Atom System Image | 24 | 7 | <input checked="" type="checkbox"/> Installed |

3. 安装最新的 Android SDK 工具：

|  Name | API | Rev. | Status |
|--|-----|--------|---|
| <input type="checkbox"/> Tools | | | |
| <input type="checkbox"/> Android SDK Tools | | 25.2.2 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Android SDK Platform-tools | | 24.0.3 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Android SDK Build-tools | | 24.0.2 | <input checked="" type="checkbox"/> Installed |
| <input type="checkbox"/> Android SDK Build-tools | | 24.0.1 | <input type="checkbox"/> Not installed |
| <input type="checkbox"/> Android SDK Build-tools | | 24 | <input type="checkbox"/> Not installed |

必须安装 Android SDK Tools 修订 25.2.2 或更高版本、Android SDK 平台工具 24.0.3 或更高版本，以及 Android SDK 生成工具 24.0.2 或更高版本。

4. 确认 **Java Development Kit** 位置为 JDK 1.8 配置：



若要在 Visual Studio 中查看此设置，请单击 **工具 > 选项 > Xamarin > Android** 设置。在 Visual Studio for Mac 中，单击 **首选项 > 项目 > SDK 位置 > Android**。

启动 **Xamarin.Android** 项目

创建新的 Xamarin.Android 项目。如果您不熟悉如何使用 Xamarin 进行 Android 开发，请参阅[Hello, Android](#)若要了解有关创建 Xamarin.Android 项目。

创建 Android 项目时，必须配置的版本设置应用到目标 Android 7.0 或更高版本。例如，若要面向的 Android 7.0 你的项目，必须配置您的项目的目标 Android API 级别 **Android 7.0 (API 24-Nougat)**。建议，将目标框架设置为 API 24 或更高版本。有关配置 Android API 级别级别的详细信息，请参阅[了解 Android API 级别](#)。

NOTE

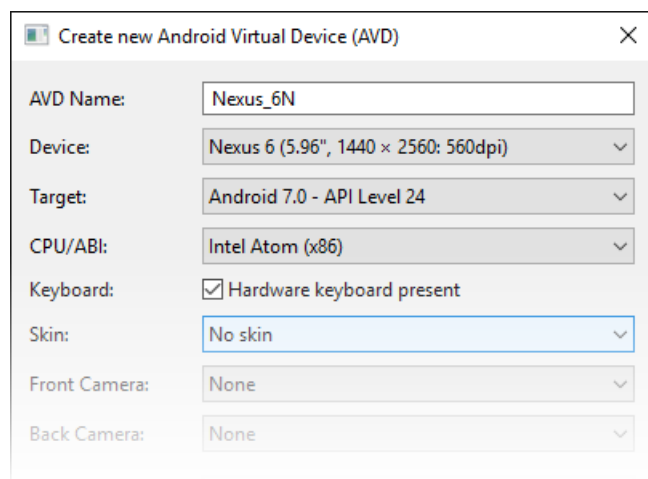
当前必须设置最低 **Android 版本到Android 7.0 (API 24-Nougat)** 若要将您的应用程序部署到 Android Nougat 设备或仿真程序。

配置仿真器或设备

如果使用仿真程序，启动 Android AVD 管理器，并创建新的设备使用以下设置：

- 设备：Nexus 5 X, Nexus 6, Nexus 6 P, Nexus 播放器 Nexus 9 或像素 c。
- 目标：Android 7.0-API 级别 24
- ABI: x86 或 x86_64

例如，此虚拟设备被配置为模拟 Nexus 6:



如果使用的物理设备，如 Nexus 5 X、6 或 9，可以通过无线 (OTA) 更新通过自动更新你的设备或下载的系统映像并直接刷新你的设备。有关手动将设备更新到 Android Nougat 的详细信息，请参阅[OTA 映像用于 Nexus 设备](#)。

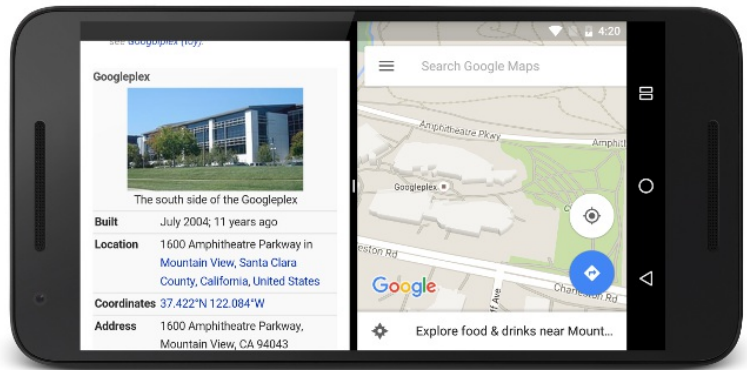
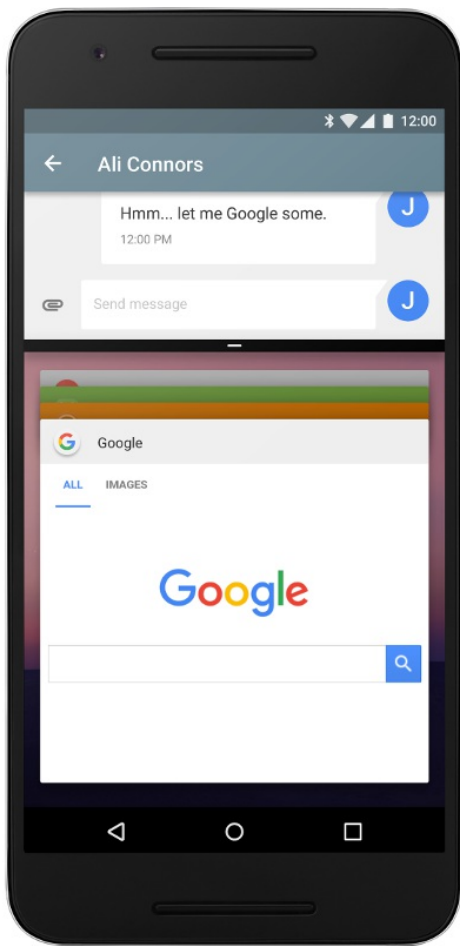
请注意，Nexus 5 设备不受 Android Nougat。

新增功能

Android Nougat 引入了各种新功能和功能，例如多窗口支持、通知增强功能和数据保护程序。以下部分重点介绍了这些特性，并提供链接，以帮助你开始在应用中使用它们。

多窗口模式

多窗口模式使用户可以同时打开两个应用具有完整的多任务处理支持。这些应用可以在拆分屏幕模式下运行并行（横向）或一个-上面--其他（纵向）。用户可以拖动以调整其大小在应用之间的分隔符，并且他们可以剪切和粘贴的内容应用程序之间。两个应用都显示在多窗口模式下，所选的活动继续运行而未选定的活动会暂停，但仍然可见。多窗口模式下不会修改 Android 活动生命周期。



你可以配置活动的 Xamarin.Android 应用程序如何支持多窗口模式下运行。例如，你可以配置在多窗口模式下设置的最小大小的默认高度和宽度的您的应用程序的属性。你可以使用新 `Activity.IsInMultiWindowMode` 属性以确定您的活动是否在多窗口模式下。例如：

```
if (!IsInMultiWindowMode) {  
    multiDisabledMessage.Visibility = ViewStates.Visible;  
} else {  
    multiDisabledMessage.Visibility = ViewStates.Gone;  
}
```

[MultiWindowPlayground](#) 示例应用包含 C# 代码演示了如何利用多个窗口具有您的应用程序用户界面。

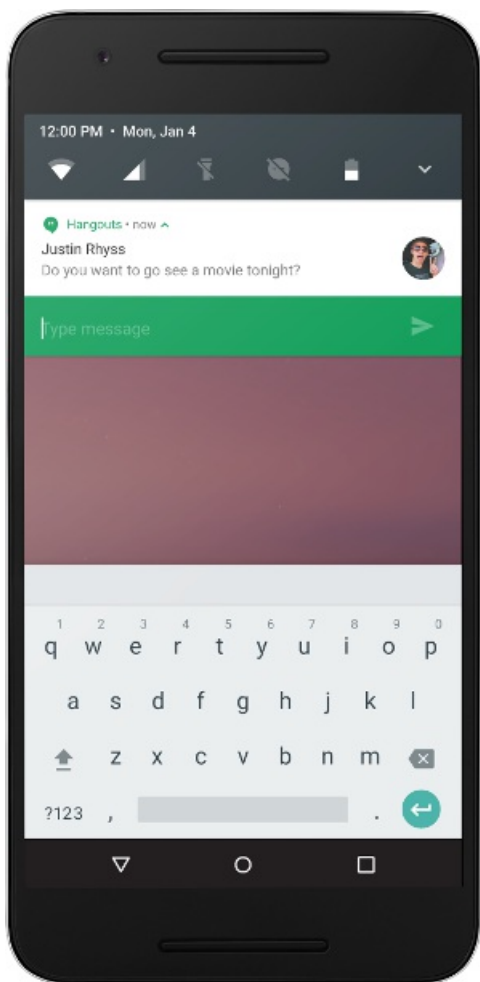
有关多窗口模式的详细信息，请参阅 [多窗口支持](#)。

增强的通知

Android Nougat 引入了重新设计的通知系统。它提供了一个新的直接答复功能，使用户快速通知的通知 UI 中直接传入短信回复。从 Android 7.0，消息可以捆绑在一起作为一个组时收到多条消息的通知开始。此外，开发人员可以自定义视图，利用系统修饰在通知中，并充分利用新的通知模板时生成通知的通知。

直接回复

当用户收到传入消息的通知时，Android Nougat 使得回复中通知的消息（而不是打开消息传送应用程序发送回复）。此内联答复功能使用户能够快速响应直接在通知接口中的短信或文本消息：



若要在您的应用程序中支持此功能，必须添加内联答复操作到你的应用通过 [RemoteInput](#) 对象，以便用户可以直接从通知 UI 回复信作为联系方式。例如，下面的代码生成 `RemoteInput` 用于接收文本输入，生成答复操作挂起意向并创建远程输入已启用的操作：

```
// Build a RemoteInput for receiving text input:
var remoteInput = new Android.Support.V4.App.RemoteInput.Builder (EXTRA_REMOTE_REPLY)
    .SetLabel (GetString (Resource.String.reply))
    .Build ();

// Build a Pending Intent for the reply action to trigger:
PendingIntent replyIntent = PendingIntent.GetBroadcast (ApplicationContext,
    conversation.ConversationId,
    GetMessageReplyIntent (conversation.ConversationId),
    PendingIntentFlags.UpdateCurrent);

// Build an Android 7.0 compatible Remote Input enabled action:
NotificationCompat.Action actionReplyByRemoteInput = new NotificationCompat.Action.Builder (
    Resource.Drawable.notification_icon,
    GetString (Resource.String.reply),
    replyIntent).AddRemoteInput (remoteInput).Build ();
```

此操作添加到通知：

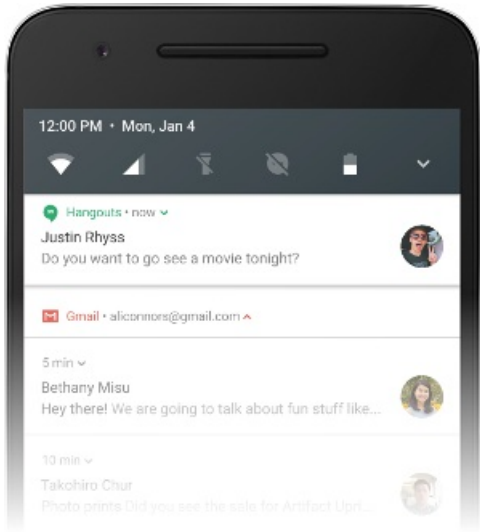
```
// Create the notification:
NotificationCompat.Builder builder = new NotificationCompat.Builder (ApplicationContext)
    .SetSmallIcon (Resource.Drawable.notification_icon)
    ...
    .AddAction (actionReplyByRemoteInput);
```

[消息传递服务](#) 示例应用包含 C# 代码演示如何扩展使用通知 `RemoteInput` 对象。有关添加内联答复详细信息到您的

应用程序的操作的 Android 7.0 或更高版本, 请参阅 [Android 答复通知](#) 主题。

捆绑的通知

Android Nougat 可以通知消息分组 (例如, 通过消息主题), 并显示组, 而不是每个单独的消息。这 [捆绑通知](#) 功能使用户可以关闭或存档中一个操作的通知组。用户可以向下滑动, 若要展开的通知, 以查看详细信息中的每个通知捆绑包:



若要支持捆绑的通知, 您的应用程序可以使用 [Builder.SetGroup](#) 方法捆绑类似的通知。有关在 Android N 捆绑的通知组的详细信息, 请参阅 [Android 捆绑通知](#) 主题。

自定义视图

Android Nougat 使您可以使用系统通知标头、操作和可扩展布局创建自定义通知视图。有关 Android Nougat 中的自定义通知视图的详细信息, 请参阅 [Android 通知增强功能](#) 主题。

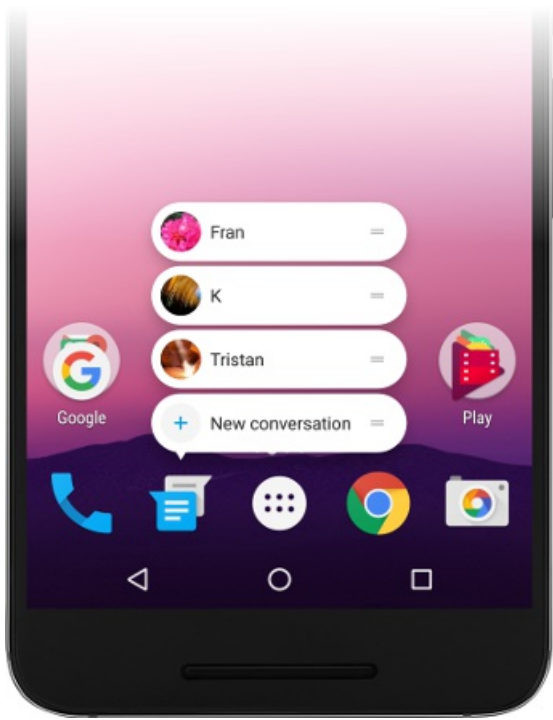
数据保护程序

从 Android Nougat 开始, 用户可以启用一个新 [数据保护程序](#) 设置阻止后台数据使用情况。此设置还向您的应用程序在前台中使用较少的数据, 只要有可能发出信号。 [ConnectivityManager](#) 延长了 Android Nougat, 以便您的应用程序可以检查, 以便您的应用程序可以致力于限制其数据使用情况数据保护程序启用时, 用户是否已启用数据保护程序。

有关 Android Nougat 中新的数据保护程序功能的详细信息, 请参阅 [Android 优化网络数据使用情况](#) 主题。

应用程序快捷方式

引入了 android 7.1 [应用程序快捷方式](#), 可以让用户快速开始常见或推荐任务与你的应用的功能。若要激活的快捷方式菜单, 用户 long 类型的值的按应用图标第二个或更多-菜单显示时有快速振动。释放按下了会导致要保持的菜单:



此功能是可用唯一 API 级别 25 或更高版本。有关 Android 7.1 中的新应用程序快捷方式功能的详细信息, 请参阅 [Android 应用程序快捷方式](#) 主题。

代码示例

多个 Xamarin.Android 示例可以用来说明如何利用 Android Nougat 功能:

- [MultiWindowPlayground](#) 演示如何使用 Android Nougat 中提供的多窗口 API。可切换到多 windows 模式, 以查看它如何影响应用程序的生命周期和行为的示例应用程序。
- [消息传递服务](#) 发送通知的简单服务使用 `NotificationCompatManager`。它还扩展了该通知 `RemoteInput` 对象, 以允许 Android Nougat 设备回复文本通过直接从通知, 而无需打开应用。
- [活动通知](#) 演示了如何使用 `NotificationManager` API 来告诉您当前正在显示你的应用程序的通知数。
- [限定的目录访问权限](#) 演示了如何使用作用域内的目录访问 API 可以轻松地访问特定的目录。这将作为无需定义一种替代方法 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 在清单中的权限。
- [直接启动](#) 说明了如何在设备加密时引导设备同时之前和之后输入任何用户 credentials(PIN/Pattern/Password) 是始终可用的存储中存储数据。

总结

本文介绍 Android Nougat 并介绍了如何安装和配置最新的工具和包 Xamarin.Android 开发 Android Nougat 上。它还提供指向示例的源代码, 以帮助您入门中创建适用于 Android Nougat 应用 Android Nougat 中提供的关键功能的概述。

相关链接

- [面向开发人员的 android 7.1](#)
- [Xamarin Android 7.0 发行说明](#)

Marshmallow 功能

2018/10/26 • [Edit Online](#)

本文可帮助你开始使用 *Xamarin.Android* 开发适用于 *Android 6.0 Marshmallow* 应用中使用。

本文概述了 *Android 6.0 Marshmallow* 中的新增功能、介绍如何进行准备 *Xamarin.Android* 进行 *Android Marshmallow* 开发, 并提供指向示例应用程序, 展示了如何利用新的 *Android Marshmallow* 在 *Xamarin.Android* 应用中的功能。

概述

[Android 6.0 Marshmallow](#), 是下一步的主要 *Android Lollipop* 之后的版本。*Xamarin.Android* 支持 *Android Marshmallow*, 包括:

- **API 23/Android 6.0 绑定** – *Android 6.0* 添加许多新 *Api*, 如下所述的新功能; 当你的目标 *API* 级别 23 时, 这些 *API* 是适用于 *Xamarin.Android* 应用。有关 *Android 6.0 Api* 的详细信息, 请参阅[Android 6.0 Api](#)。



尽管 *Marshmallow* 版本主要侧重于“波兰语和质量”, 但它还为 *Xamarin.Android* 开发人员提供感兴趣的许多新功能。这些功能包括:

- **运行时权限**–此增强功能使用户能够在运行时批准的安全权限的情况。
- **身份验证的改进**–从 *Android Marshmallow* 开始, 应用现在可以使用指纹传感器来进行身份验证用户和一个新 *确认凭据* 功能最大程度减少需要输入密码。
- **应用链接**–此功能有助于消除无应用选择器弹出通过自动将应用与 *web* 域相关联。
- **直接共享**–可以定义 *直接共享目标*, 让共享快速、直观的用户; 此功能允许用户与其他应用共享内容。
- **语音交互**–这个新的 *API* 可用于构建对话式语音功能到你的应用。
- **4 K 显示模式**–在 *Android Marshmallow*, 您的应用程序可以请求 4k 支持它的硬件上的显示分辨率。
- **音频的新功能** – *Marshmallow* 从开始, *Android* 现在支持 *MIDI* 协议。它还提供了新类, 以创建数字音频捕获和播放对象, 并提供用于将音频和输入设备相关联的新 *API* 挂钩。
- **视频的新功能** – *Marshmallow* 提供了一个新类, 可帮助应用程序呈现音频和视频流同步; 此类还提供对动态播放速度的支持。
- **Android for Work** – *Marshmallow* 包括增强的控件为公司拥有的单用户设备。它支持无提示安装和卸载的应用程序的设备所有者、自动接受系统更新、改进了的证书管理、数据使用情况跟踪、权限管理和工作状态

通知。

- **材料设计支持库**–新的 *设计支持库* 提供设计组件和模式，使你更轻松地将内置到应用 Material Design 外观和感觉。

此外，Android m，已发布了很多核心 Android 库更新并且这些更新为 Android M 和早期版本的 Android 提供的新功能。

此外，Android Marshmallow 与已发布了很多核心 Android 库更新并且这些更新为 Android Marshmallow 和早期版本的 Android 提供的新功能。本文介绍如何开始使用 Android Marshmallow 构建应用程序，并提供在 Android 6.0 中重点介绍新功能的概述。

要求

以下是所需的基于 Xamarin 的应用中使用新的 Android Marshmallow 功能：

- **Xamarin.Android** – Xamarin.Android 5.1.7.12 或更高版本必须安装并配置了 Visual Studio 或 Xamarin Studio。
- **Visual Studio for Mac**或**Visual Studio** –如果使用 Visual Studio for Mac 版本 5.9.7.22 或更高版本。如果使用的 Visual Studio，版本 3.11.1537 或更高版本的 Visual Studio 的 Xamarin 工具是必需的。
- **Android SDK** – Android SDK 6.0 (API 23) 或更高版本必须安装通过 Android SDK 管理器。
- **Java 开发人员工具包** – Xamarin.Android 需要 [JDK 1.8](#) 或更高，如果你正在开发的 API 级别 24 或更高版本 (JDK 1.8 还支持 API 级别低于 24，包括 Marshmallow)。如果使用的自定义控件或窗体预览程序，则需要 JDK 1.8 的 64 位版本。

你可以继续使用 [JDK 1.7](#) 如果您是开发专门针对 API 级别 23 或更早版本。

入门

若要开始使用 Xamarin.Android 的 Android Marshmallow，必须下载并安装最新工具和 SDK 包，然后才能创建 Android Marshmallow 项目：

1. 安装最新的 Xamarin 更新从稳定通道。
2. 安装 Android 6.0 Marshmallow SDK 包和工具。
3. 创建新的 Xamarin.Android 项目面向 Android 6.0 Marshmallow (API 级别 23)。
4. 为 Android Marshmallow 配置仿真器或设备。

以下部分解释了每个步骤：









安装 Xamarin 更新

若要更新 Xamarin，使其包括支持 Android 6.0 Marshmallow，更改到的更新通道稳定并安装所有更新。有关从更新通道安装更新的详细信息，请参阅 [更改更新通道](#)。


















安装 Android 6.0 SDK

若要为 Android Marshmallow 创建 Xamarin.Android 项目，您必须首先使用 Android SDK 管理器安装 Android 6.0 SDK：

- 启动 Android SDK 管理器 (在 Visual Studio for Mac 中，使用 **工具 > SDK 管理器**；在 Visual Studio 中，使用 **工具 > Android > Android SDK 管理器**) 并安装最新的 Android SDK Tools：

|  Name | API | Rev. | Status |
|--|-----|--------|---|
|  Tools | | | |
|  Android SDK Tools | | 24.3.4 |  Installed |
|  Android SDK Platform-tools | | 23.0.1 |  Installed |
|  Android SDK Build-tools | | 23.0.1 |  Installed |

- 此外，安装最新**Android 6.0** SDK 包：

| | | | |
|--|----|---|---|
| <input checked="" type="checkbox"/>  Android 6.0 (API 23) | | | |
|  Documentation for Android SDK | 23 | 1 |  Installed |
|  SDK Platform | 23 | 1 |  Installed |
|  Samples for SDK | 23 | 2 |  Installed |
|  Android TV ARM EABI v7a System Image | 23 | 2 |  Installed |
|  Android TV Intel x86 Atom System Image | 23 | 2 |  Installed |
|  ARM EABI v7a System Image | 23 | 3 |  Installed |
|  Intel x86 Atom_64 System Image | 23 | 3 |  Installed |
|  Intel x86 Atom System Image | 23 | 3 |  Installed |

必须安装 Android SDK Tools 修订 24.3.4 或更高版本。有关使用 Android SDK 管理器安装 Android 6.0 SDK 的详细信息，请参阅[SDK 管理器](#)。

启动 Xamarin.Android 项目

创建新的 Xamarin.Android 项目。如果您不熟悉如何使用 Xamarin 进行 Android 开发，请参阅[Hello, Android](#)，了解如何创建 Android 项目。

创建 Android 项目时，必须配置的版本设置应用到目标 Android 6.0 MarshMallow。若要为 Marshmallow 针对你的项目，必须配置的项目**API 级别 23 (Xamarin.Android 6.0 版支持)**。有关配置 Android API 级别级别的详细信息，请参阅[了解 Android API 级别](#)。

配置仿真器或设备

如果使用仿真程序，启动 Android AVD 管理器，并创建新的设备使用以下设置：

- 设备：Nexus 5、6 或 9。
- 目标：Android 6.0-API 级别 23
- ABI: x86

例如，此虚拟设备被配置为模拟 Nexus 5:

| | |
|---------------|---|
| AVD Name: | <input type="text" value="AndroidM"/> |
| Device: | <input ,="" 1080="" 1920:="" type="text" value="Nexus 5 (4.95" xxhdpi)"="" ×=""/> |
| Target: | <input type="text" value="Android 6.0 - API Level 23"/> |
| CPU/ABI: | <input type="text" value="Intel Atom (x86)"/> |
| Keyboard: | <input checked="" type="checkbox"/> Hardware keyboard present |
| Skin: | <input type="text" value="HVGA"/> |
| Front Camera: | <input type="text" value="None"/> |
| Back Camera: | <input type="text" value="None"/> |

如果使用的物理设备，如 Nexus 5, 6, 或 9, 可以安装 Android Marshmallow 的预览图像。有关将设备更新到 Android Marshmallow 的详细信息，请参阅[硬件系统映像](#)。

新增功能

许多 Android Marshmallow 中引入的更改被侧重于改进 Android 用户体验、提高性能，并修复的 bug。但是，Marshmallow 还引入了一些广泛更改 Android 平台的基础知识。以下部分突出显示这些增强功能，并提供链接，以帮助你开始在应用程序中使用新的 Android Marshmallow 功能。

运行时权限

Android 权限系统已显著优化，并简化自 Android 棒棒糖。在 Android Marshmallow 中的用户授予权限的情况的基础，在运行时中，而不是在安装时间。若要支持此功能在 Android Marshmallow 上及更高版本，设计应用时提示用户获取在运行时（在所需权限的上下文中）的权限。此更改使用户更轻松地开始使用您的应用程序立即因为它简化了安装和升级您的应用程序的过程。

请参阅[在 Android Marshmallow 请求运行时权限](#)有关 Xamarin.Android 应用中实现运行时权限（包括代码示例）的详细信息。Xamarin 还提供了示例应用，演示了在 Android Marshmallow（及更高版本），运行时权限的工作原理：[RuntimePermissions](#)。

此示例应用演示如下：

- 如何检查以及在运行时的请求权限。
- 如何声明适用于 Android M 设备的权限。

若要使用此示例应用：

1. 点击**照相机或联系人**按钮以显示权限请求对话框。
2. 授予查看照相机或联系人片段的权限。

有关在 Android Marshmallow 中新的运行时权限功能的详细信息，请参阅[使用系统权限](#)。

身份验证增强功能

Android Marshmallow 包含使您不再需要的密码的两个身份验证增强功能：

- **指纹身份验证**—使用指纹扫描用户进行身份验证。
- **确认凭据**—基于多长时间，设备已解锁的用户进行身份验证。

链接和下一步所述的示例应用程序可以借助这些新功能来帮助你熟悉。

指纹身份验证

在设备上支持的具有指纹扫描硬件，可以使用新 `FingerprintManager` 类，以对用户进行身份验证。有关在 Android Marshmallow 指纹身份验证功能的详细信息，请参阅[指纹身份验证](#)。

Xamarin 提供了示例应用，演示了如何使用已注册的指纹进行身份验证应用程序中的用户：[FingerprintDialog](#)。

若要使用此示例应用：

1. 触摸**采购按钮**以打开指纹身份验证对话框。
2. 在你已注册的指纹进行身份验证中进行扫描。

请注意，此示例应用程序要求使用指纹读取器的设备。此应用不存储你的指纹（或你的密码）。

语音交互

在 Android Marshmallow 中引入的新的语音交互功能允许您的应用程序的用户使用他们的语音以确认操作并从选项列表中选择。有关语音交互的详细信息，请参阅[语音交互 API 概述](#)。

请参阅[添加到 Android 应用使用的语音交互会话](#)有关在 Xamarin.Android 应用中实现的语音交互（包括代码示例）的详细信息。提供了示例应用演示如何在 Xamarin.Android 应用程序中使用语音交互 API：[语音交互](#)。

确认凭据

使用新**确认凭据**功能的 Android Marshmallow 可以释放用户无需记住，并且通过进行身份验证根据多长时间，其设备已解锁其输入特定于应用的密码。若要执行此操作，您使用的新 `SetUserAuthenticationValidityDurationSeconds` 方法的 `KeyGenerator`。使用 `KeyGuardManager` 的 `CreateConfirmDeviceCredentialIntent` 方法重新进行身份验证从您的应用程序中的用户。有关在 Android Marshmallow 此新功能的详细信息，请参阅[确认凭据](#)。

Xamarin 提供了示例应用，演示了如何在应用中使用设备凭据（如 PIN、图案或密码）：[ConfirmCredential](#)

若要使用此示例应用：

1. 设置你的设备上安全锁定屏幕（安全 > 安全 > **Screenlock**）。

2. 点击采购按钮, 然后确认安全锁定屏幕凭据。

Chrome 自定义选项卡

在用户点击 URL 时, 应用程序开发人员面临一个选择: 应用程序可以启动浏览器或使用基于应用程序内浏览器 `WebView`。这两个选项提出了挑战-启动浏览器是可自定义, 而不是大量上下文切换 `WebView` 与浏览器不共享状态。此外, 使用的 `WebView` 可以添加额外的维护开销。

*Chrome 自定义选项卡*使您可以轻松地完美地而无需离开您的应用程序用户显示与 Chrome 强大的网站。此功能可为您的应用程序提供更好地控制用户的 web 体验, 使本机之间的转换和 web 内容更加无缝而不必求助于 `WebView`。Chrome 的外观和通过自定义以下大家也会影响您的应用程序:

- 工具栏颜色
- 进入和退出动画
- Chrome 工具栏和溢出菜单中的自定义操作
- 开始前 chrome 和内容预提取 (的加载速度更快)

若要充分利用此功能的 Xamarin.Android 应用程序中, 下载并安装[Android 支持自定义选项卡库](#)。有关此功能的详细信息, 请参阅[Chrome 自定义选项卡](#)。

材料设计支持库

Android Lollipop 引入[Material Design](#)作为新的设计语言来刷新 Android 体验 (请参阅[材料主题](#)有关 Xamarin.Android 应用中使用材料设计)。Google 引入了与 Android Marshmallow *Android 设计支持库*为了简化应用程序开发人员采用材料设计外观和感觉。此库包括以下组件:

- **CoordinatorLayout** - 新的 `CoordinatorLayout` 小组件是与类似, 但比功能更强大 `FrameLayout`。可以使用 `CoordinatorLayout` 作为子视图的容器或一个顶级的布局, 和它提供了 `layout_anchor` 特性的定位点相对于其他视图的视图。
- **折叠工具栏** - 新的 `CollapsingToolbarLayout` 是折叠的应用栏, 它是包装 `Toolbar`。(请注意, *应用程序栏*是什么以前称为 *操作栏*。)
- **浮点操作按钮** - 表示应用程序的接口上的主要操作圆形按钮。
- **浮点标签编辑文本** - 使用新 `TextInputLayout` 小组件 (用于包装 `EditText`) 时提示用户输入文本, 则隐藏显示浮动的标签。
- **导航视图** - 新 `NavigationView` 小组件可帮助你为用户更轻松地导航的方式使用导航抽屉。
- **Snackbar** - 新 `SnackBar` 小组件是一个轻型的反馈机制 (类似于 toast 通知), 在屏幕上, 使其不显示最重要的屏幕上的其他元素的底部显示一个简短的消息。
- **材料的选项卡** - 新 `TabLayout` 小组件提供了水平布局, 用于为应用程序中实现的顶级导航方法显示选项卡。

若要充分利用[设计支持库](#)在 Xamarin.Android 应用中, 下载并安装 Xamarin [Xamarin 支持库设计](#) NuGet 包。

请参阅[美观的材料设计与 Android 支持设计库](#)有关使用 Xamarin.Android 应用中的材料设计支持库 (包括代码示例) 的详细信息。Xamarin 提供了示例应用, 演示了新的 Android 设计库在 Xamarin.Android - [Cheesecake](#)。此示例演示了设计库的以下功能:

- 折叠工具栏
- 浮动的操作按钮
- 视图锚定
- NavigationView
- Snackbar

有关设计库的详细信息, 请参阅[Android 设计支持库](#) Android 开发人员的博客中。

其他库更新

除了 Android Marshmallow, Google 已宣布对几个核心 Android 库的相关的更新。Xamarin 提供了 Xamarin.Android 支持这些更新通过多个预览版本的 NuGet 包:

- [Google Play Services](#) – Google Play 服务的最新版本包括新[应用程序邀请](#)功能, 从而使用户能够与朋友共享自己的应用程序。有关此功能的详细信息, 请参阅[使用 Google 的应用程序邀请的展开您的应用程序的市场宣传](#)。
- [Android 支持库](#)–这些 Nuget 提供的同时提供向后兼容版本的 Android 框架 Api 属性仅适用于库 Api 的功能。
- [Android 可穿戴库](#)–此 NuGet 包括 Google Play 服务绑定。可穿戴库的最新版本为 Android Wear 平台带来了新功能 (包括用于自定义应用更易于导航)。

总结

本文引入 Android Marshmallow 并介绍了如何安装和 Marshmallow 上配置的最新工具和 Xamarin.Android 开发的包。它还提供有关 Xamarin.Android 开发的最令人兴奋的新 Android Marshmallow 功能的概述。

相关链接

- [Android 6.0 Marshmallow](#)
- [获取 Android SDK](#)
- [功能概述](#)
- [发行说明](#)
- [RuntimePermissions \(示例\)](#)
- [ConfirmCredential \(示例\)](#)
- [FingerprintDialog \(示例\)](#)

棒棒糖功能

2018/10/26 • [Edit Online](#)

本文提供了在 Android 5.0 (Lollipop) 中引入的新功能的高级别概述。这些功能包括新的用户界面样式称为材料主题还包含新的支持功能, 如动画、视图阴影和 drawable 色调。Android 5.0 还包括增强的通知、两个新的 UI 小组件、新的作业计划程序和少量新的 Api 来提高存储、网络、连接性和多媒体功能。

棒棒糖概述

Android 5.0 (Lollipop) 引入了新的设计语言, *Material Design*, 并为之支持强制转换的新功能, 可让应用更加轻松、更直观地使用。使用 Material Design Android 5.0 不只提供 Android 手机改动; 它还提供一组新的设计规则, 用于基于 Android 的平板电脑、台式计算机、监视器和智能电视。这些设计规则强调简单性和 minimalism 同时充分利用熟悉触觉属性 (如实际的图面和边缘提示), 以帮助用户快速、直观地了解到该接口。

材料主题是 Android UI 设计原则的体现。本文首先介绍材料主题支持功能:

- **动画** – 触摸反馈动画 活动转换动画 查看状态转换动画和显示效果。
- **查看阴影和提升** – 视图现在具有 `elevation` 属性; 与视图更高版本 `elevation` 值强制转换可查看大阴影背景上的。
- **颜色功能** – *Drawable 色调* 使您可以通过更改其颜色, 重用图像资产并突出的颜色提取可帮助你动态您的应用程序基于图像中颜色主题。

很多材料主题功能已内置到 Android 5.0 UI 体验, 而其他人必须显式添加到应用程序。例如, 某些标准视图 (如按钮) 时, 应用必须启用大多数视图阴影已包括触摸反馈动画。

除了通过材料主题带来的 UI 改进, Android 5.0 还在本文中包含几个介绍了其他新功能:

- **增强的通知** – Android 5.0 中的通知已大幅更新使用全新的外观、对锁屏通知的支持和新 *平视* 通知显示格式。
- **新的 UI 小组件** – 新的 `RecyclerView` 小组件可简化应用程序以提供大型数据集和复杂信息和新 `CardView` 小组件提供的简化类似于卡的显示格式, 用于显示文本和映像。
- **新的 Api** – Android 5.0 将添加新 Api, 可用于多个网络支持, 改进的蓝牙连接、更简单的存储管理, 以及更灵活地控制多媒体播放器和摄像机设备。新的作业日程安排功能是可用于运行任务以异步方式在计划的时间。此功能有助于提高电池使用寿命, 例如, 计划任务在插入设备时就位和收费。

要求

以下是所需的基于 Xamarin 的应用中使用新的 Android 5.0 功能:

- **Xamarin.Android** – Xamarin.Android 4.20 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for mac。
- **Android SDK** – Android 5.0 (API 21) 或更高版本必须安装通过 Android SDK 管理器。
- **Java 开发人员工具包** – Xamarin.Android 需要 [JDK 1.8](#) 或更高, 如果你正在开发的 API 级别 24 或更高版本 (JDK 1.8 还支持 API 级别低于 24, 包括棒棒糖形)。如果使用的自定义控件或窗体预览程序, 则需要 JDK 1.8 的 64 位版本。

你可以继续使用 [JDK 1.7](#) 如果您是开发专门针对 API 级别 23 或更早版本。

设置 Android 5.0 项目

若要创建 Android 5.0 项目时，必须安装最新工具和 SDK 包。使用以下步骤设置 Xamarin.Android 项目面向 Android 5.0:

1. 安装 Xamarin.Android 工具并激活 Xamarin 许可证。请参阅[设置和安装](#)有关安装 Xamarin.Android 的详细信息。
2. 如果你使用 Visual Studio for Mac, 安装最新的 Android 5.0 更新。
3. 启动 Android SDK 管理器 (在 Visual Studio for Mac 中, 使用工具>打开 **Android SDK 管理器...**) 并安装 Android SDK Tools 23.0.5 或更高版本:

| Name | API | Rev. | Status |
|----------------------------|-----|--------|-----------|
| Tools | | | |
| Android SDK Tools | | 23.0.5 | Installed |
| Android SDK Platform-tools | | 21 | Installed |
| Android SDK Build-tools | | 21.1 | Installed |

此外, 安装最新的 Android 5.0 SDK 包 (API 21 或更高版本):

| | | | |
|--|----|---|-----------|
| Android 5.0 (API 21) | | | |
| Documentation for Android SDK | 21 | 1 | Installed |
| SDK Platform | 21 | 1 | Installed |
| Android TV ARM EABI v7a System Image | 21 | 1 | Installed |
| Android TV Intel x86 Atom System Image | 21 | 1 | Installed |
| ARM EABI v7a System Image | 21 | 1 | Installed |
| Intel x86 Atom_64 System Image | 21 | 1 | Installed |
| Intel x86 Atom System Image | 21 | 1 | Installed |
| Google APIs | 21 | 1 | Installed |
| Google APIs ARM EABI v7a System Image | 21 | 2 | Installed |
| Google APIs Intel x86 Atom_64 System Image | 21 | 2 | Installed |
| Google APIs Intel x86 Atom System Image | 21 | 2 | Installed |

有关使用 Android SDK 管理器的详细信息, 请参阅[SDK 管理器](#)。

4. 创建新的 Xamarin.Android 项目。如果您不熟悉如何使用 Xamarin 进行 Android 开发, 请参阅[Hello, Android](#), 了解如何创建 Android 项目。创建 Android 项目时, 请确保配置的 Android 5.0 版本设置。在 Visual Studio for Mac 中, 导航到项目选项>构建>常规并设置目标框架到**Android 5.0 (Lollipop)** 或更高版本:

Target framework: Android 5.0 (Lollipop)

下项目选项>构建>**Android** 应用程序, 设置最小和目标 Android 版本自动-使用目标框架版本:

Minimum Android version Automatic – use target framework version

Target Android version Automatic – use target framework version

5. 配置要测试你的应用的仿真程序或 Android 设备。如果要使用模拟器, 请参阅[Android 仿真程序安装程序](#)若要了解如何使用 Xamarin Studio 或 Visual Studio 中配置用于 Android 模拟器。如果使用 Android 设备, 请参阅[设置预览版 SDK](#)若要了解如何为 Android 5.0 更新你的设备。若要配置用于运行和调试 Xamarin.Android 应用程序在 Android 设备, 请参阅[设置设备进行开发](#)。

注意: 如果要更新现有的 Android 项目是面向 Android 的 L 预览, 则必须更新目标框架并**Android** 版本上面所述的值。

重要的更改

以前已发布 Android 应用程序可能会影响 Android 5.0 中的更改。具体而言, Android 5.0 使用新的运行时和显著更改的通知格式。

Android 运行时

Android 5.0 使用而不是 Dalvik 默认运行时作为新 Android 运行时（图片）。艺术实现几个主要的新增功能：

- **提前时间的 (AOT) 编译** – AOT 编译应用程序代码之前首次启动应用程序可以提高应用性能。安装应用时，画生成已编译的应用可执行文件的目标设备。
- **改进了垃圾回收 (GC)** – 画中的 GC 改进方面还可以提高应用性能。现在垃圾回收使用一个 GC 暂停，而不是两个，并更及时的并发 GC 操作完成。
- **改进了应用程序调试** – 画面提供了更多诊断的详细信息，以帮助分析异常和故障报告。

现有应用程序应进行任何修改下画 – 除了利用技术到上一 Dalvik 运行时唯一的应用，这可能无法工作下画。有关这些更改的详细信息，请参阅[验证应用行为在 Android 运行时（艺术）](#)。

通知更改

通知发生了重大更改 Android 5.0 中：

- **以不同方式处理声音和振动** – 通知听起来和振动现在由 `Notification.Builder` 而不是 `Ringtone`，`MediaPlayer`，并 `Vibrator`。
- **新的配色方案** – 材料主题根据通知用深文本呈现在白色或非常浅色背景上。此外，Android 来协调使用系统配色方案可能修改中通知图标的 alpha 通道。
- **锁屏通知** – 现在可以在设备锁屏上显示通知。
- **平视** – 高优先级的通知现在显示在小的浮动窗口（危险警告通知）时该设备已解锁，并打开屏幕。

在大多数情况下，移植到 Android 5.0 的现有应用程序通知功能需要执行以下步骤：

1. 转换代码以使用 `Notification.Builder`（或 `NotificationsCompat.Builder`）用于创建通知。
2. 验证你的现有通知资产可在新材料主题配色方案中查看。
3. 决定您的通知时显示在锁屏上应具有何种可见性。如果通知不是公共的哪些内容应显示在锁屏上？
4. 设置你的通知的类别，因此它们在新的 Android 5.0 中正确处理 *请勿打扰* 模式。

如果您的通知显示传输控件，显示媒体播放状态，使用 `RemoteControlClient`，或调用 `ActivityManager.GetRecentTasks`，请参阅[重要行为更改](#)有关更新适用于 Android 通知的详细信息 5.0。

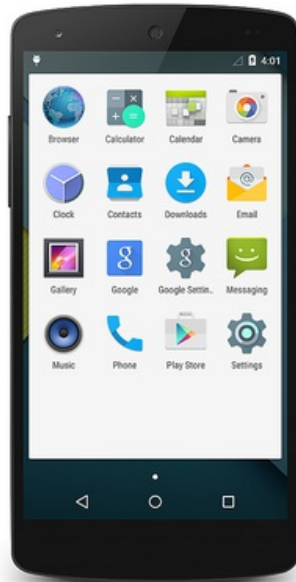
在 Android 中创建通知的信息，请参阅[本地通知](#)。[兼容性](#) 本文的部分说明了如何创建通知来向下兼容使用早期版本的 Android。

材料主题

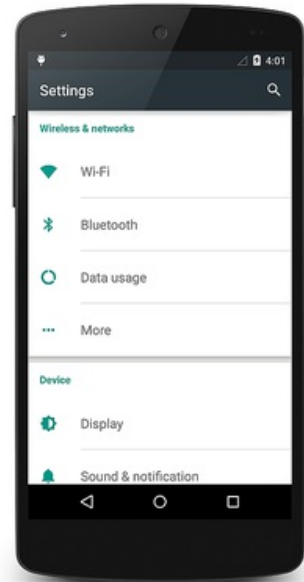
新的 Android 5.0 材料主题为 Android 用户界面的外观和感觉带来了重大改动。可视元素现在使用边用粗体显示的图形、版式和打印目的设计的明亮的颜色对其执行的图面。材料主题的示例中的以下屏幕截图所示：



Home Screen



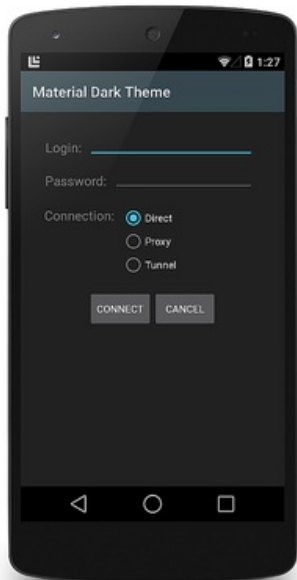
Apps Screen



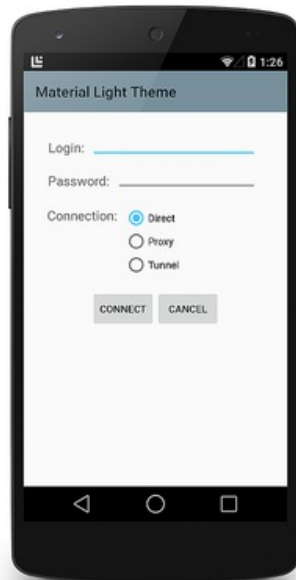
Settings Screen

Android 5.0 向您致意显示在左侧的主屏幕。中心的屏幕截图是第一个屏幕的应用列表中，并在右侧屏幕截图是设置屏幕。Google [Material Design](#)规范说明新材料主题概念背后的基础设计规则。

材料主题包括可以在应用中使用的三种内置风格：`Theme.Material` 深色主题（默认值），`Theme.Material.Light` 主题，和 `Theme.Material.Light.DarkActionBar` 主题：



Theme.Material



Theme.Material.Light



Theme.Material.Light.DarkActionBar

有关在 Xamarin.Android 应用中使用材料主题功能的详细信息，请参阅[材料主题](#)。

动画

Android 5.0 提供了触摸反馈动画、活动过渡动画和视图状态过渡动画，以使应用界面更加直观使用。此外，Android 5.0 应用程序可以使用显示效果动画来隐藏或显示视图。可以使用曲线运动呈现需要如何快速配置设置或缓慢的动画。

触摸反馈动画

视图具有触摸时，触摸反馈动画向用户提供可视反馈。例如，按钮现在显示波纹效果时接触—这是在 Android 5.0

默认触摸反馈动画。Ripple 动画实现的新 `RippleDrawable` 类。波纹效果可以配置为视图的边界处结束或扩展到视图的边界之外。例如，下面的屏幕截图序列触摸动画期间说明按钮中波纹效果：



初始触控接触与按钮出现在左侧的第一个图像中的剩余序列（从左到右）说明了如何波纹效果向外扩散到按钮的边缘。在 ripple 动画结束时，视图将返回到其原始的外观。默认波纹动画发生在第二个，一小部分，但可以自定义动画的长度的时间更长或更短长度。

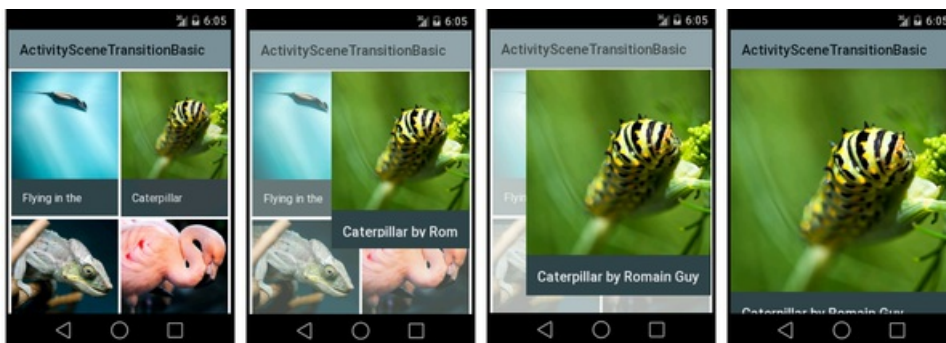
触摸反馈动画在 Android 5.0 中的详细信息，请参阅[自定义触摸反馈](#)。

活动过渡动画

当转换到另一个活动，活动过渡动画为用户提供 visual 连续性的意义。应用程序可以指定三种类型的过渡动画：

- **输入过渡**-为当某个活动进入场景。
- **退出过渡**-为活动退出场景时。
- **共享元素过渡**-为当普遍适用于两个活动的视图更改为，第一个活动将转换为下一步。

例如，以下一系列屏幕截图显示了共享的元素转换：



共享的元素（毛虫照片）是一个多个视图中的第一个活动;它将放大成为第二个活动的形式，第一个活动将转换为第二个中的唯一视图。

输入转换的动画类型

Enter 键转换为 Android 5.0 提供了三种类型的动画：

- **Explode 动画**-放大场景的中心中的视图。
- **滑动动画**-将视图中移动从一个场景的边缘。
- **淡入动画**-淡场景的视图。

退出过渡动画类型

对退出过渡，Android 5.0 提供了三种类型的动画：

- **Explode 动画**-收缩到中心的场景的视图。
- **滑动动画**-将一个视图移到的一个场景的边缘。
- **淡入动画**-淡入淡出场景的视图。

共享元素转换的动画类型

共享的元素转换支持多个类型的动画，如：

- 更改视图的布局或剪辑边界。

- 更改的规模和视图的旋转。
- 更改视图的大小和扩展类型。

有关在 Android 5.0 中的活动过渡动画的详细信息，请参阅[自定义活动转换](#)。

视图状态过渡动画

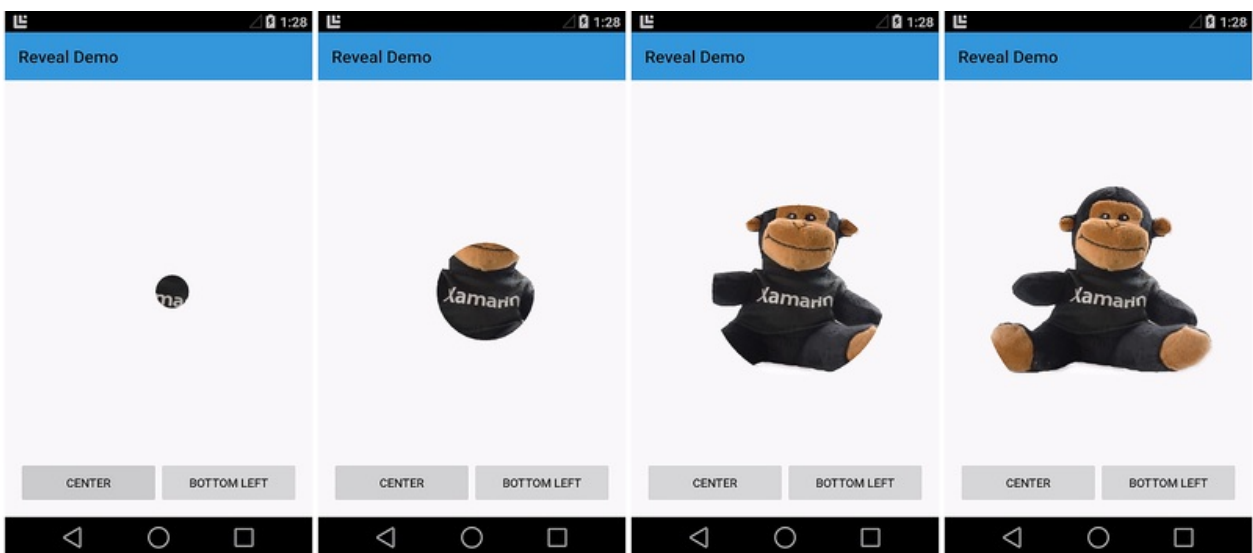
Android 5.0 使动画运行时的视图状态更改。您可以使用以下方法之一对视图状态转换进行动画处理：

- 创建进行动画处理的特定视图相关联的状态更改的绘图。新 `AnimatedStateListDrawable` 类可用于创建显示视图状态更改之间的动画的绘图。
- 定义的视图状态更改时运行的动画功能。新 `StateListAnimator` 类允许您定义的视图状态更改时运行动画。

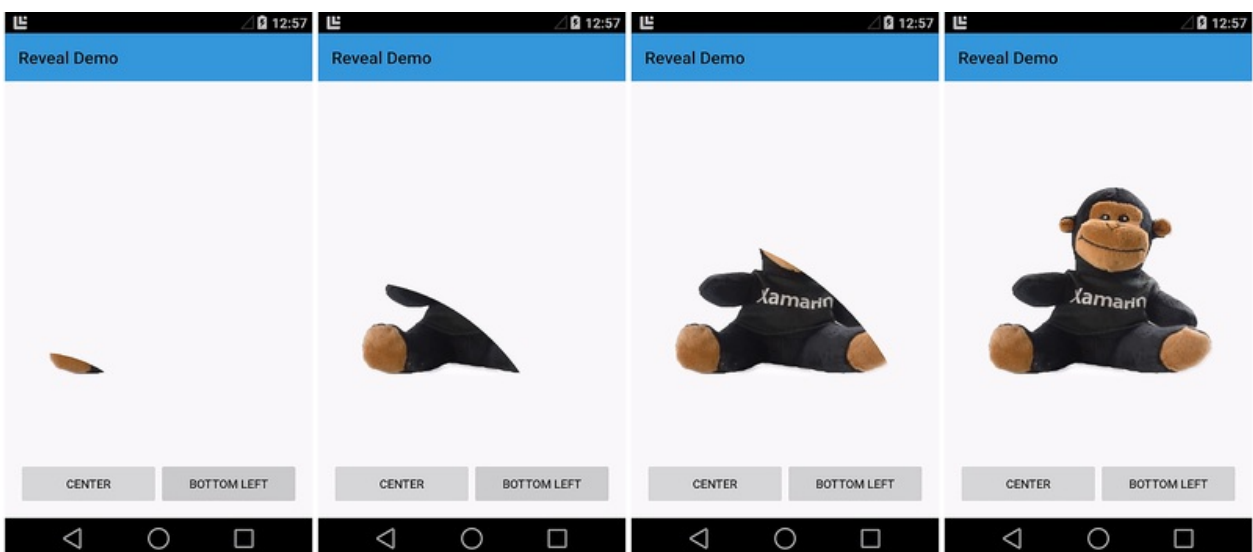
有关在 Android 5.0 中的视图状态过渡动画的详细信息，请参阅[进行动画处理视图状态更改](#)。

显示效果

显示效果是一个剪辑的圆的更改半径，以显示或隐藏视图。可以通过设置剪辑圆的初始和最终半径控制这种效果。以下一系列屏幕截图演示了显示的效果动画从中心的屏幕：



下一步的序列演示了一个发生从屏幕的左下角的显示效果动画：



显示动画，可以撤消;也就是说，剪辑圆圈可以缩小以隐藏视图而不是放大以显示该视图。

中的 Android 5.0 显示效果的详细信息，请参阅[使用显示效果](#)。

曲线的动作

除了这些动画功能, Android 5.0 还提供了新的 Api, 使您能够指定动画的时间和运动曲线。Android 5.0 使用这些曲线内插在动画期间在临时和空间移动。Android 5.0 中定义了三种曲线:

- **快速_出_线性_中**-快速加快并加速动画结束时才将继续。
- **快速_出_慢速_中**-加快快速和缓慢减速的动画结束时。
- **线性_出_慢速_中**-与峰值速度缓慢的开头为动画减速。

你可以使用新 `PathInterpolator` 类, 以指定动画内插进行。 `PathInterpolator` 是遍历动画路径根据指定的控制点和运动曲线内插器。有关如何在 Android 5.0 中指定曲线的运动设置的详细信息, 请参阅[使用曲线运动](#)。

查看阴影和提升

在 Android 5.0 中, 可以指定 *提升* 视图通过设置一个新的 `z` 属性。一个大于 `z` 值导致要投影更大的背景, 使视图显示转换为浮点型上面在后台更高版本上的视图。可以通过配置来设置视图的初始权限提升其 `elevation` 布局中的属性。

下面的示例演示了一个空进行强制转换阴影 `TextView` 控制当其提升属性设置为 2dp、4dp 和 6dp, 分别:



查看卷影设置可以是静态的 (如上所示), 或它们可在动画以使视图显示视图的背景上暂时上升。可以使用 `ViewPropertyAnimator` 类进行动画处理视图的提升。视图的仰角是其布局的总和 `elevation` 设置加上 `translationZ` 属性, 可通过设置 `ViewPropertyAnimator` 方法调用。

有关 Android 5.0 中的视图阴影的详细信息, 请参阅[定义阴影和剪辑视图](#)。

颜色功能

Android 5.0 提供了用于管理应用中的颜色的两项新功能:

- **可绘制色调**可以通过更改布局属性更改颜色的图像资产。
- **突出的颜色提取**使动态自定义应用的颜色主题与显示的图像的颜色调色板进行协调。

可绘制色调

Android 5.0 布局识别新 `tint` 属性, 可以用于绘图的颜色设置而无需创建多个版本的这些资产, 以显示不同的颜色。若要使用此功能, 您定义位图为 alpha 掩码和使用 `tint` 属性来定义资产的颜色。这使得您可以一次创建资产并在布局以匹配你的主题中的这些颜色。

在下面的示例中, 单个图像资产-带有透明背景的白色徽标-用于创建色彩变体:



此徽标会显示上面蓝色圆圈为背景，如以下示例所示。在左侧图像，而无需徽标的显示方式 `tint` 设置。在中心图中，该徽标的 `tint` 属性设置为暗灰色。在右侧，映像中 `tint` 设置为浅灰色：



有关可绘制、在 Android 5.0 中的详细信息，请参阅[Drawable](#)。

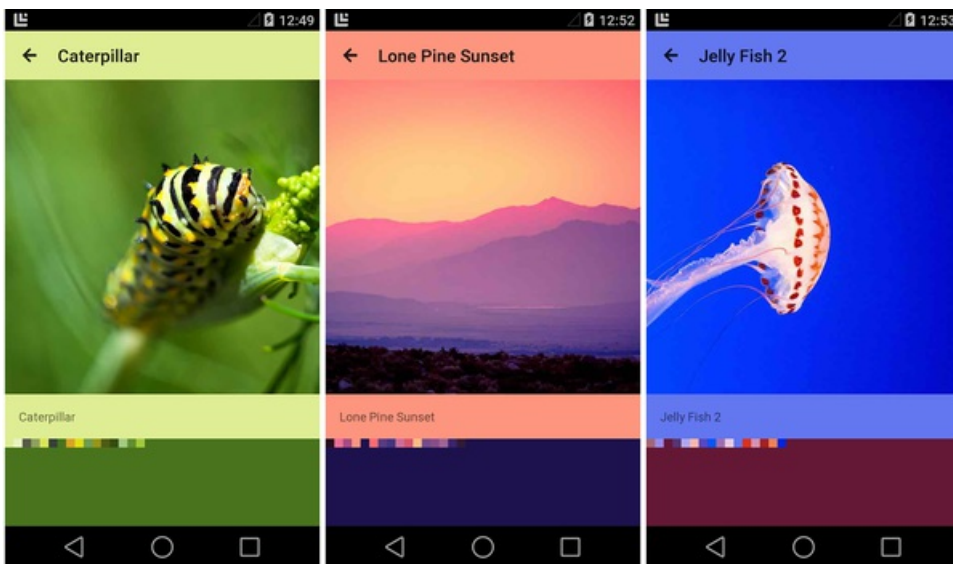
突出的颜色提取

新的 Android 5.0 `Palette` 类使您可以从图像中提取颜色，以便您可以动态地将它们应用到自定义调色板。

`Palette` 类从图像中提取六种颜色和标签这些颜色根据其相对的彩色饱和度和亮度级别：

- 充满活力
- 充满活力的深色
- 充满活力的光
- 静音
- 静音的深色
- 静音的光

例如，在下面的屏幕截图，照片查看应用程序从上显示的图像中提取突出的颜色，并使用这些颜色调整的应用程序配色方案，以匹配图像：



在上面的屏幕中，在操作栏设置为提取"充满活力浅色"颜色和背景设置为提取"充满活力深色"颜色。在上述每个示例中，小型彩色的平方值的行是包括在内，以说明了从图像中提取的调色板颜色。

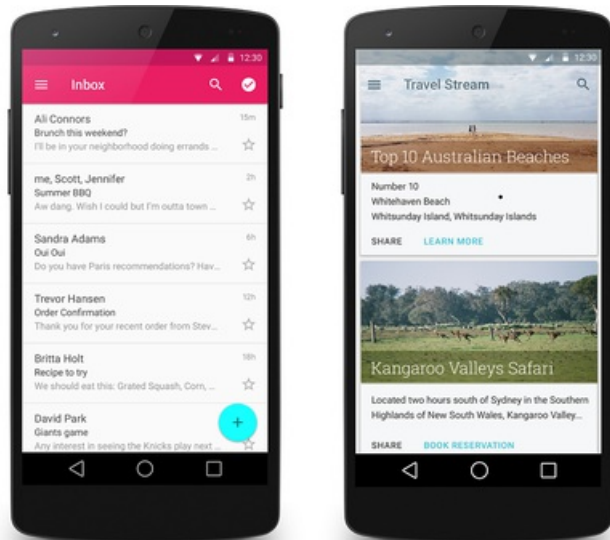
有关在 Android 5.0 中的颜色提取有关的详细信息，请参阅[从映像中提取突出显示颜色](#)。

新的 UI 小组件

Android 5.0 引入了两个新的 UI 小组件：

- `RecyclerView` – 一组视图，显示的可滚动项列表。
- `CardView` – 具有圆角的基本布局。

这两个小组件包括中随附材料主题功能; 支持例如，`RecyclerView` 用于添加和删除视图，使用动画和 `CardView` 使用查看使每个卡似乎浮在背景的阴影。在下面的屏幕截图显示了这些新的小组件的示例：



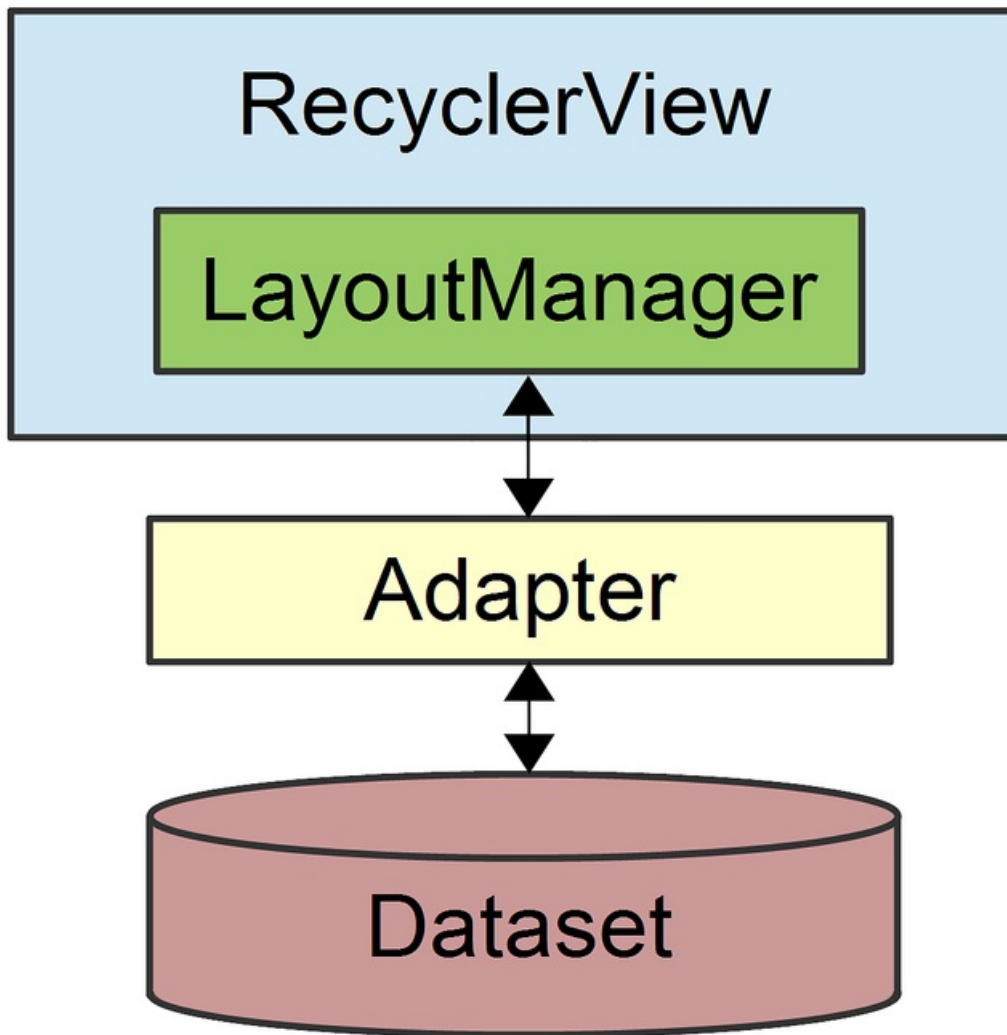
在左侧的屏幕截图是一种 `RecyclerView` 上使用电子邮件应用和屏幕截图中右侧是举例说明 `CardView` 旅游预订应用中使用的。

RecyclerView

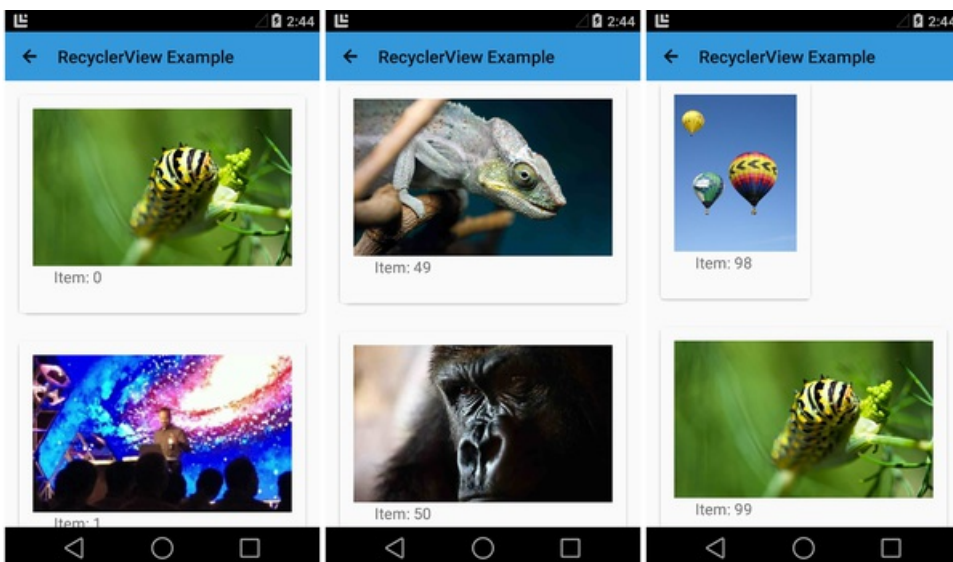
`RecyclerView` 类似于 `ListView`，但更好地适用于大型集的视图或动态更改的元素的列表。如 `ListView`，指定适配器以访问基础数据集。但是，与不同 `ListView`，则使用 `布局管理器` 若要定位项目内 `RecyclerView`。布局管理器还负责视图回收; 它管理项将不再对用户可见的视图的重用。

当你使用 `RecyclerView` 小组件中，您必须指定 `LayoutManager` 和适配器。此图中所示 `LayoutManager` 是适配器之间

的媒介和 RecyclerView :

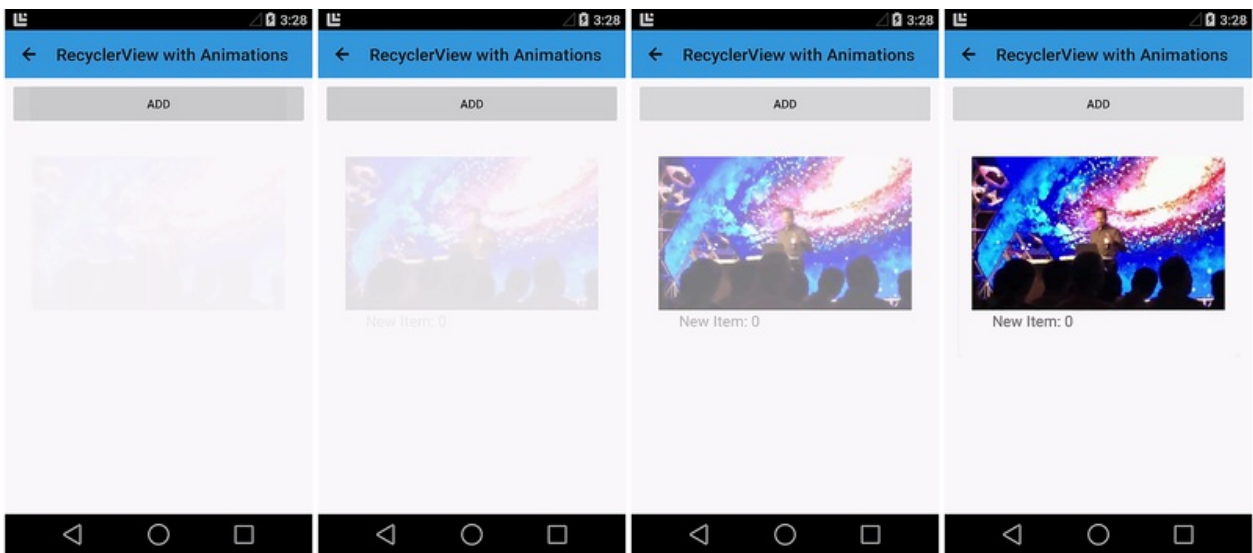


下面的屏幕截图显示 RecyclerView , 其中包含 100 个项 (每个项组成 ImageView 和一个 TextView):



RecyclerView 轻松地处理此大型数据集—滚动列表以结束开始从在此示例中的列表的应用程序需要仅几秒钟。

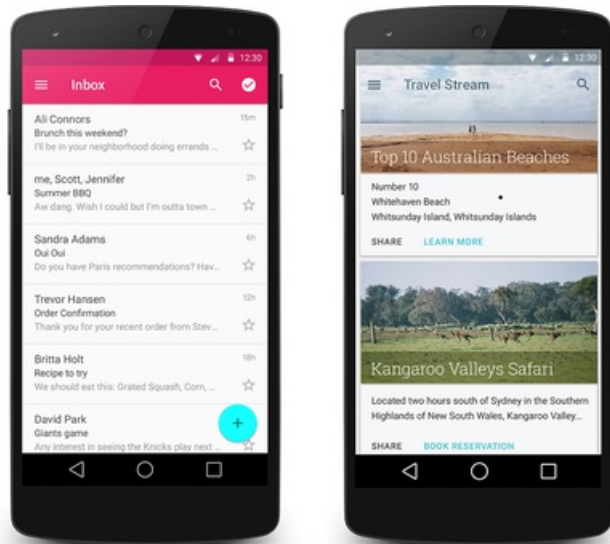
RecyclerView 此外支持动画;事实上, 默认情况下启用用于添加和删除项的动画。当某项添加到 RecyclerView , 它淡在这一序列的屏幕截图中所示:



有关详细信息 `RecyclerView`，请参阅[RecyclerView](#)。

CardView

`CardView` 是一个简单的视图，用于模拟带有圆角的浮动卡。因为 `CardView` 具有内置视图阴影，它提供了用于轻松对你要向应用添加视觉深度。以下屏幕截图显示三个面向文本的示例 `CardView`：

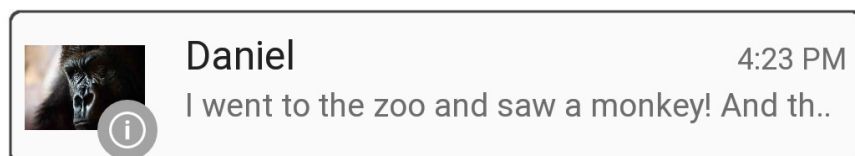


在上面的示例数据卡的每个包含 `TextView`；通过设置背景色 `cardBackgroundColor` 属性。

有关详细信息 `CardView`，请参阅[CardView](#)。

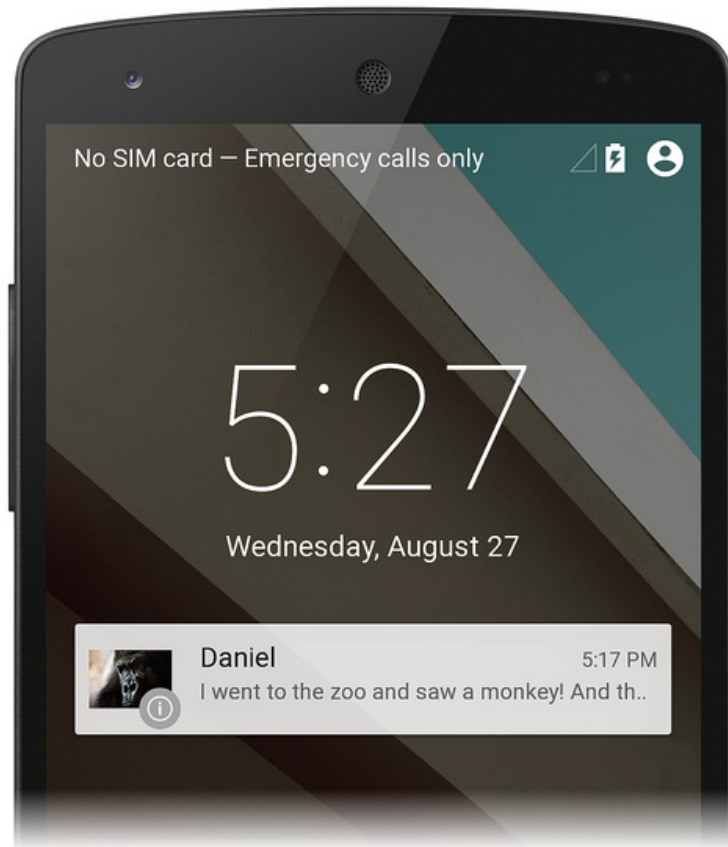
增强的通知

已使用新的可视化格式和新功能显著更新 Android 5.0 中的通知系统。通知 Android 5.0 中拥有了新外观。例如，Android 5.0 中的通知现在深文本通过使用浅色背景：



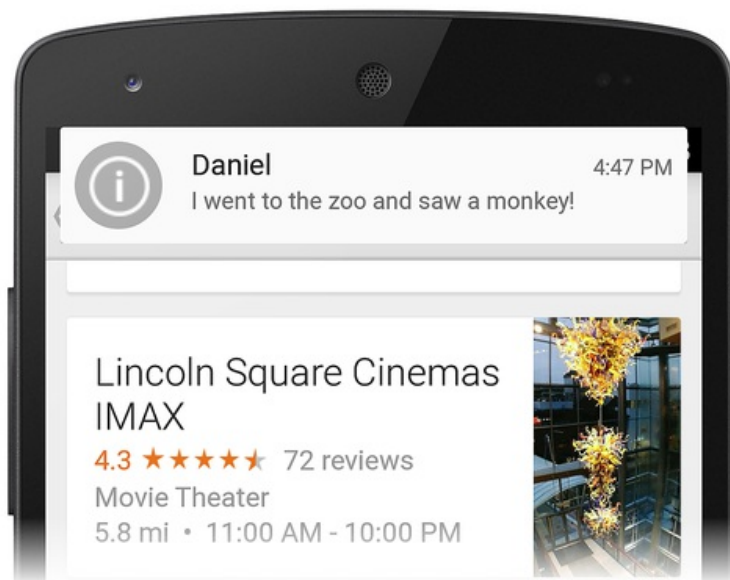
当大图标显示在通知（如上面的示例中所示）时，Android 5.0 会小图标的徽章显示大图标上。

在 Android 5.0，通知也会出现在设备锁屏上。例如，下面是使用一条通知锁屏的示例屏幕截图：



用户可以双击锁屏以解锁设备，并跳转到源自该通知的应用的通知或轻扫以取消通知。通知有一个新 *可见性* 设置确定可以在锁屏上显示内容的数量。用户可以选择是否允许在锁屏通知中显示的敏感内容。

Android 5.0 引入了新的高优先级通知演示文稿格式称为 *平视*。危险警告通知从屏幕顶部几秒钟向下滑动，然后返回到屏幕的顶部通知阴影撤回。危险警告通知使系统用户界面而不会中断当前正在运行的活动将用户的重要信息。下面的示例演示了显示在应用程序之上的简单危险警告通知：



危险警告通知通常用于以下事件：

- 新的下一条消息
- 传入的电话呼叫
- 指示电池电量不足
- 警报

仅当它具有高或最大优先级设置时，android 5.0 格式平视显示通知。

在 Android 5.0 中，可以提供通知元数据，帮助 Android 进行排序和更智能地显示通知。Android 5.0 组织根据优先级、可见性和类别的通知。通知类别用于筛选器可以显示的通知，当设备处于 *请勿打扰* 模式。

有关创建和启动通知与最新的 Android 5.0 功能的详细信息，请参阅[本地通知](#)。

新 API

除了上面所述的新外观和体验功能，Android 5.0 将添加新 Api 用于扩展现有的多媒体功能、存储和无线/连接功能。此外，Android 5.0 包括了新的 Api，为新的作业计划程序功能提供支持。

照相机

Android 5.0 为增强的照相机功能提供了几个新 Api。新 `Android.Hardware.Camera2` 命名空间包含的功能，为访问单个摄像机设备连接到 Android 设备。此外，`Android.Hardware.Camera2` 模型作为管道的每个照相机设备：它接受捕获请求、捕获映像，并输出结果。这种方法使应用程序排队到摄像机设备的多个捕获请求。

以下 Api 实现这些新功能：

- `CameraManager.GetCameraIdList` – 可帮助你以编程方式访问相机的设备;您使用 `CameraManager.OpenCamera` 连接到特定的摄像机设备。
- `CameraCaptureSession` – 捕获或流式处理从照相机设备映像。您实现 `CameraCaptureSession.CaptureListener` 接口以处理新映像捕获的事件。
- `CaptureRequest` – 定义捕获参数。
- `CaptureResult` – 提供的映像捕获操作的结果。

有关新相机 Android 5.0 中的 Api 的详细信息，请参阅[媒体](#)。

音频播放

Android 5.0 更新 `AudioTrack` 更好的音频播放的类：

- `ENCODING_PCM_FLOAT` – 配置 `AudioTrack` 以接受更好的动态范围、更大的空间和更高质量（这得益于增加的精度）的浮点格式的音频数据。此外，浮点格式有助于避免音频剪辑。
- `ByteBuffer` – 现在可以提供音频数据到 `AudioTrack` 作为字节数组。
- `WRITE_NON_BLOCKING` – 此选项可以简化缓冲和多线程处理对于某些应用。

有关详细信息 `AudioTrack` 改进在 Android 5.0 中，请参阅[媒体](#)。

媒体播放控件

Android 5.0 引入了新 `Android.Media.MediaController` 类，它取代了 `RemoteControlClient`。

`Android.Media.MediaController` 提供简化的传输控制 Api 并提供线程安全控件的 UI 上下文以外的播放。以下新的 Api 处理传输控件：

- `Android.Media.Session.MediaSession` – 媒体控制处理多个控制器的会话。在调用 `MediaSession.GetSessionToken` 请求您的应用程序使用与会话进行交互的令牌。
- `MediaController.TransportControls` – 负责处理传输等命令**播放，停止，并跳过**。

此外，你可以使用新 `Android.App.Notification.MediaStyle` 类能够富通知内容（如提取和显示唱片集画面）相关联的媒体会话。

有关在 Android 5.0 中的新媒体播放控件功能的详细信息，请参阅[媒体](#)。

存储

Android 5.0 更新存储访问框架, 以使其容易使应用程序使用目录和文档:

- 若要选择的目录子树, 可以生成并发送 `Android.Intent.Action.OPEN_DOCUMENT_TREE` 意向。此意向会导致系统显示所有支持的子树选择; 的提供程序实例然后, 用户浏览并选择一个目录。
- 若要创建和管理新文档或下一个子树的任意位置的目录, 你使用新 `CreateDocument`, `RenameDocument`, 并 `DeleteDocument` 方法的 `DocumentsContract`。
- 若要获取所有共享的存储设备上的媒体目录路径, 请调用新 `Android.Content.Context.GetExternalMediaDirs` 方法。

有关新存储 Android 5.0 中的 Api 的详细信息, 请参阅[存储](#)。

无线和连接性

Android 5.0 将添加以下 API 用于无线和连接性的增强功能:

- 新多网络Api, 使应用程序以查找并建立连接之前, 选择具有特定功能的网络。
- 启用 Android 5.0 设备, 使其作为低功耗蓝牙外围设备的蓝牙广播功能。
- NFC 增强, 使其更轻松地与其他设备共享数据使用近场通信功能。

有关新的无线和连接 Android 5.0 中的 Api 的详细信息, 请参阅[无线和连接](#)。

作业计划

Android 5.0 引入了新 `JobScheduler` API, 可帮助用户最小化电池耗尽, 通过计划在插入设备时才运行某些任务和收费。此作业计划程序功能还可用于计划任务运行时满足条件, 更适合该任务, 如通过 Wi-fi 网络, 而不是按流量计费的网络连接设备时下载大型文件。

有关新的作业计划 Android 5.0 中的 Api 的详细信息, 请参阅[计划作业](#)。

总结

本文为 Xamarin.Android 应用程序开发人员提供的 Android 5.0 中的重要新功能的概述:

- 材料主题
- 动画
- 查看阴影和提升
- 颜色功能, 如 `drawable`、并突出显示颜色提取
- 新 `RecyclerView` 和 `CardView` 小组件
- 通知增强功能
- 相机、音频播放、media 控件、存储、无线连接和作业计划的新 Api

如果 Xamarin Android 开发新手, 请阅读[设置和安装](#)来帮助你开始使用 Xamarin.Android。Hello, Android是极好的入门学习如何创建 Android 项目。

相关链接

- [Android L 开发者预览版](#)
- [获取 Android SDK](#)
- [材料设计](#)
- [材料设计原则](#)

KitKat 功能

2018/11/13 • [Edit Online](#)

Android 4.4 (KitKat) 是加载了太多的用户和开发人员的功能。本指南重点介绍了这些功能的几个, 并提供代码示例和实现详细信息, 以帮助您充分利用 KitKat。

概述

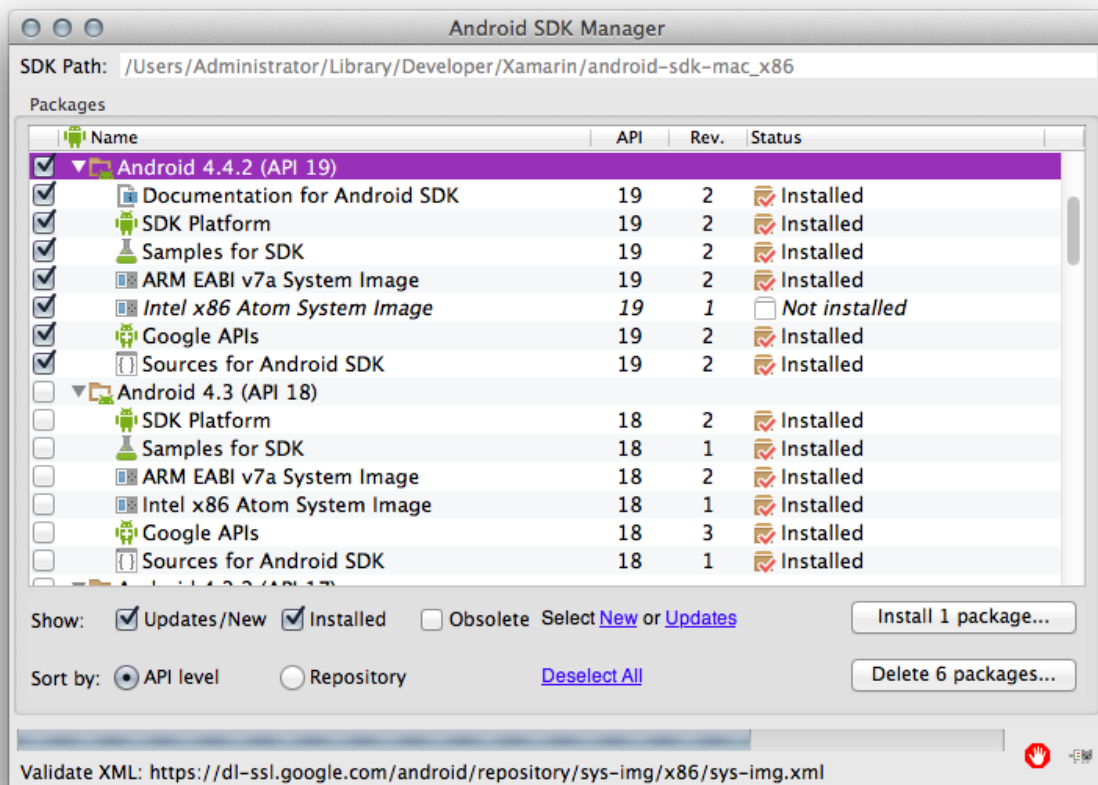
Android 4.4 (API 级别 19), 也称为"KitKat", 已在后期 2013年发布。KitKat 提供了多种新功能和改进, 包括:

- **用户体验**-与转换 framework、半透明状态和导航栏和的全屏沉浸式模式下的简单动画帮助用户更好的体验。
- **用户的内容**-用户文件管理简化了存储访问框架; 打印图片、网站和其他内容是改进了打印 Api, 更容易。
- **硬件**-到使用 NFC 基于主机的卡仿真的 NFC 卡将所有应用; 运行与的低功率传感器 `SensorManager`。
- **开发人员工具**-截屏视频中使用 Android Debug Bridge 客户端, 可用的操作的应用程序作为 Android SDK 的一部分。

本指南提供有关现有 Xamarin.Android 应用程序迁移到 KitKat, 以及 KitKat 的高级概述为 Xamarin.Android 开发人员指导。

要求

若要开发 Xamarin.Android 应用程序使用 KitKat, 需要Xamarin.Android 4.11.0或更高版本和 Android 4.4 (API 级别 19) 安装通过 Android SDK 管理器中, 如以下屏幕截图所示:



将您的应用程序迁移到 KitKat

本部分提供了一些第一个响应项以帮助转换到 Android 4.4 的现有应用程序。

检查系统版本

如果应用程序需要能够与较旧版本的 Android 兼容，请务必将任何 KitKat 特定的代码包装在系统版本检查，如下面的代码示例所示：

```
if (Build.VERSION.SdkInt >= BuildVersionCodes.Kitkat) {
    //KitKat only code here
}
```

警报批处理

Android 使用警报服务在指定时间唤醒在后台中的应用。KitKat 这一步是通过采用批处理警报以保留电源。这意味着，而非唤醒的确切时间每个应用，KitKat 首选组注册的相同的时间间隔内唤醒并唤醒它们在同一时间的多个应用程序。若要告知 Android 在指定的时间间隔内唤醒应用，调用 `SetWindow` 上 `AlarmManager`、传入最小和最大时间（毫秒）之前唤醒应用程序，可以等待，以及要执行的操作在唤醒。下面的代码提供了需要唤醒半小时和一小时从窗口中设置的时间之间的应用程序的示例：

```
AlarmManager alarmManager = (AlarmManager)GetSystemService(AlarmService);
alarmManager.SetWindow (AlarmType.Rtc, AlarmManager.IntervalHalfHour, AlarmManager.IntervalHour,
pendingIntent);
```

若要继续在准确的时间唤醒应用，请使用 `SetExact`、传入应用程序应唤醒的确切时间和要执行的操作：

```
alarmManager.SetExact (AlarmType.Rtc, AlarmManager.IntervalDay, pendingIntent);
```

KitKat 不再允许您设置确切的重复警报。使用的应用程序 `SetRepeating` 并且需要确切的警报，以便将现在需要手动触发每个警报。

外部存储

外部存储现在分为两种类型的唯一的应用程序，并由多个应用程序共享的数据存储。读取和写入外部存储的应用程序的特定位置需要任何特殊权限。现在与共享存储上的数据进行交互需要 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 权限。这种情况下可以造成两个类型进行分类：

- 如果在调用方法来获取文件或目录路径 `Context` -例如， `GetExternalFilesDir` 或 `GetExternalCacheDirs`
 - 您的应用程序需要任何额外权限。
- 如果可将文件或目录路径的访问属性或调用方法上 `Environment`，如 `GetExternalStorageDirectory` 或 `GetExternalStoragePublicDirectory` 你的应用需要 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 权限。

NOTE

`WRITE_EXTERNAL_STORAGE` 意味着 `READ_EXTERNAL_STORAGE` 权限，因此只需将一个权限设置。

SMS 合并

KitKat 简化了通过聚合由用户选择一个默认应用程序中的所有 SMS 内容的用户消息传送。开发人员负责使应用程序的默认消息传送应用程序，可选择并运行正常的代码中和生活中如果不选择该应用程序。有关过渡到 KitKat SMS 应用程序的详细信息，请参阅[前您 SMS 应用程序的准备工作](#) KitKat来自 Google 的指南。

WebView 应用程序

`WebView` KitKat 而陷入功能改进。最大的变化提高加载到内容的安全性 `WebView`。虽然大多数面向较旧的 API 版本的应用程序应按预期方式工作的测试应用程序使用的 `WebView` 强烈建议类。有关受影响的 `WebView` Api 的详细信息，请参阅 android[迁移到 WebView 中 Android 4.4](#)文档。

用户体验

KitKat 附带了几个新的 Api 来增强用户体验，包括新的过渡框架，用于处理属性动画和主题的半透明 UI 选项。下面介绍了这些更改。

转换 Framework

转换 framework 可以轻松地实现动画。KitKat 允许您执行只需一行代码，使用一个简单的属性的动画或自定义使用的转换场景。

简单的属性动画

新的 Android 转换库简化了代码隐藏属性动画。该框架使您能够使用最少的代码的简单动画。例如，下面的代码示例使用 `TransitionManager.BeginDelayedTransition` 若要进行动画处理显示和隐藏 `TextView`：

```

using Android.Transitions;

public class MainActivity : Activity
{
    LinearLayout linear;
    Button button;
    TextView text;

    protected override void onCreate (Bundle bundle)
    {
        base.onCreate (bundle);
        setContentView (Resource.Layout.Main);

        linear = FindViewById<LinearLayout> (Resource.Id.linearLayout);
        button = FindViewById<Button> (Resource.Id.button);
        text = FindViewById<TextView> (Resource.Id.textView);

        button.Click += (o, e) => {

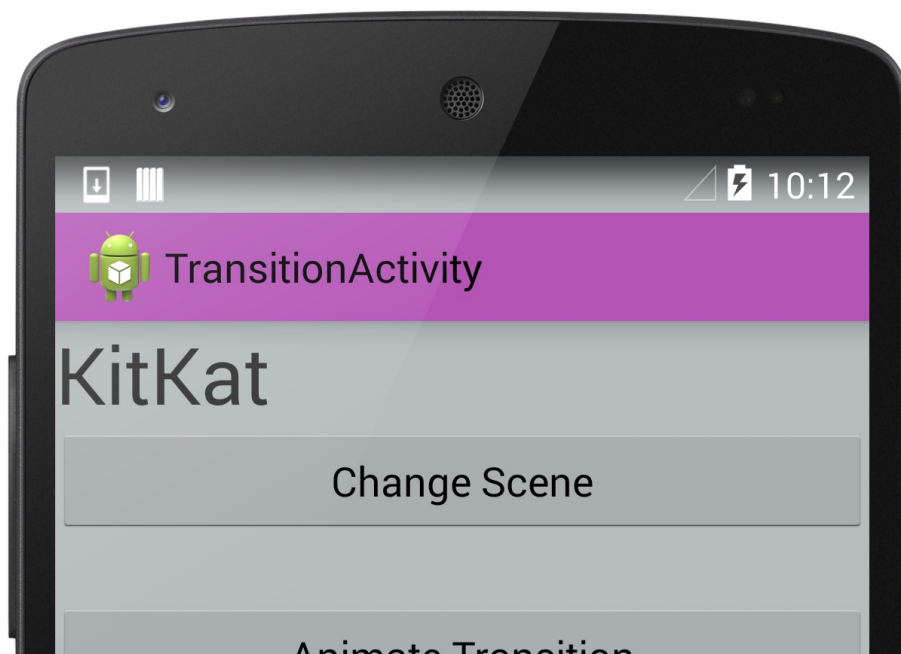
            TransitionManager.BeginDelayedTransition (linear);

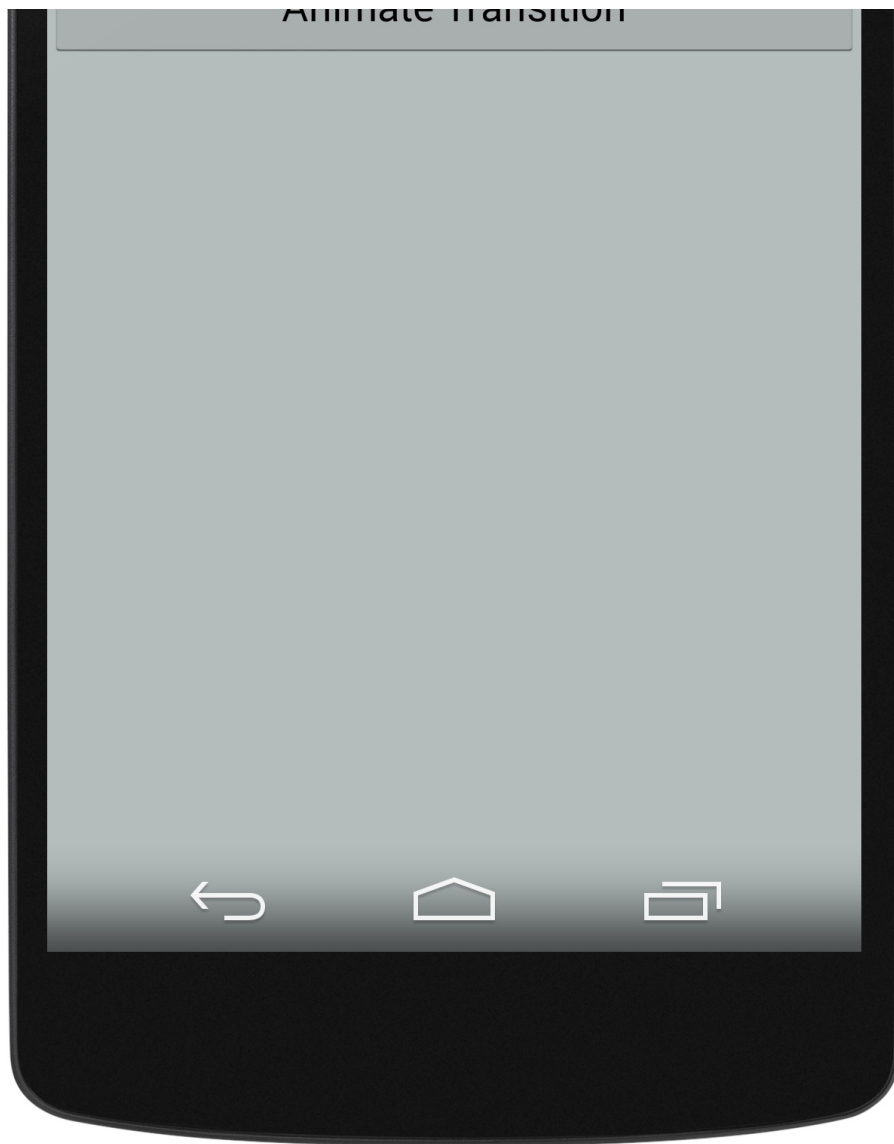
            if(text.Visibility != ViewStates.Visible)
            {
                text.Visibility = ViewStates.Visible;
            }
            else
            {
                text.Visibility = ViewStates.Invisible;
            }
        };
    }
}

```

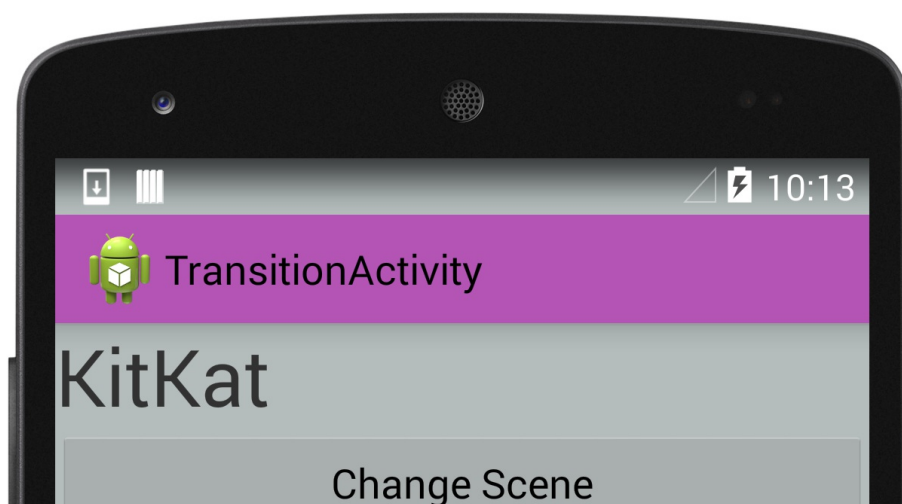
上面的示例中使用转换框架来创建自动、不断变化的属性值之间的默认转换。因为动画处理通过一行代码，就可以轻松地这与较旧版本的 Android 兼容通过包装 `BeginDelayedTransition` 调用中系统版本检查。请参阅[迁移你的应用程序到 KitKat](#)部分，了解详细信息。

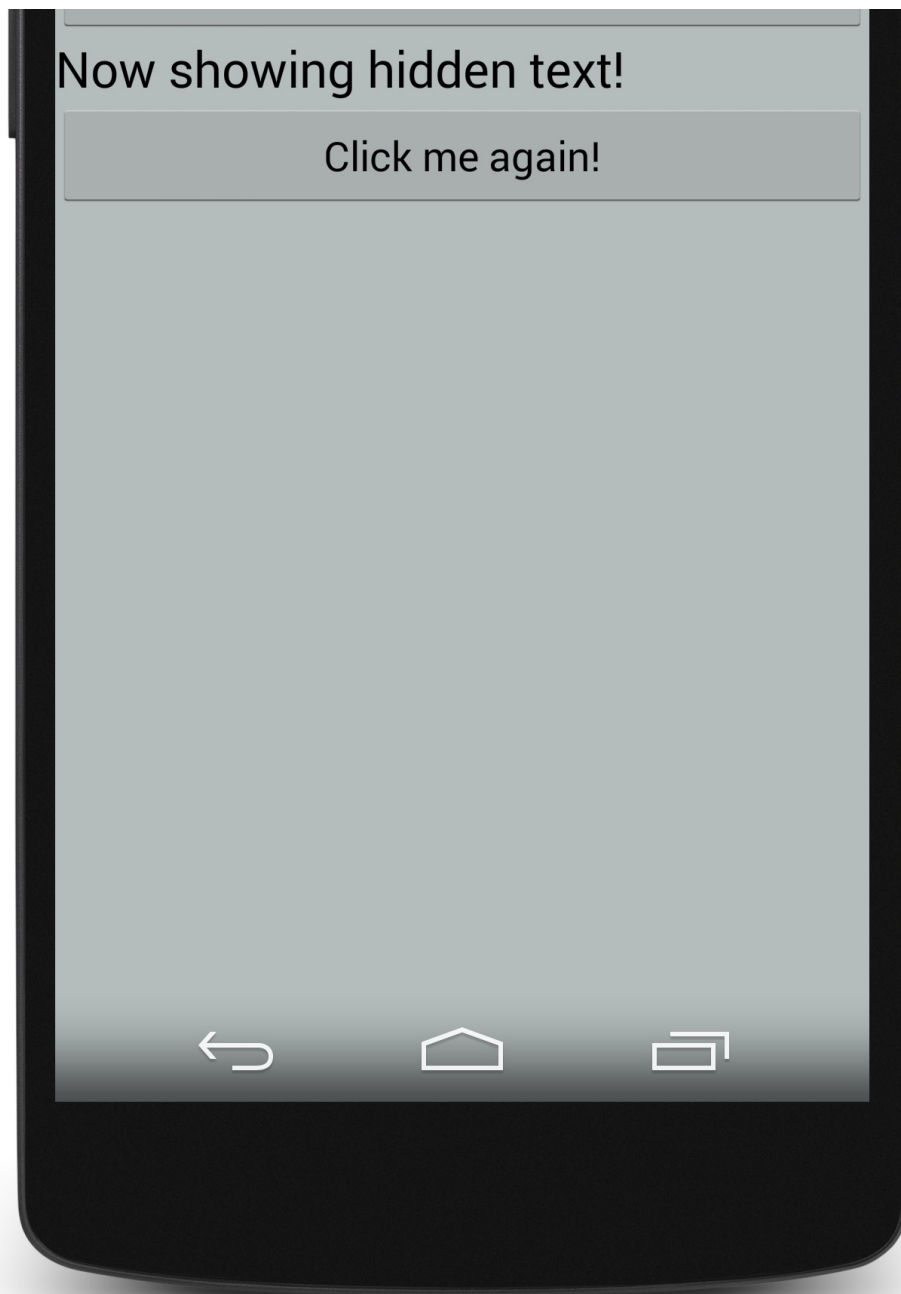
下面的屏幕截图显示了动画之前应用：





下面的屏幕截图显示了应用的动画结束后：





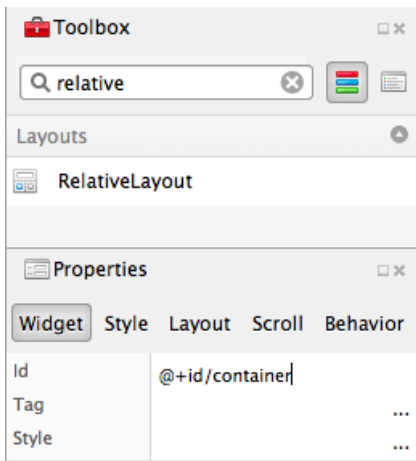
您可以获得更好地控制转换时将在下一节中介绍的场景。

Android 场景

[场景](#)作为转换 framework 为开发人员提供更好地控制动画的一部分引入。场景在 UI 中创建动态区域：指定容器和多个版本中或为 XML 内容的容器中，"场景"和 Android 余下的场景之间的转换进行动画处理的工作。Android 场景允许您开发端上生成具有最少的工作的复杂动画。

其中包含动态内容的静态用户界面元素称为 *容器*或*场景基*。下面的示例使用 Android 设计器来创建

`RelativeLayout` 调用 `container`：



示例布局还定义了名为的按钮 `sceneButton` 如下 `container` 。此按钮将触发转换。

在容器内的动态内容要求两个新的 Android 布局。这些布局指定只将代码内容。下面的代码示例定义了一种称为布局 `Scene1` 分别包含两个文本字段读取"工具包"和"Kat"和第二个布局称为 `Scene2` 包含反转相同的文本字段。XML 是按如下所示：

Scene1.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<merge xmlns:android="http://schemas.android.com/apk/res/android">
    <TextView
        android:id="@+id/textA"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Kit"
        android:textSize="35sp" />
    <TextView
        android:id="@+id/textB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/textA"
        android:text="Kat"
        android:textSize="35sp" />
</merge>
```

Scene2.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<merge xmlns:android="http://schemas.android.com/apk/res/android">
    <TextView
        android:id="@+id/textB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Kat"
        android:textSize="35sp" />
    <TextView
        android:id="@+id/textA"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/textB"
        android:text="Kit"
        android:textSize="35sp" />
</merge>
```

上面的示例使用 `merge` 以使视图代码更短和简化的视图层次结构。可以阅读更多有关 `merge` 布局 [此处](#)。

通过调用创建一个场景 `Scene.GetSceneForLayout` ，并在容器对象中，资源 ID 场景的布局文件，以及当前传递

Context，如下面的代码示例所示：

```
RelativeLayout container = findViewById<RelativeLayout> (Resource.Id.container);

Scene scene1 = Scene.GetSceneForLayout(container, Resource.Layout.Scene1, this);
Scene scene2 = Scene.GetSceneForLayout(container, Resource.Layout.Scene2, this);

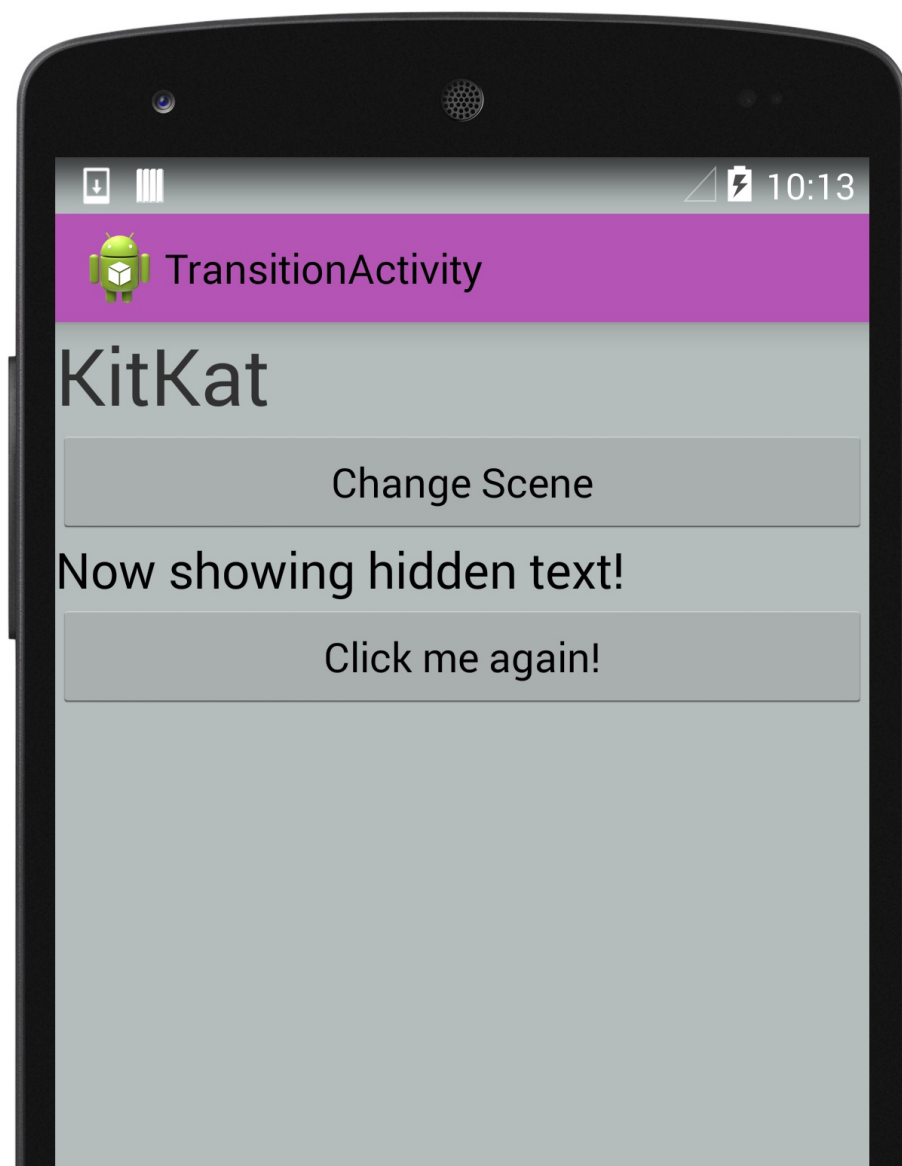
scene1.Enter();
```

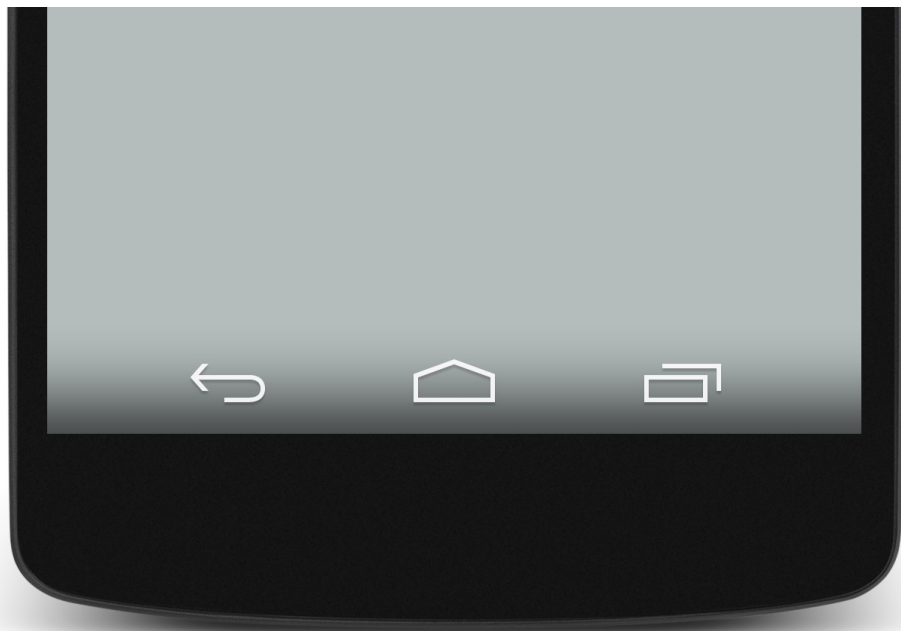
单击的按钮翻转 Android 进行动画处理的默认转换值在两个场景之间：

```
sceneButton.Click += (o, e) => {
    Scene temp = scene2;
    scene2 = scene1;
    scene1 = temp;

    TransitionManager.Go (scene1);
};
```

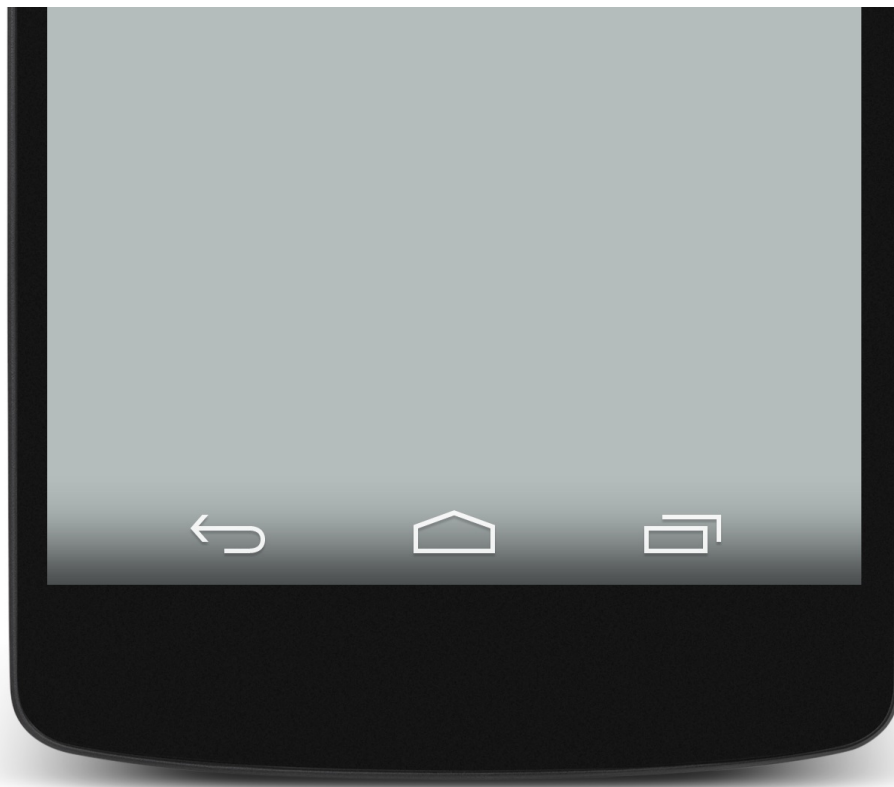
下面的屏幕截图演示了之前动画场景：





下面的屏幕截图演示了动画结束后的场景：



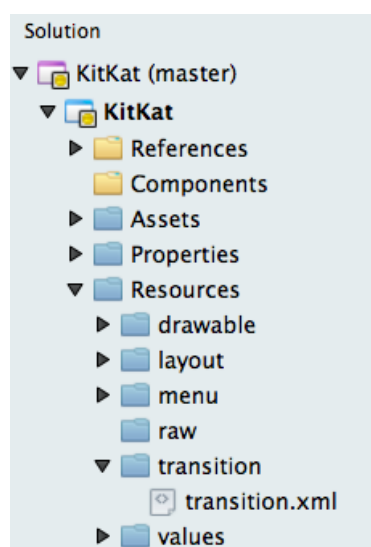


NOTE

没有已知的 [bug](#) 在 Android 转换会导致场景的库创建使用 `GetSceneForLayout` 用户完成某个动作的第二个时间导航时中断。
Java 解决方法描述 [此处](#)。

场景中的自定义转换

可以在 xml 资源文件中定义的自定义转换 `transition` 目录下 `Resources`，如下面的屏幕截图所示：



下面的代码示例定义一个转换的 5 秒之间进行动画处理，并使用 [超过了想要内插器](#)：

```
<changeBounds
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="5000"
    android:interpolator="@android:anim/overshoot_interpolator" />
```

在活动中创建转换使用 [TransitionInflater](#), 如下面的代码所示:

```
Transition transition = TransitionInflater.From(this).InflateTransition(Resource.Transition.transition);
```

新的转换, 然后添加到 `Go` 开始动画的调用:

```
TransitionManager.Go (scene1, transition);
```

半透明 UI

KitKat 提供更好地控制主题通过使用可选的半透明状态和导航栏在应用程序。您可以更改用于定义 Android 主题的同个 XML 文件中的系统 UI 元素的透明度。KitKat 引入了以下属性:

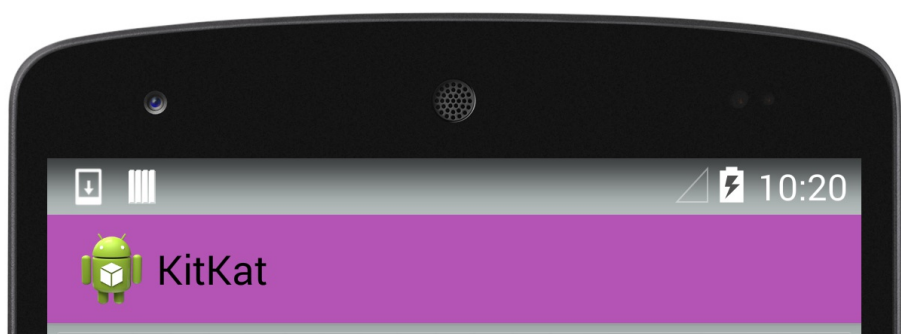
- `windowTranslucentStatus` - 设置为 true, 使顶部状态栏半透明。
- `windowTranslucentNavigation` - 设置为 true, 可在底部导航栏半透明。
- `fitsSystemWindows` - 设置为 translucent 顶部或底部栏默认情况下移动下的透明的 UI 元素的内容。此属性设置为 true 是使内容在重叠的半透明系统 UI 元素的简单方法。

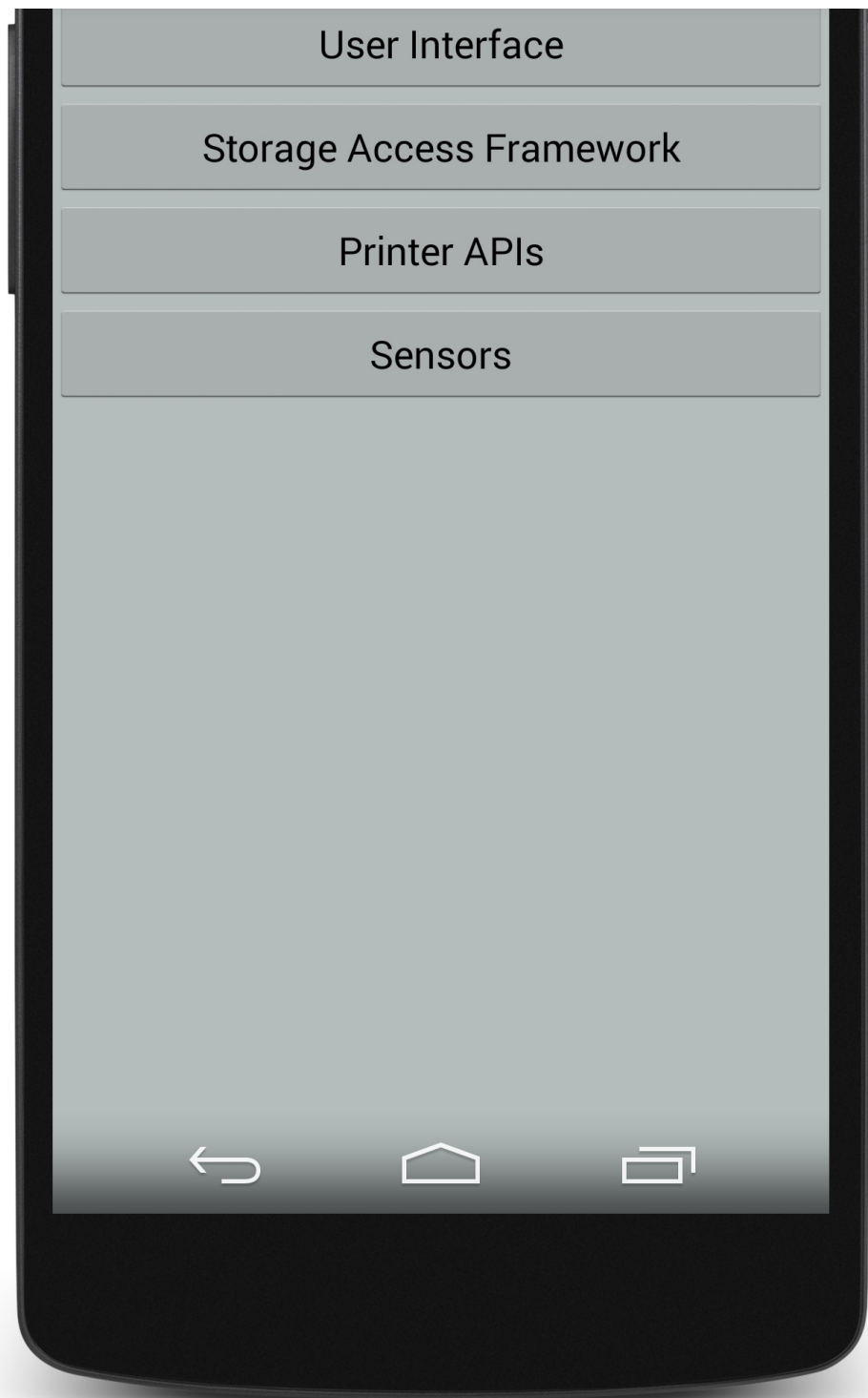
下面的代码定义了与半透明状态和导航条主题:

```
<?xml version="1.0" encoding="UTF-8" ?>
<resources>
    <style name="KitKatTheme" parent="android:Theme.Holo.Light">
        <item name="android:windowBackground">@color/xamgray</item>
        <item name="android:windowTranslucentStatus">true</item>
        <item name="android:windowTranslucentNavigation">true</item>
        <item name="android:fitsSystemWindows">true</item>
        <item name="android:actionBarStyle">@style/ActionBar.Solid.KitKat</item>
    </style>

    <style name="ActionBar.Solid.KitKat" parent="@android:style/Widget.Holo.Light.ActionBar.Solid">
        <item name="android:background">@color/xampurple</item>
    </style>
</resources>
```

下面的屏幕截图显示了与半透明状态和导航栏上方主题:





用户的内容

存储访问框架

存储访问框架 (SAF) 是用户与存储的内容, 例如图像、视频和文档交互的新方法。而不是向用户显示一个对话框, 以选择一个应用程序来处理内容, KitKat 打开新的用户界面, 使用户能够访问其数据存储在一个聚合的位置。选择内容后用户将返回到请求内容的应用程序, 并将继续正常进行的应用体验。

此更改需要开发人员端上的两个操作: 首先, 需要从提供程序的内容的应用程序需要更新为新请求内容的方式。第二个, 将数据写入到的应用程序 `ContentProvider` 需要进行修改以使用新的框架。这两种方案都依赖于新 `DocumentsProvider` API。

DocumentsProvider

在 KitKat, 与交互 `ContentProviders` 抽象化与 `DocumentsProvider` 类。这意味着, SAF 并不关心数据的实际, 只要它是可通过访问 `DocumentsProvider` API。本地提供程序, 云服务和外部存储设备所有使用该接口, 并将被视为相同的方式, 用户和开发人员提供一个位置与用户的内容进行交互。

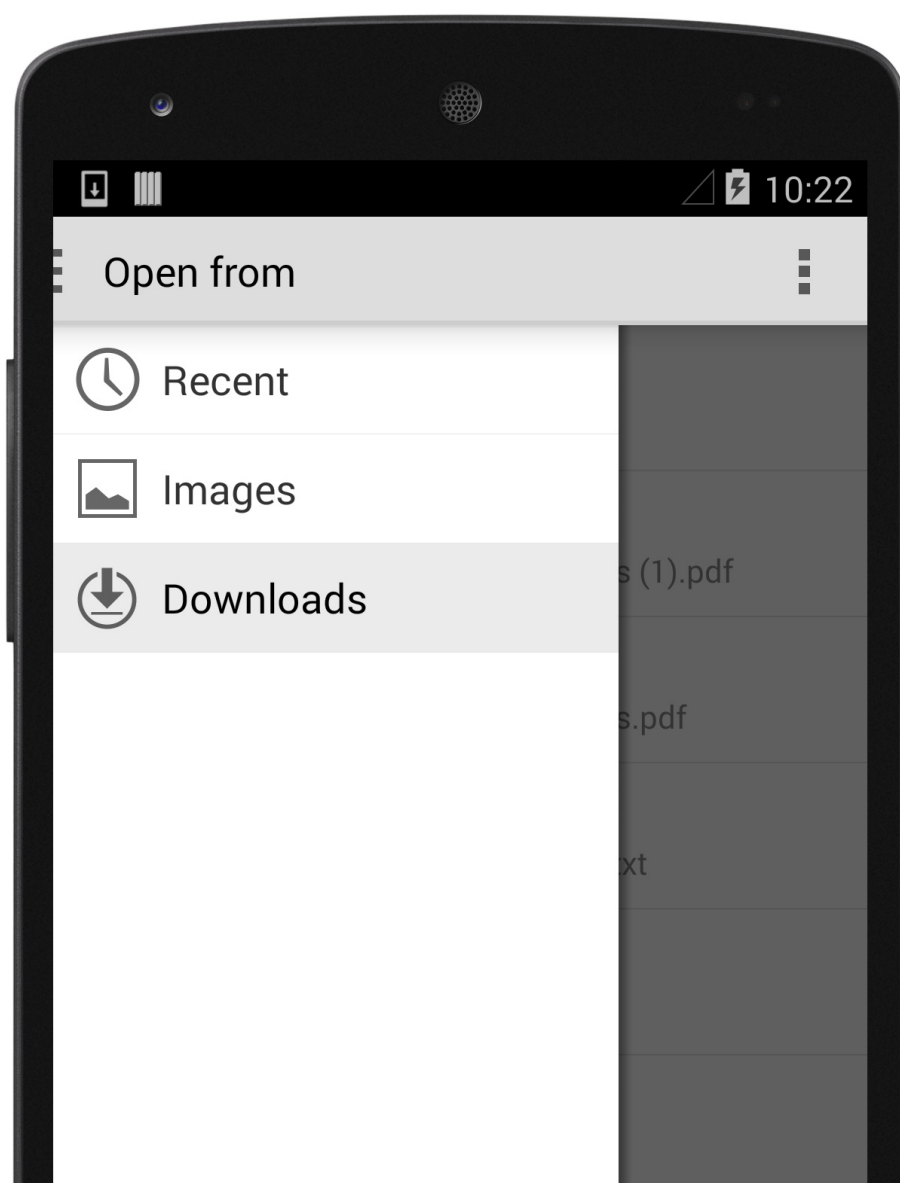
本部分介绍如何加载和保存与存储访问框架的内容。

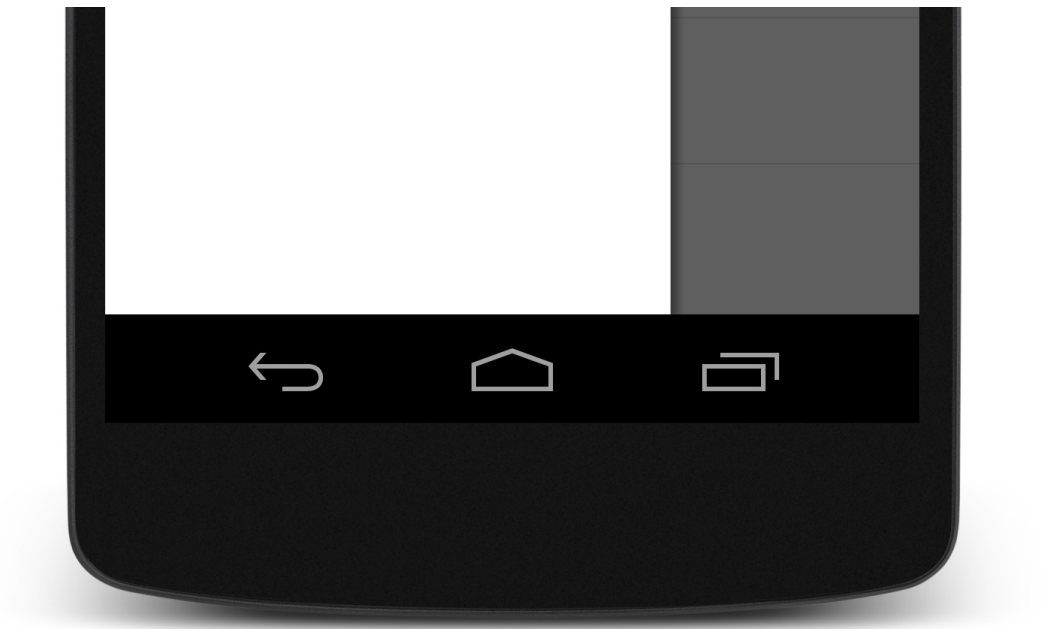
从提供程序的请求内容

我们可以告诉 KitKat 我们想要选择使用 SAF UI 内容 `ActionOpenDocument` 意向, 表示我们想要连接到与该设备所有内容提供商。您可以添加一些筛选到此目的通过指定 `CategoryOpenable`, 将返回这意味着可以打开 (即可访问性, 可使用内容) 的唯一内容。KitKat 还允许筛选的内容与 `MimeType`。例如, 以下由指定的图像的图像结果的筛选器代码 `MimeType`:

```
Intent intent = new Intent (Intent.ActionOpenDocument);
intent.AddCategory (Intent.CategoryOpenable);
intent.SetType ("image/*");
StartActivityForResult (intent, save_request_code);
```

调用 `StartActivityForResult` 启动 SAF UI, 用户可以浏览以选择一个映像:





用户已选择一个映像之后, `OnActivityResult` 返回 `Android.Net.Uri` 的所选的文件。下面的代码示例显示用户的图像选择:

```
protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
{
    base.OnActivityResult(requestCode, resultCode, data);

    if (resultCode == Result.Ok && data != null && requestCode == save_request_code) {
        imageView = FindViewById<ImageView> (Resource.Id.imageView);
        imageView.SetImageURI (data.Data);
    }
}
```

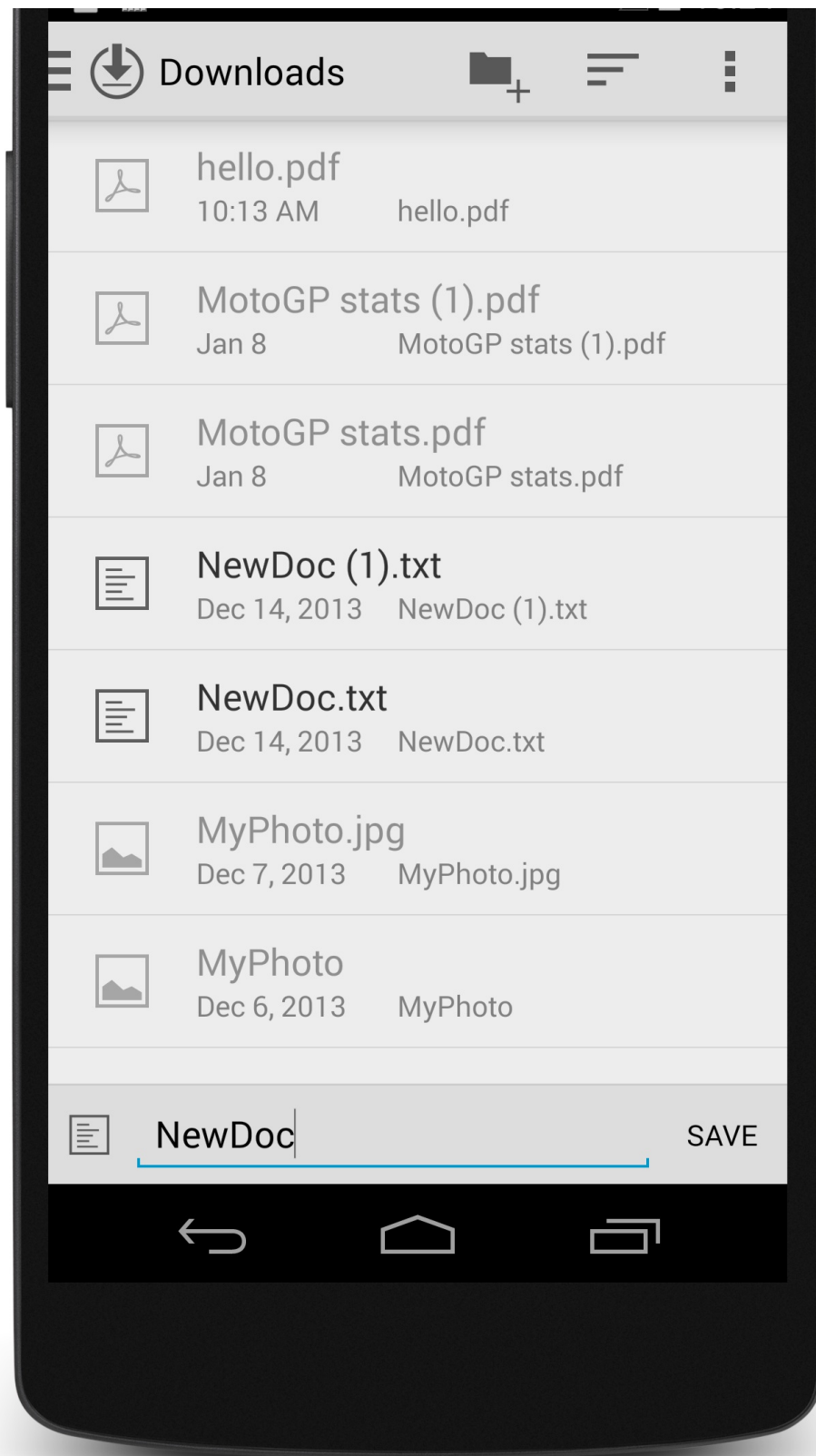
将内容写到提供程序

除了从 SAF UI 加载内容, KitKat 还允许您将内容保存到任何 `ContentProvider` 实现 `DocumentProvider` API。正在保存内容使用 `Intent` 与 `ActionCreateDocument` :

```
Intent intentCreate = new Intent (Intent.ActionCreateDocument);
intentCreate.AddCategory (Intent.CategoryOpenable);
intentCreate.SetType ("text/plain");
intentCreate.PutExtra (Intent.ExtraTitle, "NewDoc");
StartActivityForResult (intentCreate, write_request_code);
```

上面的代码示例加载 SAF UI, 让用户更改文件名称, 然后选择一个目录来容纳新文件:





当用户按**保存**，`OnActivityResult` 传递 `Android.Net.Uri` 的新创建的文件，可以通过访问 `data.Data`。Uri 可以用于数据流式传输到新的文件：

```
protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
{
    base.OnActivityResult(requestCode, resultCode, data);

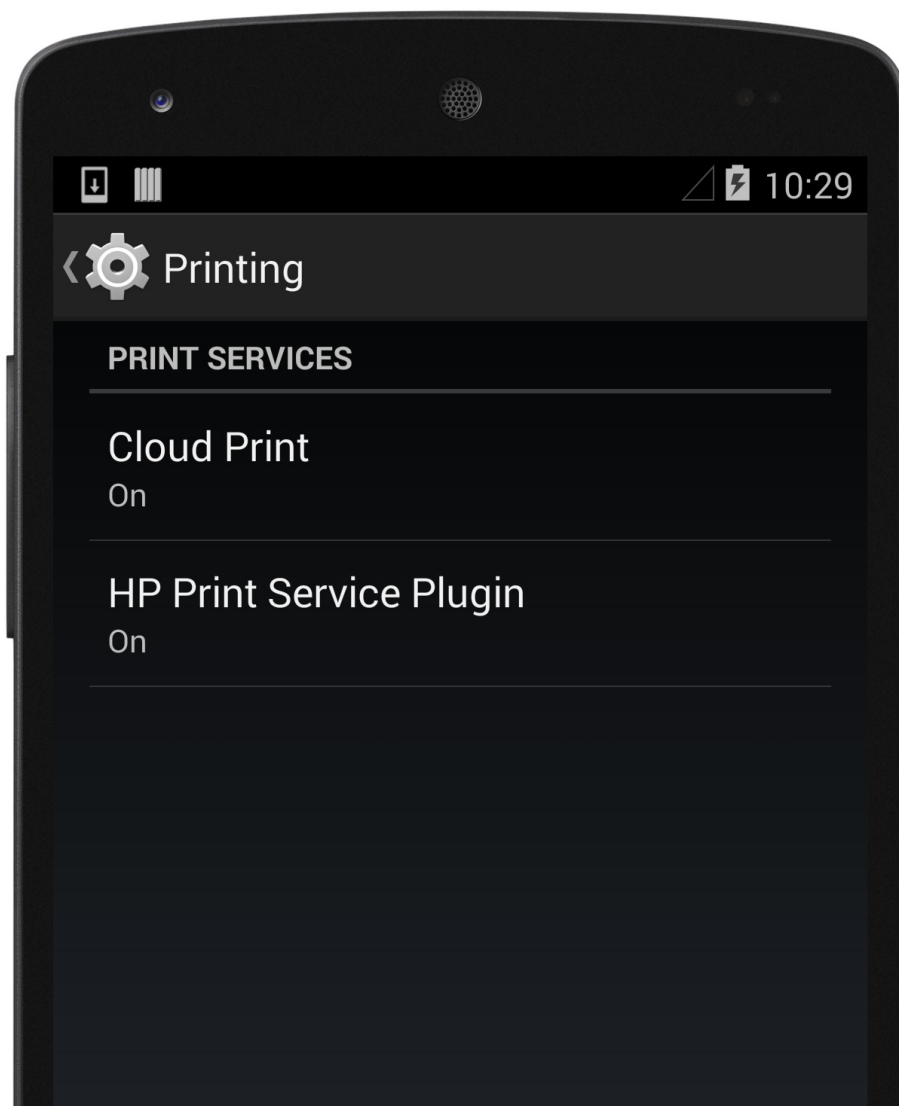
    if (resultCode == Result.Ok && data != null && requestCode == write_request_code) {
        using (Stream stream = ContentResolver.OpenOutputStream(data.Data)) {
            Encoding u8 = Encoding.UTF8;
            string content = "Hello, world!";
            stream.Write (u8.GetBytes(content), 0, content.Length);
        }
    }
}
```

请注意, `ContentResolver.OpenOutputStream(Android.Net.Uri)` 返回 `System.IO.Stream`, 因此可以用.NET 编写整个流式处理过程。

有关加载的详细信息, 创建和编辑内容与存储访问框架, 请参阅[Android 文档存储访问框架](#)。

打印

打印内容通过引入简化 KitKat[打印服务](#)和 `PrintManager`。KitKat 也是能够充分利用的第一个 API 版本[Google 云打印服务 Api](#)使用[Google 云打印应用程序](#)。大多数设备会自动随 KitKat 下载 Google 云打印应用程序和[HP 打印服务插件](#)当他们首次连接到 WiFi。用户可以通过导航到检查其自己的设备的打印设置设置 > 系统 > 打印:





NOTE

虽然打印 Api 设置为使用 Google 云打印默认情况下, Android 仍允许开发人员准备打印内容使用新的 Api, 并将其发送到其他应用程序可以处理打印。

打印 HTML 内容

会自动创建 KitKat `PrintDocumentAdapter` 的 web 视图与 `WebView.CreatePrintDocumentAdapter`。打印 web 内容是之间协调的工作量 `WebViewClient`, 等待要加载的 HTML 内容, 可让活动知道选项菜单中提供打印选项和 Activity, 等待到用户选择打印选项并调用 `Print` 上 `PrintManager`。本部分介绍了屏幕上打印所需的基本设置 HTML 内容。

请注意, 加载和打印的 web 内容需要 Internet 权限:

| Required permissions | |
|----------------------|--|
| | <input checked="" type="checkbox"/> Internet |
| | <input type="checkbox"/> KillBackgroundProcesses |
| | <input type="checkbox"/> LocationHardware |
| | <input type="checkbox"/> ManageAccounts |

打印菜单项

打印选项通常会在中显示的活动选项菜单。选项菜单可让用户在活动上执行操作。它是在屏幕的右上角, 如下所示:

tivity



Print

可以在中定义的附加菜单项 **菜单** 目录下 **资源**。下面的代码定义了一个示例菜单项调用 **打印**:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_print"
        android:title="Print"
        android:showAsAction="never" />
</menu>
```

与在活动中的选项菜单的交互是通过 `OnCreateOptionsMenu` 和 `OnOptionsItemSelected` 方法。`OnCreateOptionsMenu` 可以添加新菜单项, 如打印选项, 从 **菜单resources** 目录中。`OnOptionsItemSelected` 侦听用户从菜单中, 选择打印选项, 并开始打印:

```
bool dataLoaded;

public override bool OnCreateOptionsMenu (IMenu menu)
{
    base.OnCreateOptionsMenu (menu);
    if (dataLoaded) {
        MenuInflater.Inflate (Resource.Menu.print, menu);
    }
    return true;
}

public override bool OnOptionsItemSelected (IMenuItem item)
{
    if (item.ItemId == Resource.Id.menu_print) {
        PrintPage ();
        return true;
    }
    return base.OnOptionsItemSelected (item);
}
```

上面的代码还定义一个变量, 名为 `dataLoaded` 来跟踪的 HTML 内容的状态。`WebViewClient` 将设置此变量为 `true` 时的所有内容已都加载, 让活动知道要将打印菜单项添加到选项菜单。

WebViewClient

作业 `WebViewClient` 是为了确保中的数据 `WebView` 是完全加载之前的打印选项出现在菜单中, 这一点与 `OnPageFinished` 方法。`OnPageFinished` 侦听的 web 内容, 以完成加载, 并告知要重新创建具有其选项菜单的活动 `InvalidateOptionsMenu`:

```

class MyWebViewClient : WebViewClient
{
    PrintHtmlActivity caller;

    public MyWebViewClient (PrintHtmlActivity caller)
    {
        this.caller = caller;
    }

    public override void OnPageFinished (WebView view, string url)
    {
        caller.dataLoaded = true;
        caller.InvalidateOptionsMenu ();
    }
}

```

`OnPageFinished` 此外设置 `dataLoaded` 值设为 `true`，因此 `OnCreateOptionsMenu` 可以重新创建使用位置中的打印选项的菜单。

`PrintManager`

下面的代码示例打印的内容 `WebView`：

```

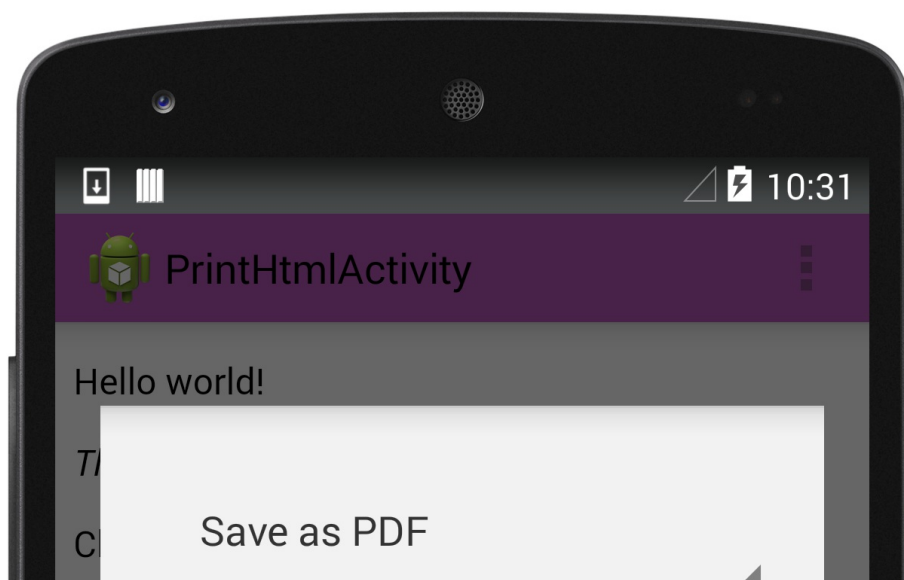
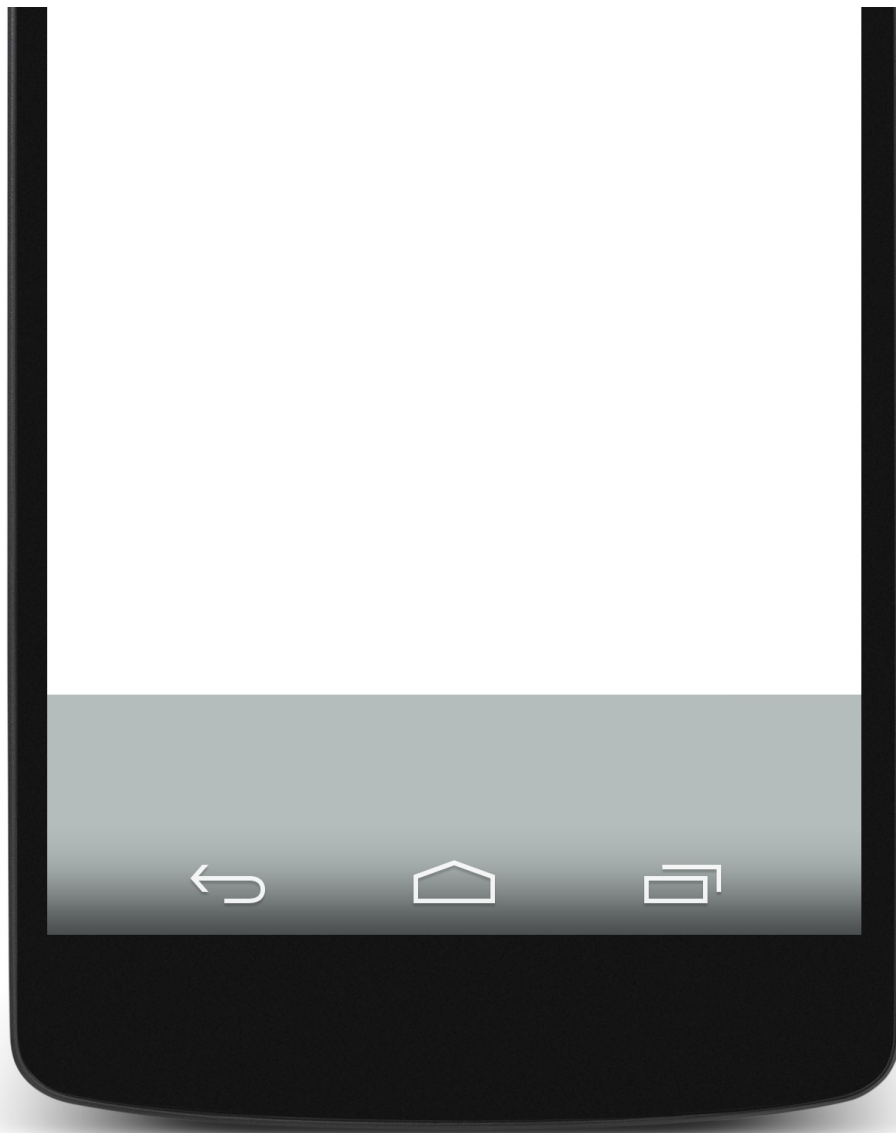
void PrintPage ()
{
    PrintManager printManager = (PrintManager)GetSystemService (Context.PRINT_SERVICE);
    PrintDocumentAdapter printDocumentAdapter = myWebView.CreatePrintDocumentAdapter ();
    printManager.Print ("MyWebPage", printDocumentAdapter, null);
}

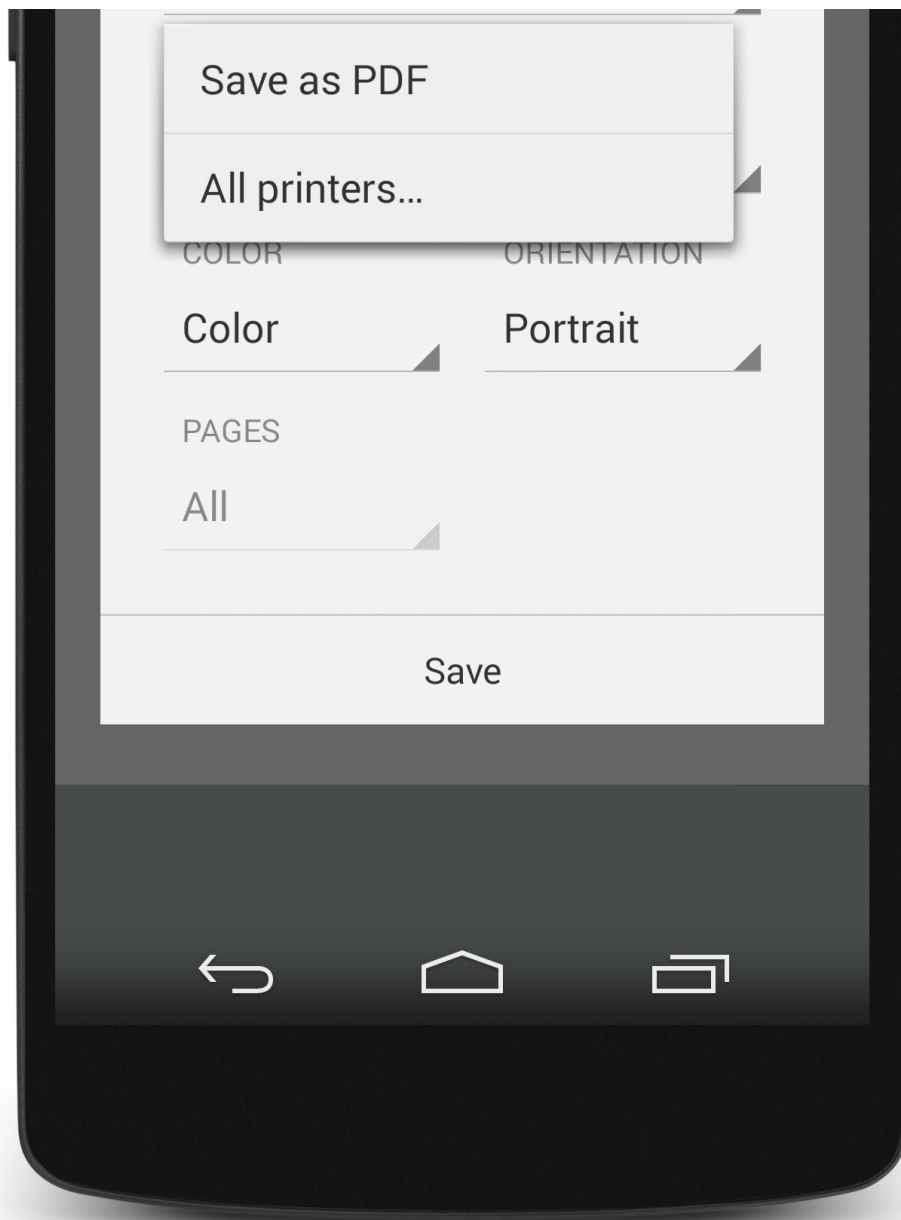
```

`Print` 将作为参数: ("MyWebPage"在此示例中)，打印作业的名称 `PrintDocumentAdapter` 它生成打印文档的内容和 `PrintAttributes` (`null` 在上面的示例)。您可以指定 `PrintAttributes` 以帮助打印页面上的内容的布局，但默认属性应处理大多数情况。

调用 `Print` 加载打印 UI，其中列出了打印作业的选项。用户界面，用户可以选择打印或 HTML 内容保存为 pdf 格式，如下面的屏幕截图所示：







硬件

KitKat 将添加几个 Api, 以适应新的设备功能。最值得注意的是基于主机的卡仿真和新 `SensorManager` 。

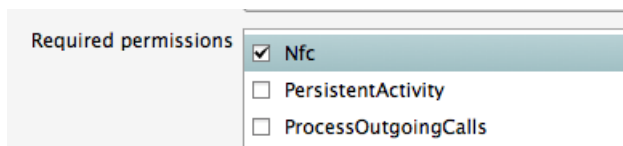
基于主机的卡 NFC 模拟机制

基于主机的卡仿真 (HCE) 允许应用程序而不依赖于运营商的专有安全元素的行为类似于 NFC 卡或 NFC 卡读卡器。设置 HCE 之前, 确保 HCE 可在设备上使用 `PackageManager.HasSystemFeature` :

```
bool hceSupport = PackageManager.HasSystemFeature(PackageManager.FeatureNfcHostCardEmulation);
```

HCE 需要这两个 HCE 功能和 `Nfc` 应用程序的注册权限 `AndroidManifest.xml` :

```
<uses-feature android:name="android.hardware.nfc.hce" />
```



若要处理，HCE 具有能够在后台运行并具有启动时用户所做的 NFC 事务，即使使用 HCE 的应用程序未运行。我们可以完成此操作通过编写 HCE 代码作为 `Service`。HCE 服务实现 `HostApuService` 接口，实现以下方法：

- `ProcessCommandApu` - 应用程序协议数据单元 (APDU) 是什么获取 NFC 读取器和 HCE 服务之间发送。此方法使用 ADPU 从读取器，并在响应中返回数据单元。
- `OnDeactivated` - `HostApuService` HCE 服务无法再使用 NFC 读取器通信时停用。

HCE 服务还需要注册应用程序清单，并使用适当的权限，意向筛选器和元数据修饰。下面的代码是一种 `HostApuService` 注册的 Android 清单使用 `Service` 属性 (有关属性的详细信息，请参阅 Xamarin[使用 Android 清单指南](#)):

```
[Service(Exported=true, Permission="android.permissions.BIND_NFC_SERVICE"),
    IntentFilter(new[] { "android.nfc.cardemulation.HOST_APDU_SERVICE" }),
    Metadata("android.nfc.cardemulation.host.apdu_service",
        Resource="@xml/hceservice")]

class HceService : HostApuService
{
    public override byte[] ProcessCommandApu(byte[] apdu, Bundle extras)
    {
        ...
    }

    public override void OnDeactivated (DeactivationReason reason)
    {
        ...
    }
}
```

上述服务，可以为要与该应用程序进行交互的 NFC 读取器但 NFC 读取器仍有没有办法知道如果此服务模拟它需要扫描的 NFC 卡。为了帮助标识该服务的 NFC 读取器，我们可以将分配该服务的唯一 *应用程序 ID (AID)*。我们指定的辅助手段，以及有关 HCE 服务的其他元数据的 xml 资源文件中注册 `Metadata` 属性 (请参阅上面的代码示例)。此资源文件指定一个或多个辅助设备筛选器-以十六进制格式的唯一标识符字符串对应于一个或多个 NFC 读取器设备的辅助工具：

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/hce_service_description"
    android:requireDeviceUnlock="false"
    android:apduServiceBanner="@drawable/service_banner">
    <aid-group android:description="@string/aid_group_description"
        android:category="payment">
        <aid-filter android:name="1111111111111111"/>
        <aid-filter android:name="0123456789012345"/>
    </aid-group>
</host-apdu-service>
```

除了帮助筛选器，xml 资源文件还提供 HCE 服务的面向用户的说明指定的辅助设备组（而不是“其他”付款应用程序）和对于付款应用程序，以向用户显示一个 260 x 96 dp 横幅。

上面所述安装程序提供了模拟的 NFC 卡的应用程序的基本构建基块。NFC 本身需要几个详细步骤和进一步测试配置。有关基于主机的卡仿真的详细信息，请参阅[Android 文档门户](#)。有关结合使用 NFC 和 Xamarin 的详细信息，请参阅[Xamarin NFC 示例](#)。

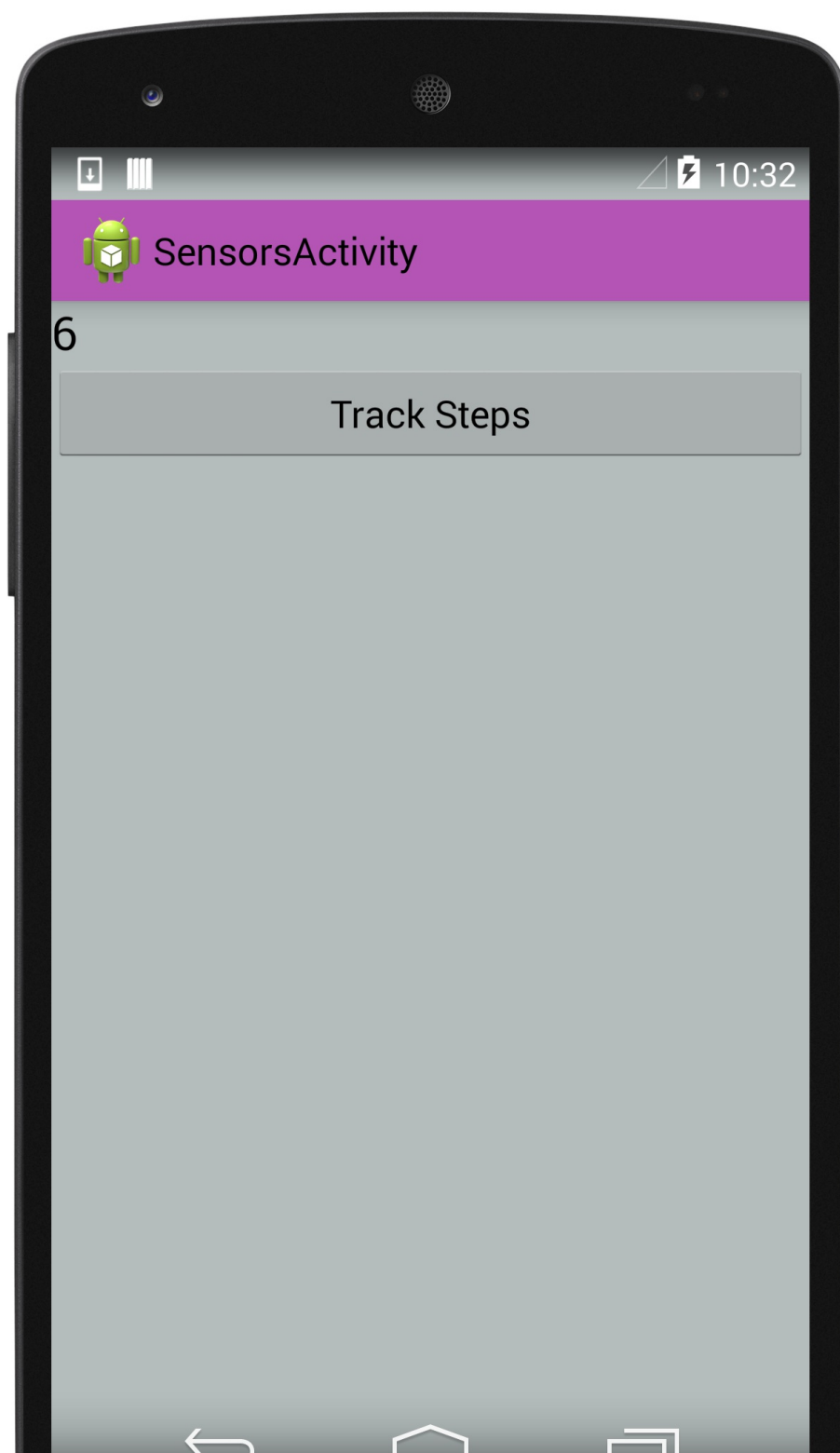
传感器

KitKat 人员能够通过设备的传感器 `SensorManager` 。 `SensorManager` ，操作系统就可以计划传递传感器信息再到批处理中的应用程序保留的电池使用寿命。

KitKat 还附带了两个新的传感器类型用于跟踪用户的步骤。这些基于加速感应器，包括：

- *StepDetector* -应用程序通知/唤醒时用户采取的步骤，并检测程序步骤发生时提供一个时间值。
- *StepCounter* -跟踪的传感器已注册用户后执行的步骤数直到下一步的设备重启。

下面的屏幕截图描绘了操作中的步计数器：





您可以创建 `SensorManager` 通过调用 `GetSystemService(SensorService)` 并将结果作为转换 `SensorManager`。若要使用的步骤计数器, 请调用 `GetDefaultSensor` 上 `SensorManager`。你可以注册传感器并侦听步骤计数的帮助中的更改 `ISensorEventListener` 接口, 如下面的代码示例所示:

```
public class MainActivity : Activity, ISensorEventListener
{
    float count = 0;

    protected override void onCreate (Bundle bundle)
    {
        base.onCreate (bundle);
        setContentView (Resource.Layout.Main);

        SensorManager senMgr = (SensorManager) GetSystemService (SensorService);
        Sensor counter = senMgr.getDefaultSensor (SensorType.StepCounter);
        if (counter != null) {
            senMgr.registerListener(this, counter, SensorDelay.Normal);
        }
    }

    public void OnAccuracyChanged (Sensor sensor, SensorStatus accuracy)
    {
        Log.info ("SensorManager", "Sensor accuracy changed");
    }

    public void OnSensorChanged (SensorEvent e)
    {
        count = e.Values [0];
    }
}
```

`OnSensorChanged` 如果步骤计数更新该应用程序在前台时进行调用。如果应用程序进入后台, 或在设备处于睡眠状态时, `OnSensorChanged` 不会调用; 但是, 步骤将继续进行计数直到 `UnregisterListener` 调用。

请记住在注册传感器的所有应用程序中, 步骤计数值是累计的。这意味着, 即使在卸载和重新安装应用程序, 并初始化 `count` 变量 0 处开始应用程序启动时, 传感器报告的值将保持的步骤时注册传感器, 使总数是否由你应用程序或另一个。可以阻止应用程序将添加到步骤计数器通过调用 `UnregisterListener` 上 `SensorManager`, 如下面的代码所示:

```
protected override void onPause()
{
    base.onPause ();
    senMgr.unregisterListener(this);
}
```

重启设备步骤计数重置为 0。您的应用程序将需要额外的代码, 以确保它报告的应用程序, 而不考虑其他应用程序使用传感器或设备的状态的准确计数。

NOTE

步骤检测和计数与 KitKat 一起提供的 API, 而不是所有手机都配备传感器。您可以检查传感器是否可通过运行

```
PackageManager.HasSystemFeature(PackageManager.FeatureSensorStepCounter);
```

, 或检查, 以确保返回的值 `getDefaultSensor` 不是 `null`。

开发人员工具

屏幕录制

KitKat 包括新屏幕, 以便开发人员可记录应用程序中操作录制功能。屏幕录制已可通过[Android Debug Bridge \(ADB\)](#)客户端, 可以下载 Android SDK 的一部分。

若要录制你的屏幕, 将设备连接, 然后, 找到你的 Android SDK 安装中, 导航到平台工具目录, 然后运行adb客户端:

```
adb shell screenrecord /sdcard/screencast.mp4
```

上述命令将记录 4Mbps 默认分辨率的默认值 3 分钟视频。若要编辑的长度, 请添加 `-时间限制` 标志。若要更改分辨率, 添加 `-比特率` 标志。以下命令将记录在 8Mbps 长时间的分钟视频:

```
adb shell screenrecord --bit-rate 8000000 --time-limit 60 /sdcard/screencast.mp4
```

可以在设备上查找你的视频-它会在您的库中录制完成后。

其他 KitKat 新增功能:

除了上面所述的更改, KitKat, 您可以:

- **使用全屏**-KitKat 引入了一个新**沉浸式模式下**浏览内容、玩游戏, 并运行其他应用程序可受益于全屏体验。
- **自定义通知**-获取有关使用系统通知的其他详细信息 `NotificationListenerService` . 这允许您在您的应用程序内的不同方式显示信息。
- **镜像可绘制资源**-可绘制资源有一个新 `autoMirrored` 通知系统的属性创建需要从左到右布局翻转的图像的镜像的版本。
- **暂停动画**-暂停和恢复使用创建的动画 `Animator` 类的新实例。
- **读取动态更改的文本**-表示动态使用新的文本更新为与新的"实时区域"的用户界面部分 `accessibilityLiveRegion` 因此在可访问性模式下会自动读取新的文本属性。
- **增强音频体验**-请噪音跟踪与 `LoudnessEnhancer` 找到的峰值数目和 RMS 的音频流 `Visualizer` 类, 并获取信息从**音频时间戳**以帮助进行音频视频同步。
- **在自定义间隔同步 ContentResolver** -KitKat 增加一些可变性执行同步请求的时间。同步 `ContentResolver` 在自定义时或通过调用间隔 `ContentResolver.RequestSync` 并传入 `SyncRequest`。
- **控制器之间区分**-在 KitKat 控制器分配可通过设备的访问的唯一整数标识符 `ControllerNumber` 属性。这使得更轻松地告诉相隔玩家在游戏中。
- **远程控制**-使用的硬件和软件端上的一些更改, KitKat 允许您打开到远程控制使用配备红外线 (ir) 发射器设备 `ConsumerIrService`, 并与新的外围设备与之交互 `RemoteController` Api。

有关上述 API 更改的详细信息, 请参阅 Google [Android 4.4 Api](#)概述。

总结

本文介绍了一些 Android 4.4 (API 级别 19) 中提供的新 Api, 并转换到 KitKat 的应用程序时涉及的最佳做法。It 体验空心的更改会影响用户的 Api, 包括 *过渡框架* 用于和新选项 *主题*。接下来, 它引入 *存储访问框架* 并 `DocumentsProvider` 类, 以及新 *打印 Api*。它探讨 *NFC 基于主机的卡仿真*, 以及如何使用 *低功率传感器*, 包括两个新的传感器, 用于跟踪用户的步骤。最后, 它演示了捕获的应用程序与实时演示 *屏幕录制*, 并提供 KitKat API 更改和新增功能的详细的列表。

相关链接

- [KitKat 示例](#)
- [Android 4.4 Api](#)
- [Android KitKat](#)

Jelly Bean 功能

2018/10/26 • [Edit Online](#)

本文档将 Android 4.1 中引入的开发人员提供的新功能的高级别概述。这些功能包括：增强的通知、Android 无线发送共享大型文件、多媒体、对等网络发现、动画、新的权限的更新的更新。

概述

Android 4.1 (API 级别 16), 也称为"Jelly Bean", 是在 2012 年 7 月 9 日的版本。本文将使用 Xamarin.Android 的开发人员提供一些 Android 4.1 中的新增功能的高级别介绍。某些引入这些新功能是为了启动活动、相机、新的声音并改进了对应用程序堆栈导航支持动画的增强功能。现可与意图剪切和粘贴。

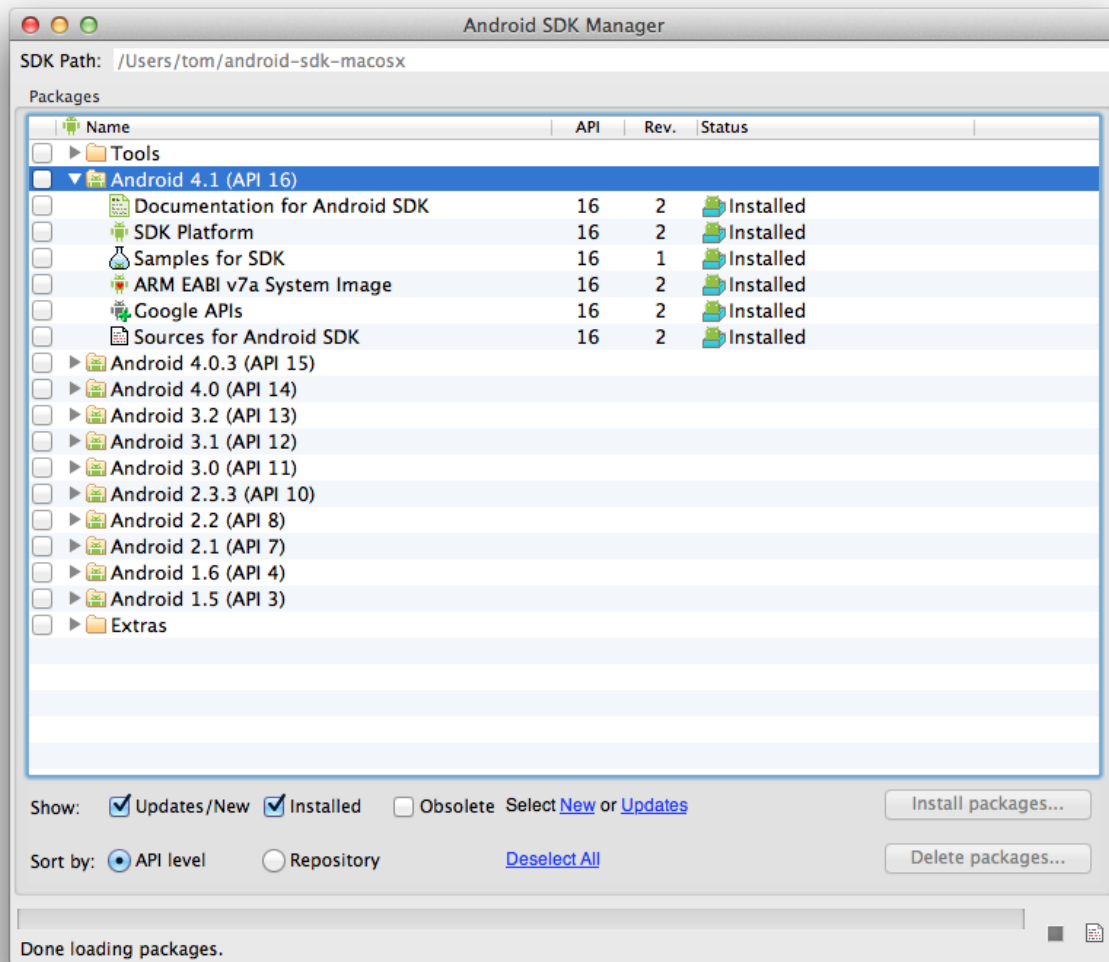
Android 应用程序的稳定性得到改进来隔离对不稳定的内容提供商的依赖关系的功能。服务也可能是独立, 以便它们仅可由访问启动它们的活动。

已添加了支持的网络服务发现使用 Bonjour、UPnP 或多播的 DNS 基于服务。就现在可以更丰富的文本、操作按钮和图像的大小已设置格式的通知。

最后几个新权限已添加 Android 4.1 中。

要求

若要开发 Xamarin.Android 应用程序使用 Jelly Bean 需要 Xamarin.Android 4.2.6-或更高版本以及 Android 4.1 (API 级别为 16) 安装通过 Android SDK 管理器, 如以下屏幕截图中所示:



新增功能

动画

可以通过使用缩放动画或自定义动画启动活动 `ActivityOptions` 类。提供以下新方法支持这些动画：

- `MakeScaleUpAnimation` - 这将创建可从起始位置和大小在屏幕上的活动窗口内扩展的动画。
- `MakeThumbnailScaleUpAnimation` - 这将从缩略图在屏幕上的指定位置中创建的动画，向上扩展。
- `MakeCustomAnimation` - 这将从应用程序中的资源创建动画。没有一个动画，以打开活动时，另一个用于活动停止时。

新 `TimeAnimator` 类提供了一个接口 `TimeAnimator.ITimeListener`，可以每次以动画帧发生更改时通知应用程序。例如，考虑以下实现 `TimeAnimator.ITimeListener`：

```
class MyTimeListener : Java.Lang.Object, TimeAnimator.ITimeListener
{
    public void OnTimeUpdate(TimeAnimator animation, long totalTime, long deltaTime)
    {
        Log.Debug("Activity1", "totalTime={0}, deltaTime={1}", totalTime, deltaTime);
    }
}
```

现在，若要使用的类的实例 `TimeAnimator` 创建，并设置侦听器：

```
var animator = new TimeAnimator();
animator.SetTimeListener(new MyTimeListener());
animator.Start();
```

作为 `TimeAnimator` 实例正在运行，它将调用 `ITimeAnimator.ITimeListener`，哪个然后将日志如何运行和多长时间它因为自上次以来已方法已调用动画器已被长时间。

应用程序堆栈导航

Android 4.1 改进了 Android 3.0 中引入了应用程序堆栈导航。通过指定 `ParentName` 属性 `ActivityAttribute`，当用户按下时，Android 可以打开正确的父活动 **向上按钮** 在操作栏中的 Android 将实例化指定的活动 `ParentName` 属性。这允许应用程序保留的活动，使某个给定的任务的层次结构。

对于大多数应用程序设置 `ParentName` 活动是足够的信息适用于 Android 的导航的应用程序堆栈; 提供正确的行为 Android 将通过创建一系列意向为每个父活动的合成必要 back 堆栈。但是，由于这是将自定义应用程序堆栈，综合的每个活动将不能将具有自然活动的已保存的状态。若要提供到综合父活动的已保存的状态，活动可能会重写 `OnPrepareNavigationUpTaskStack` 方法。此方法接收 `TaskStackBuilder` 将有一系列的意图的实例对象，将使用 Android 创建 back 堆栈。活动可能会修改这些意图，以便创建综合的活动时，它将接收正确的状态信息。

对于更复杂的方案，可用于处理导航的行为，并构造 back 堆栈 Activity 类上有新的方法：

- `OnNavigateUp` - 通过重写此方法就可以执行自定义操作时向上按下按钮。
- `NavigateUpTo` - 调用此方法将导致应用程序以从当前活动导航到给定目的指定的活动。
- `ParentActivityIntent` - 这用于获取意向，它将启动当前活动的父活动。
- `ShouldUpRecreateTask` - 此方法用于查询是否必须创建综合的 back 堆栈以导航到父活动。返回 `true` 如果必须创建综合的堆栈。
- `FinishAffinity` - 调用此方法将完成当前活动和所有它下面当前任务中的活动具有相同的任务关联。
- `OnCreateNavigateUpTaskStack` - 此方法被重写时需要具有完全控制如何创建综合的堆栈。

照相机

没有新界面，`Camera.IAutoFocusMoveCallback`，这可用于检测何时自动聚焦已开始或停止移动。以下代码段中，可以看到此新接口的示例：

```
public class AutoFocusCallbackActivity : Activity, Camera.IAutoFocusCallback
{
    public void OnAutoFocus(bool success, Camera camera)
    {
        // camera is an instance of the camera service object.

        if (success)
        {
            // Auto focus was successful - do something here.
        }
        else
        {
            // Auto focus didn't happen for some reason - react to that here.
        }
    }
}
```

新类 `MediaActionSound` 提供的 API 的一组用于生成适用于各种媒体操作的声音。有多个操作可能出现的照相机，这些定义通过枚举 `Android.Media.MediaActionSoundType`：

- `MediaActionSoundType.FocusComplete` - 此将重点放完成时，将会播放的声音。
- `MediaActionSoundType.ShutterClick` - 拍摄静止图像图片时，将播放这听起来。
- `MediaActionSoundType.StartVideoRecording` - 使用这听起来指示开始的视频录制。

- `MediaActionSoundType.StopVideoRecording` – 若要指示视频录制的末尾，将播放这听起来。

举例说明如何使用 `MediaActionSound` 类可以在以下代码片段中看到：

```
var mediaPlayer = new MediaActionSound();

// Preload the sound for a shutter click.
mediaActionPlayer.Load(MediaActionSoundType.ShutterClick);
var button = FindViewById<Button>(Resource.Id.MyButton);

// Play the sound on a button click.
button.Click += (sender, args) => mediaPlayer.Play(MediaActionSoundType.ShutterClick);

// This releases the preloaded resources. Don't make any calls on
// mediaPlayer after this.
mediaActionPlayer.Release();
```

连接

Android 无线发送

Android 无线发送是一种基于 NFC 技术，允许两个 Android 设备与彼此通信。Android 4.1 传输的大型文件提供更好的支持。使用新的方法时 `NfcAdapter.SetBeamPushUri()` Android 都将改用备用传输机制（例如蓝牙）之间实现快速传输速度。

网络服务发现

Android 4.1 包含新的 API 的多路广播基于 DNS 的服务发现。这允许应用程序以检测并通过 Wi-fi 连接到其他设备，如打印机、相机和媒体设备。这些新的 API 位于 `Android.Net.Nsd` 包。

若要创建的服务，可供其他服务，`NsdServiceInfo` 类用于创建一个对象，将定义服务的属性。然后将此对象提供给 `NsdManager.RegisterService()` 的实现以及 `NsdManager.ResolveListener`。实现 `NsdManager.ResolveListener` 用于通知的成功注册和撤消服务注册。

若要发现的网络和实现上的服务 `Nsd.DiscoveryListener` 传递给 `NsdManager.discoverServices()`。

网络使用情况

新的方法，`ConnectivityManager.IsActiveNetworkMetered`，允许检查它是否连接到按流量计费的网络设备。此方法可用于帮助管理数据使用情况的准确地通知用户可能昂贵的数据操作的费用。

WiFi 直接服务发现

`WifiP2pManager` 类以支持 Android 4.0 中引入 *zeroconf*。Zeroconf（零个网络配置）是一组技术，允许设备（计算机、打印机、手机）以自动连接到网络的人工网络运营商或特殊的配置服务器干预。

在 Jelly Bean `WifiP2pManager` 可以发现附近的设备使用 *Bonjour* 或 *Upnp*。Bonjour 是 zeroconf Apple 的实现。Upnp 还支持 zeroconf 的网络协议的设置。以下方法添加到 `WifiP2pManager` 以支持 Wi-fi 服务发现：

- `AddLocalService()` – 使用此方法发现的对方通过 Wi-fi 宣布作为服务的应用程序。
- `AddServiceRequest()` – 此方法是将服务发现请求发送到框架。它用于初始化的 Wi-fi 服务发现。
- `SetDnsSdResponseListeners()` – 此方法用于注册要调用的接收从 Bonjour 发现请求的响应的回调。
- `SetUpnpServiceResponseListener()` – 此方法用于注册要调用的接收的响应发现请求 Upnp 的回调。

内容提供商

`ContentResolver` 类已收到新的方法，`AcquireUnstableContentProvider`。此方法允许应用程序获取“不稳定”的内容提供商。通常情况下，如果应用程序获取内容提供商，而该内容提供程序发生崩溃，因此将应用程序。使用此方法调用，如果发生故障，内容提供商，将不崩溃应用程序。相反，`Android.OS.DeathObjectException` 将引发对内容提供程序的调用以通知的应用程序内容提供商具有已得到解决。与其他应用程序中的内容提供商进行交互时，“不稳定”的内容提供商非常有用 – 它不太可能从其他应用程序的错误代码，将会影响另一个应用程序。

复制和粘贴到意向

`Intent` 类现在可以 `ClipData` 通过与其关联的对象 `Intent.ClipData` 属性。此方法允许为额外的数据，从剪贴板将待传输的意图。实例 `ClipData` 可以包含一个或多个 `ClipData.Item`。`ClipData.Item` 以下类型的项：

- **文本**– 这是任何文本，任一 HTML 字符串或其格式支持内置的 Android 样式的任何字符串跨越。
- **意向**– 任何 `Intent` 对象。
- **Uri**– 这可以是任何 URI，例如 HTTP 书签或内容提供程序的 URI。

隔离的服务

独立的服务是一种服务，在其自己的特殊进程下运行和不具有其自己的任何权限。与该服务的唯一通信是何时启动该服务并将绑定到它通过服务 API。可以通过设置的属性声明为独立的服务 `IsolatedProcess="true"` 在 `ServiceAttribute` 装饰服务类。

媒体

新 `Android.Media.MediaCodec` 类提供了对低级别的媒体编解码器的 API。应用程序可以查询系统以找出哪些较低级别的编解码器是可在设备上。

新 `Android.Media.Audiofx.AudioEffect` 添加了子类以支持其他音频预上捕获音频处理过程：

- `Android.Media.Audiofx.AcousticEchoCanceller` — 此类为预处理音频用于从捕获的音频信号从远程方删除信号。例如，从语音通信应用程序删除 echo。
- `Android.Media.AudiofxAutomaticGainControl` — 此类用于通过在提升或降低输入的信号，以便输出信号是常量规范化捕获的信号。
- `Android.Media.Audiofx.NoiseSuppressor` — 此类将从捕获的信号删除背景噪音。

并非所有设备将都支持这些效果。该方法 `AudioEffect.IsAvailable` 应由运行该应用程序在设备上是否受支持的音频效果相关的应用程序调用。

`MediaPlayer` 类现在支持具有无缝播放 `SetNextMediaPlayer()` 方法。这种新方法指定下一步的 `MediaPlayer` 时当前媒体播放器完成其播放启动。

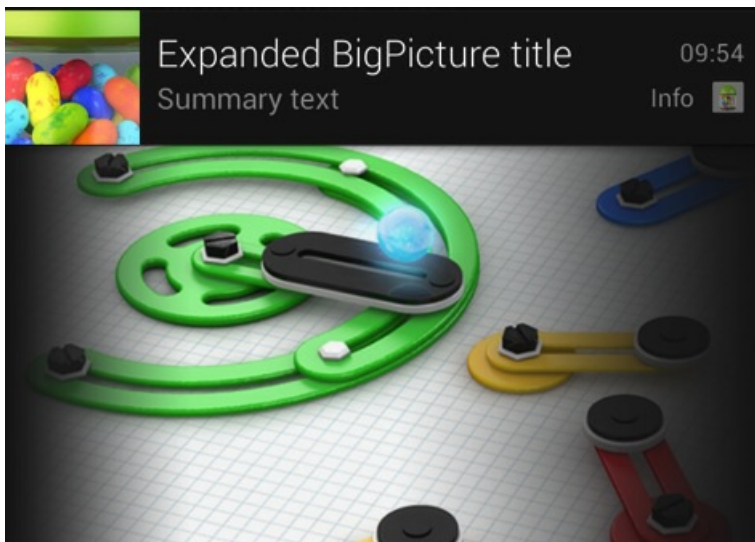
以下新类提供用于选择播放媒体的标准机制和 UI：

- `MediaRouter` — 此类允许应用程序来控制路由到外部扬声器的设备或其他设备中的媒体通道。
- `MediaRouterActionProvider` 和 `MediaRouteButton` — 这些类帮助提供一致的用户界面，用于选择和播放媒体。

通知

Android 4.1 允许应用程序更大的灵活性和控制能力的显示通知。应用程序现在可以向用户显示更大、更好的通知。新的方法，`NotificationBuilder.SetStyle()` 允许新建三个新样式之一上通知设置：

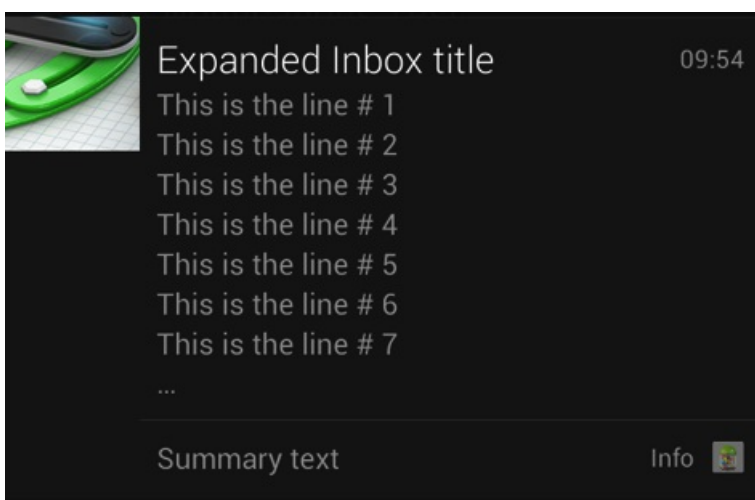
- `Notification.BigPictureStyle` — 这是将生成通知，其中将包含映像的一个帮助器类。下图显示了一个通知，其中大映像的示例：



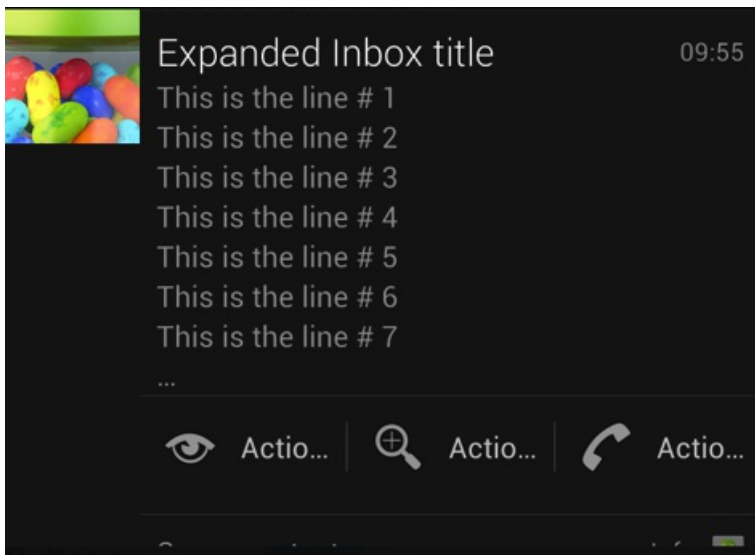
- `Notification.BigTextStyle` – 这是将生成将具有多行文本，如电子邮件通知的帮助器类。以下屏幕截图中可以看到此新的通知样式的示例：



- `Notification.InboxStyle` – 这是将生成包含一组字符串，如电子邮件中的代码片段的通知，如以下屏幕截图中所示的帮助器类：



就可以通知使用正常或更大样式时，底部的通知消息添加最多两个操作按钮。出现这种可在以下屏幕截图中，操作按钮是可见底部的通知：



`Notification` 类已收到新常量, 允许开发人员指定一个五个优先级别的通知。这些可以设置通知使用 `Priority` 属性。

权限

已添加以下新权限:

- `READ_EXTERNAL_STORAGE` -应用程序需要只读的访问到外部存储。当前所有应用程序默认情况下, 具有读取访问权限, 但未来版本的 Android 将需要应用程序显式请求读取访问权限。
- `READ_USER_DICTIONARY` -允许对用户的 word 词典的读取访问。
- `READ_CALL_LOG` -允许应用程序通过读取调用日志来获取有关传入和传出调用的信息。
- `WRITE_CALL_LOG` -允许应用程序能够在手机上写入调用日志。
- `WRITE_USER_DICTIONARY` -允许应用程序要写入到用户的 word 词典。

需要注意的重要更改 `READ_EXTERNAL_STORAGE` - 当前由 Android 自动授予此权限。未来版本的 Android 将需要应用程序请求之前此权限授予该权限。

总结

本文介绍了一些新的 API 的 Android 4.1 (API 级别为 16) 中可用。它突出显示某些动画和进行动画处理的活动, 该活动发布的更改, 并引入了用于使用 Bonjour 或 UPnP 等协议的其他设备的网络发现的新 API。API 的其他更改是将突出显示, 例如剪切和粘贴数据通过意向, 能够使用独立的服务或"不稳定"的内容提供商的功能。

这篇文章, 然后接着介绍更新通知, 并讨论某些 Android 4.1 中引入了新权限

相关链接

- [时间的动画示例 \(示例\)](#)
- [Android 4.1 Api](#)
- [任务和后退堆栈](#)
- [使用后退和向上导航](#)

Ice Cream Sandwich 功能

2018/10/26 • [Edit Online](#)

本指南介绍了几个与 Android 4 API-Ice Cream Sandwich 应用程序开发人员的新功能。它介绍了几种新的用户界面技术, 然后检查各种 Android 4 提供的数据之间的应用程序和设备之间共享的新功能。

概述

Android OS 版本 4.0 (API 级别 14) 表示主要的改编 Android 操作系统的并包括大量的重要更改和升级, 包括:

- **更新用户界面**– 几个新的 UI 功能使开发人员能够更多电源和接口时它们创建应用程序用户的灵活性。这些新功能包括: `GridLayout`, `PopupMenu`, `Switch` 小组件中, 和 `TextureView`。
- **更好的硬件加速**– 2D 呈现现在所有 Android 控件在 GPU 上的发生。此外, 硬件加速, 默认情况下为针对 Android 4.0 开发的所有应用程序中。
- **新的数据 Api**– 新访问不是以前正式访问, 如日历数据和设备所有者的用户配置文件的数据。
- **应用数据共享**– 应用程序和设备之间共享数据状态变得过通过技术如 `ShareActionProvider`, 这样就可以轻松从操作栏中, 创建共享操作并 Android 无线发送有关近场通信 (NFC), 便于管理单元中相互邻近的设备之间共享数据。

在本文中, 我们将探索这些功能和其他 Android 4.0 API, 对所做的更改, 我们将介绍如何通过 Xamarin.Android 使用的每个功能。

用户界面功能

各种新的用户界面技术是适用于 Android 4, 其中包括:

- **GridLayout** – 支持的控件的 2D 网格布局。
- **切换小部件** – 允许之间切换为 ON 或 OFF。
- **TextureView** – 使视频和 OpenGL 内容视图中的。
- **导航栏** – 包含虚拟按钮后, home 和多任务。

此外, 其他 UI 元素得到了增强, 如 `PopupMenu`, 这是现在更轻松地使用, 和选项卡, 它具有更精美的外观。

共享功能

Android 4 包括多种新技术, 让我们跨设备和应用程序之间共享数据。它还提供对各种类型的数据未之前提供, 如日历信息和设备所有者的用户配置文件的访问。在本部分中我们将介绍各种功能 Android 4 提供该地址这些领域包括:

- **Android 无线发送** – 允许数据通过 NFC 共享。
- **ShareActionProvider** – 创建提供程序, 允许开发人员指定操作栏中的共享操作。
- **用户配置文件** – 提供对设备所有者的配置文件数据的访问。
- **日历 API** – 从日历提供程序提供对日历数据的访问。

x86 仿真程序

ICS 尚不支持 x86 开发仿真程序。x86 仿真程序仅在使用 Android 2.3.3, API 级别 10 支持。请参阅[配置 x86 仿真程序](#)有关详细信息。

总结

本文介绍了各种现适用于 Android 4 的新技术。我们讨论了新的用户界面功能等`GridLayout`，`PopupMenu`，并开
关小组件。我们还了解了一些用于控制系统 UI，以及如何使用新的支持`TextureView`。然后，我们讨论了各种新共
享和技术。我们介绍如何`Android` 无线发送让我们使用的设备之间共享信息`NFC`，讨论新`日历API`，还介绍了如何
使用内置的`ShareActionProvider`。最后，我们探讨了如何使用`ContactsContract`访问用户配置文件数据提供程序。

相关链接

- [Ice Cream Sandwich 示例](#)
- [TextureViewDemo \(示例\)](#)
- [CalendarDemo \(示例\)](#)
- [选项卡布局教程](#)
- [Ice Cream Sandwich](#)
- [Android 4.0 平台](#)

简介 Contentprovider

2018/10/26 • • [Edit Online](#)

Android 操作系统使用的内容提供商来方便地对媒体文件、联系人和日历信息等的共享数据的访问。本文介绍 `ContentProvider` 类, 并提供如何使用它的两个示例。

内容提供程序概述

一个 `ContentProvider` 封装的数据存储库, 并提供一个 API 来访问它。提供程序存在通常还提供一个 UI 用于显示/管理数据的 Android 应用程序的一部分。使用内容提供商的主要好处, 让其他应用程序能够轻松地访问封装的数据使用的提供程序客户端对象 (称为 `ContentResolver`)。在一起, 内容提供商和内容冲突解决程序提供一致的应用程序间 API 进行生成和使用简单的数据访问。任何应用程序可以选择使用 `ContentProviders` 在内部管理数据以及将其公开给其他应用程序。

一个 `ContentProvider`, 还需要为应用程序提供自定义搜索建议, 或如果你想要提供应用程序以粘贴到其他应用程序中的复杂数据复制的功能。本文档演示如何访问和生成 `ContentProviders` 使用 `Xamarin.Android`。

本部分的结构如下所示:

- 其工作原理-内容概述 `ContentProvider` 旨在为, 以及如何配合工作。
- 使用内容提供商-示例: 访问联系人列表。
- 使用 **ContentProvider** 共享数据-编写和使用方 `ContentProvider` 在同一应用程序。

`ContentProviders` 并对其数据的游标通常用于填充 `Listview`。请参阅 [Listview](#) 和 [适配器指南](#) 有关如何使用这些类的详细信息。

`ContentProviders` 公开的 Android (或其他应用程序), 可以轻松在应用程序中包括来自其他源的数据。它们使你能够访问和呈现数据, 如联系人列表、照片或从应用程序中的日历事件, 让用户使用该数据进行交互。

自定义 `ContentProviders` 是由其他应用程序 (包括自定义搜索和复制/粘贴等特殊用途) 打包你的数据在你自己的应用, 内部使用或供使用的简便方法。

在本部分中的主题提供使用和编写一些简单示例 `ContentProvider` 代码。

相关链接

- [ContactsAdapter 演示 \(示例\)](#)
- [SimpleContentProvider \(示例\)](#)
- [内容提供程序开发人员指南](#)
- [ContentProvider 类引用](#)
- [ContentResolver 类引用](#)
- [ListView 类引用](#)
- [CursorAdapter 类引用](#)
- [UriMatcher 类引用](#)
- [Android.Provider](#)
- [ContactsContract 类引用](#)

如何在内容提供程序的效果

2018/10/26 • [Edit Online](#)

有两个类中所涉及 `ContentProvider` 交互：

- **ContentProvider** –实现以标准方式公开一组数据的 API。主要方法为查询、Insert、Update 和 Delete。
- **ContentResolver** –与通信的静态代理 `ContentProvider` 来访问其数据，或者从同一应用程序内或从另一个应用程序。

内容提供商通常由一个 SQLite 数据库，但 API，则意味着，使用代码不需要知道有关基础 SQL 的任何信息。查询都是通过 Uri 使用常量引用列名称（若要减少对基础数据结构的依赖项），和一个 `ICursor` 返回有关使用代码来循环访问。

使用 ContentProvider

`ContentProviders` 公开其功能通过在中注册的 Uri **AndroidManifest.xml** 的应用程序的发布的数据。是一种约定，Uri 和公开的数据列应该可用作常量来轻松地绑定到数据。Android 的内置 `ContentProviders` 所有提供方便的类具有引用中的数据结构的常量 `Android.Providers` 命名空间。

内置提供程序

Android 提供访问各种系统和用户数据使用 `ContentProviders`：

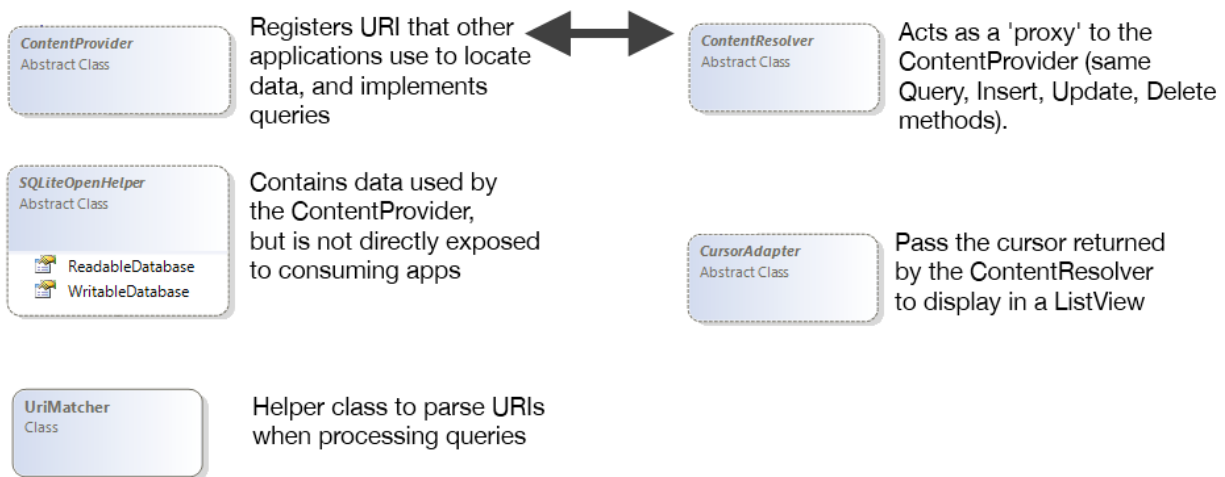
- **浏览器**–书签和浏览器历史记录（需要权限 `READ_HISTORY_BOOKMARKS` 和/或 `WRITE_HISTORY_BOOKMARKS`）。
- **CallLog** –还是与设备接收了最新的调用。
- **联系人**–详细的用户的联系人列表，其中包括人员、手机、照片和组中的信息。
- **MediaStore** –用户的设备的内容：音频（相册、艺术家、流派、播放列表）、（包括缩略图）的图像和视频。
- **设置**–系统范围内的设备设置和首选项。
- **UserDictionary** –用于预测文本输入的用户定义的字典的内容。
- **语音邮件**–语音邮件消息的历史记录。

类概述

处理时使用的主要类 `ContentProvider` 如下所示：

Content Provider application

Consuming application



在此图中，`ContentProvider` 实现查询，并注册 URI 的其他应用程序用来查找数据。`ContentResolver` 充当到的代理 `ContentProvider`（查询、插入、更新和删除方法）。`SQLiteOpenHelper` 包含使用的数据 `ContentProvider`，但系统不直接公开给使用应用。`CursorAdapter` 传递返回的游标 `ContentResolver` 中显示 `ListView`。`UriMatcher` 是处理查询时分析 Uri 的帮助器类。

每个类的用途如下所述：

- **ContentProvider** –实现此抽象类的方法来公开数据。该 API 可供其他类和应用程序中通过添加到类定义的 Uri 属性。
- **SQLiteOpenHelper** –可帮助实现由公开的 SQLite 数据存储 `ContentProvider`。
- **UriMatcher** –使用 `UriMatcher` 中你 `ContentProvider` 实现，以帮助管理用于查询内容的 Uri。
- **ContentResolver** –使用代码使用 `ContentResolver` 访问 `ContentProvider` 实例。两个类一起进程间通信问题，从而允许应用程序之间轻松共享数据处理。使用代码永远不会创建 `ContentProvider` 类显式；相反，数据访问通过创建基于公开的 Uri 的游标 `ContentProvider` 应用程序。
- **CursorAdapter** –使用 `CursorAdapter` 或 `SimpleCursorAdapter` 以显示数据通过访问 `ContentProvider`。

`ContentProvider` API 允许使用者执行各种操作的数据，例如：

- 查询数据，以返回列表或单个记录。
- 修改单个记录。
- 添加新记录。
- 删除记录。

本文档包含使用系统提供的示例，`ContentProvider`，以及一个简单的只读的示例实现一个自定义 `ContentProvider`。

使用联系人 ContentProvider

2018/10/26 • [Edit Online](#)

使用访问公开的数据的代码 `ContentProvider` 不需要引用 `ContentProvider` 根本类。相反, Uri 用于基于公开的数据创建游标 `ContentProvider`。Android 使用 Uri 来搜索系统的应用程序的公开 `ContentProvider` 具有此标识符。Uri 是一个字符串, 通常在反向 DNS 格式如 `com.android.contacts/data`。

而不是让开发人员请记住此字符串中, Android 联系人提供程序公开其元数据中的 `android.provider.ContactsContract` 类。此类用于确定的 Uri `ContentProvider` 以及表和列, 可查询的名称。

某些数据类型也需要访问的特殊权限。内置的联系人列表需要 `android.permission.READ_CONTACTS` 中的权限 **AndroidManifest.xml** 文件。

有三种方法从 Uri 创建一个游标:

1. **ManagedQuery()** – 首选的方法中 Android 2.3 (API 级别 10) 及更早版本, `ManagedQuery` 返回游标, 并还会自动管理刷新的数据和关闭游标。在 Android 3.0 (API 级别 11) 中已弃用此方法。
2. **ContentResolver.Query()** – 返回非托管的游标, 这意味着它必须是刷新, 在代码中显式关闭。
3. **CursorLoader().LoadInBackground()** – 在 Android 3.0 (API 级别 11) 中引入 `CursorLoader` 现已使用的首选的方式 `ContentProvider`。 `CursorLoader` 查询 `ContentResolver` 使 UI 不会阻止在后台线程上。在较旧版本的 Android 使用 v4 兼容性库可访问此类。

每种方法具有相同的基本输入集:

- **Uri** – 的完全限定的名 `ContentProvider`。
- **投影** – 光标选择哪些列规范。
- **所选内容** – 与 SQL 类似 `WHERE` 子句。
- **SelectionArgs** – 参数替换选定内容中。
- **SortOrder** – 要作为排序依据的列。

为查询创建输入

`ContactsProvider` 示例代码执行针对 Android 的内置联系人提供程序的非常简单查询。不需要知道实际 Uri 或列名称的查询联系人所需的所有信息 `ContentProvider` 为公开的常量可 `ContactsContract` 类。

无论使用哪种方法来检索光标, 这些相同的对象用作参数中所示 `ContactsProvider/ContactsAdapter.cs` 文件:

```
var uri = ContactsContract.Contacts.ContentUri;
string[] projection = {
    ContactsContract.Contacts.InterfaceConsts.Id,
    ContactsContract.Contacts.InterfaceConsts.DisplayName,
    ContactsContract.Contacts.InterfaceConsts.PhotoId,
};
```

对于本例, 请 `selection`, `selectionArgs` 并 `sortOrder` 将它们设置为忽略 `null`。

从内容提供商 Uri 创建游标

一旦创建参数对象, 它们可在以下三种方法之一:

使用托管的查询

面向 Android 2.3 (API 级别 10) 的应用程序或之前应使用此方法：

```
var cursor = activity.ManagedQuery(uri, projection, null, null, null);
```

此游标将由 Android 管理，因此不需要将其关闭。

使用 `ContentResolver`

访问 `ContentResolver` 直接以显示对光标 `ContentProvider` 可以完成如下所示：

```
var cursor = activity.ContentResolver(uri, projection, null, null, null);
```

此游标是受管理，因此不再需要时，必须关闭它。确保代码是否关闭游标处于打开状态，否则，将出现错误。

```
cursor.Close();
```

或者，可以调用 `StartManagingCursor()` 和 `StopManagingCursor()` 管理光标。托管的游标自动停用并重新查询活动时
被停止并重新启动。

使用 `CursorLoader`

应用程序构建适用于 Android 3.0 (API 级别 11) 或更高版本应该使用此方法：

```
var loader = new CursorLoader (activity, uri, projection, null, null, null);  
var cursor = (ICursor)loader.LoadInBackground();
```

`CursorLoader` 可确保所有游标操作都将在后台线程，并可以智能地重复使用现有游标在活动实例之间活动重新启动时（例如，由于配置更改）而不是，再次重新加载数据。

此外可以使用早期的 Android 版本 `CursorLoader` 通过使用类 [v4 支持库](#)。

显示具有自定义适配器的游标数据

若要显示联系人图片我们将使用自定义适配器，以便我们可以手动解决 `PhotoId` 对图像文件路径引用。

若要使用自定义适配器显示数据，该示例使用 `CursorLoader` 将检索到本地集中的所有联系人数据 `FillContacts` 方法从 `ContactsProvider/ContactsAdapter.cs`：

```

void FillContacts ()
{
    var uri = ContactsContract.Contacts.ContentUri;
    string[] projection = {
        ContactsContract.Contacts.InterfaceConsts.Id,
        ContactsContract.Contacts.InterfaceConsts.DisplayName,
        ContactsContract.Contacts.InterfaceConsts.PhotoId
    };
    // CursorLoader introduced in Honeycomb (3.0, API11)
    var loader = new CursorLoader(activity, uri, projection, null, null, null);
    var cursor = (ICursor)loader.LoadInBackground();
    contactList = new List<Contact> ();
    if (cursor.MoveToFirst ()) {
        do {
            contactList.Add (new Contact{
                Id = cursor.GetLong (cursor.GetColumnIndex (projection [0])),
                DisplayName = cursor.GetString (cursor.GetColumnIndex (projection [1])),
                PhotoId = cursor.GetString (cursor.GetColumnIndex (projection [2]))
            });
        } while (cursor.MoveNext());
    }
}

```

然后, 实现的 BaseAdapter 方法使用 `contactList` 集合。就像它是与任何其他集合实现适配器-没有任何特殊的处理
此处因为数据源自 `ContentProvider` :

```

Activity activity;
public ContactsAdapter (Activity activity)
{
    this.activity = activity;
    FillContacts ();
}
public override int Count {
    get { return contactList.Count; }
}
public override Java.Lang.Object GetItem (int position)
{
    return null; // could wrap a Contact in a Java.Lang.Object to return it here if needed
}
public override long GetItemId (int position)
{
    return contactList [position].Id;
}
public override View GetView (int position, View convertView, ViewGroup parent)
{
    var view = convertView ?? activity.LayoutInflater.Inflate (Resource.Layout.ContactListItem, parent, false);
    var contactName = view.FindViewById<TextView> (Resource.Id.ContactName);
    var contactImage = view.FindViewById<ImageView> (Resource.Id.ContactImage);
    contactName.Text = contactList [position].DisplayName;
    if (contactList [position].PhotoId == null) {
        contactImage = view.FindViewById<ImageView> (Resource.Id.ContactImage);
        contactImage.SetImageResource (Resource.Drawable.ContactImage);
    } else {
        var contactUri = ContentUris.WithAppendedId (ContactsContract.Contacts.ContentUri, contactList
[position].Id);
        var contactPhotoUri = Android.Net.Uri.WithAppendedPath (contactUri, Contacts.Photos.ContentDirectory);
        contactImage.SetImageURI (contactPhotoUri);
    }
    return view;
}

```

(如果存在), 会显示该图像在设备上的图像文件中使用的 Uri。应用程序如下所示:



使用类似的代码模式，你的应用程序可以访问各种系统数据包括用户的照片、视频和音乐。某些数据类型需要特殊权限以便在项目的请求**AndroidManifest.xml**。

显示与 SimpleCursorAdapter 游标数据

此外可以使用显示光标 `SimpleCursorAdapter` (尽管仅名称将显示，但不照片)。此代码演示如何使用 `ContentProvider` 与 `SimpleCursorAdapter` (此代码不会显示在此示例中)：

```
var uri = ContactsContract.Contacts.ContentUri;
string[] projection = {
    ContactsContract.Contacts.InterfaceConsts.Id,
    ContactsContract.Contacts.InterfaceConsts.DisplayName
};
var loader = new CursorLoader (this, uri, projection, null, null, null);
var cursor = (ICursor)loader.LoadInBackground();
var fromColumns = new string[] {ContactsContract.Contacts.InterfaceConsts.DisplayName};
var toControlIds = new int[] {Android.Resource.Id.Text1};
adapter = new SimpleCursorAdapter (this, Android.Resource.Layout.SimpleListItem1, cursor, fromColumns,
toControlsIds);
listView.Adapter = adapter;
```

请参阅[Listview 和适配器](#)有关详细信息实现 `SimpleCursorAdapter`。

相关链接

- [ContactsAdapter 演示 \(示例\)](#)

创建自定义 ContentProvider

2018/10/26 • [Edit Online](#)

上一节演示了如何使用内置的 `ContentProvider` 实现中的数据。本部分将介绍如何生成自定义 `ContentProvider`，然后使用其数据。

有关 Contentprovider

内容提供程序类必须继承 `ContentProvider`。它应包含用于对查询进行响应的内部数据存储和常量以帮助使用代码进行的数据的有效请求，它应公开的 Uri 和 MIME 类型。

URI (颁发机构)

`ContentProviders` 使用 Uri 在 Android 中访问的。公开的应用程序 `ContentProvider` 设置它将对在做出响应的 Uri 及其 **AndroidManifest.xml** 文件。安装应用程序时，这些 Uri 注册，以便其他应用程序可以访问它们。

适用于 Android 的 Mono 中的内容提供程序类应有 `[ContentProvider]` 属性指定的 Uri (或 Uri)，应将其添加到 **AndroidManifest.xml**。

Mime 类型

MIME 类型的典型格式由两部分组成。Android `ContentProviders` 通常使用这两个字符串的 MIME 类型的第一部分：

1. `vnd.android.cursor.item` - 若要表示的单个行，使用 `ContentResolver.CursorItemType` 常量在代码中。
2. `vnd.android.cursor.dir` - 对于多个行，使用 `ContentResolver.CursorDirType` 常量在代码中。

MIME 类型的第二部分是特定于应用程序，并应使用反向 DNS 标准的与 `vnd.` 前缀。示例代码使用 `vnd.com.xamarin.sample.Vegetables`。

数据模型元数据

使用应用程序需要构造 Uri 查询访问不同类型的数据。基本 Uri 可以扩展来引用特定表的数据，并且还包括参数以筛选结果。此外必须声明的列和子句与生成的游标用于显示数据。

若要确保仅有效的 Uri 查询构造，它是惯用提供作为常数值的有效字符串。这使得能够更轻松地访问 `ContentProvider` 因为它会使值可通过代码完成功能发现并防止在字符串中的拼写错误。

在前面的示例 `android.provider.ContactsContract` 类公开的元数据的联系人数据。对于我们自定义 `ContentProvider` 我们只需将公开为类本身的常量。

实现

有三个步骤创建和使用自定义到 `ContentProvider`：

1. 创建一个数据库类-实现 `SQLiteOpenHelper`。
2. 创建 `ContentProvider` 类-实现 `ContentProvider` 与数据库的实例，元数据公开为常量值和方法来访问数据。
3. 访问 `ContentProvider` 通过其 **Uri** -填充 `CursorAdapter` 使用 `ContentProvider`，可通过其 Uri。

前面所述，`ContentProviders` 可以从应用程序而非定义位置使用。在此示例中使用的数据在同一应用程序，但请记住，其他应用程序还可以访问它，只要他们知道的 Uri 和架构（这通常会公开为常量值）的信息。

创建数据库

大多数 `ContentProvider` 实现将基于 `SQLite` 数据库。数据库中的代码示

例 **SimpleContentProvider/VegetableDatabase.cs** 创建一个非常简单的两个列数据库，如所示：

```
class VegetableDatabase : SQLiteOpenHelper {
    const string create_table_sql =
        "CREATE TABLE [vegetables] ([_id] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, [name] TEXT NOT NULL UNIQUE)";
    const string DatabaseName = "vegetables.db";
    const int DatabaseVersion = 1;

    public VegetableDatabase(Context context) : base(context, DatabaseName, null, DatabaseVersion) { }
    public override void OnCreate(SQLiteDatabase db)
    {
        db.ExecSQL(create_table_sql);
        // seed with data
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Vegetables')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Fruits')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Flower Buds')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Legumes')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Bulbs')");
        db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Tubers')");
    }
    public override void OnUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        throw new NotImplementedException();
    }
}
```

数据库实现本身不需要任何特殊的注意事项，以通过公开 `ContentProvider`，但是如果你想要将绑定 `ContentProvider`'s 数据到 `ListView` 然后控件名为的唯一整数列 `_id` 必须属于结果集。请参阅 [ListView 和适配器](#) 使用的更多详细信息的文档 `ListView` 控件。

创建 ContentProvider

本部分的其余部分介绍有关如何分步说明 **SimpleContentProvider/VegetableProvider.cs** 生成的示例类。

初始化数据库

第一步是为子类 `ContentProvider` 并添加它将使用的数据库。

```
public class VegetableProvider : ContentProvider
{
    VegetableDatabase vegeDB;
    public override bool OnCreate()
    {
        vegeDB = new VegetableDatabase(Context);
        return true;
    }
}
```

代码的其余部分将窗体可用于发现和查询数据的实际内容的提供程序实现。

添加元数据的使用者

有四种不同类型的元数据，我们将在公开 `ContentProvider` 类。颁发机构是必需的仅由约定完成其余部分。

- 颁发机构 – `ContentProvider` 特性 *必须* 添加到类，以便它是否已注册的 Android 安装应用程序。
- Uri** – `CONTENT_URI`，这样就可以方便使用在代码中公开为常量。它应匹配颁发机构，但包含的方案和基路径。

- **MIME 类型**—结果和单个结果的列表被视为不同的内容类型，因此我们定义两个 MIME 类型来表示它们。
- **InterfaceConsts**—为每个数据列名称，提供常量值，以便使用代码可以轻松地发现并不会有键入错误的风险的情况下引用它们。

此代码显示了其中每一项的实现方式，将从上一步添加到数据库定义：

```
[ContentProvider(new string[] { CursorTableAdapter.VegetableProvider.AUTHORITY })]
public class VegetableProvider : ContentProvider
{
    public const string AUTHORITY = "com.xamarin.sample.VegetableProvider";
    static string BASE_PATH = "vegetables";
    public static readonly Android.Net.Uri CONTENT_URI = Android.Net.Uri.Parse("content://" + AUTHORITY + "/" +
    BASE_PATH);
    // MIME types used for getting a list, or a single vegetable
    public const string VEGETABLES_MIME_TYPE = ContentResolver.CursorDirBaseType +
    "/vnd.com.xamarin.sample.Vegetables";
    public const string VEGETABLE_MIME_TYPE = ContentResolver.CursorItemBaseType +
    "/vnd.com.xamarin.sample.Vegetables";
    // Column names
    public static class InterfaceConsts {
        public const string Id = "_id";
        public const string Name = "name";
    }
    VegetableDatabase vegeDB;
    public override bool OnCreate()
    {
        vegeDB = new VegetableDatabase(Context);
        return true;
    }
}
```

实现 URI 分析帮助程序

因为使用的代码使用 Uri 发出请求的 `ContentProvider`，我们需要能够分析这些请求以确定要返回的数据。

`UriMatcher` 类可帮助分析 Uri 后已初始化，`Uri` 模式的 `ContentProvider` 支持。

`UriMatcher` 在示例中用于初始化两个 Uri:

1. `"com.xamarin.sample.VegetableProvider/vegetables"`—请求，以便返回蔬菜的完整列表。
2. `"com.xamarin.sample.VegetableProvider/vegetables/#"`—其中#是数值参数的占位符 (`_id` 的数据库中的行)。星号占位符 ("*") 也可用于匹配一个文本参数。

在代码中我们使用常量来指代的元数据值，如颁发机构和基本_路径。将执行分析，以确定要返回的数据的 Uri 的方法中使用返回代码。

```
const int GET_ALL = 0; // return code when list of Vegetables requested
const int GET_ONE = 1; // return code when a single Vegetable is requested by ID
static UriMatcher uriMatcher = BuildUriMatcher();
static UriMatcher BuildUriMatcher()
{
    var matcher = new UriMatcher(UriMatcher.NoMatch);
    // Uris to match, and the code to return when matched
    matcher.AddURI(AUTHORITY, BASE_PATH, GET_ALL); // all vegetables
    matcher.AddURI(AUTHORITY, BASE_PATH + "/#", GET_ONE); // specific vegetable by numeric ID
    return matcher;
}
```

此代码是所有私有的 `ContentProvider` 类。请参阅[Google UriMatcher 文档](#)有关进一步信息。

实现 QueryMethod

最简单 `ContentProvider` 实现的方法是 `Query` 方法。使用下面的实现 `UriMatcher` 分析 `uri` 参数并调用正确的数据库的方法。如果 `uri` 包含一个 ID 参数, 则将整数解析出 (使用 `LastPathSegment`) 和数据库查询中使用。

```
public override Android.Database.ICursor Query(Android.Net.Uri uri, string[] projection, string selection,
string[] selectionArgs, string sortOrder)
{
    switch (uriMatcher.Match(uri)) {
        case GET_ALL:
            return GetFromDatabase();
        case GET_ONE:
            var id = uri.LastPathSegment;
            return GetFromDatabase(id); // the ID is the last part of the Uri
        default:
            throw new Java.Lang.IllegalArgumentException("Unknown Uri: " + uri);
    }
}

Android.Database.ICursor GetFromDatabase()
{
    return vegeDB.ReadableDatabase.RawQuery("SELECT _id, name FROM vegetables", null);
}

Android.Database.ICursor GetFromDatabase(string id)
{
    return vegeDB.ReadableDatabase.RawQuery("SELECT _id, name FROM vegetables WHERE _id = " + id, null);
}
```

`GetType` 必须也重写方法。可能会调用此方法, 以确定将为给定的 Uri 返回的内容类型。这可能会告知使用方应用程序如何处理该数据。

```
public override String GetType(Android.Net.Uri uri)
{
    switch (uriMatcher.Match(uri)) {
        case GET_ALL:
            return VEGETABLES_MIME_TYPE; // list
        case GET_ONE:
            return VEGETABLE_MIME_TYPE; // single item
        default:
            throw new Java.Lang.IllegalArgumentException("Unknown Uri: " + uri);
    }
}
```

实现其他替代

我们的简单示例不允许进行编辑或删除数据, 但必须实现的 `Insert`、`Update` 和 `Delete` 方法因此将它们添加不包含实现:

```
public override int Delete(Android.Net.Uri uri, string selection, string[] selectionArgs)
{
    throw new Java.Lang.UnsupportedOperationException();
}

public override Android.Net.Uri Insert(Android.Net.Uri uri, ContentValues values)
{
    throw new Java.Lang.UnsupportedOperationException();
}

public override int Update(Android.Net.Uri uri, ContentValues values, string selection, string[] selectionArgs)
{
    throw new Java.Lang.UnsupportedOperationException();
}
```


完成基本 `ContentProvider` 实现。一旦已安装应用程序，它公开的数据可同时在应用程序内，而且还知道要对其进行引用的 Uri 的任何其他应用程序。

访问 ContentProvider

一次 `VegetableProvider` 已实现，对其进行访问可以通过相同方法在本文档开头的联系人提供程序为：获取使用指定的 Uri 游标，然后使用适配器访问的数据。

将 ListView 绑定到 ContentProvider

若要填充 `ListView` 数据中，我们将使用未筛选的蔬菜列表相对应的 Uri。我们在代码中使用的常量值 `VegetableProvider.CONTENT_URI`，我们知道它解析为 `com.xamarin.sample.vegetableprovider/vegetables`。我们 `VegetableProvider.Query` 实现将返回一个游标，然后绑定到 `ListView`。

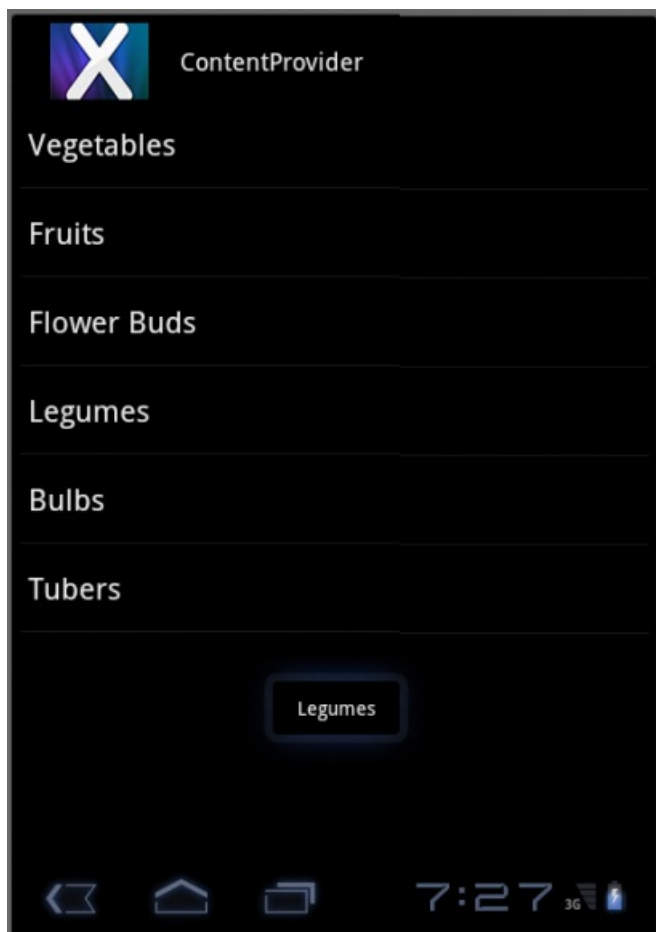
中的代码 `SimpleContentProvider/HomeScreen.cs` 显示了以显示来自数据是简单 `ContentProvider`：

```
listView = FindViewById<ListView>(Resource.Id.List);
string[] projection = new string[] { VegetableProvider.InterfaceConsts.Id,
VegetableProvider.InterfaceConsts.Name } ;
string[] fromColumns = new string[] { VegetableProvider.InterfaceConsts.Name };
int[] toControlIds = new int[] { Android.Resource.Id.Text1 };

// CursorLoader introduced in Honeycomb (3.0, API_11)
var loader = new CursorLoader(this,
    VegetableProvider.CONTENT_URI, projection, null, null, null);
cursor = (ICursor)loader.LoadInBackground();

// Create a SimpleCursorAdapter
adapter = new SimpleCursorAdapter(this, Android.Resource.Layout.SimpleListItem1, cursor, fromColumns,
toControlIds);
listView.Adapter = adapter;
```

生成的应用程序如下所示：



检索一个项从 ContentProvider

使用方应用程序可能还想要访问的数据, 这可以通过构造一个不同的 Uri (例如) 是指特定行的单个行。

使用 `ContentResolver` 直接以访问单个项, 通过使用所需的 Uri 构建 `Id`。

```
Uri.WithAppendedPath(VegetableProvider.CONTENT_URI, id.ToString());
```

完整的方法如下所示:

```
protected void OnListItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
    var id = e.Id;
    string[] projection = new string[] { "name" };
    var uri = Uri.WithAppendedPath(VegetableProvider.CONTENT_URI, id.ToString());
    ICursor vegeCursor = ContentResolver.Query(uri, projection, null, new string[] { id.ToString() }, null);
    string text = "";
    if (vegeCursor.MoveToFirst()) {
        text = vegeCursor.GetInt(0) + " " + vegeCursor.GetString(1);
        Android.Widget.Toast.MakeText(this, text, Android.Widget.ToastLength.Short).Show();
    }
    vegeCursor.Close();
}
```

相关链接

- [SimpleContentProvider \(示例\)](#)

地图和位置

2018/10/26 • [Edit Online](#)

位置服务

本指南介绍 Android 应用程序中的位置感知, 并说明了如何获取 Google 位置服务 API 中使用 Android 位置服务 API, 以及在打印位置可用的提供程序的用户的位置。

地图

本文介绍如何使用 Xamarin.Android 使用地图和位置。它涵盖了从利用内置地图应用程序直接使用 Google 映射 Android API V2 的所有内容。此外, 它说明了如何使用单个 API 来使用位置服务, 它允许应用程序以获取通过单元格 tower 位置、Wi-fi 或 GPS 位置修补程序。

位置服务

2018/11/13 • [Edit Online](#)

本指南介绍 Android 应用程序中的位置感知, 并说明了如何获取 Google 位置服务 API 中使用 Android 的位置服务 API, 以及可用的浮点混合的位置提供程序的用户的位置。

位置服务概述

Android 提供对单元格 tower 位置、Wi-fi 和 GPS 等各种位置技术的访问。每个位置技术的详细信息抽象化通过 *位置提供程序*, 允许应用程序而不考虑使用的提供程序相同的方式获取位置。本指南介绍浮点混合的位置提供程序, Google Play Services 后, 它可以智能地确定获取基于哪些提供程序和设备的的使用方式的设备的位置的最佳方法的一部分。Android 位置服务 API, 并演示如何与系统位置通信服务中使用 `LocationManager`。本指南的第二部分探讨了 Android 位置服务 API 使用 `LocationManager`。

作为常规经验, 应用程序应首选使用浮点混合的位置提供程序, 回退较旧 Android 位置服务 API 仅在必要时。

位置基础知识

在 Android 中, 无论您选择使用位置数据的哪些 API 的几个概念保持不变。本部分介绍位置提供程序和与位置相关的权限。

位置提供程序

在内部使用多种技术, 以查明用户的位置。使用的硬件上的类型取决于 *位置提供程序* 用于收集数据的作业。Android 使用三个位置提供程序:

- **GPS 提供程序** – GPS 提供最准确的位置, 使用最高的能力, 并室外可达效果最佳。此提供程序使用 GPS 和辅助的 GPS 的组合 (**aGPS**), 这会返回收集的移动电话 towers GPS 数据。
- **网络提供商**–提供的 WiFi 和手机网络数据, 包括收集的单元格 towers aGPS 数据组合。它使用较少的电量比 GPS 提供程序, 但返回不同的准确性的位置数据。
- **被动的提供程序**–使用请求的其他应用程序或服务提供程序以生成位置数据的应用程序中的"非法携带"选项。这是一个不太可靠但的节能选项非常适合于不需要常量的位置更新, 若要运行应用程序。

位置提供程序并不总是可用。例如, 我们可能想要为应用程序, 使用 GPS 但 GPS 可能在设置中, 已关闭或设备可能不会在所有具有 GPS。如果特定的提供程序不可用, 则选择该提供程序可能会返回 `null`。

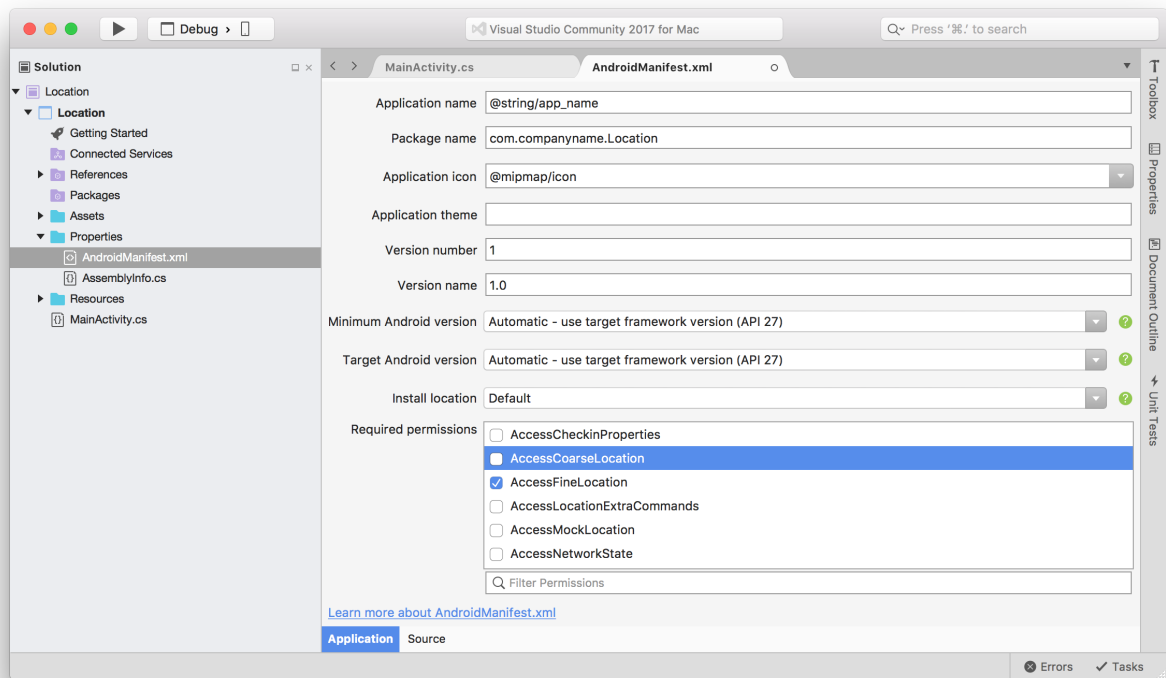
位置权限

位置感知应用程序需要访问设备的硬件传感器接收 GPS、Wi-fi 和移动电话网络数据。通过应用程序的 Android 清单中的适当权限控制的访问。有两个权限–具体取决于应用程序的要求和所选的 API, 你将想要允许一:

- `ACCESS_FINE_LOCATION` – 允许对 GPS 的应用程序访问权限。所需 *GPS 提供程序* 并 *被动的提供程序* 选项 (*被动的提供程序* 必须有权访问 GPS 数据收集的另一个应用程序或服务)。可选的权限 *网络提供商*。
- `ACCESS_COARSE_LOCATION` – 允许对手机网络和 Wi-fi 位置的应用程序访问权限。所需 *网络提供商* 如果 `ACCESS_FINE_LOCATION` 未设置。

对于面向 API 21 (Android 5.0 Lollipop) 版本的应用或更高版本, 可以启用 `ACCESS_FINE_LOCATION` 并仍在没有 GPS 硬件的设备上运行。如果你的应用需要 GPS 硬件, 您应显式添加 `android.hardware.location.gps` `uses-feature` Android 清单元素。有关详细信息, 请参阅 Android [使用功能](#) 元素引用。

若要设置权限, 请展开属性中的文件夹 **Solution Pad**, 然后双击 **AndroidManifest.xml**。将下面列出的权限所需的权限:



设置这些权限之一告知 Android 应用程序需要从用户的权限才能访问的位置提供程序。设备的运行 API 级别 22 (Android 5.1) 或更低时将要求用户每次安装应用程序授予这些权限。在设备运行 API 级别 23 (Android 6.0) 或更高版本, 应用应运行时权限之前执行检查发出请求的位置提供程序。

NOTE

注意: 设置 `ACCESS_FINE_LOCATION` 意味着对这两个粗糙和正常位置数据的访问。应无需设置这两个权限, 仅最小您的应用程序工作所需的权限。

此代码片段示范了如何检查应用具有的权限 `ACCESS_FINE_LOCATION` 权限:

```
if (ContextCompat.CheckSelfPermission(this, Manifest.Permission.AccessFineLocation) == Permission.Granted)
{
    StartRequestingLocationUpdates();
    isRequestingLocationUpdates = true;
}
else
{
    // The app does not have permission ACCESS_FINE_LOCATION
}
```

应用程序必须为容错的用户权限将授予 (或已吊销权限) 的方案, 有一种方法来适当地处理这种情况。请参阅[权限指南](#)上实现运行时权限的更多详细信息会检查在 Xamarin.Android 中。

使用浮点混合的位置提供程序

浮点混合的位置提供程序是为 Android 应用程序以从设备接收位置更新, 因为它会有效地在运行时提供最佳的位置信息, 以电池高效的方式选择的位置提供程序的首选的方法。例如, 移动办公了室外用户获取使用 GPS 读取的最佳位置。如果用户然后走室内, 其中 GPS 的工作效果不佳 (如果有), 浮点混合的位置提供程序可能会自动切换到 WiFi, 这非常适合更好的室内。

浮点混合的位置提供程序 API 提供了多种其他工具来使位置感知应用程序, 包括地理围栏和活动监视。在本部分中, 我们将重点介绍有关设置的基础知识 `LocationClient`, 建立提供程序, 以及获取用户的位置。

浮点混合的位置提供程序的一部分[Google Play Services](#)。Google Play Services 包必须安装并正确配置中的浮点混合的位置提供程序 API 应用程序工作, 并且设备必须具有对 Google Play Services APK 安装。

在之前 Xamarin.Android 应用程序可以使用浮点混合的位置提供程序, 它必须添加 **Xamarin.GooglePlayServices.Maps** 到项目的包。此外, 以下 `using` 语句应添加到引用如下所述的类的任何源文件:

```
using Android.Gms.Common;
using Android.Gms.Location;
```

检查是否已安装 Google Play Services

然后运行时异常, Xamarin.Android 会崩溃, 如果它尝试在未安装 Google Play 服务时使用浮点混合的位置提供程序 (或过期), 会发生。如果未安装 Google Play 服务, 然后在应用程序应回退到 Android 上文所述的位置服务。Google Play 服务已过时, 如果应用程序可能向用户询问他们更新已安装的版本的 Google Play 服务显示一条消息。

此代码段是如何对 Android 活动可以通过编程方式检查是否已安装 Google Play 服务的示例:

```
bool IsGooglePlayServicesInstalled()
{
    var queryResult = GoogleApiAvailability.Instance.IsGooglePlayServicesAvailable(this);
    if (queryResult == ConnectionResult.Success)
    {
        Log.Info("MainActivity", "Google Play Services is installed on this device.");
        return true;
    }

    if (GoogleApiAvailability.Instance.IsUserResolvableError(queryResult))
    {
        // Check if there is a way the user can resolve the issue
        var errorString = GoogleApiAvailability.Instance.GetErrorString(queryResult);
        Log.Error("MainActivity", "There is a problem with Google Play Services on this device: {0} - {1}",
            queryResult, errorString);

        // Alternately, display the error to the user.
    }

    return false;
}
```

FusedLocationProviderClient

若要与浮点混合的位置提供程序进行交互, Xamarin.Android 应用程序必须具有的实例

`FusedLocationProviderClient`。此类公开必需的方法来订阅位置更新和检索设备的最后一个已知的位置。

`OnCreate` 活动的方法是一个合适的地方获取对引用 `FusedLocationProviderClient`, 如下面的代码段中所示:

```
public class MainActivity: AppCompatActivity
{
    FusedLocationProviderClient fusedLocationProviderClient;

    protected override void OnCreate(Bundle bundle)
    {
        fusedLocationProviderClient = LocationServices.GetFusedLocationProviderClient(this);
    }
}
```

获取最后一个已知的位置

`FusedLocationProviderClient.GetLastLocationAsync()` 方法提供了有关 Xamarin.Android 应用程序快速获取具有最少

的编码开销的设备的最后一个已知的位置简单、非阻止方式。

此代码片段演示如何使用 `GetLastLocationAsync` 方法来检索设备的位置：

```
async Task GetLastLocationFromDevice()
{
    // This method assumes that the necessary run-time permission checks have succeeded.
    getLastLocationButton.SetText(Resource.String.getting_last_location);
    Android.Locations.Location location = await fusedLocationProviderClient.GetLastLocationAsync();

    if (location == null)
    {
        // Seldom happens, but should code that handles this scenario
    }
    else
    {
        // Do something with the location
        Log.Debug("Sample", "The latitude is " + location.Latitude);
    }
}
```

订阅到位置更新

Xamarin.Android 应用程序还可以从使用浮点混合的位置提供程序订阅到位置更新

`FusedLocationProviderClient.RequestLocationUpdatesAsync` 方法，此代码片段中所示：

```
await fusedLocationProviderClient.RequestLocationUpdatesAsync(locationRequest, locationCallback);
```

此方法采用两个参数：

- `Android.Gms.Location.LocationRequest` – 一个 `LocationRequest` 对象是 Xamarin.Android 应用程序如何将参数传递的浮点混合的位置提供程序的工作原理。`LocationRequest` 保存信息此类应进行频繁的请求或准确位置更新应为重要程度如何。例如，一个重要位置请求将导致设备时要使用 GPS，并因此更多能耗，确定位置。此代码片段演示如何创建 `LocationRequest` 对于高精度的位置，检查大约每 5 分钟一次位置更新（但不是早于请求之间的两分钟）。浮点混合的位置提供程序将使用 `LocationRequest` 作为要使用时尝试确定设备位置的位置提供程序的指导原则：

```
LocationRequest locationRequest = new LocationRequest()
    .SetPriority(LocationRequest.PriorityHighAccuracy)
    .SetInterval(60 * 1000 * 5)
    .SetFastestInterval(60 * 1000 * 2);
```

- `Android.Gms.Location.LocationCallback` – 为了接收位置更新，Xamarin.Android 应用程序必须子类 `LocationProvider` 抽象类。此类公开由浮点混合的位置提供程序来更新应用程序的位置信息可能是调用该两种方法。这将下面更详细地讨论。

若要通知的位置更新的 Xamarin.Android 应用程序，浮点混合的位置提供程序将调用

`LocationCallback.OnLocationResult(LocationResult result)`。`Android.Gms.Location.LocationResult` 参数将包含更新位置信息。

当浮点混合的位置提供程序位置数据的可用性中检测到更改时，它将调用

`LocationProvider.OnLocationAvailability(LocationAvailability locationAvailability)` 方法。如果 `LocationAvailability.IsLocationAvailable` 属性返回 `true`，则可以假定，将设备位置结果报告 `OnLocationResult` 一样准确且为最新的所需的 `LocationRequest`。如果 `IsLocationAvailable` 为 `false`，则不将返回的任何位置结果 `OnLocationResult`。

此代码片段是一个示例实现 `LocationCallback` 对象：

```

public class FusedLocationProviderCallback : LocationCallback
{
    readonly MainActivity activity;

    public FusedLocationProviderCallback(MainActivity activity)
    {
        this.activity = activity;
    }

    public override void OnLocationAvailability(LocationAvailability locationAvailability)
    {
        Log.Debug("FusedLocationProviderSample", "IsLocationAvailable:
{0}",locationAvailability.IsLocationAvailable);
    }

    public override void OnLocationResult(LocationResult result)
    {
        if (result.Locations.Any())
        {
            var location = result.Locations.First();
            Log.Debug("Sample", "The latitude is :" + location.Latitude);
        }
        else
        {
            // No locations to work with.
        }
    }
}

```

使用 Android 位置服务 API

Android 的位置服务是用于在 Android 上使用位置信息的较旧的 API。位置数据收集的硬件传感器和系统服务，可在访问该应用程序中收集 `LocationManager` 类和一个 `ILocationListener`。

位置服务最适合用于必须在没有 Google Play Services 安装的设备运行的应用程序。

位置服务是一种特殊服务由系统管理。系统服务与设备硬件进行交互，并始终在运行。若要利用我们的应用程序中的位置更新，我们将订阅位置更新系统的位置服务使用从 `LocationManager` 和一个 `RequestLocationUpdates` 调用。

若要获取用户的位置使用 Android 位置服务涉及到几个步骤：

1. 获取对引用 `LocationManager` 服务。
2. 实现 `ILocationListener` 位置更改时，接口和句柄事件。
3. 使用 `LocationManager` 为指定的提供程序的请求位置更新。 `ILocationListener` 前一步骤中将用来接收从回调 `LocationManager`。
4. 应用程序它不再适合时接收更新，请停止位置更新。

位置管理器

我们就可以访问的实例的系统位置服务 `LocationManager` 类。 `LocationManager` 是一个特殊类，可以让我们使用的系统位置服务进行交互，对其调用方法。应用程序可以获取对的引用 `LocationManager` 通过调用 `GetSystemService` 并传入服务类型，如下所示：

```

LocationManager locationManager = (LocationManager) GetSystemService(Context.LocationService);

```

`OnCreate` 为获取对引用的好时机 `LocationManager`。它是最好随时 `LocationManager` 作为类变量，以便我们可以在活动生命周期的各个点调用它。

从 `LocationManager` 请求位置更新

一旦应用程序后，对引用 `LocationManager`，它需要告知 `LocationManager` 哪种类型的位置信息所必需的并更新该信息的频率。执行此操作通过调用 `RequestLocationUpdates` 上 `LocationManager` 对象，并传入更新和将接收位置更新的回调的一些标准。此回调是一种类型，必须实现 `ILocationListener` 接口（在本指南后面的更多详细信息中所述）。

`RequestLocationUpdates` 方法告诉系统位置服务，你的应用程序想要开始接收位置更新。此方法允许您指定的提供程序，以及时间和距离阈值来控制更新频率。例如，下面的方法请求从 GPS 位置提供程序每隔 2000 毫秒，更新的位置，并仅当位置更改多个 1 metre:

```
// For this example, this method is part of a class that implements ILocationListener, described below
locationManager.RequestLocationUpdates(LocationManager.GpsProvider, 2000, 1, this);
```

应用程序应仅根据需要为很好地运行应用程序请求位置更新。这会保留的电池使用寿命，并创建用户更好的体验。

响应来自 `LocationManager` 的更新

应用程序已请求从更新后 `LocationManager`，它可以从服务接收的信息，通过实现 `ILocationListener` 接口。此接口提供了四个方法以侦听服务的位置和位置提供程序，`OnLocationChanged`。系统将调用 `OnLocationChanged` 用户的位置发生更改时够格作为根据请求位置更新时所设置的条件的条件的位置更改。

下面的代码演示中的方法 `ILocationListener` 接口：

```
public class MainActivity : AppCompatActivity, ILocationListener
{
    TextView latitude;
    TextView longitude;

    public void OnLocationChanged (Location location)
    {
        // called when the location has been updated.
    }

    public OnProviderDisabled(string locationProvider)
    {
        // called when the user disables the provider
    }

    public OnProviderEnabled(string locationProvider)
    {
        // called when the user enables the provider
    }

    public OnStatusChanged(string locationProvider, Availability status, Bundle extras)
    {
        // called when the status of the provider changes (there are a variety of reasons for this)
    }
}
```

取消订阅到 `LocationManager` 更新

为了节省系统资源，应用程序应取消订阅到位置更新越早越好。`RemoveUpdates` 方法会示意 `LocationManager` 停止将更新发送到我们的应用程序。例如，活动可以调用 `RemoveUpdates` 在 `OnPause` 方法，以便我们能够节省电量，如果应用程序不需要位置更新其活动时不在屏幕上：

```
protected override void OnPause ()
{
    base.OnPause ();
    locationManager.RemoveUpdates (this);
}
```

如果你的应用程序需要获取在后台位置更新，你将想要创建订阅的系统位置服务的自定义服务。请参阅[后台处理及](#)

[Android 服务指南](#)以获取详细信息。

确定 `LocationManager` 的最佳位置提供程序

更高版本的应用程序将 GPS 设置为位置提供程序。但是, GPS 可能不能在所有情况下, 例如设备是否室内或不具有 GPS 接收器。如果是这样, 则结果是 `null` 返回提供程序。

若要获取你的应用程序能够 GPS 不可用时, 您使用 `GetBestProvider` 方法来请求应用程序启动时的最佳可用 (支持设备和用户启用) 的位置提供程序。而不是特定提供程序中传递, 可以判断 `GetBestProvider` 如准确性和电源的使用的提供程序的要求 `Criteria` 对象。 `GetBestProvider` 返回给定条件的最佳提供程序。

下面的代码演示如何获取最佳可用的提供程序, 并请求位置更新时使用它:

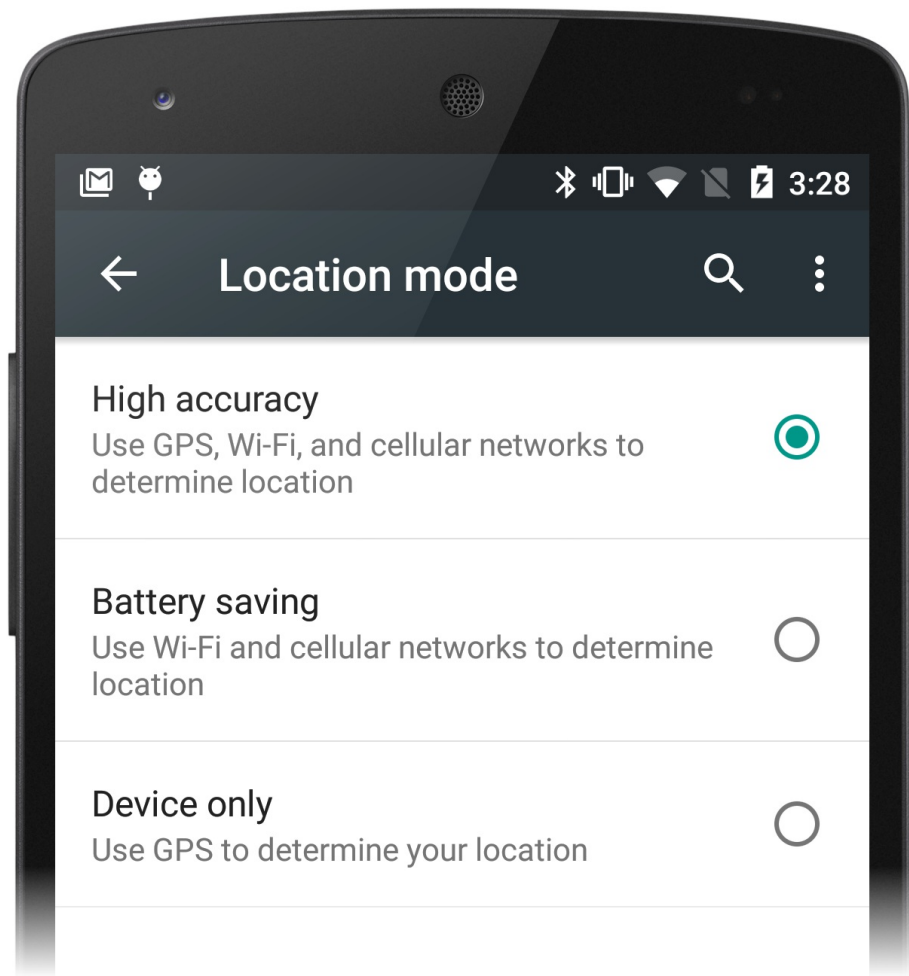
```
Criteria locationCriteria = new Criteria();
locationCriteria.Accuracy = Accuracy.Coarse;
locationCriteria.PowerRequirement = Power.Medium;

locationProvider = locationManager.GetBestProvider(locationCriteria, true);

if(locationProvider != null)
{
    locationManager.RequestLocationUpdates (locationProvider, 2000, 1, this);
}
else
{
    Log.Info(tag, "No location providers available");
}
```

NOTE

如果用户已禁用所有位置提供程序 `GetBestProvider` 将返回 `null`。若要查看此代码在真实设备上的工作方式, 请务必启用 GPS、Wi-fi 和移动电话网络下的 **Google 设置 > 位置 > 模式** 此屏幕截图中所示:



下面的屏幕截图演示了如何位置应用程序正在运行使用 `GetBestProvider` :



请记住，`GetBestProvider` 不会动态更改提供程序。然而，它将一次在活动生命周期期间确定最佳可用的提供程序。如果提供程序状态将更改已设置后，应用程序将需要额外的代码中 `ILocationListener` 方法— `OnProviderEnabled`，`OnProviderDisabled`，并 `OnStatusChanged`—来处理所有与相关的可能性提供程序开关。

总结

本指南介绍如何获取使用 Android 的位置服务和 Google 位置服务 API 中的浮点混合的位置提供程序的用户的位

置。

相关链接

- [位置（示例）](#)
- [FusedLocationProvider（示例）](#)
- [Google Play 服务](#)
- [条件类](#)
- [LocationManager 类](#)
- [LocationListener 类](#)
- [LocationClient API](#)
- [LocationListener API](#)
- [LocationRequest API](#)

如何使用 Xamarin.Android 使用 Google 地图和位置

2018/10/26 • [Edit Online](#)

本文介绍如何使用 Xamarin.Android 使用地图和位置。它涵盖了从利用内置地图应用程序直接使用 Google 映射 Android API V2 的所有内容。

映射概述

映射技术是对移动设备的通用补充。台式计算机和便携式计算机不往往具有内置的位置感知。但是，移动设备使用此类应用程序查找的设备并显示变化的位置信息。Android 应用了功能强大、内置显示地图使用位置可能在设备上可用的硬件的位置数据的技术。本文介绍一系列在 Xamarin.Android 的地图应用程序需要产品/服务，包括：

- 使用内置地图应用程序以快速添加映射功能。
- 使用地图 API 以控制地图的显示。
- 使用各种技术添加图形叠加。

在本部分中的主题介绍各种映射功能。首先，它们说明了如何利用 Android 的内置地图应用程序以及如何显示的位置的街道全景视图。然后他们将讨论如何使用地图 API 来将直接在应用程序中，介绍如何控制位置和一个代码图，显示这两个映射功能合并，以及如何添加图形叠加。

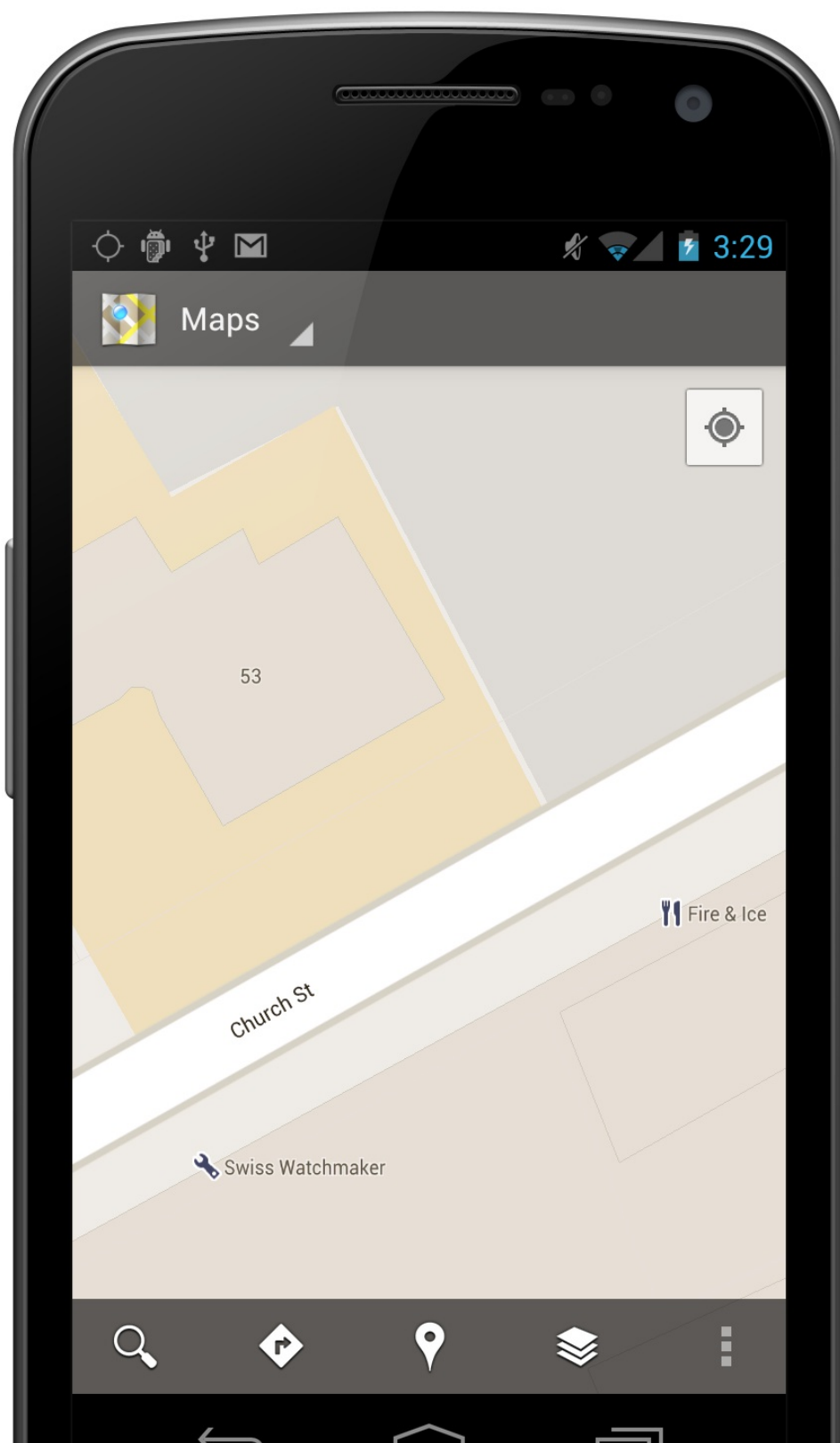
相关链接

- [MapsAndLocationDemo_v3 \(sample\)](#)
- [活动生命周期](#)
- [获取 Google Maps API 密钥](#)
- [意向列表：调用在 Android 设备上的 Google 应用程序](#)
- [位置和地图](#)

启动映射应用程序

2018/10/26 • [Edit Online](#)

有关在 Xamarin.Android 中的映射的最简单方法是利用内置地图应用程序如下所示：





使用地图应用程序时，该映射不会应用程序的一部分。相反，你的应用程序将启动地图应用程序和加载外部地图。下一节探讨如何使用 Xamarin.Android 以启动与上述的映射。

创建意向

使用映射应用程序是只需使用相应的 URI 创建一个意向，将操作设置为 ActionView，并调用 StartActivity 方法。例如，下面的代码会启动以在给定的纬度和经度为中心的映射应用程序：

```
var geoUri = Android.Net.Uri.Parse ("geo:42.374260,-71.120824");
var mapIntent = new Intent (Intent.ActionView, geoUri);
StartActivity (mapIntent);
```

此代码是只需启动上面的屏幕截图中所示的映射。除了指定纬度和经度，映射的 URI 方案支持多个其他选项。

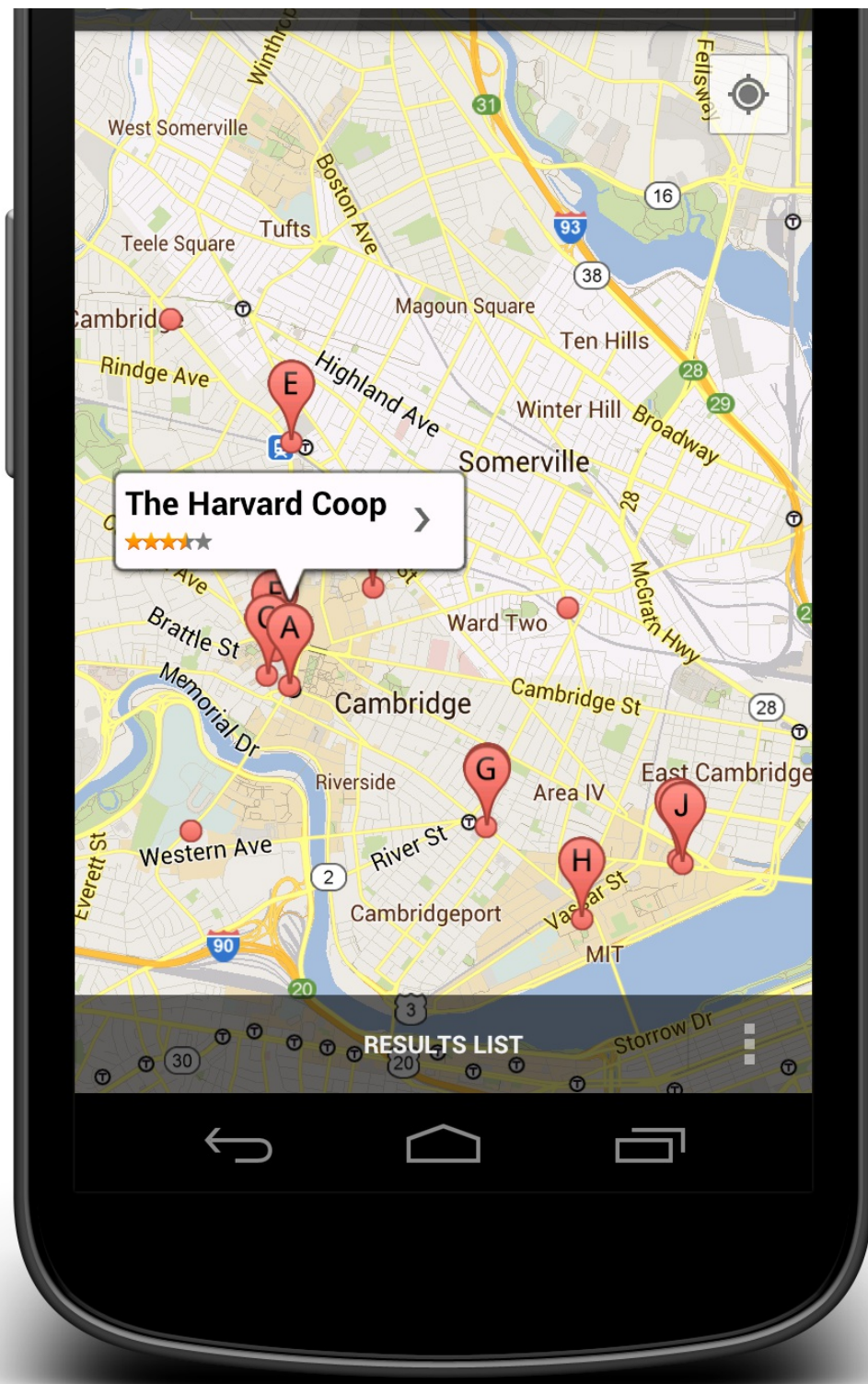
异地 URI 方案

上面的代码使用异地方案来创建一个 URI。此 URI 方案支持几种格式，如下所示：

- `geo:latitude,longitude` – 将打开地图应用程序以 lat/lon 中心。
- `geo:latitude,longitude?z=zoom` – 将打开的地图应用程序在 lat/lon 居中和缩放后为指定的级别。缩放级别范围可以从 1 到 23: 1 显示整个地球和 23 是最接近的缩放级别。
- `geo:0,0?q=my+street+address` – 将打开到街道地址的位置的地图应用程序。
- `geo:0,0?q=business+near+city` – 将打开地图应用程序并显示带批注的搜索结果。

执行查询（即街道地址或搜索词）的 uri 的版本使用 Google 的地理编码器也服务检索然后地图显示的位置。例如，URI `geo:0,0?q=coop+Cambridge` 导致映射如下所示：





有关异地 URI 方案的详细信息，请参阅[在地图上显示位置](#)。

街道视图

除了异地方案中，Android 还支持从意向加载街道的视图。从 Xamarin.Android 启动街道视图应用程序的示例如下所示：



4:42

53 Church Street



Google

© 2011 Google



若要启动的街道视图，只需使用 `google.streetview` URI 方案，如下面的代码中所示：

```
var streetViewUri = Android.Net.Uri.Parse (
    "google.streetview:cbll=42.374260,-71.120824&cbp=1,90,,0,1.0&mz=20");
var streetViewIntent = new Intent (Intent.ActionView, streetViewUri);
StartActivity (streetViewIntent);
```

使用上面的 `google.streetview` URI 方案采用以下形式：

```
google.streetview:cbll=lat,lng&cbp=1,yaw,,pitch,zoom&mz=mapZoom
```

正如您所看到的有几个参数支持，如下所示：

- `lat` – 要在街道视图中显示的位置的纬度。
- `lng` – 要在街道视图中显示的位置的经度。
- `pitch` – 垂直向上街道视图全景图，从中心中度 90 度是直接向下和-90 度为单位的角度。
- `yaw` – 由北的度数的街道视图全景视图中心顺时针测量。
- `zoom` – 放大乘数街道视图全景，其中 1.0 = 正常缩放，2.0 = 缩放的 2 x 3.0 = 缩放的 4 x，等等。
- `mz` – 从街道视图转到映射应用程序时，将使用地图缩放级别。

使用内置映射应用程序或街道视图快速添加映射支持的简单方法。但是，Android 的地图 API 提供更好地控制映射体验。

应用程序中使用 Google 地图 API

2018/11/6 • [Edit Online](#)

使用地图应用程序是很好，但有时您想要直接在您的应用程序中包括地图。除了内置映射应用程序，还提供了 Google [适用于 Android 的本机映射 API](#)。地图 API 是适用于想要维护的映射体验的更好地控制的情况。可以使用地图 API 的操作包括：

- 以编程方式更改映射的角度来看。
- 添加和自定义标记。
- 批注具有叠加的映射。

现已弃用的 Google Maps Android API v1 与 Google Maps Android API v2 是一部分 [Google Play Services](#)。可以使用 Google Maps Android API 之前，Xamarin.Android 应用程序必须满足一些强制性先决条件。

Google 地图 API 的先决条件

需要之前可以使用地图 API 中，执行几个步骤包括：

- [获取地图 API 密钥](#)
- [安装 Google Play Services SDK](#)
- [从 NuGet 安装 Xamarin.GooglePlayServices.Maps 包](#)
- [指定所需的权限](#)
- (可选) [使用 Google Api 创建一个仿真程序](#)

获取 Google 地图 API 密钥

第一步是获取 Google Maps API 密钥（请注意，不能重复使用传统的 Google Maps v1 API 的 API 密钥）。有关如何获取和使用 Xamarin.Android 使用的 API 密钥的信息，请参阅[获取 Google 地图 API 密钥](#)。

安装 Google Play Services SDK

Google Play Services 是一种技术将来自 Google，Android 应用程序可以充分利用各种 Google 功能，例如 Google +、应用内计费 and 映射。这些功能都在 Android 设备上可以访问作为后台服务，它包含在 [Google Play Services APK](#)。

与 Google Play Services 进行交互的 android 应用程序通过 Google Play 服务客户端库。此库包含的接口和类，如映射单个服务。下图显示了 Android 应用程序和 Google Play Services 之间的关系：



Android 地图 API 提供 Google Play 服务的一部分。Xamarin.Android 应用程序可以使用地图 API 之前，必须使用安装 Google Play Services SDK [Android SDK 管理器](#)。以下屏幕截图显示的位置在 Android SDK 管理器中找不到 Google Play 服务客户端：

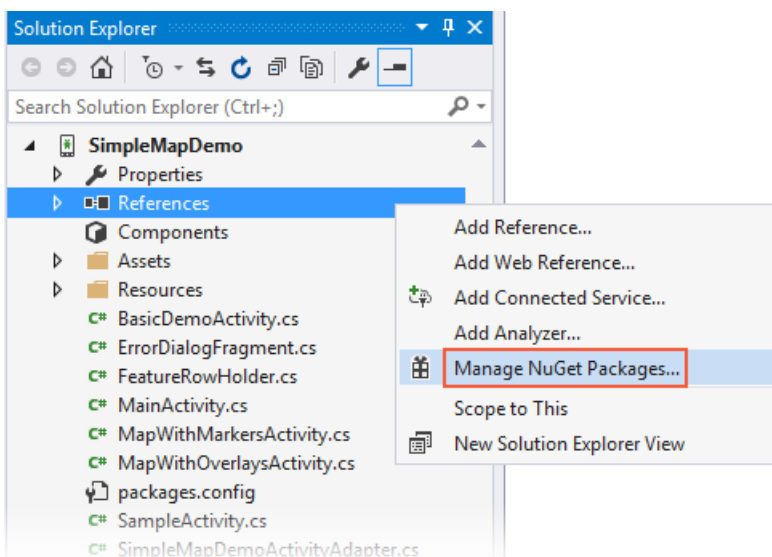
| | | | |
|--|-----|---|--|
| Extras | | | |
| <input type="checkbox"/> + Android Support Repository | 47 | <input checked="" type="checkbox"/> Installed | |
| <input type="checkbox"/> + Android Auto Desktop Head Unit emulator | 1.1 | <input type="checkbox"/> Not installed | |
| <input type="checkbox"/> + Google Play services | 39 | <input checked="" type="checkbox"/> Installed | |
| <input type="checkbox"/> + Google Repository | 46 | <input checked="" type="checkbox"/> Installed | |
| <input type="checkbox"/> + Google Play APK Expansion library | 1 | <input type="checkbox"/> Not installed | |
| <input type="checkbox"/> + Google Play Licensing Library | 1 | <input type="checkbox"/> Not installed | |
| <input type="checkbox"/> + Google Play Billing Library | 5 | <input type="checkbox"/> Not installed | |
| <input type="checkbox"/> + Android Auto API Simulators | 1 | <input type="checkbox"/> Not installed | |
| <input type="checkbox"/> + Google USB Driver | 11 | <input checked="" type="checkbox"/> Installed | |
| <input type="checkbox"/> + Google Web Driver | 2 | <input type="checkbox"/> Not installed | |

NOTE

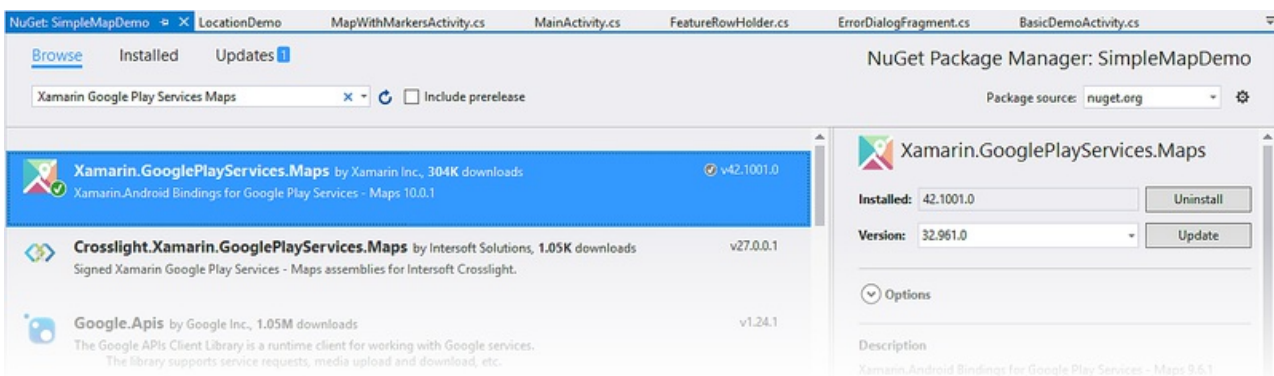
Google Play services APK 是可能不会显示在所有设备上的许可的产品。如果未安装，然后 Google Maps 不会在该设备。

从 NuGet 安装 Xamarin.GooglePlayServices.Maps 包

[Xamarin.GooglePlayServices.Maps](#) 包包含 Google Play Services 地图 API 的 Xamarin.Android 绑定。若要添加 Google Play 服务映射包，请右键单击引用在解决方案资源管理器中单击项目文件夹管理 **NuGet 包...**：



这将打开 **NuGet 包管理器**。单击浏览并输入 **Xamarin Google Play Services** 地图搜索字段中。选择 **Xamarin.GooglePlayServices.Maps** 然后单击 **安装**。(如果以前已安装此包，请单击 **更新**。):



请注意，还会安装以下依赖项包：

- **Xamarin.GooglePlayServices.Base**
- **Xamarin.GooglePlayServices.Basement**

- **Xamarin.GooglePlayServices.Tasks**

指定所需的权限

应用程序必须确定要使用 Google Maps API 的硬件和权限要求。Google Play Services SDK 将自动授予一些权限并不需要开发人员显式将其添加到**AndroidManifest.XML**:

- 对网络状态的访问-地图 API 必须能够检查是否它可以下载地图图块。
- **Internet** 访问-都下载地图图块和与 API 访问 Google Play 服务器通信所需访问 Internet。

必须在指定的以下权限和功能**AndroidManifest.XML** Google Maps Android api:

- **OpenGL ES v2** -应用程序必须声明 OpenGL ES v2 的要求。
- **Google 地图 API 密钥** - API 密钥用于确认注册并有权使用 Google Play Services 应用程序。请参阅[获取 Google 地图 API 密钥](#)有关此密钥的详细信息。
- 请求的旧的 **Apache HTTP 客户端**-面向 Android 9.0 (API 级别 28) 的应用, 或更高版本必须指定旧 Apache HTTP 客户端是一个可选的库使用。
- 对基于 **Google Web 服务**的访问-应用程序需要有权访问 Google 的返回 Android 地图 API 的 web 服务。
- **Google Play 服务通知的权限**-必须授予应用程序从 Google Play 服务接收远程通知的权限。
- **访问位置提供程序**-这些是可选的权限。将允许 `GoogleMap` 类, 以在地图上显示设备的位置。

NOTE

非常旧版本的 Google Play SDK 所需的应用以请求 `WRITE_EXTERNAL_STORAGE` 权限。此要求是不再需要使用 Google Play 服务的最新的 Xamarin 绑定。

以下代码片段示范了必须添加到的设置**AndroidManifest.XML**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionName="4.5"
package="com.xamarin.docs.android.mapsandlocationdemo2" android:versionCode="6">
    <uses-sdk android:minSdkVersion="23" android:targetSdkVersion="28" />

    <!-- Google Maps for Android v2 requires OpenGL ES v2 -->
    <uses-feature android:glEsVersion="0x00020000" android:required="true" />

    <!-- Necessary for apps that target Android 9.0 or higher -->
    <uses-library android:name="org.apache.http.legacy" android:required="false" />

    <!-- Permission to receive remote notifications from Google Play Services -->
    <!-- Notice here that we have the package name of our application as a prefix on the permissions. -->
    <uses-permission android:name="<PACKAGE_NAME>.permission.MAPS_RECEIVE" />
    <permission android:name="<PACKAGE_NAME>.permission.MAPS_RECEIVE" android:protectionLevel="signature" />

    <!-- These are optional, but recommended. They will allow Maps to use the My Location provider. -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application android:label="@string/app_name">
        <!-- Put your Google Maps V2 API Key here. -->
        <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="YOUR_API_KEY" />
        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />
    </application>
</manifest>
```


除了请求的权限`AndroidManifest.XML`, 应用还必须执行的运行时权限检查 `ACCESS_COARSE_LOCATION` 和 `ACCESS_FINE_LOCATION` 权限。请参阅[Xamarin.Android 权限指南](#)以获取有关执行运行时权限检查的详细信息。

使用 Google Api 创建一个仿真程序

在物理 Android 设备与 Google Play 服务未安装, 就可以创建用于开发的仿真程序映像。有关详细信息请参阅[设备管理器](#)。

GoogleMap 类

一旦满足先决条件, 就可以开始开发应用程序并使用 Android 地图 API。`GoogleMap`类是主要的 Xamarin.Android 应用程序将用于显示和适用于 Android 与 Google 地图进行交互的 API。此类具有下列职责:

- 与 Google Play 服务授权与 Google web 服务应用程序进行交互。
- 下载、缓存和显示地图图块。
- 显示 UI 控件, 如平移和缩放到用户。
- 在地图上绘制标记和几何形状。

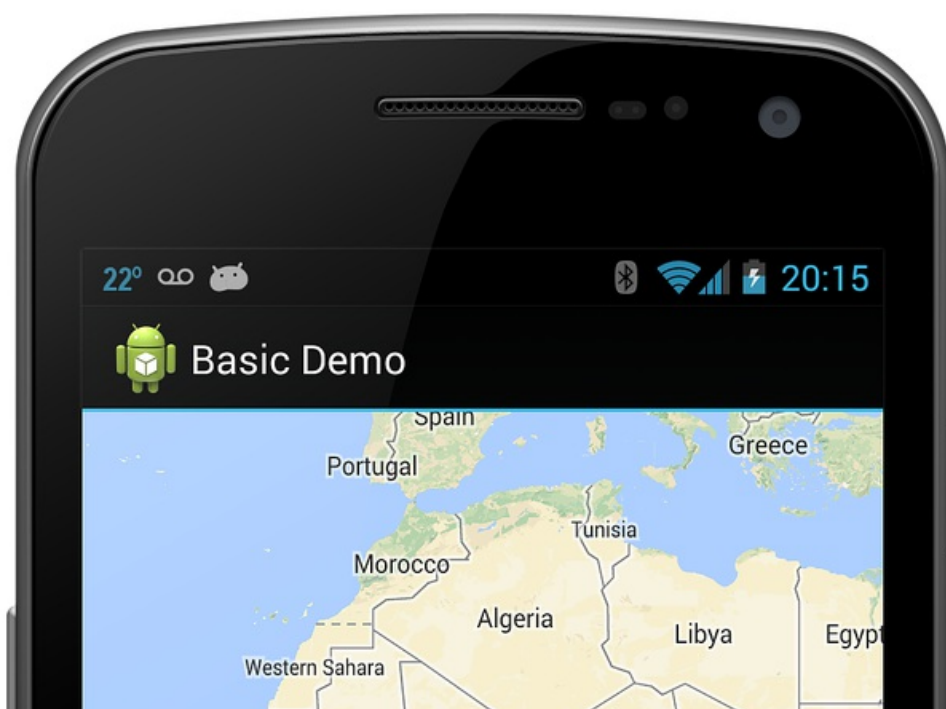
`GoogleMap` 添加到两种方式之一中的活动:

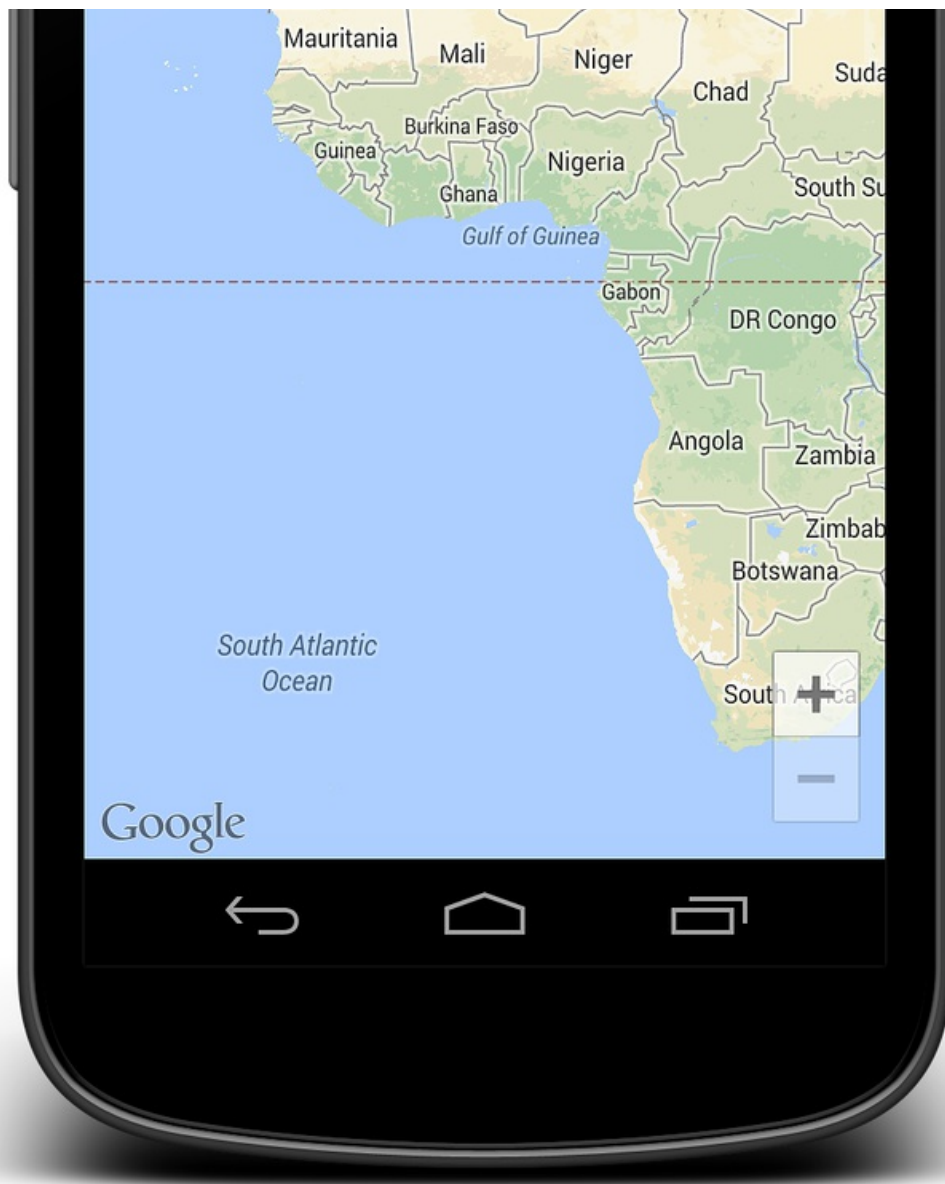
- **MapFragment** - `MapFragment`充当主机是专用片段 `GoogleMap` 对象。`MapFragment` 需要 Android API 级别 12 或更高版本。可以使用较旧版本的 Android `SupportMapFragment`。本指南将重点介绍使用 `MapFragment` 类。
- **MapView** - `MapView`是专用的视图子类, 可充当主机 `GoogleMap` 对象。此类的用户必须转发到的活动生命周期方法的所有 `MapView` 类。

每个容器公开 `Map` 返回的实例的属性 `GoogleMap`。首选项应授予给 `MapFragment`类, 如它是一种更简单的 API, 减少了开发人员必须手动实现的量样板代码。

向活动添加 MapFragment

下面的屏幕截图是一个简单的示例 `MapFragment` :





与其他片段类相似，有两种方法来添加 `MapFragment` 到活动：

- **以声明方式** - `MapFragment` 活动可以通过 XML 布局文件中添加。以下 XML 代码片段示范如何使用 `fragment` 元素：

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.MapFragment" />
```

- **以编程方式** - `MapFragment` 可以以编程方式使用实例化 `MapFragment.newInstance()` 方法，然后添加到活动。此代码段显示了实例化的最简单方法 `MapFragment` 对象，并向活动中添加：

```
var mapFrag = MapFragment.newInstance();
activity.FragmentManager.beginTransaction()
    .Add(Resource.Id.map_container, mapFrag, "map_fragment")
    .Commit();
```

可以配置 `MapFragment` 对象通过传递 `GoogleMapOptions` 对象传递给 `newInstance`。此部分中讨论 `GoogleMap` 属性本指南中稍后显示。

`MapFragment.getMapAsync` 方法用来初始化 `GoogleMap` 的片段托管并获取对由托管的映射对象的引用 `MapFragment`。此方法采用一个对象，实现 `IONMapReadyCallback` 接口。

此接口具有单个方法 `IONMapReadyCallback.onMapReady(MapFragment map)` 很可能要与之交互的应用时，将调用的 `GoogleMap` 对象。以下代码片段演示如何对 Android 活动可以将其初始化 `MapFragment` 并实现 `IONMapReadyCallback` 接口：

```
public class MapWithMarkersActivity : AppCompatActivity, IONMapReadyCallback
{
    protected override void onCreate(Bundle bundle)
    {
        base.onCreate(bundle);
        setContentView(Resource.Layout.MapLayout);

        var mapFragment = (MapFragment) FragmentManager.findFragmentById(Resource.Id.map);
        mapFragment.getMapAsync(this);

        // remainder of code omitted
    }

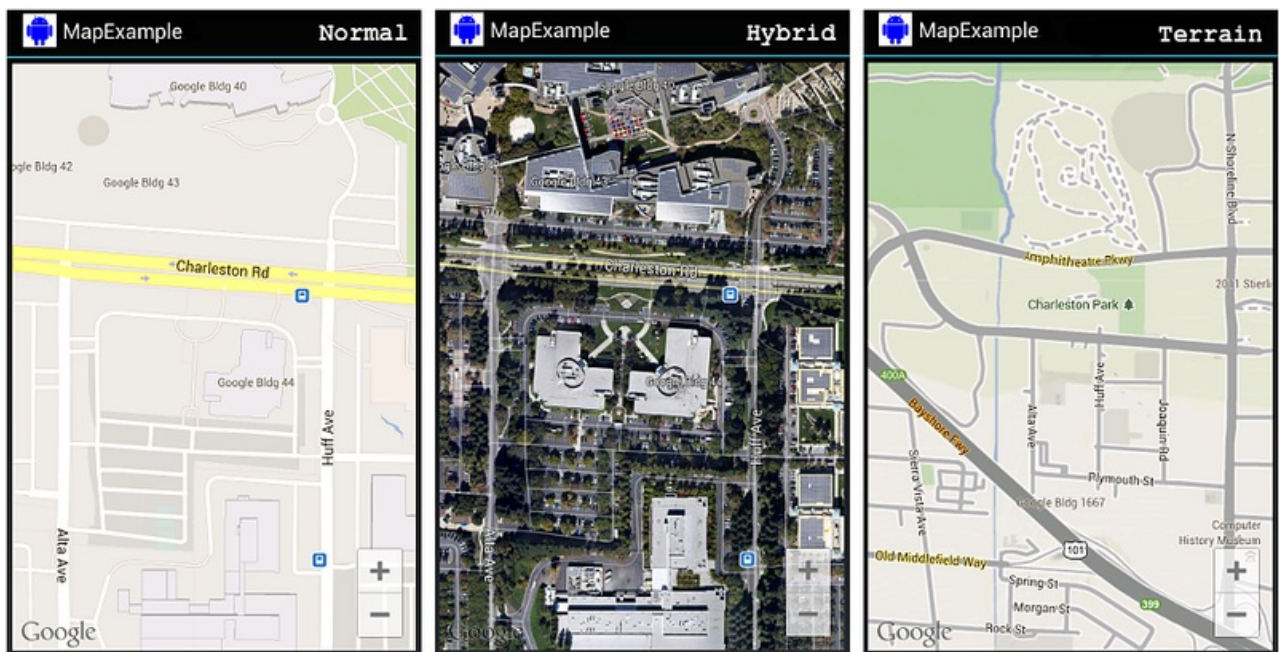
    public void onMapReady(GoogleMap map)
    {
        // Do something with the map, i.e. add markers, move to a specific location, etc.
    }
}
```

将类型映射

有五种不同类型的映射从 Google Maps API 可用：

- **正常**-这是默认映射类型。它显示道路和重要自然功能，此外还（如建筑物和桥）感兴趣的某些 artificial 点。
- **附属**-此图显示了附属摄影。
- **混合**-此映射显示附属摄影和道路映射。
- **地形**-这主要显示了与部分道路的地形特征。
- **无**-此映射将不会加载任何磁贴，呈现为一个空网格。

下图显示了三个不同类型的地图，从左到右（正常、混合，地形）：



`GoogleMap.MapType` 属性用于设置或更改显示哪种类型的映射。下面的代码段演示如何显示附属映射。

```
public void OnMapReady(GoogleMap map)
{
    map.MapType = GoogleMap.MapTypeHybrid;
}
```

GoogleMap 属性

`GoogleMap` 定义多个可以控制功能和地图的外观的属性。配置的初始状态的一种方法 `GoogleMap` 是将传递 `GoogleMapOptions` 对象创建时 `MapFragment`。下面的代码段是使用的一个示例 `GoogleMapOptions` 对象创建时 `MapFragment`：

```
GoogleMapOptions mapOptions = new GoogleMapOptions()
    .InvokeMapType(GoogleMap.MapTypeSatellite)
    .InvokeZoomControlsEnabled(false)
    .InvokeCompassEnabled(true);

FragmentManager.BeginTransaction();
mapFragment = MapFragment.NewInstance(mapOptions);
fragTx.Add(Resource.Id.map, mapFragment, "map");
fragTx.Commit();
```

配置的其他方式 `GoogleMap` 通过在操作属性是 `UiSettings` 的 `map` 对象。下一步的代码示例演示如何配置 `GoogleMap` 显示缩放控件和指南针：

```
public void OnMapReady(GoogleMap map)
{
    map.UiSettings.ZoomControlsEnabled = true;
    map.UiSettings.CompassEnabled = true;
}
```

与 GoogleMap 交互

Android 地图 API 提供的 Api, 使活动更改角度来看, 添加标记、放置自定义覆盖或绘制几何形状。本部分将讨论如何完成这些任务在 `Xamarin.Android` 中的一些。

更改视区

映射是在屏幕上, 根据 Mercator 投影为平面平面来建模。地图视图是**照相机**直接向下查找此平面上。可以通过更改位置、缩放、倾斜, 并影响控制照相机的位置。**CameraUpdate**类用于移动照相机位置。**CameraUpdate**不直接实例化对象, 而是映射 API 提供**CameraUpdateFactory**类。

一次**CameraUpdate**创建对象, 它作为参数传递到**GoogleMap.MoveCamera**或**GoogleMap.AnimateCamera**方法。**MoveCamera**方法更新时立即映射**AnimateCamera**方法提供平滑的动画转换。

此代码段是如何使用一个简单示例**CameraUpdateFactory**若要创建**CameraUpdate**的递增一个缩放级别的地图的缩放级别:

```
MapFragment mapFrag = (MapFragment) FragmentManager.FindFragmentById(Resource.Id.my_mapfragment_container);
mapFrag.GetMapAsync(this);
...

public void OnMapReady(GoogleMap map)
{
    map.MoveCamera(CameraUpdateFactory.ZoomIn());
}
```

地图 API 提供了**CameraPosition**其中将聚合所有照相机的位置的可能值。此类的实例可以提供给**CameraUpdateFactory.NewCameraPosition**方法将返回**CameraUpdate**对象。地图 API 还包括**CameraPosition.Builder** fluent API 提供用于创建类**CameraPosition**对象。下面的代码段显示了创建的示例**CameraUpdate**从**CameraPosition**并使用该更改照相机的位置上**GoogleMap**:

```
public void OnMapReady(GoogleMap map)
{
    LatLng location = new LatLng(50.897778, 3.013333);

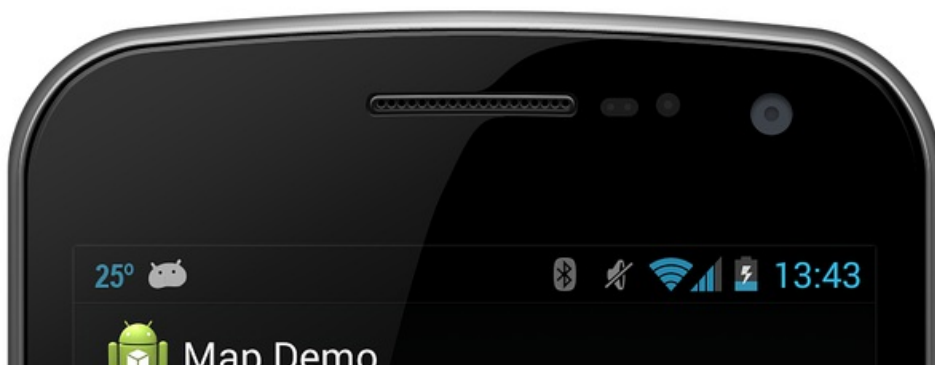
    CameraPosition.Builder builder = CameraPosition.InvokeBuilder();
    builder.Target(location);
    builder.Zoom(18);
    builder.Bearing(155);
    builder.Tilt(65);

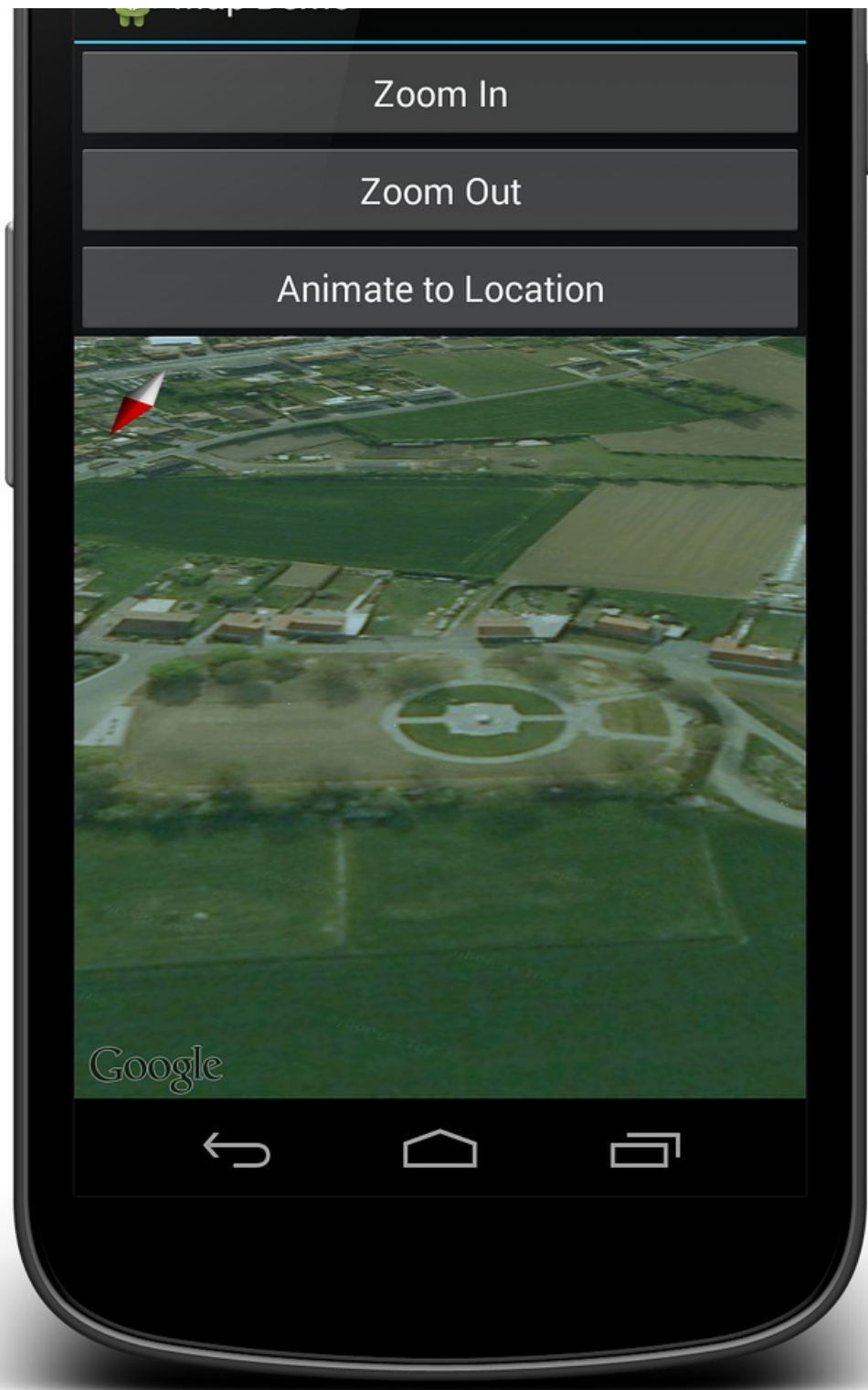
    CameraPosition cameraPosition = builder.Build();

    CameraUpdate cameraUpdate = CameraUpdateFactory.NewCameraPosition(cameraPosition);

    map.MoveCamera(cameraUpdate);
}
```

在上一代码段中, 由表示在地图上的特定位置**LatLng**类。缩放级别设置为 18 这是由 Google 地图缩放任意度量值。影响是指南针的度量值从北部顺时针旋转。倾斜属性控制的查看角度, 它指定从垂直 25 度角。以下屏幕截图显示**GoogleMap**后执行前面的代码:





在地图上绘制

Android 地图 API 提供了 API 的用于在地图上绘制的以下项：

- **标记**-这些是用于标识在地图上的单个位置的特殊图标。
- **覆盖**-这是可以使用用于标识某个集合的位置或在地图上的区域的图像。
- **线条、多边形和圆形**-这些是允许活动向映射添加形状的 Api。

标记

地图 API 提供了[标记](#)类封装所有有关在地图上的单个位置的数据。默认情况下，标记类使用 Google 地图提供的一个标准图标。可以自定义标记的外观并响应用户单击它。

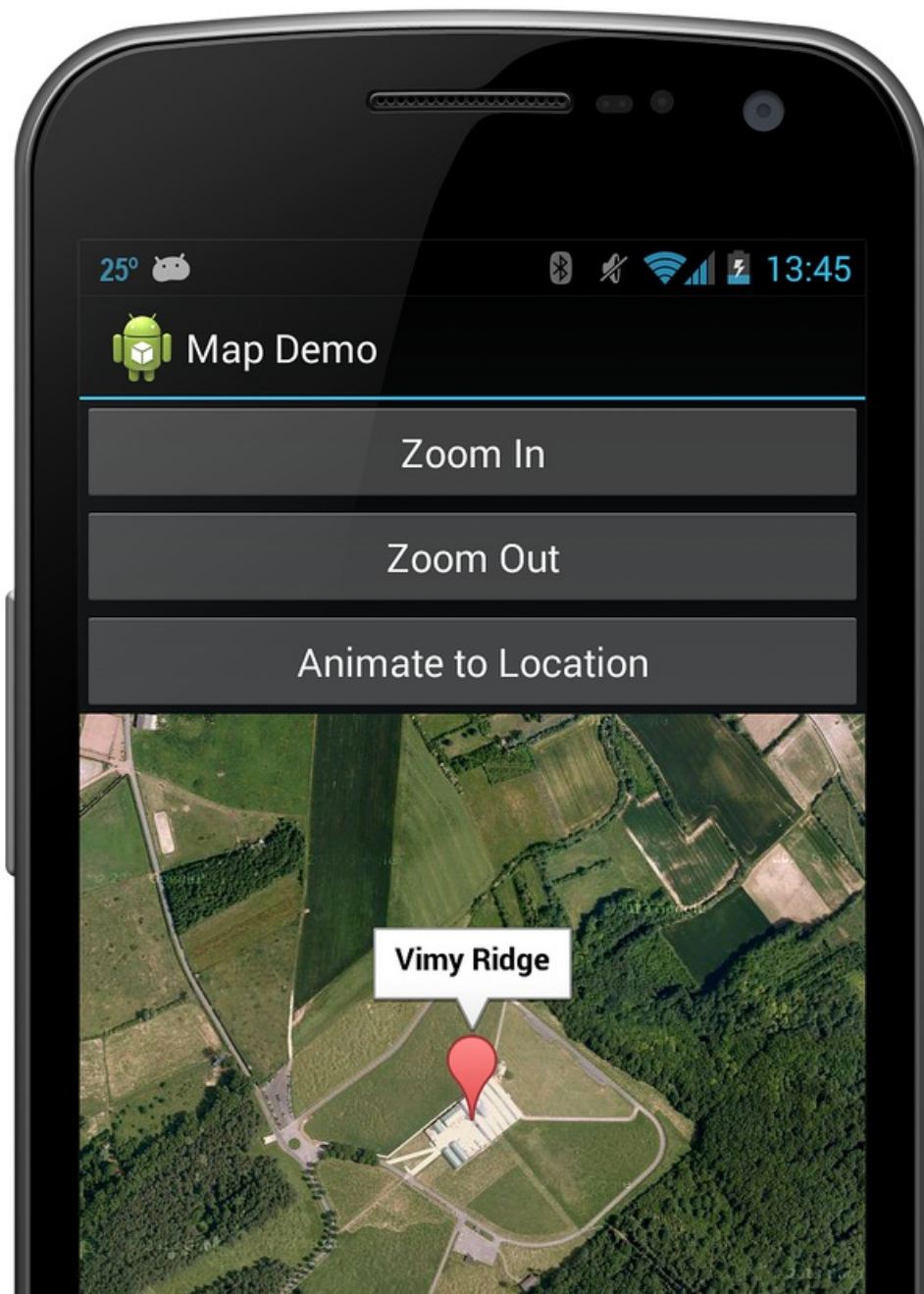
添加一个标记

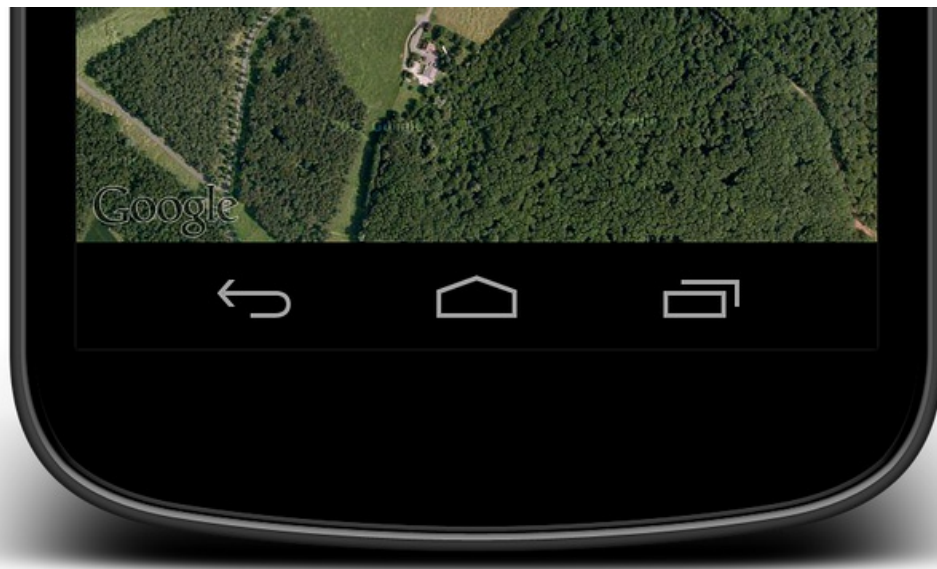
若要向映射添加一个标记，有必要创建一个新[MarkerOptions](#)对象，然后调用[AddMarker](#)方法 `GoogleMap` 实例。此方法将返回[标记](#)对象。

```
public void OnMapReady(GoogleMap map)
{
    MarkerOptions markerOpt1 = new MarkerOptions();
    markerOpt1.SetPosition(new LatLng(50.379444, 2.773611));
    markerOpt1.SetTitle("Vimy Ridge");

    map.AddMarker(markerOpt1);
}
```

标记的标题将显示在[信息窗口](#)时在用户点击该标记。下面的屏幕截图显示了此标记如下所示：





自定义标记

可以自定义图标标记由调用 `MarkerOptions.InvokeIcon` 方法将标记添加到代码图时。此方法采用 `BitmapDescriptor` 对象，其中包含要呈现图标所需的数据。`BitmapDescriptorFactory` 类提供了一些帮助器方法来简化创建 `BitmapDescriptor`。以下列表介绍了其中的某些方法：

- `DefaultMarker(float colour)` – 使用默认 Google 地图标记，但更改颜色。
- `FromAsset(string assetName)` – 从资产文件夹中的指定文件中使用自定义图标。
- `FromBitmap(Bitmap image)` – 指定的位图用作图标。
- `FromFile(string fileName)` – 从指定路径处的文件创建自定义图标。
- `FromResource(int resourceId)` – 从指定的资源创建自定义图标。

下面的代码段显示了创建青色彩色的默认标记的示例：

```
public void OnMapReady(GoogleMap map)
{
    MarkerOptions markerOpt1 = new MarkerOptions();
    markerOpt1.SetPosition(new LatLng(50.379444, 2.773611));
    markerOpt1.SetTitle("Vimy Ridge");

    var bmDescriptor = BitmapDescriptorFactory.DefaultMarker (BitmapDescriptorFactory.HueCyan);
    markerOpt1.InvokeIcon(bmDescriptor);

    map.AddMarker(markerOpt1);
}
```

信息 windows

信息 windows 是该弹出窗口，用于向用户显示的信息，当用户点击特定标记的特殊窗口。默认情况下信息窗口会显示标记的标题的内容。如果未分配有标题，将不显示任何信息窗口。只能有一个信息窗口中显示的一次。

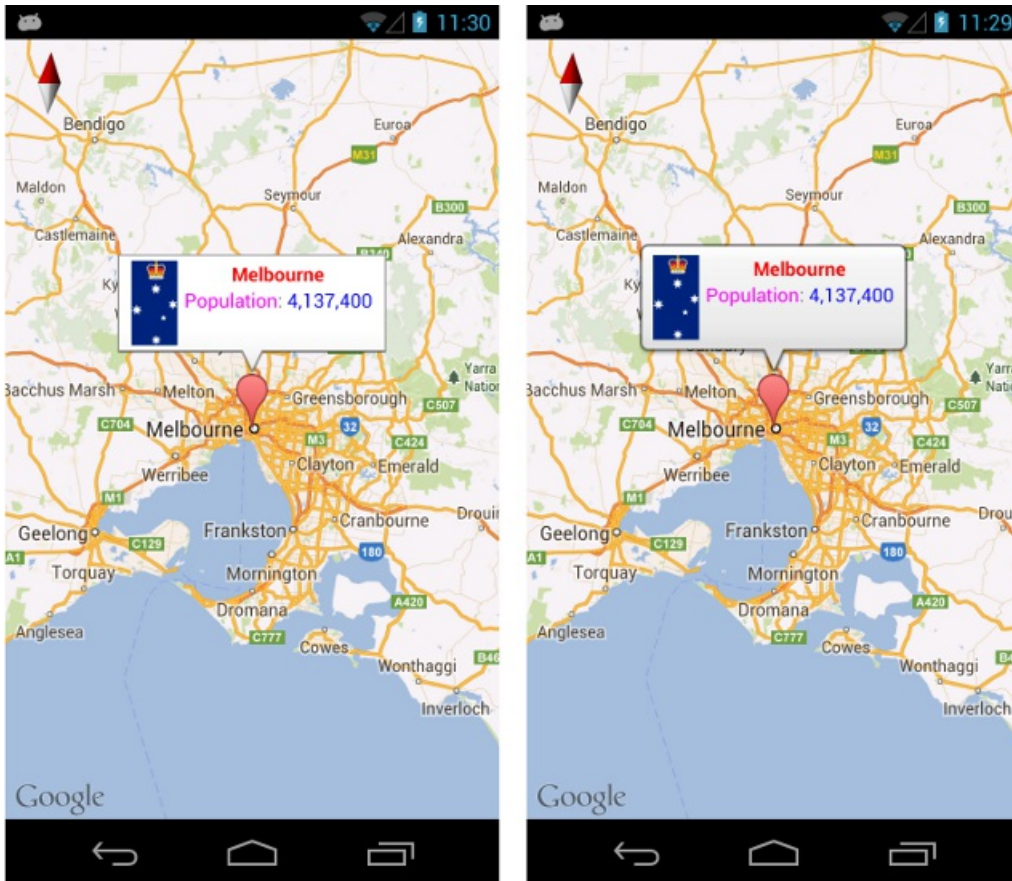
可以通过实现自定义信息窗口 `GoogleMap.InfoWindowAdapter` 接口。此接口上有两个重要方法：

- `public View GetInfoWindow(Marker marker)` – 若要自定义的信息窗口获得一个标记，调用此方法。如果它返回 `null`，则将使用默认的窗口呈现。如果此方法返回的视图，该视图将被放置在信息窗口框架。

- `public View GetInfoContents(Marker marker)` – 将仅调用此方法, 如果 `GetInfoWindow` 返回 `null`。此方法可返回 `null` 时的信息窗口内容的默认呈现时要使用的值。否则, 此方法应返回具有信息窗口的内容的视图。

信息窗口不是实时视图-Android 改为将视图转换为静态位图和显示的图像。这意味着, 信息窗口无法响应的任何触摸事件或手势, 也不将其自动进行自我更新。若要更新的信息窗口, 为调用所需 `GoogleMap.ShowInfoWindow` 方法。

下图显示了某些自定义的信息窗口的一些示例。在左侧图像都有其自定义, 而右侧图像具有其窗口的内容和使用圆角自定义的内容:



GroundOverlays

与不同的标记, 标识在地图上的特定位置, 请 `GroundOverlay` 是用于标识的位置或在地图上的某个区域集合的映像。

添加 `GroundOverlay`

向地图添加地面覆盖层是类似于向地图添加一个标记。首先, `GroundOverlayOptions` 创建对象。此对象然后作为参数传递 `GoogleMap.AddGroundOverlay` 方法, 它将返回 `GroundOverlay` 对象。此代码片段是向映射添加地面覆盖层的示例:

```
BitmapDescriptor image = BitmapDescriptorFactory.FromResource(Resource.Drawable.polarbear);
GroundOverlayOptions groundOverlayOptions = new GroundOverlayOptions()
    .Position(position, 150, 200)
    .InvokeImage(image);
GroundOverlay myOverlay = googleMap.AddGroundOverlay(groundOverlayOptions);
```

下面的屏幕截图显示在地图上覆盖此层:





有三个简单类型的可以添加到地图的几何图：

- **折线**-这是一系列相互连接的线段。它可以将标记在地图上的路径，或创建几何形状。
- **圆形**-这将在地图上绘制一个圆圈。
- **多边形**-这是一个闭合的形状用于标记在地图上的区域。

折线

一个**折线**是一系列连续 `LatLng` 对象指定每个线段的顶点。通过首先创建创建一条折线 `PolylineOptions` 对象并向其中添加点。 `PolylineOption` 对象然后传递给 `GoogleMap` 对象通过调用 `AddPolyline` 方法。

```
PolylineOption rectOptions = new PolylineOption();
rectOptions.Add(new LatLng(37.35, -122.0));
rectOptions.Add(new LatLng(37.45, -122.0));
rectOptions.Add(new LatLng(37.45, -122.2));
rectOptions.Add(new LatLng(37.35, -122.2));
rectOptions.Add(new LatLng(37.35, -122.0)); // close the polyline - this makes a rectangle.

googleMap.AddPolyline(rectOptions);
```

圆形

由第一个实例化创建圆形 `CircleOption` 对象将指定米在中心和圆的半径。调用在地图上绘制圆形 `GoogleMap.AddCircle`。下面的代码段显示了如何绘制一个圆：

```
CircleOptions circleOptions = new CircleOptions ();
circleOptions.InvokeCenter (new LatLng(37.4, -122.1));
circleOptions.InvokeRadius (1000);

googleMap.AddCircle (circleOptions);
```

多边形

`Polygon` 类似于 `Polyline` s，但未处于打开状态结束。 `Polygon` s 是封闭式的循环，并且具有填充其内部。 `Polygon` 在完全相同的方式创建 `Polyline`，除非 `GoogleMap.AddPolygon` 调用方法。

与不同 `Polyline`、`Polygon` 自结束。将通过关闭多边形 `AddPolygon` 通过绘制一条连接的第一个和最后一个点的直线的方法。下面的代码段将通过相同区域中的上一代码段创建一个实心矩形 `Polyline` 示例。

```
PolygonOptions rectOptions = new PolygonOptions();
rectOptions.Add(new LatLng(37.35, -122.0));
rectOptions.Add(new LatLng(37.45, -122.0));
rectOptions.Add(new LatLng(37.45, -122.2));
rectOptions.Add(new LatLng(37.35, -122.2));
// notice we don't need to close off the polygon

googleMap.AddPolygon(rectOptions);
```

响应用户事件

有三种类型的用户可能有一个带有地图的交互：

- **标记单击**-用户单击某个标记。
- **标记拖动**-用户已长时间-单击 mparger 上
- **信息窗口中单击**-用户单击信息窗口上。

将下面更详细地讨论每个这些事件。

单击事件标记

`MarkerClicked` 用户点击一个标记时引发事件。此事件接受 `GoogleMap.MarkerClickEventArgs` 对象作为参数。此类包含两个属性：

- `GoogleMap.MarkerClickEventArgs.Handled` – 此属性应设置为 `true` 以指示事件处理程序已使用该事件。如果此值设置为 `false` 则默认行为将发生除了事件处理程序的自定义行为。
- `Marker` – 此属性是对引发的标记的引用 `MarkerClick` 事件。

此代码片段演示的示例 `MarkerClick`，会将照相机的位置更改为地图上的新位置：

```
void MapOnMarkerClick(object sender, GoogleMap.MarkerClickEventArgs markerClickEventArgs)
{
    markerClickEventArgs.Handled = true;

    var marker = markerClickEventArgs.Marker;
    if (marker.Id.Equals(gotMauiMarkerId))
    {
        LatLng InMaui = new LatLng(20.72110, -156.44776);

        // Move the camera to look at Maui.
        PositionPolarBearGroundOverlay(InMaui);
        googleMap.AnimateCamera(CameraUpdateFactory.NewLatLngZoom(InMaui, 13));
        gotMauiMarkerId = null;
        polarBearMarker.Remove();
        polarBearMarker = null;
    }
    else
    {
        Toast.MakeText(this, $"You clicked on Marker ID {marker.Id}", ToastLength.Short).Show();
    }
}
```

将事件标记

当用户想要将标记拖动时，引发此事件。默认情况下，标记不是可拖动。一个标记，可以设置为可拖动通过设置 `Marker.Draggable` 属性设置为 `true` 或通过调用 `MarkerOptions.Draggable` 方法替换 `true` 作为参数。

若要将标记拖动，用户必须首先长时间单击标记，那么他们的手指必须保持在代码图上。当用户的手指在屏幕上拖动围绕时，将移动标记。当用户的手指抬起出屏幕时，该标记将保持不变。

以下列表描述可拖动标记，将生成的各种事件：

- `GoogleMap.MarkerDragStart(object sender, GoogleMap.MarkerDragStartEventArgs e)` – 在用户第一次拖动标记时，引发此事件。
- `GoogleMap.MarkerDrag(object sender, GoogleMap.MarkerDragEventArgs e)` – 按标记进行拖动，将引发此事件。
- `GoogleMap.MarkerDragEnd(object sender, GoogleMap.MarkerDragEndEventArgs e)` – 当用户已完成时，将引发此事件将标记。

每个 `EventArgs` 包含一个名为的单个属性 `PO`，它是引用 `Marker` 对象正被拖动。

信息窗口中单击事件

可以一次显示一个信息窗口。当用户单击图中的信息窗口中时，将引发 `map` 对象 `InfoWindowClick` 事件。下面的代码段显示了如何绑定到事件处理程序：

```
public void OnMapReady(GoogleMap map)
{
    map.InfoWindowClick += MapOnInfoWindowClick;
}

private void MapOnInfoWindowClick (object sender, GoogleMap.InfoWindowClickEventArgs e)
{
    Marker myMarker = e.Marker;
    // Do something with marker.
}
```

请注意，信息窗口是一个静态 `View` 其呈现为地图上的图像。任何小组件，如按钮、复选框或放置在信息窗口内的文本视图将静态，并且不能响应任何其整型用户事件。

相关链接

- [SimpleMapDemo](#)
- [Google Play 服务](#)
- [Google 映射 Android API v2](#)
- [Google Play 服务的 APK](#)
- [获取 Google Maps API 密钥](#)
- [使用库](#)
- [使用功能](#)

获取 Google Maps API 密钥

2018/10/26 • [Edit Online](#)

若要在 Android 中使用 Google 地图功能，您需要注册为 Google 地图 API 密钥。执行此操作时，你只需将应用程序中看到而不是映射空白网格。必须获取 Google Maps Android API v2 密钥-从较旧的 Google Maps Android API 密钥 v1 的密钥将不可用。

获取地图 API v2 密钥涉及以下步骤：

1. 检索用于对应用程序签名的密钥存储的 sha-1 指纹。
2. 在 Google Api 控制台中创建一个项目。
3. 获取 API 密钥。

获取签名的密钥指纹

若要请求来自 Google 地图 API 密钥，您需要知道用于签名应用程序的密钥存储的 sha-1 指纹。通常情况下，这意味着您需要确定调试密钥存储，sha-1 指纹，然后 sha-1 指纹为用于发布应用程序进行签名的密钥存储。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

默认情况下使用的应用程序可以是 Xamarin.Android 的调试版本进行签名的密钥存储在以下位置找到：

C:\用户\[USERNAME]\AppData\本地\Xamarin\适用于 Android Mono\debug.keystore

可通过从 JDK 运行 `keytool` 命令来获取有关密钥存储的信息。此工具通常 Java 的 bin 目录中找到：

C:\Program Files (x86)\Java\jdk [版本]\bin\keytool.exe

运行 keytool 使用以下命令（使用如上所示的文件路径）：

```
keytool -list -v -keystore [STORE FILENAME] -alias [KEY NAME] -storepass [STORE PASSWORD] -keypass [KEY PASSWORD]
```

Debug.keystore 示例

对于默认调试密钥（这自动为您创建用于调试），使用以下命令：

- [Visual Studio](#)
- [Visual Studio for Mac](#)

```
keytool.exe -list -v -keystore "C:\Users\[USERNAME]\AppData\Local\Xamarin\Mono for Android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

生产密钥

当将应用部署到 Google Play，它必须是[使用私钥签名](#)。`keytool` 将需要使用专用密钥的详细信息和生成 sha-1 指纹用于创建生产 Google 地图 API 密钥运行。请记住更新 **AndroidManifest.xml** 部署前的正确 Google Maps API 密钥的文件。

Keytool 输出

您应看到类似于以下输出控制台窗口中的内容：

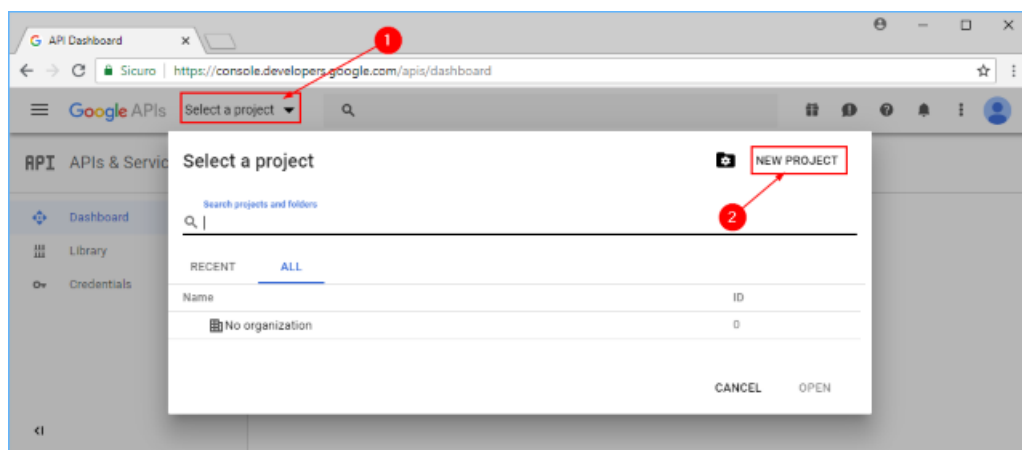
```
Alias name: androiddebugkey
Creation date: Jan 01, 2016
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4aa9b300
Valid from: Mon Jan 01 08:04:04 UTC 2013 until: Mon Jan 01 18:04:04 PST 2033
Certificate fingerprints:
    MD5: AE:9F:95:D0:A6:86:89:BC:A8:70:BA:34:FF:6A:AC:F9
    SHA1: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75
Signature algorithm name: SHA1withRSA
Version: 3
```

将使用 sha-1 指纹 (之后列出**SHA1**) 在本指南后面。

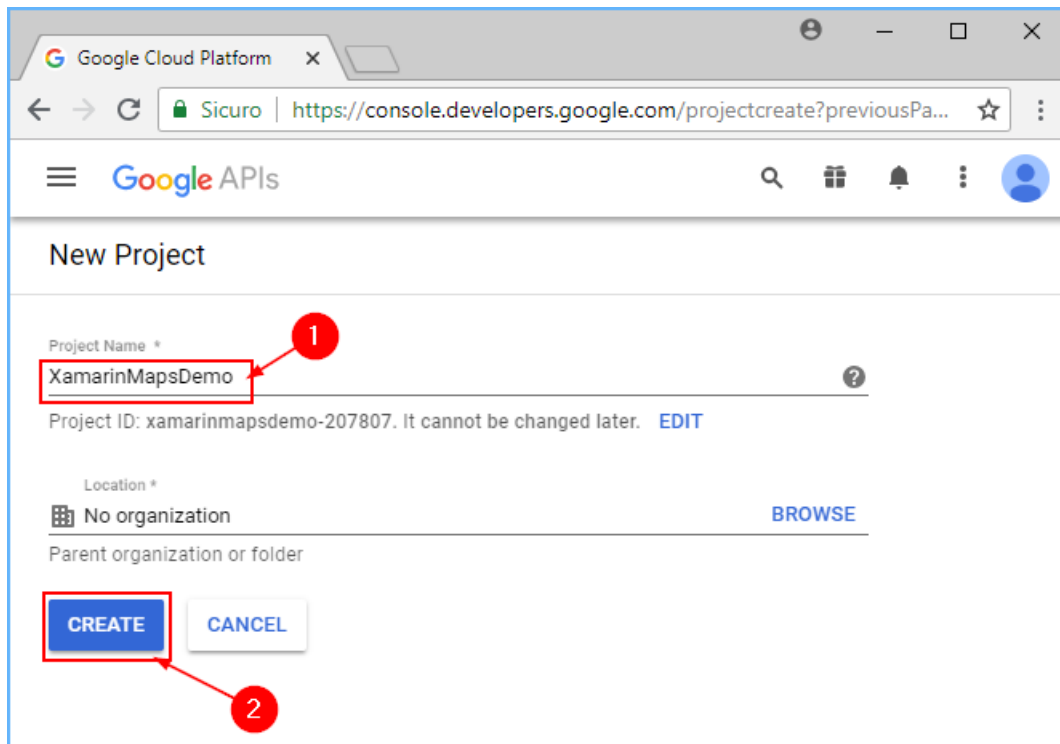
创建 API 项目

检索签名的密钥存储的 sha-1 指纹后, 有必要在 Google Api 控制台中创建一个新的项目 (或向现有项目添加 Google Maps Android API v2 服务)。

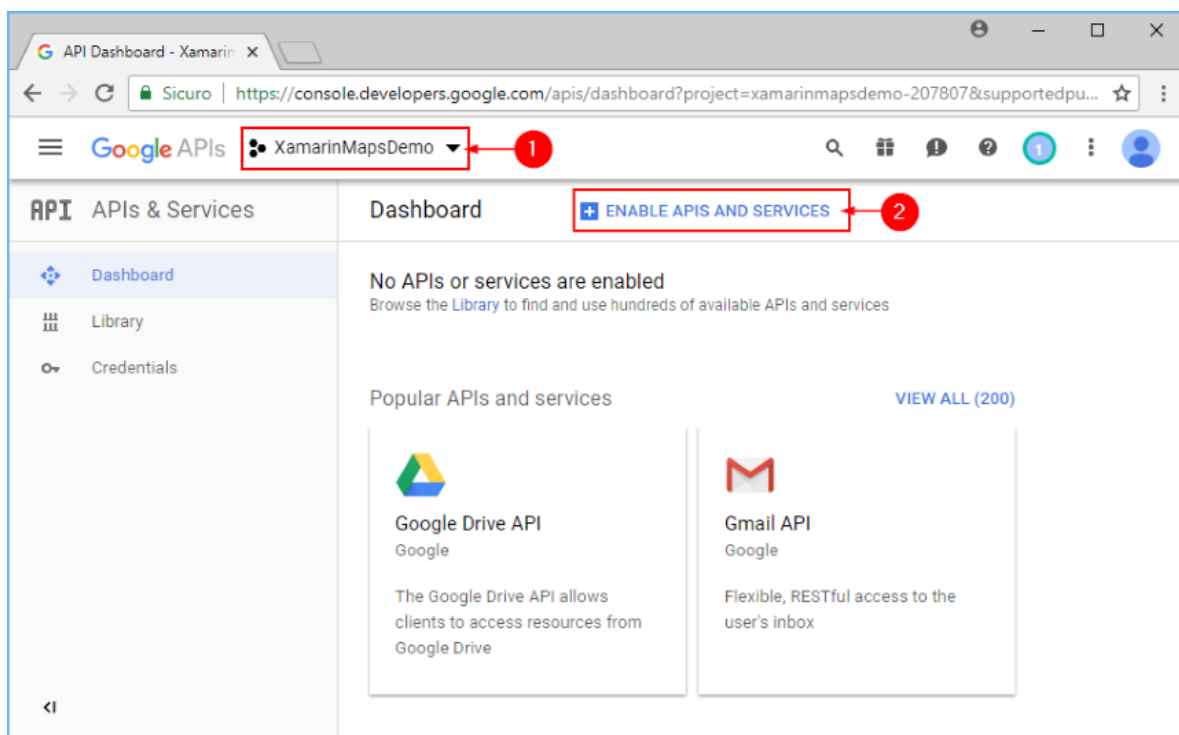
1. 在浏览器中, 导航到[Google 开发人员控制台 API 和服务仪表板](https://console.developers.google.com/apis/dashboard)然后单击选择一个项目。单击项目名称, 或通过单击创建新新的项目:



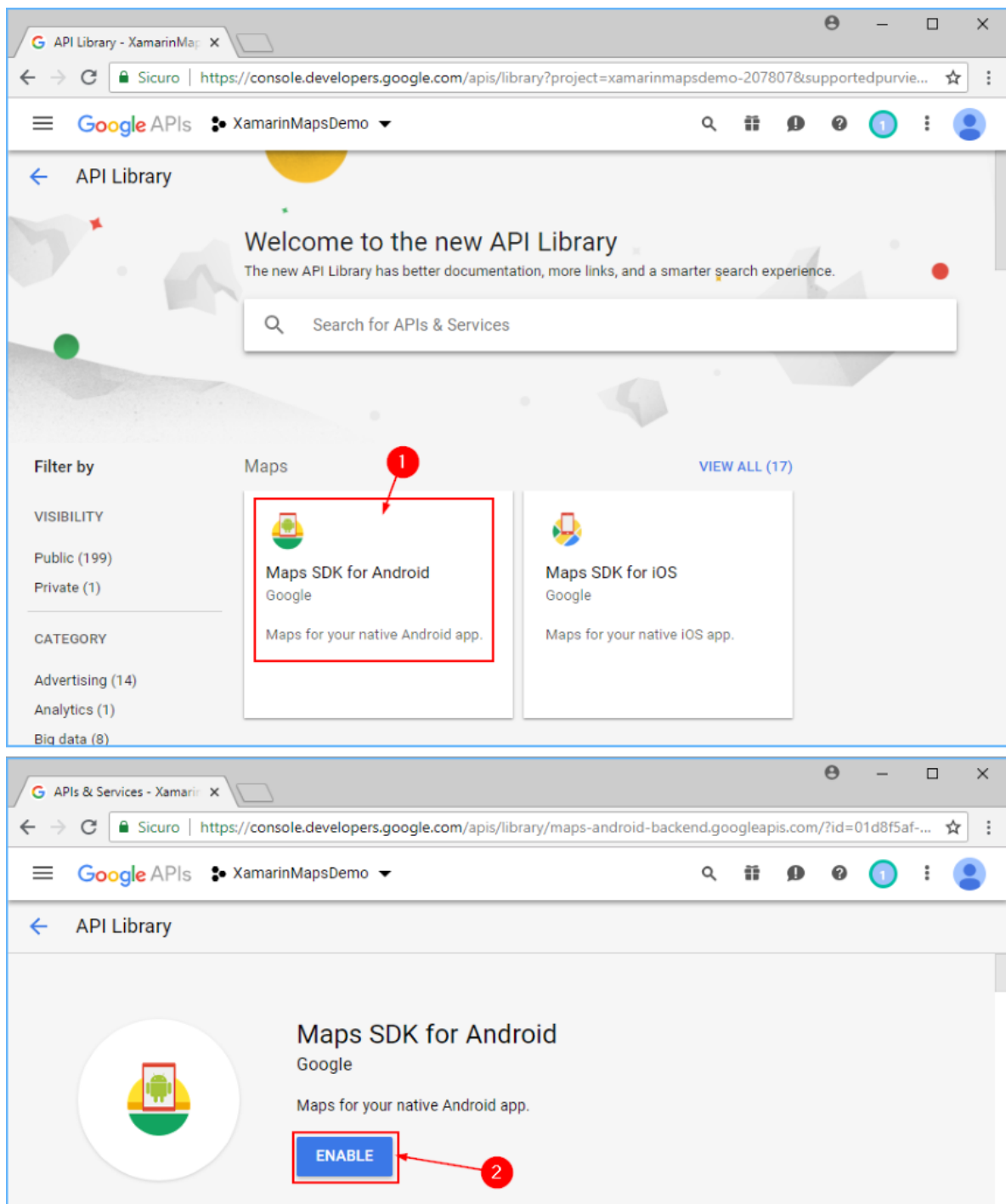
2. 如果您创建了一个新项目, 请输入中的项目名称**新的项目**显示对话框。此对话框将制造取决于你的项目名称的唯一项目 ID。接下来, 单击**创建**按钮在此示例中所示:



3. 后一分钟左右, 创建项目, 并转到仪表板项目页。在这里, 单击启用的 **API 和服务**:



4. 从API 库页上, 单击**Maps SDK for Android**。在下一页上, 单击**启用**若要打开此项目的服务:

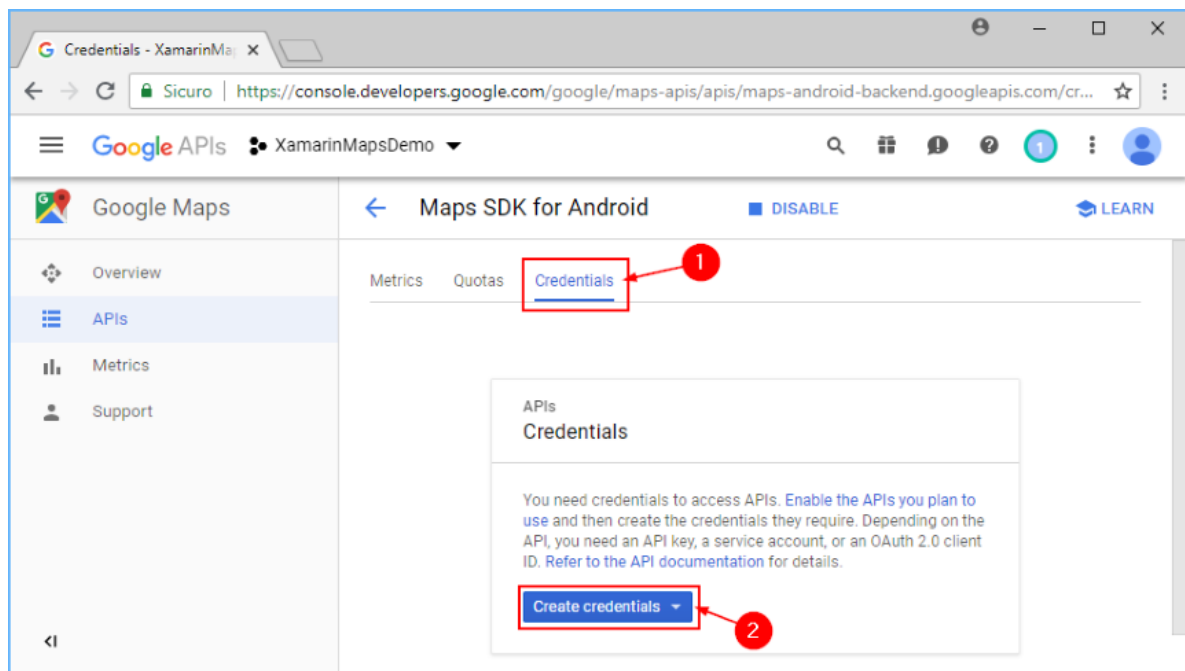


此时已创建的 API 项目和 Google Maps Android API v2 已添加到它。但是，不能在项目中使用此 API 之前为其创建凭据。下一部分介绍如何创建 API 密钥和允许列表的 Xamarin.Android 应用程序，以便其有权使用此密钥。

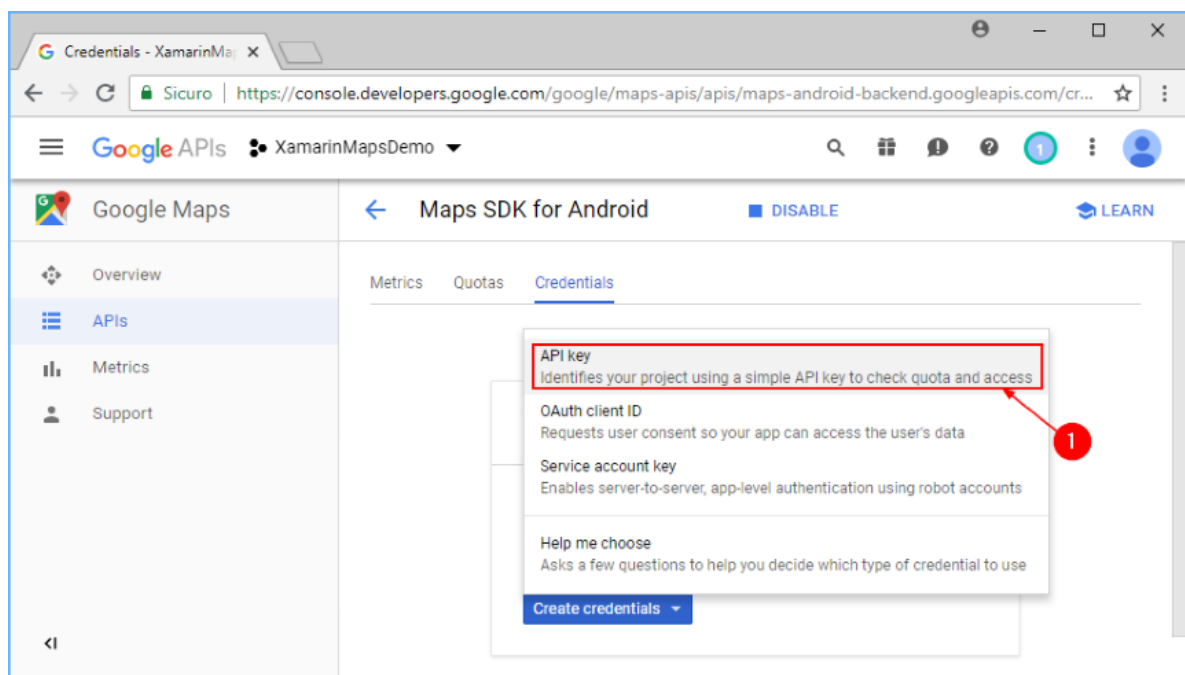
获取 API 密钥

之后 **Google Developer Console** API 项目已创建的有必要创建 Android API 密钥。Xamarin.Android 应用程序必须具有 API 密钥之前向他们授予对 Android 的 Map API v2 的访问。

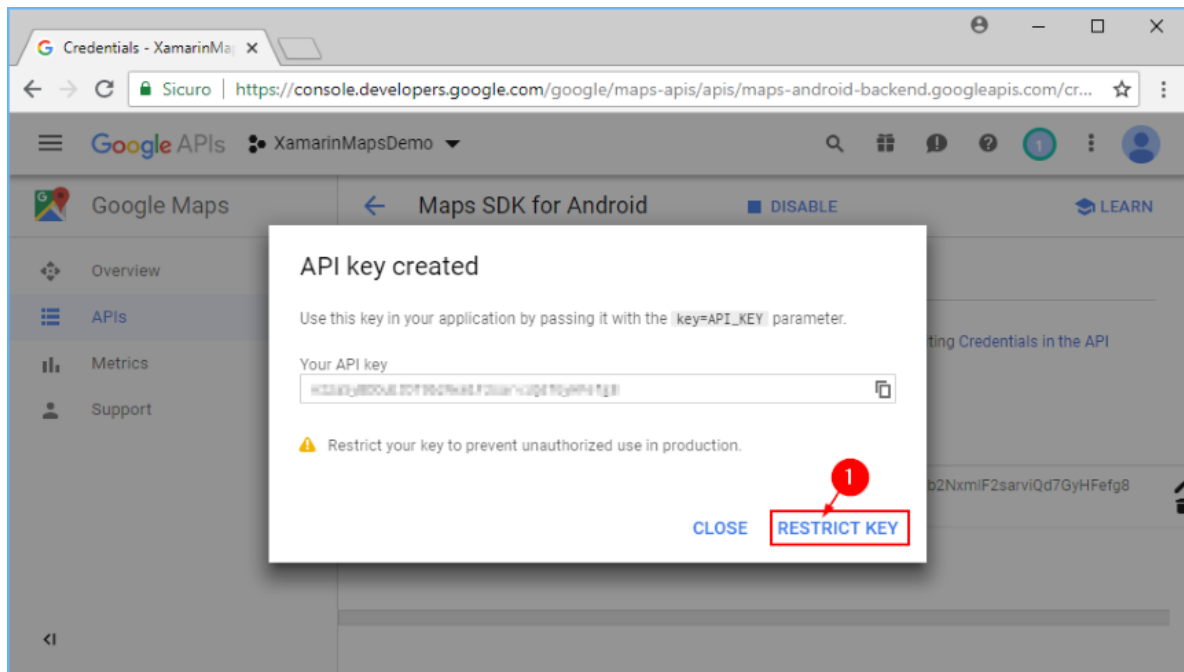
1. 中 **Maps SDK for Android** 显示的页 (单击后启用上一步中)，请转到 **凭据** 选项卡，单击 **创建凭据** 按钮：



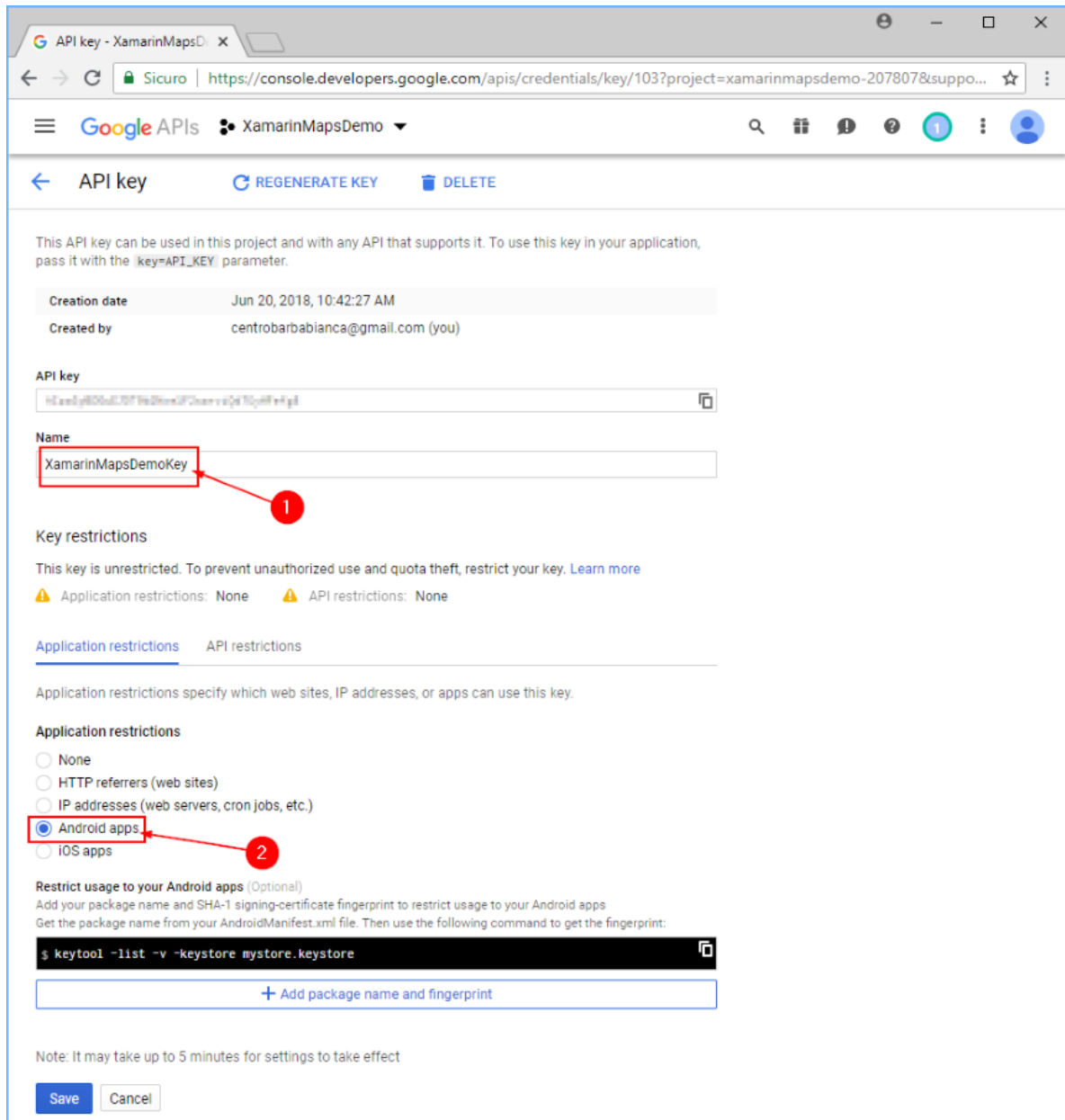
2. 单击**API 密钥**:



3. 单击此按钮后，将生成的 API 密钥。接下来是需要限制此密钥，以便你的应用可以使用此密钥调用 Api。单击**限制密钥**：



4. 更改名称字段从API 密钥 1到将帮助您记住该密钥的使用的名称 (**XamarinMapsDemoKey**此示例中使用)。接下来，单击**Android 应用**单选按钮：



5. 若要添加的 sha-1 指纹, 请单击 + 添加包的名称和指纹:

Restrict usage to your Android apps

Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps

[Learn more](#)

Get the package name from your AndroidManifest.xml file. Then use the following command to get the fingerprint:

```
$ keytool -list -v -keystore mystore.keystore
```

+ Add package name and fingerprint

6. 输入你的应用包名称并输入 sha-1 证书指纹 (通过获得 `keytool` 本指南中前面所述)。在以下示例中, 包名称 `XamarinMapsDemo` 是输入后, 跟从获取 sha-1 证书指纹 `debug.keystore`:

Restrict usage to your Android apps

Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps

[Learn more](#)

Get the package name from your AndroidManifest.xml file. Then use the following command to get the fingerprint:

```
$ keytool -list -v -keystore mystore.keystore
```

| Package name | SHA-1 certificate fingerprint | |
|---|--|---|
| <code>com.xamarin.docs.android.map</code> | <code>F2:46:F8:6B:92:1B:F9:4A:61:F9:8D:C7:B9:5F:25:41:A3:69:CA:5C</code> | × |

+ Add package name and fingerprint

7. 请注意, 为了使 APK 来访问 Google 地图, 您必须包括 sha-1 指纹, 包使用对 APK 进行签名每个密钥存储 (调试和发布) 的名称。例如, 如果一台计算机用于调试和生成发布 APK 的另一台计算机, 您应包括 sha-1 证书指纹从第一台计算机的调试密钥存储和从的发布密钥存储的 sha-1 证书指纹第二台计算机。单击 + 添加包的名称和指纹以添加另一个指纹和包名称, 在此示例中所示:

Restrict usage to your Android apps

Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps

[Learn more](#)

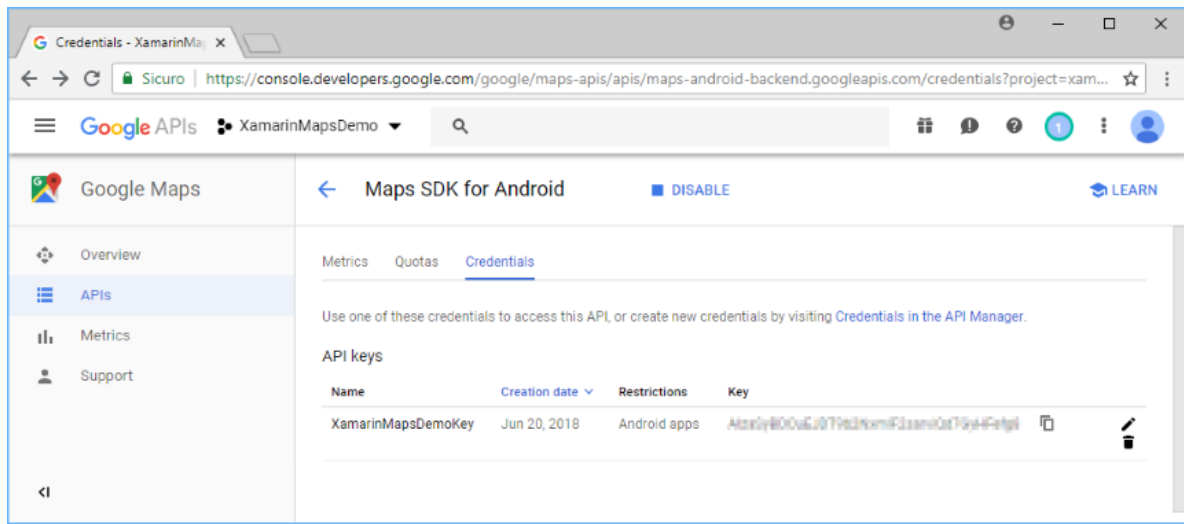
Get the package name from your AndroidManifest.xml file. Then use the following command to get the fingerprint:

```
$ keytool -list -v -keystore mystore.keystore
```

| Package name | SHA-1 certificate fingerprint | |
|---|--|---|
| <code>com.xamarin.docs.android.map</code> | <code>F2:46:F8:6B:92:1B:F9:4A:61:F9:8D:C7:B9:5F:25:41:A3:69:CA:5C</code> | × |
| <code>com.xamarin.docs.android.map</code> | <code>63:38:10:73:71:3E:D4:E4:EF:87:FB:7B:D8:3C:A2:1E:E3:D4:95:A</code> | × |

+ Add package name and fingerprint

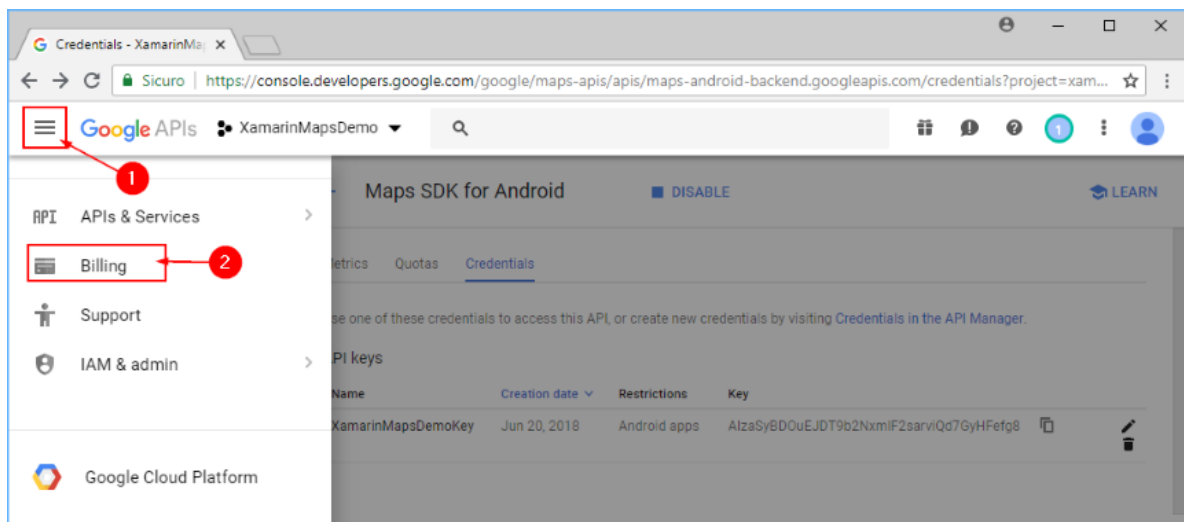
8. 单击“保存”按钮保存更改。接下来, 你将返回到你的 API 密钥的列表。如果有其他前面创建的 API 密钥, 它们还将此处列出。在此示例中, 列出了只有一个 API 密钥 (在前面的步骤中创建):



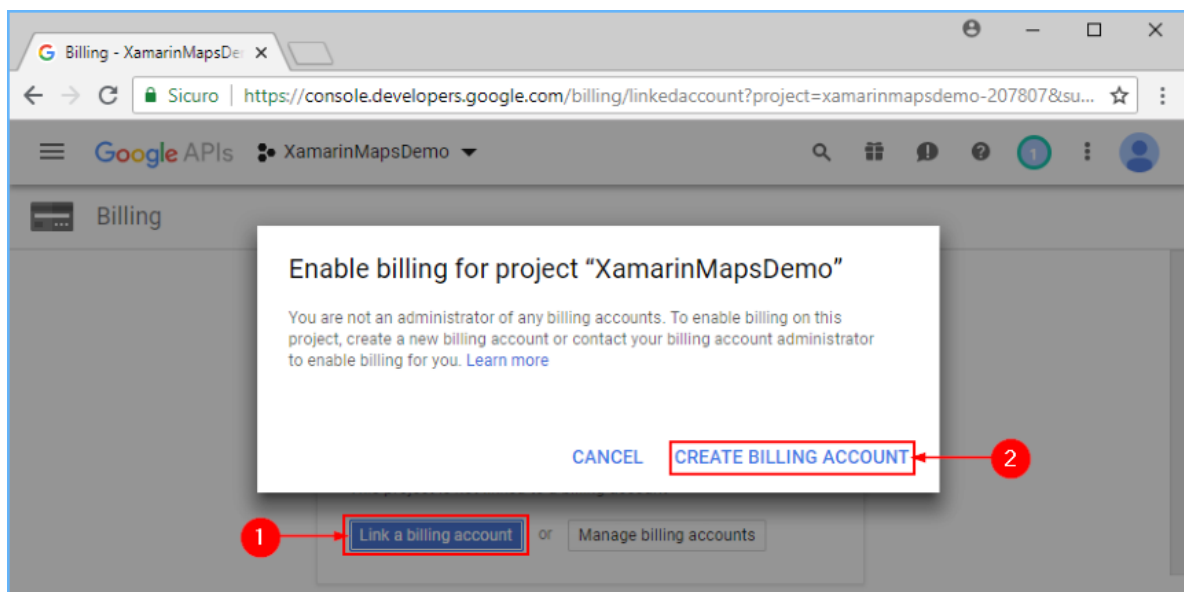
将项目连接到一个可计费帐户

从 2018 年 6 月 11 日，API 密钥将无法工作如果项目不连接到计费帐户（即使该服务仍然免费提供给移动应用）。

1. 单击汉堡菜单按钮，然后选择计费页：



2. 将项目链接到计费帐户时，通过单击计费帐户链接跟创建计费帐户在显示弹出窗口（如果还没有帐户，我们将帮助您创建一个新）：



将密钥添加到你的项目

最后，添加到此 API 密钥 **AndroidManifest.XML** Xamarin.Android 应用程序的文件。在以下示例中，

`YOUR_API_KEY` 是替换为在上一步骤中生成的 API 密钥：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionName="4.10" package="com.xamarin.docs.android.mapsandlocationdemo"
    android:versionCode="10">
    ...
    <application android:label="@string/app_name">
        <!-- Put your Google Maps V2 API Key here. -->
        <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="YOUR_API_KEY" />
        <meta-data android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
    </application>
</manifest>
```

相关链接

- [Google Api 控制台](#)
- [Google 地图 API 密钥](#)
- [keytool](#)

Android 语音

2018/10/26 • [Edit Online](#)

本文介绍如何使用功能非常强大 `Android.Speech` 命名空间的基础知识。从诞生, `Android` 已经能够识别语音并将其输出为文本。它是一个相对简单的过程。对于文本到语音, 但是, 该过程是更为复杂, 因为不仅不语音引擎必须考虑到, 但还语言可用且已安装从文本到语音 (TTS) 系统。

语音概述

具有系统, 其中"了解"人类语音和 enunciates 正在键入的内容 — 语音转文本和文本到语音转换 — 如自然的沟通, 与我们的设备在需求提高是日益增长区域中的移动开发。很多情况下, 具有文本将转换为语音, 或反之, 是非常有用的工具, 可将合并到 `android` 应用程序的功能。

例如, 使用向下移动电话使用, 同时还能降低 `clamp`, 用户需要运行他们的设备一手免费方式。数不清的不同 `Android` 窗体因素 — 例如 `Android Wear` —, 不断扩大的那些包含能够使用 `Android` 设备 (如平板电脑和记事本) 创建了更大的焦点上 TTS 的众多应用程序。

`Google` 提供使用 `Android.Speech` 命名空间中更丰富的 `Api` 开发人员, 以涵盖大多数情况下的"语音识别"(如 `for the blind` 而设计的软件) 的设备。该命名空间包含设施以允许文本转换为通过语音 `Android.Speech.Tts`, 用于执行转换, 以及许多的引擎控制 `RecognizerIntent` s 允许语音, 以将转换为文本。

在设施的语音, 以了解需要, 在基于使用的硬件的限制。不太可能在设备成功解释所说到它在每个语言中可用的所有内容。

要求

有本指南中, 你的设备麦克风和扬声器以外没有特殊要求。

解释语音的 `Android` 设备的核心是使用 `Intent` 相对应的 `OnActivityResult`。它是重要的是, 不过, 若要识别不理解语音 — 但解释为文本。差异十分重要。

了解和解释之间的差异

了解简单定义是您将能够通过的语调和上下文确定的讲述进行的真正意义。只需解释意味着需要单词和输出它们在另一个窗体中。

请考虑使用日常对话中的以下简单示例:

喂, 你好吗?

而无需转折点 (侧重于特定的单词或单词的某些部分), 它是一个简单的问题。但是, 如果慢的速度应用于线条, 侦听的人会检测或提问者不是太高兴, 并可能需要 cheering 提问者是 `unwell`。如果焦点位于"是", 请求的人员是通常更感兴趣的响应。

不带相当强大音频处理以使利用转折点和一定程度的人工智能 (AI) 以了解上下文, 软件甚至不能以了解所说的内容 — 的最简单的手机可以执行操作是将语音转换为文本。

设置

使用语音系统之前, 最好始终检查并确保设备的麦克风。将有意义尝试 `Kindle` 或 `Google` 注意板上没有安装麦克风的情况下运行你的应用。

下面的代码示例演示如果麦克风可用, 如果没有, 请查询创建警报。如果没有麦克风可用此时您将退出活动或禁用语音录制功能。

```

string rec = Android.Content.PM.PackageManager.FeatureMicrophone;
if (rec != "android.hardware.microphone")
{
    var alert = new AlertDialog.Builder(recButton.Context);
    alert.SetTitle("You don't seem to have a microphone to record with");
    alert.SetPositiveButton("OK", (sender, e) =>
    {
        return;
    });
    alert.Show();
}

```

创建意向

语音系统意向使用特定类型的调用的意图 `RecognizerIntent`。此意向控制大量参数，包括多长时间等待用静音，直到记录被认为比，以识别并输出，任何其他语言和要包括在任何文本 `Intent` 的模式对话框作为指令的方式。此代码片段 `VOICE` 是 `readonly int` 用于识别中的 `OnActivityResult`。

```

var voiceIntent = new Intent(RecognizerIntent.ActionRecognizeSpeech);
voiceIntent.PutExtra(RecognizerIntent.ExtraLanguageModel, RecognizerIntent.LanguageModelFreeForm);
voiceIntent.PutExtra(RecognizerIntent.ExtraPrompt,
Application.Context.GetString(Resource.String.messageSpeakNow));
voiceIntent.PutExtra(RecognizerIntent.ExtraSpeechInputCompleteSilenceLengthMillis, 1500);
voiceIntent.PutExtra(RecognizerIntent.ExtraSpeechInputPossiblyCompleteSilenceLengthMillis, 1500);
voiceIntent.PutExtra(RecognizerIntent.ExtraSpeechInputMinimumLengthMillis, 15000);
voiceIntent.PutExtra(RecognizerIntent.ExtraMaxResults, 1);
voiceIntent.PutExtra(RecognizerIntent.ExtraLanguage, Java.Util.Locale.Default);
StartActivityForResult(voiceIntent, VOICE);

```

语音的转换

从语音解释的文本将中传递 `Intent`，其返回时该活动完成后，通过访问

`GetStringArrayListExtra(RecognizerIntent.ExtraResults)`。这将返回 `IList<string>`，而该索引可用于并显示，具体取决于语言中调用方意向请求数量（其指定 `RecognizerIntent.ExtraMaxResults`）。与任何列表，它是值得选择，以确保没有要显示数据。

当侦听的返回值时 `StartActivityForResult`，则 `OnActivityResult` 方法有提供。

在以下示例中，`textBox` 是 `TextBox` 用于输出决定什么。它同样可用于传递给解释器，并从某种形式的文本，该应用程序可以比较文本和分支到的应用程序其他部分。

```
protected override void OnActivityResult(int requestCode, Result resultVal, Intent data)
{
    if (requestCode == VOICE)
    {
        if (resultVal == Result.Ok)
        {
            var matches = data.GetStringArrayListExtra(RecognizerIntent.ExtraResults);
            if (matches.Count != 0)
            {
                string textInput = textBox.Text + matches[0];
                textBox.Text = textInput;
                switch (matches[0].Substring(0, 5).ToLower())
                {
                    case "north":
                        MovePlayer(0);
                        break;
                    case "south":
                        MovePlayer(1);
                        break;
                }
            }
            else
            {
                textBox.Text = "No speech was recognised";
            }
        }
        base.OnActivityResult(requestCode, resultVal, data);
    }
}
```

文本到语音转换

文本到语音转换刚好与之相反的语音到文本并不依赖于两个关键组件;文本到语音转换引擎正在设备上安装并正在安装一种语言。

默认值, Android 设备有很大程度上, 安装 Google TTS 服务和至少一种语言。这建立当设备首次设置时, 并且将根据设备是时 (例如, 在德国设置手机将安装德语的语言, 而另一个 America 中将具有美国英语)。

步骤 1-实例化 TextToSpeech

`TextToSpeech` 可能需要最多 3 个参数前, 两个需要与第三个可选的 (`AppContext` , `IOnInitListener` , `engine`)。侦听器用于绑定到的服务和失败的测试具有被任意数量的可用 Android 的文本到语音转换引擎的引擎。至少, 设备将有 Google 的引擎。

步骤 2-查找可用的语言

`Java.Util.Locale` 类包含一个名为的有用方法 `GetAvailableLocales()`。然后可以对已安装的语言测试这一系列语音引擎支持的语言。

这是普通的问题生成的"了解"语言的列表。始终会默认语言 (用户设置时他们首次设置其设备的语言), 因此, 在此示例 `List<string>` 具有 "Default" 作为第一个参数, 将根据的结果填充列表的其余部分

`textToSpeech.IsLanguageAvailable(locale)`。

```

var langAvailable = new List<string>{ "Default" };
var localesAvailable = Java.Util.Locale.GetAvailableLocales().ToList();
foreach (var locale in localesAvailable)
{
    var res = textToSpeech.IsLanguageAvailable(locale);
    switch (res)
    {
        case LanguageAvailableResult.Available:
            langAvailable.Add(locale.DisplayLanguage);
            break;
        case LanguageAvailableResult.CountryAvailable:
            langAvailable.Add(locale.DisplayLanguage);
            break;
        case LanguageAvailableResult.CountryVarAvailable:
            langAvailable.Add(locale.DisplayLanguage);
            break;
    }
}
langAvailable = langAvailable.OrderBy(t => t).Distinct().ToList();

```

此代码将调用[TextToSpeech.IsLanguageAvailable](#)要测试是否已在设备上存在给定区域设置的语言包。此方法返回[LanguageAvailableResult](#)，指示是否可传递的区域设置的语言。如果 [LanguageAvailableResult](#) 指示的语言是 [NotSupported](#)，没有任何语音包可用（即使对于下载），然后针对该语言。如果 [LanguageAvailableResult](#) 设置为 [MissingData](#)，则可能如下所述步骤 4 中下载新的语言包。

步骤 3-设置速度和间距

Android 允许用户通过更改 alter 语音的声音 [SpeechRate](#) 和 [Pitch](#)（速度和语音的音的频率）。这是由从 0 为 1，与"normal"语音为两个 1。

步骤 4-测试和加载新语言

下载新的语言使用执行 [Intent](#)。此目的的结果将导致[OnActivityResult](#)要调用的方法。与语音到文本的示例不同（使用哪一种[RecognizerIntent](#)作为 [PutExtra](#) 参数 [Intent](#)），测试和加载 [Intent](#) s 是 [Action](#) -基于：

- [TextToSpeech.Engine.ActionCheckTtsData](#) -从平台中启动活动 [TextToSpeech](#) 引擎来验证正确安装和在设备上的语言资源的可用性。
- [TextToSpeech.Engine.ActionInstallTtsData](#) -启动提示用户下载需要的语言的活动。

下面的代码示例说明了如何使用这些操作来测试语言资源和下载新的语言：

```

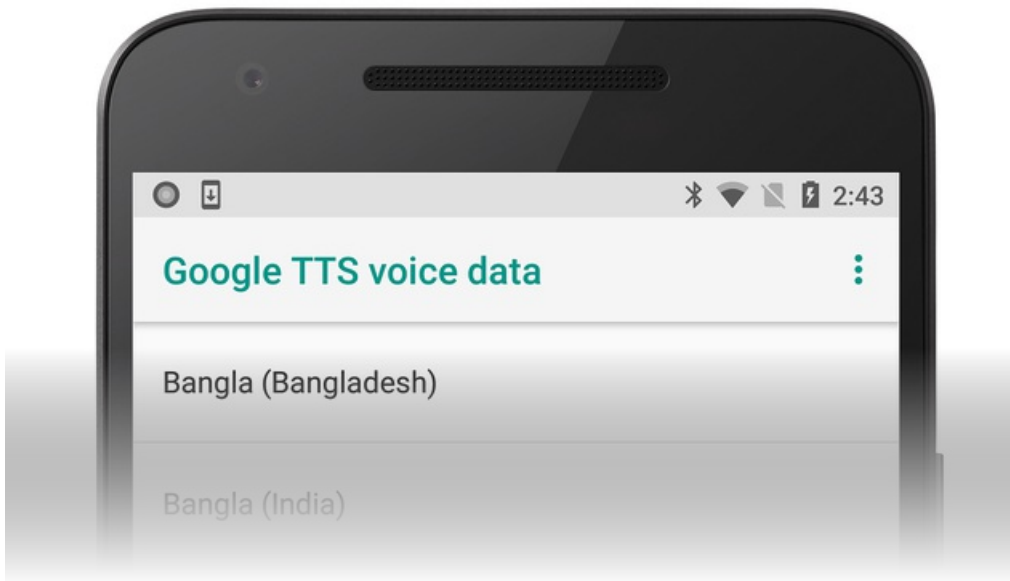
var checkTTSIntent = new Intent();
checkTTSIntent.SetAction(TextToSpeech.Engine.ActionCheckTtsData);
StartActivityForResult(checkTTSIntent, NeedLang);
//
protected override void OnActivityResult(int req, Result res, Intent data)
{
    if (req == NeedLang)
    {
        var installTTS = new Intent();
        installTTS.SetAction(TextToSpeech.Engine.ActionInstallTtsData);
        StartActivity(installTTS);
    }
}

```

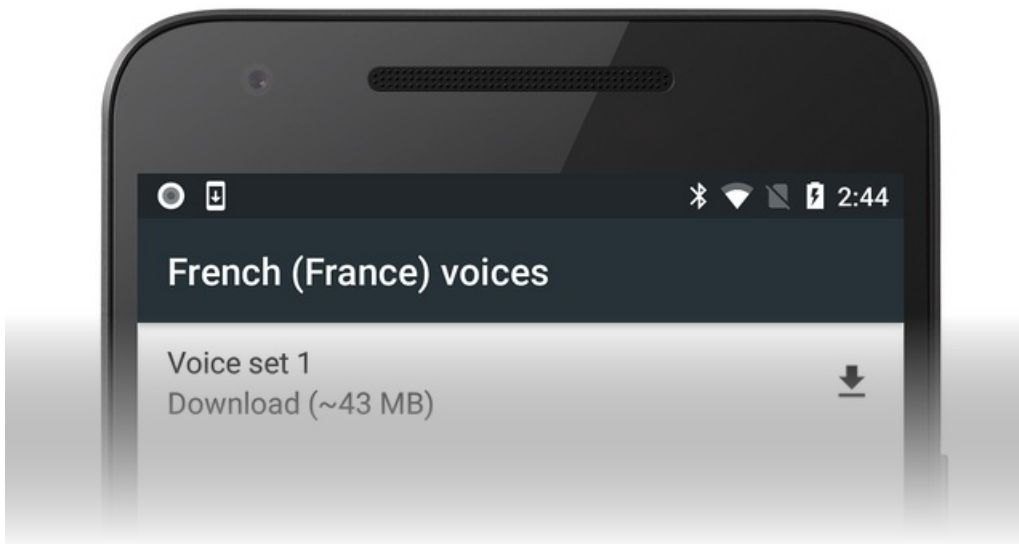
[TextToSpeech.Engine.ActionCheckTtsData](#) 语言资源的可用性测试。 [OnActivityResult](#) 此测试完成时调用。如果需要下载，语言资源 [OnActivityResult](#) 激发一次 [TextToSpeech.Engine.ActionInstallTtsData](#) 操作启动的活动，允许用户下载需要的语言。请注意，此 [OnActivityResult](#) 实现不会检查 [Result](#) 代码，因为在此简化的示例中，确定已经需要下载的语言包。

[TextToSpeech.Engine.ActionInstallTtsData](#) 操作的原因Google TTS 语音数据活动以选择要下载的语言显示给用

户：



例如，用户可能选择法语，并单击下载图标下载法语语音数据：



在下载完成后将自动发生此类数据的安装。

步骤 5-IOOnInitListener

有关用于转换文本到语音，接口方法的活动 `OnInit` 来实现 (这是指定的实例化的第二个参数 `TextToSpeech` 类)。此初始化该侦听器并测试结果。

侦听器应测试两个 `OperationResult.Success` 和 `OperationResult.Failure` 最小值。下面的示例显示了这一：

```
void TextToSpeech.IOOnInitListener.OnInit(OperationResult status)
{
    // if we get an error, default to the default language
    if (status == OperationResult.Error)
        textToSpeech.SetLanguage(Java.Util.Locale.Default);
    // if the listener is ok, set the lang
    if (status == OperationResult.Success)
        textToSpeech.SetLanguage(lang);
}
```


总结

在本指南中我们已经探讨将文本到语音和语音转换为文本和如何将其包含在你自己的应用中的可能的方法的基础知识。而它们并未涵盖每个特定情况下，您现在应具有基本了解如何解释语音、如何安装新语言，以及如何提高您的应用程序的 inclusivity。

相关链接

- [Xamarin.Forms DependencyService](#)
- [文本到语音转换（示例）](#)
- [语音转文本（示例）](#)
- [Android.Speech 命名空间](#)
- [Android.Speech.Tts 命名空间](#)

Java 集成概述

2018/11/2 • [Edit Online](#)

Java 生态系统包括组件的不同和巨大的集合。许多这些组件可用于减少开发 Android 应用程序所需的时间。本文档将介绍并提供一些开发人员可以使用这些现有的 Java 组件以提高其 Xamarin.Android 应用程序的开发体验的方式的高级概述。

概述

给定的 Java 生态系统的程度，它是很有可能在 Java 中已编码为 Xamarin.Android 应用程序所需的任何给定的功能。因此，很吸引人尝试并创建 Xamarin.Android 应用程序时重复使用这些现有的库。

有三个可能的方法可重复使用 Xamarin.Android 应用程序中的 Java 库：

- **创建 Java 绑定库**—使用此技术，Xamarin.Android 项目用于创建 C# Java 类型的包装。然后，Xamarin.Android 应用程序可以引用 C# 包装器创建此项目，然后使用 `.jar` 文件。
- **Java 本机接口**—*Java 本机接口*(JNI) 是一个框架，允许非 Java 代码 (如 c + + 或 C#) 调用或调用通过运行的 Java 代码 JVM 内部。
- **将代码移植**—此方法包括获取 Java 源代码，并再将其转换为 C#。这可以手动或通过使用如锐化的自动化的工具。

前两个技术的核心是 *Java 本机接口*(JNI)。JNI 是一个框架，允许不以 Java 编写的应用程序的 Java 虚拟机中运行的 Java 代码进行交互。Xamarin.Android 使用 JNI 来创建 *绑定* 为 C# 代码。

第一种方法是绑定 Java 库的自动化程度更高、声明性方法。它涉及到使用 Visual Studio for Mac 或 Visual Studio 项目类型提供的 Xamarin.Android – Java 绑定库。若要成功创建这些绑定，Java 绑定库可能仍需要一些手动修改，但没有那么多一样将纯 JNI 方法。请参阅[绑定 Java 库](#)有关 Java 绑定库的详细信息。

第二种方法，使用 JNI，在更低级别工作，但可以提供更精细的控制和访问通常不能通过 Java 绑定库可访问的 Java 方法。

第三种方法是从以前的两个全然不同：移植到 Java 中的代码 C#。移植到另一种语言中的代码是一个非常费力的过程，但可以减少工作量的一种工具帮助调用 *锐化*。提升是开放源代码工具，它是 Java-到-C# 转换器。

总结

本文档提供的一些不同的方式通过 Java 库，可以在 Xamarin.Android 应用程序中重复使用的高级概述。它引入了绑定的概念和管理可调用包装器，并探讨了用于 Java 将代码移植到选项 C#。

相关链接

- [体系结构](#)
- [绑定 Java 库](#)
- [使用 JNI](#)
- [锐化](#)
- [Java 本机接口](#)

Android 可调用包装器

2018/10/26 • [Edit Online](#)

每当 Android 运行时将调用托管的代码时，android 可调用包装器 (ACWs) 是必需的。需要这些包装器，因为没有方法在运行时与艺术 (Android 运行时) 注册类。(具体而言，[JNI DefineClass\(\) 函数](#) Android 运行时不支持。) Android 可调用包装器因此弥补缺少的运行时类型注册支持。

每次 Android 代码要执行所需 `virtual` 接口的方法或 `overridden` 或托管代码中实现时，Xamarin.Android 必须提供 Java 代理，以便此方法被调度到相应的托管类型。这些 Java 代理类型是与托管类型实现相同的构造函数并声明任何重写的基类和接口方法具有“相同”的基类和 Java 接口列表的 Java 代码。

Android 可调用包装器生成的 `monodroid.exe` 程序期间 [生成过程](#)：(直接或间接) 继承的所有类型生成 `Java.Lang.Object`。

Android 可调用包装器命名

Android 可调用包装器的包名称基于要导出的类型的程序集限定名称的 MD5SUM。此命名方法实现相同的完全限定类型名称，将变得可由不同的程序集而不会引入打包错误。

由于此 MD5SUM 命名方案，不能按名称直接访问您的类型。例如，以下 `adb` 命令不会起作用，因为类型名称 `my.ActivityType` 默认情况下不生成：

```
adb shell am start -n My.Package.Name/my.ActivityType
```

此外，可能会看到如下所示的错误，如果你尝试按名称引用的类型：

```
java.lang.ClassNotFoundException: Didn't find class "com.company.app.MainActivity"
on path: DexPathList[[zip file "/data/app/com.company.App-1.apk"] ...
```

如果您执行需要访问对按名称的类型，可以声明该类型在特性声明中的名称。例如，下面是声明的完全限定名称的活动的代码 `My.ActivityType`：

```
namespace My {
    [Activity]
    public partial class ActivityType : Activity {
        /* ... */
    }
}
```

`ActivityAttribute.Name` 可以设置属性来显式声明此活动的名称：

```
namespace My {
    [Activity(Name="my.ActivityType")]
    public partial class ActivityType : Activity {
        /* ... */
    }
}
```

添加此属性设置后，`my.ActivityType` 可以按名称从外部代码和访问 `adb` 脚本。`Name` 属性可以为许多不同类型，包括 `Activity`，`Application`，`Service`，`BroadcastReceiver`，并 `ContentProvider`：

- [ActivityAttribute.Name](#)
- [ApplicationAttribute.Name](#)
- [ServiceAttribute.Name](#)
- [BroadcastReceiverAttribute.Name](#)
- [ContentProviderAttribute.Name](#)

在 Xamarin.Android 5.0 中引入了基于 MD5SUM ACW 命名。有关命名属性的详细信息，请参阅[RegisterAttribute](#)。

实现接口

有时您可能需要实现一个 Android 接口，例如[Android.Content.IComponentCallbacks](#)。由于所有 Android 类和接口扩展[Android.Runtime.IJavaObject](#)接口，现在的问题是：我们如何实现 `IJavaObject` ？

上面回答该问题的：原因 Android 的所有类型都需要实现 `IJavaObject`，以便 Xamarin.Android 具有 Android，即为给定类型的 Java 代理提供的 Android 可调用包装器。由于 **monodroid.exe** 只寻找 `Java.Lang.Object` 子类，和 `Java.Lang.Object` 实现 `IJavaObject`，答案会显而易见：子类 `Java.Lang.Object`：

```
class MyComponentCallbacks : Java.Lang.Object, Android.Content.IComponentCallbacks {

    public void OnConfigurationChanged (Android.Content.Res.Configuration newConfig)
    {
        // implementation goes here...
    }

    public void OnLowMemory ()
    {
        // implementation goes here...
    }
}
```

实现详细信息

此页的其余部分提供了实现详细信息，如有更改，恕不另行通知(和只是因为开发人员会想知道这怎么回事，此处提供)。

例如，假定有以下 C# 源：

```
using System;
using Android.App;
using Android.OS;

namespace Mono.Samples.HelloWorld
{
    public class HelloAndroid : Activity
    {
        protected override void OnCreate (Bundle savedInstanceState)
        {
            base.OnCreate (savedInstanceState);
            SetContentView (R.layout.main);
        }
    }
}
```

Mandroid.exe 程序将生成以下 Android 可调用包装器：

```

package mono.samples.helloWorld;

public class HelloAndroid
    extends android.app.Activity
{
    static final String __md_methods;
    static {
        __md_methods = "n_onCreate:(Landroid/os/Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\n" + "";
        mono.android.Runtime.register (
            "Mono.Samples.HelloWorld.HelloAndroid, HelloWorld, Version=1.0.0.0,
            Culture=neutral, PublicKeyToken=null", HelloAndroid.class, __md_methods);
    }

    public HelloAndroid ()
    {
        super ();
        if (getClass () == HelloAndroid.class)
            mono.android.TypeManager.Activate (
                "Mono.Samples.HelloWorld.HelloAndroid, HelloWorld, Version=1.0.0.0,
                Culture=neutral, PublicKeyToken=null", "", this, new java.lang.Object[] { });
    }

    @Override
    public void onCreate (android.os.Bundle p0)
    {
        n_onCreate (p0);
    }

    private native void n_onCreate (android.os.Bundle p0);
}

```

请注意，它也保留的基类，和 `native` 为托管代码中重写每个方法提供了方法声明。

使用 JNI

2018/10/26 • [Edit Online](#)

Xamarin.Android 允许编写 Android 应用中的 C# 而不是 Java。多个程序集提供使用 Xamarin.Android 提供的 Java 库, 包括 Mono.Android.dll 和 Mono.Android.GoogleMaps.dll 绑定。但是, 对于每个可能的 Java 库, 不提供了绑定, 并且提供的绑定可能不会绑定每个 Java 类型和成员。若要使用未绑定的 Java 类型和成员, 可以使用 Java 本机接口 (JNI)。本文将演示如何使用 JNI 与 Java 类型和成员从 Xamarin.Android 应用程序进行交互。

概述

它并不总是需要或不能创建托管可调用包装 (MCW) 来调用 Java 代码。在许多情况下, "内联" JNI 是完全可接受和用于一次性使用未绑定 Java 成员。通常是使用 JNI 调用上比以生成整个 jar 绑定一个 Java 类的单个方法简单得多。

Xamarin.Android 提供了 `Mono.Android.dll` 程序集, 提供适用于 Android 的绑定 `android.jar` 库。类型和成员中不存在 `Mono.Android.dll` 中不存在类型和 `android.jar` 可能由手动将其绑定。若要绑定 Java 类型和成员, 请使用 **Java 本机接口 (JNI)** 若要查找类型、读写字段, 并调用方法。

在 Xamarin.Android 中的 JNI API 是从概念上讲非常类似于 `System.Reflection` 在 .NET 中的 API: 它可以为您要查找类型和成员名称, 通过读取和写入字段值, 调用方法和的详细信息。可以使用 JNI 和 `Android.Runtime.RegisterAttribute` 自定义特性来声明可以绑定到支持重写的虚方法。可以将绑定接口, 以便它们可以实现在 C#。

本文档说明:

- 如何 JNI 引用类型。
- 如何查找、读取和写入的字段。
- 如何查找和调用方法。
- 如何公开虚方法来从托管代码重写。
- 如何公开接口。

要求

JNI, 如通过公开 [Android.Runtime.JNIEnv 命名空间](#), 可在每个版本的 Xamarin.Android。若要绑定 Java 类型和接口, 必须使用 Xamarin.Android 4.0 或更高版本。

托管的可调用包装器

一个托管可调用包装器 (MCW) 是 *绑定* Java 类或接口, 从而将所有的 JNI 机制因此该客户端 C# 无需代码担心的 JNI 底层的复杂性。大多数的 `Mono.Android.dll` 包含托管的可调用包装器。

托管的可调用包装器有两种用途:

1. 封装 JNI 使用, 从而使客户端代码不需要了解的有关底层的复杂性。
2. 让可以子类 Java 类型和实现 Java 接口。

第一个用途是纯粹是为了方便使用和复杂程度的封装, 以便使用者具有一套简单、托管的要使用的类。这需要使用的各种 [JNIEnv](#) 成员在本文后面部分所述。请记住, 管理可调用包装器并不是必需-"内联" JNI 使用是完全可以接受, 并可用于一次性使用未绑定 Java 成员。子类和接口的实现需要使用托管的可调用包装器。

Android 可调用包装器

Android 可调用包装器 (ACW) 是必需的每当需要调用托管的代码; Android 运行时 (图片) 需要这些包装器, 因为没有方法在运行时与艺术注册类。(具体而言, [DefineClass](#) JNI 函数不受 Android 运行时。Android 可调用包装器因此弥补缺少的运行时类型注册支持。)

每当 Android 代码需要执行的虚拟或接口方法重写或实现是在托管代码中, Xamarin.Android 必须提供 Java 代理, 以便此方法获取调度到相应的托管类型。这些 Java 代理类型是具有与托管类型实现相同的构造函数并声明任何重写的基类和接口方法"相同"的基类和 Java 接口列表的 Java 代码。

Android 可调用包装器生成的 **monodroid.exe** 程序期间 [生成过程](#), 并为 (直接或间接) 继承的所有类型生成 [Java.Lang.Object](#)。

实现接口

有时您可能需要实现一个 Android 接口 (如 [Android.Content.IComponentCallbacks](#))。

所有 Android 类和接口扩展 [Android.Runtime.IJavaObject](#) 接口; 因此, 所有 Android 类型必须实现 `IJavaObject`。Xamarin.Android 利用这一事实-它使用 `IJavaObject`, 让 Android Java 代理 (Android 可调用包装) 针对给定托管类型。因为 **monodroid.exe** 只寻找 `Java.Lang.Object` 子类 (必须实现 `IJavaObject`)、生成子类 `Java.Lang.Object` 为我们提供了一种在托管代码中实现接口方法。例如:

```
class MyComponentCallbacks : Java.Lang.Object, Android.Content.IComponentCallbacks {
    public void OnConfigurationChanged (Android.Content.Res.Configuration newConfig) {
        // implementation goes here...
    }
    public void OnLowMemory () {
        // implementation goes here...
    }
}
```

实现详细信息

本文的其余部分提供了实现详细信息, 如有更改, 恕不另行通知(和此处只是因为开发人员可能会好奇这怎么回事实质上提供)。

例如, 假定有以下 C# 源:

```
using System;
using Android.App;
using Android.OS;

namespace Mono.Samples.HelloWorld
{
    public class HelloAndroid : Activity
    {
        protected override void OnCreate (Bundle savedInstanceState)
        {
            base.OnCreate (savedInstanceState);
            SetContentView (R.layout.main);
        }
    }
}
```

Mandroid.exe 程序将生成以下 Android 可调用包装器:

```

package mono.samples.helloWorld;

public class HelloAndroid extends android.app.Activity {
    static final String __md_methods;
    static {
        __md_methods =
            "n_onCreate:(Landroid/os/Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\n" +
            "";
        mono.android.Runtime.register (
            "Mono.Samples.HelloWorld.HelloAndroid, HelloWorld, Version=1.0.0.0, Culture=neutral,
            PublicKeyToken=null",
            HelloAndroid.class,
            __md_methods);
    }

    public HelloAndroid ()
    {
        super ();
        if (getClass () == HelloAndroid.class)
            mono.android.TypeManager.Activate (
                "Mono.Samples.HelloWorld.HelloAndroid, HelloWorld, Version=1.0.0.0, Culture=neutral,
                PublicKeyToken=null",
                "", this, new java.lang.Object[] { });
    }

    @Override
    public void onCreate (android.os.Bundle p0)
    {
        n_onCreate (p0);
    }

    private native void n_onCreate (android.os.Bundle p0);
}

```

请注意保留的基类，并为托管代码中重写每个方法提供了本机方法声明。

ExportAttribute 和 ExportFieldAttribute

通常情况下，Xamarin.Android 会自动生成包含 ACW; 的 Java 代码这一代基于时的类派生自 Java 类和重写现有的 Java 方法的类和方法名称。但是，在某些情况下，代码生成是不足够，如下所述：

- Android 支持操作名称中的布局 XML 特性，例如 `android:onClick` XML 特性。如果指定，增加的视图实例将尝试查找 Java 方法。
- `Java.io.Serializable` 接口要求 `readObject` 和 `writeObject` 方法。由于它们不是此接口的成员，因此我们相应的托管的实现不公开这些方法对 Java 代码。
- `Android.os.Parcelable` 接口需要实现类必须具有一个静态字段 `CREATOR` 类型的 `Parcelable.Creator`。生成的 Java 代码需要一些显式字段。与我们的标准方案，是用 Java 代码的输出字段无法从托管代码。

代码生成不提供解决方案，以生成具有随机名称的任意 Java 方法，因为从 Xamarin.Android 4.2 开始 `ExportAttribute` 和 `ExportFieldAttribute` 已引入了提供上述方案的解决方案。这两个属性位于 `Java.Interop` 命名空间：

- `ExportAttribute` – 指定方法名称和其预期的异常类型（若要为提供显式“抛出”在 Java 中）。在方法上使用它时，该方法将“导出”生成调度代码对相应 JNI 调用托管方法的 Java 方法。这可以用于 `android:onClick` 和 `java.io.Serializable`。
- `ExportFieldAttribute` – 指定的字段名称。它驻留在充当字段初始值设定项的方法。这可以用于 `android.os.Parcelable`。

`ExportAttribute` 示例项目演示了如何使用这些属性。

ExportAttribute 和 ExportFieldAttribute 故障排除

- 由于缺少打包失败 **Mono.Android.Export.dll** -如果您使用过 `ExportAttribute` 或 `ExportFieldAttribute` 您的代码或依赖库中的一些方法, 您必须添加 **Mono.Android.Export.dll**。此程序集是隔离, 以支持通过 Java 回调代码。它是独立于 **Mono.Android.dll** 因为它对应用程序添加了额外的大小。
- 在发布版本 `MissingMethodException` 发生的导出方法-中发布版本, `MissingMethodException` 发生的导出方法。(此问题已修复在 Xamarin.Android 的最新版本。)

ExportParameterAttribute

`ExportAttribute` 和 `ExportFieldAttribute` 运行时代码可以使用该 Java 提供的功能。此运行时代码生成的 JNI 方法取决于这些属性通过访问托管的代码。因此, 没有任何现有的 Java 方法的托管的方法绑定;因此, 从托管的方法签名生成 Java 方法。

但是, 这种情况下不是完全决定的。最值得注意的是, 这是在托管的类型和 Java 类型, 如之间的一些高级映射, 则返回 true:

- `InputStream`
- `OutputStream`
- `XmlPullParser`
- `XmlResourceParser`

如果需要为导出的方法, 这些类型 `ExportParameterAttribute` 必须用于显式提供相应的参数或返回值的类型。

批注特性

在 Xamarin.Android 4.2 我们转换 `IAnnotation` 到属性 (`System.Attribute`) 和添加了的对 Java 包装器中的批注生成的实现类型。

这意味着以下方向更改:

- 绑定生成器生成 `Java.Lang.DeprecatedAttribute` 从 `java.Lang.Deprecated` (虽然它应该是 `[Obsolete]` 在托管代码中)。
- 这并不意味着, 现有 `Java.Lang.Deprecated` 类会消失。(如果存在此类使用情况), 可以为常用的 Java 对象仍使用这些基于 Java 的对象。将有 `Deprecated` 和 `DeprecatedAttribute` 类。
- `Java.Lang.DeprecatedAttribute` 类标记为 `[Annotation]`。继承自的自定义属性时 `[Annotation]` 特性, msbuild 任务将生成该自定义属性的 Java 批注 (`@Deprecated`) 在 Android 可调用包装器 (ACW)。
- 批注无法生成到类、方法和导出字段 (这是在托管代码中的方法)。

如果未注册包含类 (批注的类本身或包含带批注的成员的类), 整个 Java 类不生成源, 包括的批注。对于方法, 可以指定 `ExportAttribute` 获取显式生成并批注的方法。此外, 它不是一项功能"生成"Java 批注类定义。换言之, 如果定义特定批注的自定义托管的属性, 需要添加另一个 jar 库, 它包含相应的 Java 批注类。添加用于定义批注类型的 Java 源代码文件是不够的。Java 编译器不能像那样 **apt**。

此外, 以下限制适用:

- 此转换过程不会考虑 `@Target` 上批注类型的批注到目前为止。
- 到属性的属性无效。改为使用为属性 getter 或 setter 的属性。

类绑定

绑定类意味着编写托管可调用包装器, 以简化的基础的 Java 类型的调用。

绑定虚拟和抽象方法, 以便允许重写从 C# 需要 Xamarin.Android 4.0。但是, 任何版本的 Xamarin.Android 可以绑定的非虚拟方法、静态方法或虚方法不支持替代情况下。

绑定通常包含以下各项：

- 一个要绑定的 Java 类型的句柄 JNI。
- JNI 字段 id 和每个绑定字段的属性。
- JNI 方法 id 和每个方法绑定方法。
- 如果子类是必需的需要具有类型 `RegisterAttribute` 上使用的类型声明的自定义属性 `RegisterAttribute.DoNotGenerateAcw` 设置为 `true`。

声明类型句柄

字段和方法查找方法要求其声明的类型引用的对象引用。按照约定，这将保存在 `class_ref` 字段：

```
static IntPtr class_ref = JNIEnv.FindClass(CLASS);
```

请参阅 [JNI 类型引用](#) 有关详细信息部分有关 `CLASS` 令牌。

绑定字段

Java 字段公开为 C# 属性，例如 Java 字段 `java.lang.System.in` 绑定为 C# 属性 `Java.Lang.JavaSystem.In`。此外，由于 JNI 区分静态字段和实例字段，实现属性时使用不同的方法。

字段绑定涉及到三个集的方法：

1. 获取字段 id 方法。获取字段 id 方法负责返回字段句柄 获取字段值 并 设置字段值 方法将使用。获取字段 id 需要知道声明类型，该字段的名称和 [JNI 类型签名](#) 的字段。
2. 获取字段值方法。这些方法需要字段句柄，并负责从 Java 读取字段的值。要使用的方法取决于字段的类型。
3. 设置字段值方法。这些方法需要字段句柄，并且负责编写在 Java 中的字段的值。要使用的方法取决于字段的类型。

静态字段 使用 `JNIEnv.GetStaticFieldID`，`JNIEnv.GetStatic*Field`，并 `JNIEnv.SetStaticField` 方法。

实例字段 使用 `JNIEnv.GetFieldID`，`JNIEnv.Get*Field`，并 `JNIEnv.SetField` 方法。

例如，静态属性 `JavaSystem.In` 可以作为实现：

```
static IntPtr in_jfieldID;
public static System.IO.Stream In
{
    get {
        if (in_jfieldId == IntPtr.Zero)
            in_jfieldId = JNIEnv.GetStaticFieldID (class_ref, "in", "Ljava/io/InputStream;");
        IntPtr __ret = JNIEnv.GetStaticObjectField (class_ref, in_jfieldId);
        return InputStreamInvoker.FromJniHandle (__ret, JniHandleOwnership.TransferLocalRef);
    }
}
```

注意：我们将使用 `InputStreamInvoker.FromJniHandle` 要转换到的 JNI 引用 `System.IO.Stream` 使用的实例，然后我们 `JniHandleOwnership.TransferLocalRef` 因为 `JNIEnv.GetStaticObjectField` 返回本地引用。

许多 `Android.Runtime` 类型具有 `FromJniHandle` 方法会将转换 JNI 引用到所需的类型。

方法绑定

Java 方法公开为 C# 方法以及 C# 属性。例如，Java 方法 `java.lang.Runtime.runFinalizersOnExit` 方法绑定为 `Java.Lang.Runtime.RunFinalizersOnExit` 方法，和 `java.lang.Object.getClass` 方法绑定为 `Java.Lang.Object.Class` 属性。

方法调用是一个两步过程：

1. 获取方法 `id` 为要调用的方法。获取方法 `id` 方法负责返回方法调用方法将使用的方法句柄。获取方法 `id` 需要知道声明类型，该方法的名称和 [JNI 类型签名](#) 的方法。
2. 调用方法。

就像使用字段，将使用以获取方法 `id` 并调用该方法的方法的静态方法和实例方法之间存在差异。

静态方法使用 `JNIEnv.GetStaticMethodID()` 查找方法 `id`，并使用 `JNIEnv.CallStatic*Method` 系列方法调用。

实例方法使用 `JNIEnv.GetMethodID` 查找方法 `id`，并使用 `JNIEnv.Call*Method` 和 `JNIEnv.CallNonvirtual*Method` 系列的调用的方法。

方法绑定是可能不止是方法调用。方法绑定还包括允许的方法重写（适用于抽象的并且非最终方法），或实现（适用于接口方法）。[支持继承接口](#)部分介绍了支持的虚拟方法和接口方法的复杂性。

静态方法

绑定的静态方法涉及到使用 `JNIEnv.GetStaticMethodID` 若要获取的方法句柄，然后使用相应

`JNIEnv.CallStatic*Method` 方法，具体取决于方法的返回类型。以下是有关绑定的示例 [Runtime.getRuntime](#) 方法：

```
static IntPtr id_getRuntime;

[Register ("getRuntime", "()Ljava/lang/Runtime;", "")]
public static Java.Lang.Runtime GetRuntime ()
{
    if (id_getRuntime == IntPtr.Zero)
        id_getRuntime = JNIEnv.GetStaticMethodID (class_ref,
            "getRuntime", "()Ljava/lang/Runtime;");

    return Java.Lang.Object.GetObject<Java.Lang.Runtime> (
        JNIEnv.CallStaticObjectMethod (class_ref, id_getRuntime),
        JniHandleOwnership.TransferLocalRef);
}
```

请注意，我们在静态字段中，存储方法句柄 `id_getRuntime`。这是一种性能优化，因此方法句柄不需要在每次调用上查找。不需要缓存方法句柄以这种方式。获取方法句柄后，`JNIEnv.CallStaticObjectMethod` 用于调用该方法。

`JNIEnv.CallStaticObjectMethod` 返回 `IntPtr` 其中包含返回的 Java 实例的句柄。

[Java.Lang.Object.GetObject<T>\(IntPtr, JniHandleOwnership\)](#) 用于将 Java 句柄转换为强类型化的对象实例。

非虚拟实例方法绑定

绑定 `final` 实例方法或实例方法不需要重写时，涉及到使用 `JNIEnv.GetMethodID` 若要获取的方法句柄，然后使用相应 `JNIEnv.Call*Method` 方法，具体取决于方法的返回类型。以下是有关绑定的示例 `Object.Class` 属性：

```
static IntPtr id_getClass;
public Java.Lang.Class Class {
    get {
        if (id_getClass == IntPtr.Zero)
            id_getClass = JNIEnv.GetMethodID (class_ref, "getClass", "()Ljava/lang/Class;");
        return Java.Lang.Object.GetObject<Java.Lang.Class> (
            JNIEnv.CallObjectMethod (Handle, id_getClass),
            JniHandleOwnership.TransferLocalRef);
    }
}
```

请注意，我们在静态字段中，存储方法句柄 `id_getClass`。这是一种性能优化，因此方法句柄不需要在每次调用上查找。不需要缓存方法句柄以这种方式。获取方法句柄后，`JNIEnv.CallStaticObjectMethod` 用于调用该方法。

`JNIEnv.CallStaticObjectMethod` 返回 `IntPtr` 其中包含返回的 Java 实例的句柄。

[Java.Lang.Object.GetObject<T>\(IntPtr, JniHandleOwnership\)](#) 用于将 Java 句柄转换为强类型化的对象实例。

绑定的构造函数

构造函数是具有名称的 Java 方法 `<init>`。只需与 Java 实例方法一样，`JNIEnv.GetMethodID` 用于查找的构造函数的句柄。与 Java 方法不同 `JNIEnv.NewObject` 方法用于调用构造函数方法句柄。返回值 `JNIEnv.NewObject` 是 JNI 本地引用：

```
int value = 42;
IntPtr class_ref = JNIEnv.FindClass ("java/lang/Integer");
IntPtr id_ctor_I = JNIEnv.GetMethodID (class_ref, "<init>", "(I)V");
IntPtr lrefInstance = JNIEnv.NewObject (class_ref, id_ctor_I, new JValue (value));
// Dispose of lrefInstance, class_ref...
```

类绑定通常将子类 `Java.Lang.Object`。子类化时 `Java.Lang.Object`、其他语义派上用场：`Java.Lang.Object` 的实例将保留通过 Java 实例的全局引用 `Java.Lang.Object.Handle` 属性。

1. `Java.Lang.Object` 默认构造函数将分配一个 Java 实例。
2. 如果该类型具有 `RegisterAttribute`，并 `RegisterAttribute.DoNotGenerateAcw` 是 `true`，然后实例 `RegisterAttribute.Name` 通过其默认构造函数创建类型。
3. 否则为 [Android 可调用包装器](#) (ACW) 对应于 `this.GetType` 通过其默认构造函数实例化。在为包创建过程中生成 android 可调用包装器每隔 `Java.Lang.Object` 为其子类 `RegisterAttribute.DoNotGenerateAcw` 未设置为 `true`。

类型不类绑定，这是预期语义：实例化 `Mono.Samples.HelloWorld.HelloAndroid` C#实例应构造 Java `mono.samples.helloworld.HelloAndroid` 实例，其中是生成 Android 可调用包装器。

对于类的绑定，这可能是正确的行为，如果 Java 类型包含一个默认构造函数和/或任何其他构造函数需要调用。否则，必须执行以下操作提供一个构造函数：

1. 调用 `Java.Lang.Object (IntPtr, JniHandleOwnership)` 而不是默认 `Java.Lang.Object` 构造函数。需要这些信息来避免创建新的 Java 实例。
2. 检查的值 `Java.Lang.Object.Handle` 之前创建的任何 Java 实例。`Object.Handle` 属性将具有一个值，而不是 `IntPtr.Zero` 如果 Android 可调用包装器构造在 Java 代码中，并且正在构造的类绑定以包含所创建的 Android 可调用包装器实例。例如，当 Android 创建 `mono.samples.helloworld.HelloAndroid` 实例，将创建 Android 可调用包装器，第一个和 Java `HelloAndroid` 构造函数将创建一个实例的相应 `Mono.Samples.HelloWorld.HelloAndroid` 类型，与 `Object.Handle` 属性将设置为在构造函数执行之前的 Java 实例。
3. 如果当前的运行时类型不相同声明类型，则相应的 Android 可调用包装器的实例必须创建并使用 `Object.SetHandle` 用于存储返回的句柄 `JNIEnv.CreateInstance`。
4. 如果当前的运行时类型声明的类型相同，然后调用 Java 构造函数，并使用 `Object.SetHandle` 用于存储返回的句柄 `JNIEnv.NewInstance`。

例如，考虑 `java.lang.Integer(int)` 构造函数。此绑定为：

```
// Cache the constructor's method handle for later use
static IntPtr id_ctor_I;

// Need [Register] for subclassing
// RegisterAttribute.Name is always ".ctor"
// RegisterAttribute.Signature is the JNI type signature of constructor
// RegisterAttribute.Connector is ignored; use ""
[Register (".ctor", "(I)V", "")]
public Integer (int value)
{
    // 1. Prevent Object default constructor execution
    : base (IntPtr.Zero, JniHandleOwnership.DoNotTransfer)

    // 2. Don't allocate Java instance if already allocated
    if (Handle != IntPtr.Zero)
        return;

    // 3. Derived type? Create Android Callable Wrapper
    if (GetType () != typeof (Integer)) {
        SetHandle (
            Android.Runtime.JNIEnv.CreateInstance (GetType (), "(I)V", new JValue (value)),
            JniHandleOwnership.TransferLocalRef);
        return;
    }

    // 4. Declaring type: lookup & cache method id...
    if (id_ctor_I == IntPtr.Zero)
        id_ctor_I = JNIEnv.GetMethodID (class_ref, "<init>", "(I)V");
    // ...then create the Java instance and store
    SetHandle (
        JNIEnv.NewObject (class_ref, id_ctor_I, new JValue (value)),
        JniHandleOwnership.TransferLocalRef);
}
```

[JNIEnv.CreateInstance](#)方法是帮助程序执行[JNIEnv.FindClass](#)，[JNIEnv.GetMethodID](#)，[JNIEnv.NewObject](#)，并且[JNIEnv.DeleteGlobalReference](#)从返回的值上[JNIEnv.FindClass](#)。有关详细信息，请参阅下一节。

支持继承，接口

子类化的 Java 类型或实现 Java 接口需要的新一代[Android 可调用包装器](#)(ACWs) 为生成每个 `Java.Lang.Object` 在打包过程的子类。通过控制 ACW 代[Android.Runtime.RegisterAttribute](#)自定义属性。

有关C#类型，[\[Register\]](#)自定义特性构造函数需要一个参数：[JNI 简化类型引用](#)为相应的 Java 类型。这允许提供 Java 之间不同的名称和C#。

Xamarin.Android 4.0 之前 [\[Register\]](#)自定义属性均不可用"alias"现有的 Java 类型。这是因为 ACW 生成过程会生成 ACWs 为每个 `Java.Lang.Object` 遇到子类。

Xamarin.Android 4.0 引入了[RegisterAttribute.DoNotGenerateAcw](#)属性。此属性指示到 ACW 生成过程跳过带批注的类型，允许的新托管可调用包装器不会导致 ACWs 正在创建包时生成的声明。这允许绑定现有的 Java 类型。例如，考虑以下简单的 Java 类 `Adder`，其中包含一种方法，`add`，用于将添加到整数并返回结果：

```
package mono.android.test;
public class Adder {
    public int add (int a, int b) {
        return a + b;
    }
}
```

`Adder` 无法作为绑定类型：

```
[Register ("mono/android/test/Adder", DoNotGenerateAcw=true)]
public partial class Adder : Java.Lang.Object {
    static IntPtr class_ref = JNIEnv.FindClass ( "mono/android/test/Adder");

    public Adder ()
    {
    }

    public Adder (IntPtr handle, JniHandleOwnership transfer)
        : base (handle, transfer)
    {
    }
}
partial class ManagedAdder : Adder {
}
```

在这里，`Adder` C#类型别名 `Adder` Java 类型。`[Register]` 特性用于指定的 JNI 名称 `mono.android.test.Adder` Java 类型和 `DoNotGenerateAcw` 属性用来抑制 ACW 生成。这将导致为 ACW 新一代 `ManagedAdder` 类型，可在正确子类 `mono.android.test.Adder` 类型。如果 `RegisterAttribute.DoNotGenerateAcw` 属性未被使用，则 Xamarin.Android 生成过程应已生成一个新 `mono.android.test.Adder` Java 类型。这会导致编译错误，因为 `mono.android.test.Adder` 类型都会出现两次，在两个单独的文件。

绑定的虚拟方法

`ManagedAdder` 子类 Java `Adder` 类型，但它不是特别有趣：C# `Adder` 类型未定义任何虚拟方法，因此 `ManagedAdder` 不能重写任何内容。

绑定 `virtual` 允许由子类重写的方法需要多项操作需要执行这分为以下两个类别：

1. 方法绑定
2. 方法注册

方法绑定

方法绑定需要添加到两个支持成员 C# `Adder` 定义：`ThresholdType`，并 `ThresholdClass`。

`ThresholdType`

`ThresholdType` 属性返回当前绑定的类型：

```
partial class Adder {
    protected override System.Type ThresholdType {
        get {
            return typeof (Adder);
        }
    }
}
```

`ThresholdType` 用于在方法绑定中确定何时应执行虚拟和非虚方法调度。应始终返回 `System.Type` 实例与声明对应的 C# 类型。

`ThresholdClass`

`ThresholdClass` 属性返回的绑定类型的 JNI 类引用：

```
partial class Adder {
    protected override IntPtr ThresholdClass {
        get {
            return class_ref;
        }
    }
}
```

`ThresholdClass` 在方法绑定中调用时所使用的非虚拟方法。

绑定实现

方法绑定实现负责的 Java 方法的运行时调用。它还包含 `[Register]` 是方法注册的一部分，将在方法注册部分中讨论的自定义特性声明：

```
[Register ("add", "(II)I", "GetAddHandler")]
public virtual int Add (int a, int b)
{
    if (id_add == IntPtr.Zero)
        id_add = JNIEnv.GetMethodID (class_ref, "add", "(II)I");
    if (GetType () == ThresholdType)
        return JNIEnv.CallIntMethod (Handle, id_add, new JValue (a), new JValue (b));
    return JNIEnv.CallNonvirtualIntMethod (Handle, ThresholdClass, id_add, new JValue (a), new JValue (b));
}
```

`id_add` 字段包含要调用的 Java 方法的方法 ID。`id_add` 值从获取 `JNIEnv.GetMethodID`，这需要声明类 (`class_ref`)，Java 方法名称 (`"add"`)，和方法的 JNI 签名 (`"(II)I"`)。

获取方法 ID 后，`GetType` 进行比较的 `ThresholdType` 以确定是否需要虚拟或非虚拟调度。虚拟调度时，必须 `GetType` 匹配 `ThresholdType`，作为 `Handle` 可能指 Java 分配的子类，这会重写该方法。

时 `GetType` 不匹配 `ThresholdType`，`Adder` 子类别 (例如通过 `ManagedAdder`)，并且 `Adder.Add` 如果子类调用，将只调用实现 `base.Add`。这种非虚拟调度情况，这是 where `ThresholdClass` 传入。`ThresholdClass` 指定的 Java 类将提供要调用的方法的实现。

方法注册

假设我们有的已更新 `ManagedAdder` 定义，这会重写 `Adder.Add` 方法：

```
partial class ManagedAdder : Adder {
    public override int Add (int a, int b) {
        return (a*2) + (b*2);
    }
}
```

请记住，`Adder.Add` 有 `[Register]` 自定义属性：

```
[Register ("add", "(II)I", "GetAddHandler")]
```

`[Register]` 自定义特性构造函数接受三个值：

1. Java 方法的名称 `"add"` 这种情况下。
2. JNI 类型签名的方法，`"(II)I"` 这种情况下。
3. 连接器方法，`GetAddHandler` 这种情况下。连接器方法将在下文。

前两个参数允许 ACW 生成过程生成一个方法声明重写的方法。生成 ACW 将包含某些下面的代码：

```

public class ManagedAdder extends mono.android.test.Adder {
    static final String __md_methods;
    static {
        __md_methods = "n_add:(II)I:GetAddHandler\n" +
            "";
        mono.android.Runtime.register (...);
    }
    @Override
    public int add (int p0, int p1) {
        return n_add (p0, p1);
    }
    private native int n_add (int p0, int p1);
    // ...
}

```

请注意, `@Override` 方法声明, 该委托给 `n_`-前缀相同名称的方法。这确保 Java 代码调用时 `ManagedAdder.add`, `ManagedAdder.n_add` 将调用, 以便重写 C# `ManagedAdder.Add` 要执行方法。

因此, 最重要的问题: 如何将 `ManagedAdder.n_add` 挂接到 `ManagedAdder.Add` ?

Java `native` 与 Java (Android 运行时) 运行时通过注册方法 [JNI RegisterNatives 函数](#)。 `RegisterNatives` 将遵循结构包含要调用的 Java 方法名称、JNI 类型签名和函数指针的数组 [JNI 调用约定](#)。函数指针必须是采用两个指针自变量跟方法参数的函数。Java `ManagedAdder.n_add` 必须通过具有以下 C 原型的函数的实现方法:

```

int FunctionName(JNIEnv *env, jobject this, int a, int b)

```

Xamarin.Android 不公开 `RegisterNatives` 方法。相反, ACW 和 MCW 一起提供的信息需要调用 `RegisterNatives`: ACW 包含方法名称和 JNI 类型签名, 现在只缺少挂接的函数指针。

这就是 [连接器方法](#) 传入。第三个 `[Register]` 自定义特性参数是在已注册的类型或不接受任何参数, 并返回的已注册类型的基类中定义的方法名称 `System.Delegate`。返回 `System.Delegate` 又是指具有正确的 JNI 函数签名的方法。最后, 该连接器方法返回的委托 **必须** 根路径, 使 GC 不会收集它, 因为委托提供给 Java。

```

#pragma warning disable 0169
static Delegate cb_add;
// This method must match the third parameter of the [Register]
// custom attribute, must be static, must return System.Delegate,
// and must accept no parameters.
static Delegate GetAddHandler ()
{
    if (cb_add == null)
        cb_add = JNINativeWrapper.CreateDelegate ((Func<IntPtr, IntPtr, int, int, int>) n_Add);
    return cb_add;
}
// This method is registered with JNI.
static int n_Add (IntPtr jnienv, IntPtr lrefThis, int a, int b)
{
    Adder __this = Java.Lang.Object.GetObject<Adder>(lrefThis, JniHandleOwnership.DoNotTransfer);
    return __this.Add (a, b);
}
#pragma warning restore 0169

```

`GetAddHandler` 方法创建 `Func<IntPtr, IntPtr, int, int, int>` 委托, 是指 `n_Add` 方法, 然后调用 [JNINativeWrapper.CreateDelegate](#)。 `JNINativeWrapper.CreateDelegate` 将提供的方法包装在 try/catch 块, 以便任何未经处理的异常处理, 将导致引发 [AndroidEvent.UnhandledExceptionRaiser](#) 事件。结果委托存储在静态 `cb_add` 变量, 以便 GC 不会释放该委托。

最后, `n_Add` 方法负责封送到相应的托管类型的 JNI 参数然后委派方法调用。

注意：请始终使用 `JniHandleOwnership.DoNotTransfer` MCW 获取对使用 Java 实例时。把它们当作本地引用 (并因此调用 `JNIEnv.DeleteLocalRef`) 将中断托管-> Java->托管堆栈转换。

完成 Adder 绑定

完全托管的绑定 `mono.android.tests.Adder` 类型是：

```
[Register ("mono/android/test/Adder", DoNotGenerateAcw=true)]
public class Adder : Java.Lang.Object {

    static IntPtr class_ref = JNIEnv.FindClass ("mono/android/test/Adder");

    public Adder ()
    {
    }

    public Adder (IntPtr handle, JniHandleOwnership transfer)
        : base (handle, transfer)
    {
    }

    protected override Type ThresholdType {
        get {return typeof (Adder);}
    }

    protected override IntPtr ThresholdClass {
        get {return class_ref;}
    }

    #region Add
    static IntPtr id_add;

    [Register ("add", "(II)I", "GetAddHandler")]
    public virtual int Add (int a, int b)
    {
        if (id_add == IntPtr.Zero)
            id_add = JNIEnv.GetMethodID (class_ref, "add", "(II)I");
        if (GetType () == ThresholdType)
            return JNIEnv.CallIntMethod (Handle, id_add, new JValue (a), new JValue (b));
        return JNIEnv.CallNonvirtualIntMethod (Handle, ThresholdClass, id_add, new JValue (a), new JValue
(b));
    }

    #pragma warning disable 0169
    static Delegate cb_add;
    static Delegate GetAddHandler ()
    {
        if (cb_add == null)
            cb_add = JNINativeWrapper.CreateDelegate ((Func<IntPtr, IntPtr, int, int, int>) n_Add);
        return cb_add;
    }

    static int n_Add (IntPtr jnienv, IntPtr lrefThis, int a, int b)
    {
        Adder __this = Java.Lang.Object.GetObject<Adder>(lrefThis, JniHandleOwnership.DoNotTransfer);
        return __this.Add (a, b);
    }
    #pragma warning restore 0169
    #endregion
}
```

限制

编写一种类型时满足以下条件：

1. 子类 `Java.Lang.Object`

2. 具有 `[Register]` 自定义属性

3. `RegisterAttribute.DoNotGenerateAcw` 为 `true`

然后 GC 交互类型对于不得有可能引用的任何字段 `Java.Lang.Object` 或 `Java.Lang.Object` 在运行时的子类。例如, 类型的字段 `System.Object` 和不允许使用任何接口类型。不能引用的类型 `Java.Lang.Object` 允许使用实例, 如 `System.String` 和 `List<int>`。此限制是为了防止 gc 过早对象集合。

如果该类型必须包含一个实例字段, 它可以指 `Java.Lang.Object` 实例, 则字段类型必须为 `System.WeakReference` 或 `GCHandle`。

绑定抽象方法

绑定 `abstract` 方法在很大程度上等同于绑定的虚拟方法。有只有两个区别:

1. 抽象方法是抽象类。它仍将保留 `[Register]` 属性和关联的方法注册, 方法绑定只需移动到 `Invoker` 类型。
2. 非 `abstract`Invoker` 创建类型的子类的抽象类型。 `Invoker` 类型必须重写基类中声明的所有抽象方法和重写的实现是绑定方法的实现, 但可以忽略非虚拟调度大小写。

例如, 假设上述 `mono.android.test.Adder.add` 方法已 `abstract`。C#绑定会更改, 以便 `Adder.Add` 了抽象, 和一个新 `AdderInvoker` 将其实现定义的类型 `Adder.Add`:

```
partial class Adder {
    [Register ("add", "(II)I", "GetAddHandler")]
    public abstract int Add (int a, int b);

    // The Method Registration machinery is identical to the
    // virtual method case...
}

partial class AdderInvoker : Adder {
    public AdderInvoker (IntPtr handle, JniHandleOwnership transfer)
        : base (handle, transfer)
    {
    }

    static IntPtr id_add;
    public override int Add (int a, int b)
    {
        if (id_add == IntPtr.Zero)
            id_add = JNIEnv.GetMethodID (class_ref, "add", "(II)I");
        return JNIEnv.CallIntMethod (Handle, id_add, new JValue (a), new JValue (b));
    }
}
```

`Invoker` 时获取对 Java 创建实例的 JNI 引用, 类型才是必需。

绑定接口

绑定接口是从概念上讲类似于绑定类, 其中包含的虚拟方法, 但许多细节在细微 (和小) 方面存在差异。请考虑以下 [Java 接口声明](#):

```
public interface Progress {
    void onAdd(int[] values, int currentIndex, int currentSum);
}
```

接口绑定都具有两个部分: C#接口定义和调用程序定义的接口。

接口定义

C#接口定义必须满足以下要求：

- 接口定义必须具有 `[Register]` 自定义属性。
- 接口定义必须扩展 `IJavaObject interface`。如果不这样做将阻止 ACWs 从 Java 接口继承。
- 每个接口方法必须包含 `[Register]` 属性，用于指定相应的 Java 方法名称、JNI 签名和连接器方法。
- 连接器方法还必须指定可以位于连接器方法的类型。

绑定时 `abstract` 和 `virtual` 内正在注册的类型的继承层次结构中搜索方法，该连接器方法。接口可具有包含正文，因此这不起作用，因此要求没有方法的指定类型，该值指示连接器方法所在的位置。在连接器方法字符串中，指定一个冒号之后的类型 `':'`，并且必须包含调用程序的类型的程序集限定的类型名称。

接口方法声明为相应的 Java 方法使用的翻译兼容类型。Java 内置类型兼容的类型是相应 C# 类型，例如 Java `int` 是 C# `int`。对于引用类型兼容的类型是可以提供适当的 Java 类型的 JNI 句柄的类型。

接口成员，将不会直接调用由 Java-将通过调用程序类型间接调用-以便允许一定程度的灵活性。

Java 进度接口可以是 [中声明C#作为](#)：

```
[Register ("mono/android/test/Adder$Progress", DoNotGenerateAcw=true)]
public interface IAdderProgress : IJavaObject {
    [Register ("onAdd", "([III)V",
        "GetOnAddHandler:Mono.Samples.SanityTests.IAdderProgressInvoker, SanityTests, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null")]
    void OnAdd (JavaArray<int> values, int currentIndex, int currentSum);
}
```

请注意，在上述我们映射 Java `int[]` 参数 `JavaArray<int>`。没有必要这样做：我们可能已经绑定到 C# `int[]`，或 `IList<int>`，或其他内容完全。选择的任何类型，则 `Invoker` 需要能够将其转换为 Java `int[]` 调用的类型。

调用程序定义

`Invoker` 类型定义必须继承 `Java.Lang.Object`、实现相应的接口，并提供在接口定义中引用的所有连接方法。还有一个不同于类绑定的更多建议：`class_ref` 字段和方法 `Id` 应为实例成员，非静态成员。

优先使用实例成员的原因都与 `JNIEnv.GetMethodID` 中 Android 运行时的行为。（这可能是 Java 的行为；它尚未测试）。`JNIEnv.GetMethodID` 查找来自于实现的接口和未声明的接口的方法时返回 `null`。请考

虑 `java.util.SortedMap<K, V>` Java 接口，实现 `java.util.Map<K, V>` 接口。映射提供清除方法，因此看起来合理

`Invoker SortedMap` 的定义将是：

```
// Fails at runtime. DO NOT FOLLOW
partial class ISortedMapInvoker : Java.Lang.Object, ISortedMap {
    static IntPtr class_ref = JNIEnv.FindClass ("java/util/SortedMap");
    static IntPtr id_clear;
    public void Clear()
    {
        if (id_clear == IntPtr.Zero)
            id_clear = JNIEnv.GetMethodID(class_ref, "clear", "()V");
        JNIEnv.CallVoidMethod(Handle, id_clear);
    }
    // ...
}
```

上述将失败，因为 `JNIEnv.GetMethodID` 将返回 `null` 查找时 `Map.clear` 方法通过 `SortedMap` 类实例。

有两个解决方案：跟踪每个方法的来源，哪个接口，并具有 `class_ref` 为每个接口，或保留为实例成员的所有内容，并派生程度最高的类类型，不是接口类型上执行方法查找。完成后者 **Mono.Android.dll**。

该调用程序定义了六个部分：构造函数中，`Dispose` 方法，`ThresholdType` 并 `ThresholdClass` 成员，`GetObject` 方法、接口方法实现和连接器方法实现。

构造函数

需要查找正在调用的实例的运行时类和实例中存储的运行时类的构造函数 `class_ref` 字段：

```
partial class IAdderProgressInvoker {
    IntPtr class_ref;
    public IAdderProgressInvoker (IntPtr handle, JNIEnvOwnership transfer)
        : base (handle, transfer)
    {
        IntPtr lref = JNIEnv.GetObjectClass (Handle);
        class_ref = JNIEnv.NewGlobalRef (lref);
        JNIEnv.DeleteLocalRef (lref);
    }
}
```

注意：`Handle` 属性必须使用在构造函数体内，而不 `handle` 参数，截止时间 Android v4.0 `handle` 基构造函数完成执行后参数可能无效。

Dispose 方法

`Dispose` 方法需要释放的构造函数中分配的全局引用：

```
partial class IAdderProgressInvoker {
    protected override void Dispose (bool disposing)
    {
        if (this.class_ref != IntPtr.Zero)
            JNIEnv.DeleteGlobalRef (this.class_ref);
        this.class_ref = IntPtr.Zero;
        base.Dispose (disposing);
    }
}
```

ThresholdType 和 ThresholdClass

`ThresholdType` 和 `ThresholdClass` 成员是相同的类绑定中找到的内容：

```
partial class IAdderProgressInvoker {
    protected override Type ThresholdType {
        get {
            return typeof (IAdderProgressInvoker);
        }
    }
    protected override IntPtr ThresholdClass {
        get {
            return class_ref;
        }
    }
}
```

GetObject 方法

一个静态 `GetObject` 支持所需的方法 `Extensions.JavaCast<T>()`：

```
partial class IAdderProgressInvoker {
    public static IAdderProgress GetObject (IntPtr handle, JNIEnvOwnership transfer)
    {
        return new IAdderProgressInvoker (handle, transfer);
    }
}
```

接口方法

需要有一个实现, 调用相应的 Java 方法通过 JNI 接口的每个方法:

```
partial class IAdderProgressInvoker {
    IntPtr id_onAdd;
    public void OnAdd (JavaArray<int> values, int currentIndex, int currentSum)
    {
        if (id_onAdd == IntPtr.Zero)
            id_onAdd = JNIEnv.GetMethodID (class_ref, "onAdd", "([III)V");
        JNIEnv.CallVoidMethod (Handle, id_onAdd, new JValue (JNIEnv.ToJniHandle (values)), new JValue
(currentIndex), new JValue (currentSum));
    }
}
```

连接器方法

连接器方法和支持基础结构负责封送处理到相应的 JNI 参数 C# 类型。Java `int[]` 参数将传递为 JNI `jintArray`, 即 `IntPtr` 中 C#。 `IntPtr` 必须封送到 `JavaArray<int>` 为了支持调用 C# 接口:

```
partial class IAdderProgressInvoker {
    static Delegate cb_onAdd;
    static Delegate GetOnAddHandler ()
    {
        if (cb_onAdd == null)
            cb_onAdd = JNINativeWrapper.CreateDelegate ((Action<IntPtr, IntPtr, IntPtr, int, int>) n_OnAdd);
        return cb_onAdd;
    }

    static void n_OnAdd (IntPtr jnienv, IntPtr lrefThis, IntPtr values, int currentIndex, int currentSum)
    {
        IAdderProgress __this = Java.Lang.Object.GetObject<IAdderProgress>(lrefThis,
JniHandleOwnership.DoNotTransfer);
        using (var _values = new JavaArray<int>(values, JniHandleOwnership.DoNotTransfer)) {
            __this.OnAdd (_values, currentIndex, currentSum);
        }
    }
}
```

如果 `int[]` 会通过首选 `JavaList<int>`, 然后 `JNIEnv.GetArray()` 也可改用:

```
int[] _values = (int[]) JNIEnv.GetArray(values, JniHandleOwnership.DoNotTransfer, typeof (int));
```

但请注意, `JNIEnv.GetArray` 将复制整个数组之间的 Vm, 因此对于大型数组这可能导致大量增加的 GC 压力。

完成调用程序定义

[完成 IAdderProgressInvoker 定义](#):

```

class IAdderProgressInvoker : Java.Lang.Object, IAdderProgress {

    IntPtr class_ref;

    public IAdderProgressInvoker (IntPtr handle, JniHandleOwnership transfer)
        : base (handle, transfer)
    {
        IntPtr lref = JNIEnv.GetObjectClass (Handle);
        class_ref = JNIEnv.NewGlobalRef (lref);
        JNIEnv.DeleteLocalRef (lref);
    }

    protected override void Dispose (bool disposing)
    {
        if (this.class_ref != IntPtr.Zero)
            JNIEnv.DeleteGlobalRef (this.class_ref);
        this.class_ref = IntPtr.Zero;
        base.Dispose (disposing);
    }

    protected override Type ThresholdType {
        get {return typeof (IAdderProgressInvoker);}
    }

    protected override IntPtr ThresholdClass {
        get {return class_ref;}
    }

    public static IAdderProgress GetObject (IntPtr handle, JniHandleOwnership transfer)
    {
        return new IAdderProgressInvoker (handle, transfer);
    }

    #region OnAdd
    IntPtr id_onAdd;
    public void OnAdd (JavaArray<int> values, int currentIndex, int currentSum)
    {
        if (id_onAdd == IntPtr.Zero)
            id_onAdd = JNIEnv.GetMethodID (class_ref, "onAdd",
                "([III)V");
        JNIEnv.CallVoidMethod (Handle, id_onAdd,
            new JValue (JNIEnv.ToJniHandle (values)),
            new JValue (currentIndex),
            new JValue (currentSum));
    }

    #pragma warning disable 0169
    static Delegate cb_onAdd;
    static Delegate GetOnAddHandler ()
    {
        if (cb_onAdd == null)
            cb_onAdd = JNINativeWrapper.CreateDelegate ((Action<IntPtr, IntPtr, IntPtr, int, int>) n_OnAdd);
        return cb_onAdd;
    }

    static void n_OnAdd (IntPtr jnienv, IntPtr lrefThis, IntPtr values, int currentIndex, int currentSum)
    {
        IAdderProgress __this = Java.Lang.Object.GetObject<IAdderProgress>(lrefThis,
            JniHandleOwnership.DoNotTransfer);
        using (var _values = new JavaArray<int>(values, JniHandleOwnership.DoNotTransfer)) {
            __this.OnAdd (_values, currentIndex, currentSum);
        }
    }

    #pragma warning restore 0169
    #endregion
}

```

JNI 对象引用

很多 JNIEnv 方法返回 *JNI 的对象引用*，这是类似于 `GHandles`。JNI 提供三种不同类型的对象引用：本地引用、全局引用和全局的弱引用。所有这三个表示为 `System.IntPtr`，但(根据 JNI 函数类型部分中)不是所有 `IntPtr` s 从返回 `JNIEnv` 方法是引用。例如，`JNIEnv.GetMethodID` 返回 `IntPtr`，但它不会返回一个对象引用，它会返回 `jmethodID`。请查阅 [JNI 函数文档](#) 有关详细信息。

创建本地引用大多数引用创建方法。Android 仅允许有限的数量的本地引用，以在任何给定时间，通常 512 存在。可以通过删除本地引用 `JNIEnv.DeleteLocalRef`。与不同的 JNI，并非所有引用 `JNIEnv` 方法的返回对象的引用返回本地引用；`JNIEnv.FindClass` 返回 *全局引用*。强烈建议快速可以可能是通过构造来删除本地引用 `Java.Lang.Object` 围绕对象并指定 `JniHandleOwnership.TransferLocalRef` 到 `Java.Lang.Object (IntPtr 处理, JniHandleOwnership 传输)` 构造函数。

创建全局引用 `JNIEnv.NewGlobalRef` 并 `JNIEnv.FindClass`。它们可以与销毁 `JNIEnv.DeleteGlobalRef`。仿真程序具有限制为 2,000 未完成的全局引用，而硬件设备具有的限制为大约 52,000 全局引用。

Android v2.2 (Froyo) 及更高版本，弱全局引用才可用。可以使用删除全局的弱引用 `JNIEnv.DeleteWeakGlobalRef`。

处理 JNI 本地引用

`JNIEnv.GetObjectField`，`JNIEnv.GetStaticObjectField`，`JNIEnv.CallObjectMethod`，`JNIEnv.CallNonvirtualObjectMethod` 并 `JNIEnv.CallStaticObjectMethod` 方法返回 `IntPtr` 其中包含 Java 对象的 JNI 本地引用或 `IntPtr.Zero` 如果 Java 返回 `null`。由于有限数量的本地（512 个条目），它是需要确保引用后，可以是在未完成的引用会及时删除。有三种本地引用可处理的方法：显式删除它们，创建 `Java.Lang.Object` 实例，用于保存它们，并使用 `Java.Lang.Object.GetObject<T>()` 创建针对这些托管的可调用包装器。

显式删除本地引用

`JNIEnv.DeleteLocalRef` 用于删除本地引用。一旦本地引用已被删除，就不能使用它，因此必须格外小心，确保 `JNIEnv.DeleteLocalRef` 是通过本地引用的最后一步。

```
IntPtr lref = JNIEnv.CallObjectMethod(instance, methodID);
try {
    // Do something with `lref`
}
finally {
    JNIEnv.DeleteLocalRef (lref);
}
```

使用 Java.Lang.Object 包装

`Java.Lang.Object` 提供了 `Java.Lang.Object (IntPtr 句柄, JniHandleOwnership 传输)` 构造函数可以用来包装现有的 JNI 引用它。`JniHandleOwnership` 参数确定如何 `IntPtr` 应视为参数：

- `JniHandleOwnership.DoNotTransfer` – 创建 `Java.Lang.Object` 实例中将创建新的全局引用 `handle` 参数，和 `handle` 保持不变。调用方负责释放到 `handle`，如果有必要。
- `JniHandleOwnership.TransferLocalRef` – 创建 `Java.Lang.Object` 实例中将创建新的全局引用 `handle` 参数，并且 `handle` 使用删除 `JNIEnv.DeleteLocalRef`。调用方必须释放 `handle`，并且必须使用 `handle` 构造函数完成执行之后。
- `JniHandleOwnership.TransferGlobalRef` – 创建 `Java.Lang.Object` 实例将接管其所有权的 `handle` 参数。调用方必须释放 `handle`。

因为 JNI 方法调用方法返回本地 refs `JniHandleOwnership.TransferLocalRef` 通常使用：

```
IntPtr lref = JNIEnv.CallObjectMethod(instance, methodID);
var value = new Java.Lang.Object (lref, JniHandleOwnership.TransferLocalRef);
```

前创建的全局引用将不会释放 `Java.Lang.Object` 实例进行垃圾回收。如果可以，则释放该实例将释放全局引用，加快垃圾回收：

```
IntPtr lref = JNIEnv.CallObjectMethod(instance, methodID);
using (var value = new Java.Lang.Object (lref, JniHandleOwnership.TransferLocalRef)) {
    // use value ...
}
```

使用 `Java.Lang.Object.GetObject<T>()`

`Java.Lang.Object` 提供了 `Java.Lang.Object.GetObject<T>(IntPtr 句柄, JniHandleOwnership 传输)` 方法，可以用来创建指定类型的托管可调用包装器。

类型 `T` 必须满足以下要求：

1. `T` 必须是引用类型。
2. `T` 必须实现 `IJavaObject` 接口。
3. 如果 `T` 不是抽象类或接口，然后 `T` 必须为一个构造函数提供的参数类型 `(IntPtr, JniHandleOwnership)`。
4. 如果 `T` 是一个抽象类或接口，有 *必须会调用程序* 可用于 `T`。调用程序是一种非抽象类型，继承 `T` 或实现 `T`，并且具有相同的名称作为 `T` 与调用程序后缀。例如，如果 `T` 是接口 `Java.Lang.IRunnable`，然后键入 `Java.Lang.IRunnableInvoker` 必须存在，并且必须包含所需 `(IntPtr, JniHandleOwnership)` 构造函数。

因为 JNI 方法调用方法返回本地 refs `JniHandleOwnership.TransferLocalRef` 通常使用：

```
IntPtr lrefString = JNIEnv.CallObjectMethod(instance, methodID);
Java.Lang.String value = Java.Lang.Object.GetObject<Java.Lang.String>( lrefString,
JniHandleOwnership.TransferLocalRef);
```

查找 Java 类型

若要查找的字段或方法中 JNI，字段或方法的声明类型必须查找第一个。

`Android.Runtime.JNIEnv.FindClass(string)` 方法用于查找 Java 类型。字符串参数是 *简化类型引用* 或 *完整类型引用* Java 类型。请参阅 [JNI 类型参考资料部分](#) 有关简化和完整的类型引用的详细信息。

注意：不同于其他所有 `JNIEnv` 方法可返回对象实例 `FindClass` 返回全局引用，而不是本地引用。

实例字段

通过操作时字段 *字段 Id*。通过获取的字段 Id `JNIEnv.GetFieldID`，这需要在类的字段定义中的字段的名称和 *JNI 类型签名* 的字段。

字段 Id 不需要释放，并且有效，只要加载相应的 Java 类型。（android 不当前不支持卸载类。）

有两个组的操作实例字段的方法：一个用于读取实例字段，另一个用于编写实例字段。所有组方法都需要读取或写入的字段值的字段 ID。

读取实例字段值

组用于读取实例字段值的方法遵循命名模式：


```
* JNIEnv.Get*Field(IntPtr instance, IntPtr fieldID);
```

其中 `*` 是字段的类型：

- `JNIEnv.GetObjectField` – 读取并不是内置类型，如任何实例字段的值 `java.lang.Object`，数组和接口类型。返回的值是 JNI 本地引用。
- `JNIEnv.GetBooleanField` – 读取的值 `bool` 实例字段。
- `JNIEnv.GetByteField` – 读取的值 `sbyte` 实例字段。
- `JNIEnv.GetCharField` – 读取的值 `char` 实例字段。
- `JNIEnv.GetShortField` – 读取的值 `short` 实例字段。
- `JNIEnv.GetIntField` – 读取的值 `int` 实例字段。
- `JNIEnv.GetLongField` – 读取的值 `long` 实例字段。
- `JNIEnv.GetFloatField` – 读取的值 `float` 实例字段。
- `JNIEnv.GetDoubleField` – 读取的值 `double` 实例字段。

正在写入实例字段值

组用于编写实例字段值的方法遵循命名模式：

```
JNIEnv.SetField(IntPtr instance, IntPtr fieldID, Type value);
```

其中 `类型` 是字段的类型：

- `JNIEnv.SetField` – 编写的并不是内置类型，如任何字段值 `java.lang.Object`，数组和接口类型。`IntPtr` 值可能为 JNI 本地引用、JNI 全局引用、JNI 弱全局引用，或 `IntPtr.Zero`（对于 `null`）。
- `JNIEnv.SetField` – 的值写入 `bool` 实例字段。
- `JNIEnv.SetField` – 的值写入 `sbyte` 实例字段。
- `JNIEnv.SetField` – 的值写入 `char` 实例字段。
- `JNIEnv.SetField` – 的值写入 `short` 实例字段。
- `JNIEnv.SetField` – 的值写入 `int` 实例字段。
- `JNIEnv.SetField` – 的值写入 `long` 实例字段。
- `JNIEnv.SetField` – 的值写入 `float` 实例字段。
- `JNIEnv.SetField` – 的值写入 `double` 实例字段。

静态字段

通过操作时的静态字段 `字段 Id`。通过获取的字段 `Id` `JNIEnv.GetStaticFieldID`，这需要在类的字段定义中的字段的名称和 `JNI 类型签名` 的字段。

字段 `Id` 不需要释放，并且有效，只要加载相应的 Java 类型。（android 不当前不支持卸载类。）

有两个组的操作的静态字段的方法：一个用于读取实例字段，另一个用于编写实例字段。所有组方法都需要读取或写入的字段值的字段 `ID`。

读取静态字段值

组用于读取静态字段值的方法遵循命名模式：

```
* JNIEnv.GetStatic*Field(IntPtr class, IntPtr fieldID);
```

其中 * 是字段的类型：

- [JNIEnv.GetStaticObjectField](#) - 读取的任何静态字段，并不是内置类型，如值 `java.lang.Object`，数组和接口类型。返回的值是 JNI 本地引用。
- [JNIEnv.GetStaticBooleanField](#) - 读取的值 `bool` 静态字段。
- [JNIEnv.GetStaticByteField](#) - 读取的值 `sbyte` 静态字段。
- [JNIEnv.GetStaticCharField](#) - 读取的值 `char` 静态字段。
- [JNIEnv.GetStaticShortField](#) - 读取的值 `short` 静态字段。
- [JNIEnv.GetStaticLongField](#) - 读取的值 `long` 静态字段。
- [JNIEnv.GetStaticFloatField](#) - 读取的值 `float` 静态字段。
- [JNIEnv.GetStaticDoubleField](#) - 读取的值 `double` 静态字段。

正在写入静态字段值

组用于写入静态字段值的方法遵循命名模式：

```
JNIEnv.SetStaticField(IntPtr class, IntPtr fieldID, Type value);
```

其中 类型 是字段的类型：

- [JNIEnv.SetStaticField](#) - 编写的并不是内置类型，如任何静态字段值 `java.lang.Object`，数组和接口类型。
`IntPtr` 值可能为 JNI 本地引用、JNI 全局引用、JNI 弱全局引用，或 `IntPtr.Zero` (对于 `null`)。
- [JNIEnv.SetStaticField](#) - 的值写入 `bool` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `sbyte` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `char` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `short` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `int` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `long` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `float` 静态字段。
- [JNIEnv.SetStaticField](#) - 的值写入 `double` 静态字段。

实例方法

通过调用实例方法 `Id`。通过获取的方法 `Id` [JNIEnv.GetMethodID](#)，其需要的类型，该方法定义的方法名称中和 [JNI 类型签名](#) 的方法。

方法 `Id` 不需要释放，并且有效，只要加载相应的 Java 类型。（android 不当前不支持卸载类。）

有两组用于调用方法的方法：一个用于几乎，调用方法，一个用于非虚拟调用方法。这两个组方法需要一个方法 `ID` 来调用的方法和非虚拟调用还要求您指定应调用哪个类实现。

接口方法内的声明类型; 仅按键查找不能查找来自扩展/继承接口的方法。请参阅后续绑定接口 / 调用程序实现部

分, 了解更多详细信息。

在类中声明的任何方法或可以查找任何基类或实现的接口。

虚拟方法调用

组用于调用方法的方法几乎遵循命名模式:

```
* JNIEnv.Call*Method( IntPtr instance, IntPtr methodID, params JValue[] args );
```

其中 `*` 是该方法的返回类型。

- `JNIEnv.CallObjectMethod` –调用的方法, 它会返回非内置类型, 如 `java.lang.Object`、数组和接口。返回的值是 JNI 本地引用。
- `JNIEnv.CallBooleanMethod` –调用的方法可返回 `bool` 值。
- `JNIEnv.CallByteMethod` –调用的方法可返回 `sbyte` 值。
- `JNIEnv.CallCharMethod` –调用的方法可返回 `char` 值。
- `JNIEnv.CallShortMethod` –调用的方法可返回 `short` 值。
- `JNIEnv.CallLongMethod` –调用的方法可返回 `long` 值。
- `JNIEnv.CallFloatMethod` –调用的方法可返回 `float` 值。
- `JNIEnv.CallDoubleMethod` –调用的方法可返回 `double` 值。

非虚拟方法调用

方法用于调用方法的一非几乎遵循命名模式:

```
* JNIEnv.CallNonvirtual*Method( IntPtr instance, IntPtr class, IntPtr methodID, params JValue[] args );
```

其中 `*` 是该方法的返回类型。非虚拟方法调用通常用于调用虚方法的基方法。

- `JNIEnv.CallNonvirtualObjectMethod` –非虚拟调用的方法, 它会返回非内置类型, 如 `java.lang.Object`、数组和接口。返回的值是 JNI 本地引用。
- `JNIEnv.CallNonvirtualBooleanMethod` –非虚拟调用的方法可返回 `bool` 值。
- `JNIEnv.CallNonvirtualByteMethod` –非虚拟调用的方法可返回 `sbyte` 值。
- `JNIEnv.CallNonvirtualCharMethod` –非虚拟调用的方法可返回 `char` 值。
- `JNIEnv.CallNonvirtualShortMethod` –非虚拟调用的方法可返回 `short` 值。
- `JNIEnv.CallNonvirtualLongMethod` –非虚拟调用的方法可返回 `long` 值。
- `JNIEnv.CallNonvirtualFloatMethod` –非虚拟调用的方法可返回 `float` 值。
- `JNIEnv.CallNonvirtualDoubleMethod` –非虚拟调用的方法可返回 `double` 值。

静态方法

通过调用静态方法 `方法 Id`。通过获取的方法 Id `JNIEnv.GetStaticMethodID`, 其需要的类型, 该方法定义的方法名称中和 `JNI 类型签名` 的方法。

方法 Id 不需要释放, 并且有效, 只要加载相应的 Java 类型。(android 不当前不支持卸载类。)

静态方法调用

组用于调用方法的方法几乎遵循命名模式：

```
* JNIEnv.CallStatic*Method( IntPtr class, IntPtr methodID, params JValue[] args );
```

其中 `*` 是该方法的返回类型。

- `JNIEnv.CallStaticObjectMethod` –调用的静态方法，它会返回非内置类型，如 `java.lang.Object`、数组和接口。返回的值是 JNI 本地引用。
- `JNIEnv.CallStaticBooleanMethod` –调用的静态方法，它将返回 `bool` 值。
- `JNIEnv.CallStaticByteMethod` –调用的静态方法，它将返回 `sbyte` 值。
- `JNIEnv.CallStaticCharMethod` –调用的静态方法，它将返回 `char` 值。
- `JNIEnv.CallStaticShortMethod` –调用的静态方法，它将返回 `short` 值。
- `JNIEnv.CallStaticLongMethod` –调用的静态方法，它将返回 `long` 值。
- `JNIEnv.CallStaticFloatMethod` –调用的静态方法，它将返回 `float` 值。
- `JNIEnv.CallStaticDoubleMethod` –调用的静态方法，它将返回 `double` 值。

JNI 类型签名

`JNI 类型签名`都`JNI 类型引用`（但不简化的类型引用）的方法除外。使用方法时，JNI 类型签名是左括号 `'('` 后，跟所有类型连接在一起（不带不分隔开来将以逗号分隔或任何其他内容）后，跟右括号的参数的类型引用 `')`，后跟方法返回类型的 JNI 类型引用。

例如，对于给定的 Java 方法：

```
long f(int n, String s, int[] array);
```

将 JNI 类型签名：

```
(Ljava/lang/String;[I)J
```

一般情况下，它是 *强* 建议使用 `javap` 命令，以确定 JNI 签名。例如，JNI 类型的签名 `java.lang.Thread.State.valueOf(String)` 方法是 `"(Ljava/lang/字符串;) Ljava/lang/线程$ 状态;"`，而 JNI 键入的签名 `java.lang.Thread.State.values` 方法是 `"() [Ljava/lang/线程$ 状态;"`。尾随分号；请注意这些是 JNI 类型签名的一部分。

JNI 类型引用

从 Java 类型引用不同 JNI 类型引用。不能使用完全限定的 Java 类型名称，例如 `java.lang.String` 使用 JNI，必须改为使用 JNI 变体 `"java/lang/String"` 或 `"Ljava/lang/String;"`，具体取决于上下文；请参阅下面有关详细信息。有四种类型的 JNI 类型引用：

- **built-in**
- 简化
- **type**
- **array**

内置类型的引用

内置类型的引用是用来引用内置值类型的单个字符。映射为按如下所示：

- "B" 有关 `sbyte`。
- "S" 有关 `short`。
- "I" 有关 `int`。
- "J" 有关 `long`。
- "F" 有关 `float`。
- "D" 有关 `double`。
- "C" 有关 `char`。
- "Z" 有关 `bool`。
- "V" 有关 `void` 方法返回类型。

简化的类型引用

简化的类型引用仅可在 `JNIEnv.FindClass(string)`。有两种方法来派生简单的类型引用：

1. 从完全限定的 Java 名称，将为每个 `'.'` 中的包名称和类型名与之前 `'/'`，和每个 `'.'` 内类型名与 `'$'`。
2. 读取输出的 `'unzip -l android.jar | grep JavaName'`。

二者会导致 Java 类型 `java.lang.Thread.State` 映射到的简化的类型引用 `java/lang/Thread$State`。

类型引用

类型引用为内置类型引用或使用简化的类型引用 `'L'` 前缀和一个 `';'` 后缀。Java 类型 `java.lang.String`，则简化的类型引用的是 `"java/lang/String"`，而类型引用为 `"Ljava/lang/String;"`。

与数组类型引用和使用 JNI 签名使用类型引用。

若要获取的类型引用另一种方式是通过读取的输出 `'javap -s -classpath android.jar fully.qualified.Java.Name'`。具体取决于类型涉及到，您可以使用构造函数声明或方法返回类型，以确定 JNI 名称。例如：

```
$ javap -classpath android.jar -s java.lang.Thread.State
Compiled from "Thread.java"
```

```
public final class java.lang.Thread$State extends java.lang.Enum{
    public static final java.lang.Thread$State NEW;
        Signature: Ljava/lang/Thread$State;
    public static final java.lang.Thread$State RUNNABLE;
        Signature: Ljava/lang/Thread$State;
    public static final java.lang.Thread$State BLOCKED;
        Signature: Ljava/lang/Thread$State;
    public static final java.lang.Thread$State WAITING;
        Signature: Ljava/lang/Thread$State;
    public static final java.lang.Thread$State TIMED_WAITING;
        Signature: Ljava/lang/Thread$State;
    public static final java.lang.Thread$State TERMINATED;
        Signature: Ljava/lang/Thread$State;
    public static java.lang.Thread$State[] values();
        Signature: ()[Ljava/lang/Thread$State;
    public static java.lang.Thread$State valueOf(java.lang.String);
        Signature: (Ljava/lang/String;)Ljava/lang/Thread$State;
    static {};
        Signature: ()V
}
```

`Thread.State` 因此，我们可以使用的签名是 Java 枚举类型，`valueOf` 方法确定的类型引用为 `Ljava/lang/线程$状态;`。

数组类型引用

数组类型引用 `'['` JNI 类型引用的前缀。指定数组时, 不能使用简化的类型引用。

例如, `int[]` 是 `"[I"`, `int[][]` 是 `"[[I"`, 并 `java.lang.Object[]` 是 `"[Ljava/lang/Object;"`。

Java 泛型和类型擦除

大多数情况下, 通过 JNI, Java 泛型所示不存在。有一些"褶皱", 但这些褶皱正在 Java 如何与泛型, 不使用 JNI 如何查找和调用泛型成员进行交互。

通过 JNI 进行交互时的泛型类型或成员和非泛型类型或成员之间没有差异。例如, 泛型类型 `java.lang.Class<T>` 也是"原始"的泛型类型 `java.lang.Class`, 这两个具有相同的简化的类型引用, `"java/lang/Class"`。

Java 本机接口支持

`Android.Runtime.JNIEnv` 是 Java 本机接口 (JNI) 有关的托管的包装。内声明的 JNI 函数 [Java 本机接口规范](#) 中使用, 但方法已更改, 以删除的显式 `JNIEnv*` 参数和 `IntPtr` 而不是使用 `jobject`, `jclass`, `jmethodID`, 等等。例如, 考虑 [JNI NewObject](#) 函数:

```
jobject NewObjectA(JNIEnv *env, jclass clazz, jmethodID methodID, jvalue *args);
```

这作为公开 `JNIEnv.NewObject` 方法:

```
public static IntPtr NewObject(IntPtr clazz, IntPtr jmethod, params JValue[] parms);
```

将两个调用之间的转换是相当简单。在 C 中您必须:

```
jobject CreateMapActivity(JNIEnv *env)
{
    jclass    Map_Class    = (*env)->FindClass(env, "mono/samples/googlemaps/MyMapActivity");
    jmethodID Map_defCtor = (*env)->GetMethodID (env, Map_Class, "<init>", "()V");
    jobject   instance     = (*env)->NewObject (env, Map_Class, Map_defCtor);

    return instance;
}
```

C#等效将是:

```
IntPtr CreateMapActivity()
{
    IntPtr Map_Class    = JNIEnv.FindClass ("mono/samples/googlemaps/MyMapActivity");
    IntPtr Map_defCtor = JNIEnv.GetMethodID (Map_Class, "<init>", "()V");
    IntPtr instance     = JNIEnv.NewObject (Map_Class, Map_defCtor);

    return instance;
}
```

保存在一个 `IntPtr` 的 Java 对象实例后, 可能需要对其执行操作。您可以使用 `JNIEnv` 方法, 如 `JNIEnv.CallVoidMethod()` 若要执行此操作, 但如果已存在相似之处 C# 包装器, 则你将想要通过 JNI 引用构造一个包装器。可以通过执行 `Extensions.JavaCast()` 扩展方法:

```
IntPtr lrefActivity = CreateMapActivity();

// imagine that Activity were instead an interface or abstract type...
Activity mapActivity = new Java.Lang.Object(lrefActivity, JniHandleOwnership.TransferLocalRef)
    .JavaCast<Activity>();
```

此外可以使用[Java.Lang.Object.GetObject\(\)](#) 方法：

```
IntPtr lrefActivity = CreateMapActivity();

// imagine that Activity were instead an interface or abstract type...
Activity mapActivity = Java.Lang.Object.GetObject<Activity>(lrefActivity,
    JniHandleOwnership.TransferLocalRef);
```

此外，所有的 JNI 函数已被修改的删除 `JNIEnv*` 参数中的每个 JNI 函数存在。

总结

直接使用 JNI 处理是一种可怕的经验，应不惜一切代价避免使用。遗憾的是，它并不总是能够避免；希望本指南适用于 Android 命中与 Mono 的未绑定的 Java 用例时将提供一些帮助。

相关链接

- [SanityTests \(示例\)](#)
- [Java 本机接口规范](#)
- [Java 本机接口函数](#)

将 Java 移植到C#

2018/10/26 • [Edit Online](#)

第三个选项为 *Xamarin.Android* 应用程序中使用 Java Java 源代码移植到C#。

概述

此方法可能感兴趣的组织的：

- 从 Java 到切换技术堆栈C#。
- 必须维护C#和相同的产品的 Java 版本。
- 想要有常用的 Java 库的.NET 版本。

有两种方法以 Java 代码移植到C#。第一种方法是手动将代码移植。这涉及到熟练的开发人员了解.NET 和 Java 和熟悉每种语言的正确惯例。这种方法有意义的最少量的代码，或希望完全摆脱 Java 到组织C#。

第二个迁移方法是尝试并自动完成该过程使用的代码转换器，如[锐化](#)。[锐化](#)是从 Versant 最初用于端口的代码已开放源代码转换器 *db4o* 从 Java 到C#。*db4o* 是一种面向对象的数据库 Versant 开发在 Java 中，并在然后移植到.NET。使用代码转换器可能会有意义的项目，必须存在于这两种语言，并且需要一些两者之间的奇偶校验。

当自动化的代码转换工具有意义的示例所示[ngit](#)项目。*Ngit* 是 Java 项目的端口 *jgit*。*Jgit* 本身是 Java 实现 [Git](#) 源代码管理系统。若要生成C#通过 Java, *ngit* 程序员使用自定义自动系统来从 *jgit* 提取 Java 代码，应用一些修补程序，以容纳转换过程中，然后运行[锐化](#)，生成的代码C#代码。这允许 *ngit* 项目能够受益于 *jgit* 完成的持续、正在进行工作。

通常非完成大量工作所涉及的引导的自动化的代码转换工具，并且这可能会证明是要使用的障碍。在许多情况下，它可能是更简单、更轻松地为端口 JavaC#手动。

相关链接

- [通过转换工具](#)

绑定 Java 库

2018/10/26 • [Edit Online](#)

Android 社区有很多您可能想在您的应用程序; 若要使用的 Java 库本指南介绍如何将 Java 库合并到 Xamarin.Android 应用程序, 通过创建绑定库。

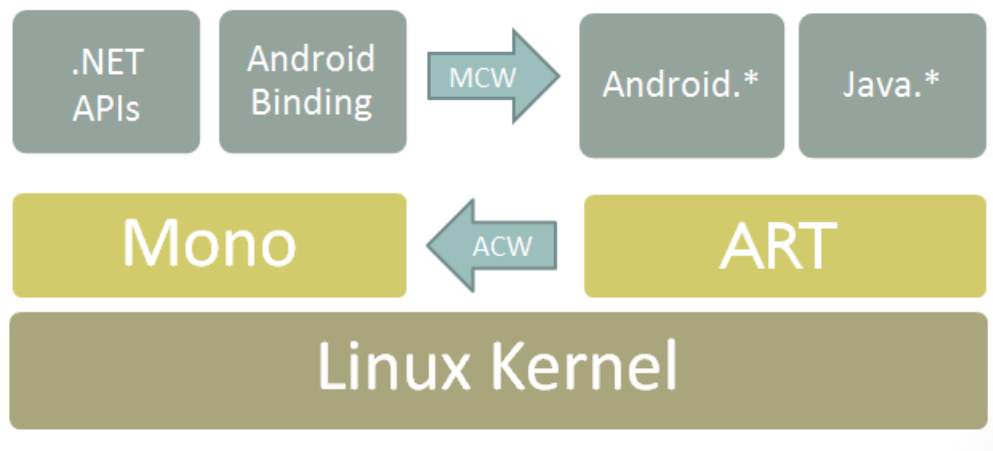
概述

适用于 Android 的第三方库生态系统是巨大的。因此, 通常最好使用比创建一个新的现有 Android 库。Xamarin.Android 提供两种方法来使用这些库:

- 创建 *绑定库* 自动包装与库 C# 包装器, 因此可以调用 Java 代码通过 C# 的调用。
- 使用 *Java 本机接口 (JNI)* 来直接调用 Java 库代码中的调用。JNI 是一个编程框架, 允许调用并由本机应用程序或库调用的 Java 代码。

本指南将介绍第一个选项: 如何创建 *绑定库*, 将一个或多个现有的 Java 库包装成可以在应用程序中链接到程序集。有关使用 JNI 的详细信息, 请参阅 [使用 JNI](#)。

Xamarin.Android 使用来实现 *绑定托管可调包装器 (MCW)*。MCW 是托管的代码需要调用 Java 代码时使用的 JNI 桥梁。子类化 Java 类型和重写虚拟方法在 Java 类型, 托管的可调用包装器还支持。同样, 只要 Android 运行时 (艺术) 代码想要调用托管的代码, 它会通过已知为 Android 可调包装器 (ACW) 的另一个 JNI 桥。这 [体系结构](#) 下图所示:



绑定库是包含管理可调包装器的 Java 类型的程序集。例如, 下面是一个 Java 类型, `MyClass`, 我们想要在绑定库自动换行:

```
package com.xamarin.mycode;

public class MyClass
{
    public String myMethod (int i) { ... }
}
```

我们生成一个绑定库, 用于以后 `.jar`, 其中包含 `MyClass`, 我们可以实例化它并从其上调用方法 C#:

```
var instance = new MyClass ();

string result = instance.MyMethod (42);
```

若要创建此绑定库，请使用 Xamarin.Android *Java 绑定库* 模板。生成的绑定项目与 MCW 类创建 .NET 程序集 **.jar** 文件，并适用于 Android 库项目中嵌入资源。您也可以为 Android 存档创建绑定库（.AAR）文件和 Eclipse Android 库项目。通过引用生成的绑定库 DLL 程序集，您可以在 Xamarin.Android 项目中重用现有的 Java 库。

当您在绑定库中引用的类型时，必须使用绑定库的命名空间。通常情况下，添加 `using` 顶部的指令在 C# 源文件，它是 .NET 命名空间版本的 Java 包名称。例如，如果您所绑定的 Java 包名称 **.jar** 所示：

```
com.company.package
```

然后可使以下 `using` 顶部的语句在 C# 源文件访问绑定中的类型 **.jar** 文件：

```
using Com.Company.Package;
```

绑定现有 Android 库时，必须要记住以下几点：

- 是否有任何库的外部依赖关系？ – 必须为 Xamarin.Android 项目中包含所需的 Android 库的任何 Java 依赖项 **ReferenceJar** 或是 **EmbeddedReferenceJar**。必须将任何本机程序集添加到作为绑定项目 **EmbeddedNativeLibrary**。
- 哪个版本的 **Android API** 是执行 **Android** 库目标？ – 不能以“降级”的 Android API 级别；确保，Xamarin.Android 绑定项目面向的相同的 API 级别（或更高）为 Android 库。
- 哪些版本的 **JDK** 用于编译库？ – 如果使用不同版本的 JDK 比在使用 xamarin.android 生成 Android 库，可能会发生绑定错误。如果可能，请重新编译使用相同的 Xamarin.Android 安装使用的 jdk 版本的 Android 库。

生成操作

创建绑定库，当您设置 *生成操作* 上 **.jar** 或 .AAR 文件合并到绑定库项目 – 每个生成操作确定如何 **.jar** 或 .AAR 文件绑定库。以下列表总结了这些生成操作：

- **EmbeddedJar** – 将嵌入 **.jar** 到作为嵌入资源生成的绑定库 DLL。这是最简单和大多数常用的生成操作。如果你想使用此选项 **.jar** 自动编译成字节代码并打包到绑定库。
- **InputJar** – 不会嵌入 **.jar** 到生成的绑定库 DLL。绑定库 DLL 会对此有依赖关系 **.jar** 在运行时。使用此选项，如果不希望包括 **.jar** 绑定库（例如，出于许可原因）中。如果使用此选项，则必须确保输入 **.jar** 可在运行您的应用程序的设备上。
- **LibraryProjectZip** – 将嵌入 .AAR 文件到生成的绑定库 DLL。它类似于 **EmbeddedJar**，只不过在绑定中，可以访问资源（以及代码）。.AAR 文件。如果你想要嵌入使用此选项 .AAR 到绑定库。
- **ReferenceJar** – 指定的引用 **.jar**：一个引用 **.jar** 是 **.jar** 到绑定的其中一个 **.jar** 或 .AAR 文件取决于。此引用 **.jar** 仅用于满足编译时依赖项。当使用此生成操作，C# 绑定不会创建引用 **.jar** 并且它不嵌入在生成的绑定库中 DLL。使用此选项时将使绑定库参考 **.jar** 但尚未尚未执行操作。此生成操作可用于打包多个 **.jars**（和/或 .AARs）到多个相互依赖的绑定库。
- **EmbeddedReferenceJar** – 将引用的嵌入 **.jar** 到生成的绑定库 DLL。如果想要创建使用此生成操作 C# 的这两个输入绑定 **.jar**（或 .AAR）和所有它的引用 **.jar**（s）绑定库中。
- **EmbeddedNativeLibrary** – 将嵌入到本机 **.so** 到绑定。用于此生成操作 **.so** 所需的文件 **.jar** 文件被绑定。它可能需要手动加载 **.so** 库，然后从 Java 库执行代码。这是如下所述。

以下指南中的更详细地介绍了操作这些生成。

此外，使用以下生成操作以帮助导入 Java API 文档和其转换为 C# XML 文档：

- **JavaDocJar** 用于指向 Javadoc 存档 Jar 符合 Maven 包样式的 Java 库（通常 `FOOBAR-javadoc*.jar`）。

- `JavaDocIndex` 用于指向 `index.html` 中的 API 参考文档 HTML 文件。
- `JavaSourceJar` 用于补充 `JavaDocJar`，以便首先从源生成 JavaDoc，然后将结果视为 `JavaDocIndex`，符合 Maven 的 Java 库包样式 (通常 `FOOBAR-sources**.jar**`)。

API 文档应为从 Java8、Java7 或 Java6 SDK (它们是所有不同的格式)，默认 doclet 或 DroidDoc 样式。

在绑定中包括本机库

可能有必要包括 `.so` 库，请在 Xamarin.Android 绑定的绑定 Java 库的一部分。Xamarin.Android 已包装的 Java 代码执行时，将无法进行 JNI 调用和错误消息 `_java.lang.UnsatisfiedLinkError: 找不到的本机方法:_` 会在 logcat 出应用程序中显示。

此解决方法是手动加载 `.so` 库通过调用 `Java.Lang.JavaSystem.LoadLibrary`。例如假设 Xamarin.Android 项目已共享库 `libpocketsphinx_jni.so` 绑定项目的生成操作中包含 `EmbeddedNativeLibrary`，以下代码片段 (使用共享的库之前执行) 将加载 `.so` 库：

```
Java.Lang.JavaSystem.LoadLibrary("pocketsphinx_jni");
```

调整到 C 的 Java Api

Xamarin.Android 绑定生成器会更改某些 Java 的惯用语言和模式，对应于 .NET 的模式。以下列表介绍了如何将 Java 映射到 C#/.NET：

- `_Setter/Getter 方法_` 在 Java 中都 `_属性_` 在 .NET 中。
- `_字段_` 在 Java 中都 `_属性_` 在 .NET 中。
- `_侦听器/侦听器接口_` 在 Java 中都 `_事件_` 在 .NET 中。回调接口中的方法的参数将由表示 `EventArgs` 子类。
- 一个 `_静态嵌套类_` 在 Java 中是 `_嵌套类_` 在 .NET 中。
- `_内部类_` 在 Java 中是 `_嵌套类_` 实例构造函数中使用 C#。

绑定方案

以下绑定方案指南可帮助你为将合并到您的应用程序绑定 Java 库 (或库)：

- [绑定。JAR](#) 是创建用于绑定库的演练 `.jar` 文件。
- [绑定。AAR](#) 是创建用于绑定库的演练。AAR 文件。阅读此演练，若要了解如何将绑定 Android Studio 库。
- [绑定 Eclipse 库项目](#) 是从 Android 库项目中创建绑定库的演练。阅读此演练，若要了解如何将绑定 Eclipse Android 库项目。
- [自定义绑定](#) 介绍了如何手动修改绑定以解决生成错误并调整所获得的 API，这样就更多 "C#-例如"。
- [绑定疑难解答](#) 列出了常见绑定错误方案，介绍了可能的原因，并提供了解决这些错误的建议。

相关链接

- [使用 JNI](#)
- [GAPI 元数据](#)
- [使用本机库](#)

绑定。JAR

2018/10/26 • [Edit Online](#)

本演练提供了从 Android 创建 Xamarin.Android Java 绑定库的分步说明。JAR 文件。

概述

Android 社区还提供了许多可能想要使用应用程序中的 Java 库。采用通常封装这些 Java 库。JAR (Java 存档文件) 格式, 但您可以将打包。它在 JAR Java 绑定库, 以便其功能是适用于 Xamarin.Android 应用。Java 绑定库的目的是使中的 Api。JAR 文件提供给 C# 代码中通过自动生成的代码包装器。

Xamarin 工具可以生成绑定库从一个或多个输入。JAR 文件。绑定库 (。DLL 程序集) 包含以下元素:

- 原始内容。JAR 文件。
- 托管可调用包装 (MCW), 这是 C# 类型对应的 Java 类型中的自动换行。JAR 文件。

生成的 MCW 代码使用 JNI (Java 本机接口) 将 API 调用转发到基础。JAR 文件。可以为任何创建绑定库。最初针对与 Android (请注意 Xamarin 工具中当前不支持非 Android Java 库的绑定) 一起使用的 JAR 文件。您还可以选择用于生成绑定库而不包括的内容。JAR 文件, 以便在 DLL 具有依赖项。JAR 在运行时。

在本指南中, 我们将逐步完成创建一个绑定库的基础知识。JAR 文件。我们将所有内容会直接的示例演示了-, 即没有自定义或绑定的调试需要的。[创建绑定使用元数据](#)提供了绑定进程不是完全自动和一定量的手动干预是必需的更高级方案的示例。Java 库绑定, 一般情况下 (其中一个基本代码示例) 的概述, 请参阅[绑定 Java 库](#)。

演练

在下面的演练中, 我们将创建一个绑定库, 用于[看毕加索会如何](#), 流行的 Android。提供了映像加载和缓存功能的 JAR。我们将使用以下步骤来绑定看毕加索会如何 **2.x.x.jar** Xamarin.Android 项目中创建新的 .NET 程序集, 我们可以使用:

1. 创建新的 Java 绑定库项目。
2. 添加。JAR 文件复制到项目。
3. 设置为相应的生成操作。JAR 文件。
4. 选择目标框架。支持 JAR。
5. 生成绑定库。

一旦我们创建绑定库, 我们将开发一个小型的 Android 应用程序演示我们调用绑定库中的 Api 的能力。在此示例中, 我们想要访问的方法看毕加索会如何 **2.x.x.jar**:

```
package com.squareup.picasso

public class Picasso
{
    ...
    public static Picasso with (Context context) { ... };
    ...
    public RequestCreator load (String path) { ... };
    ...
}
```

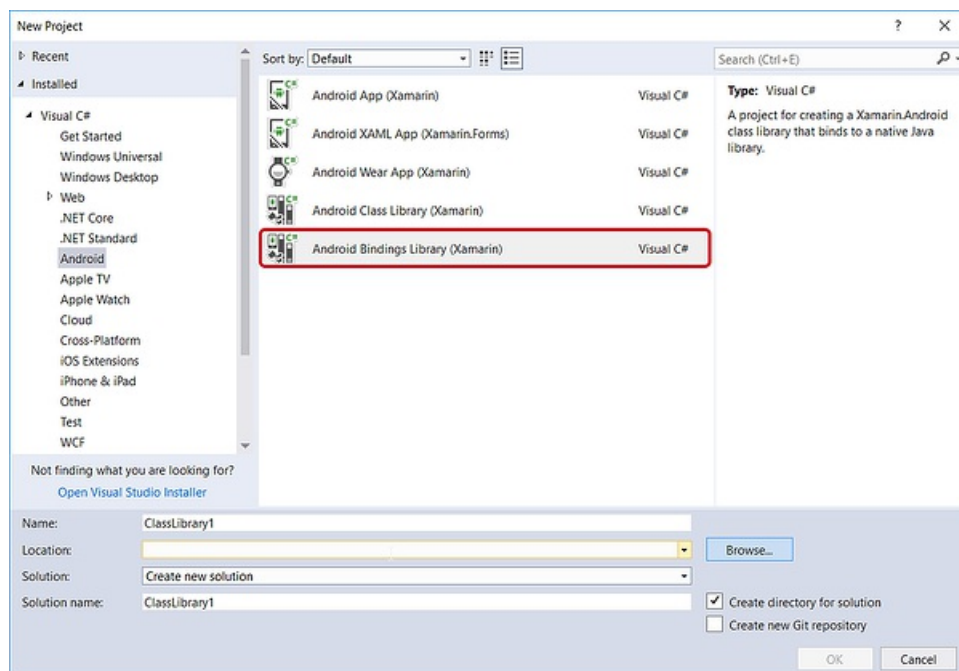
我们生成一个绑定库，用于以后看毕加索会如何 **2.x.x.jar**，我们可以调用这些方法从C#。例如：

```
using Com.Squareup.Picasso;
...
Picasso.With (this)
    .Load ("http://mydomain.myimage.jpg")
    .Into (imageView);
```

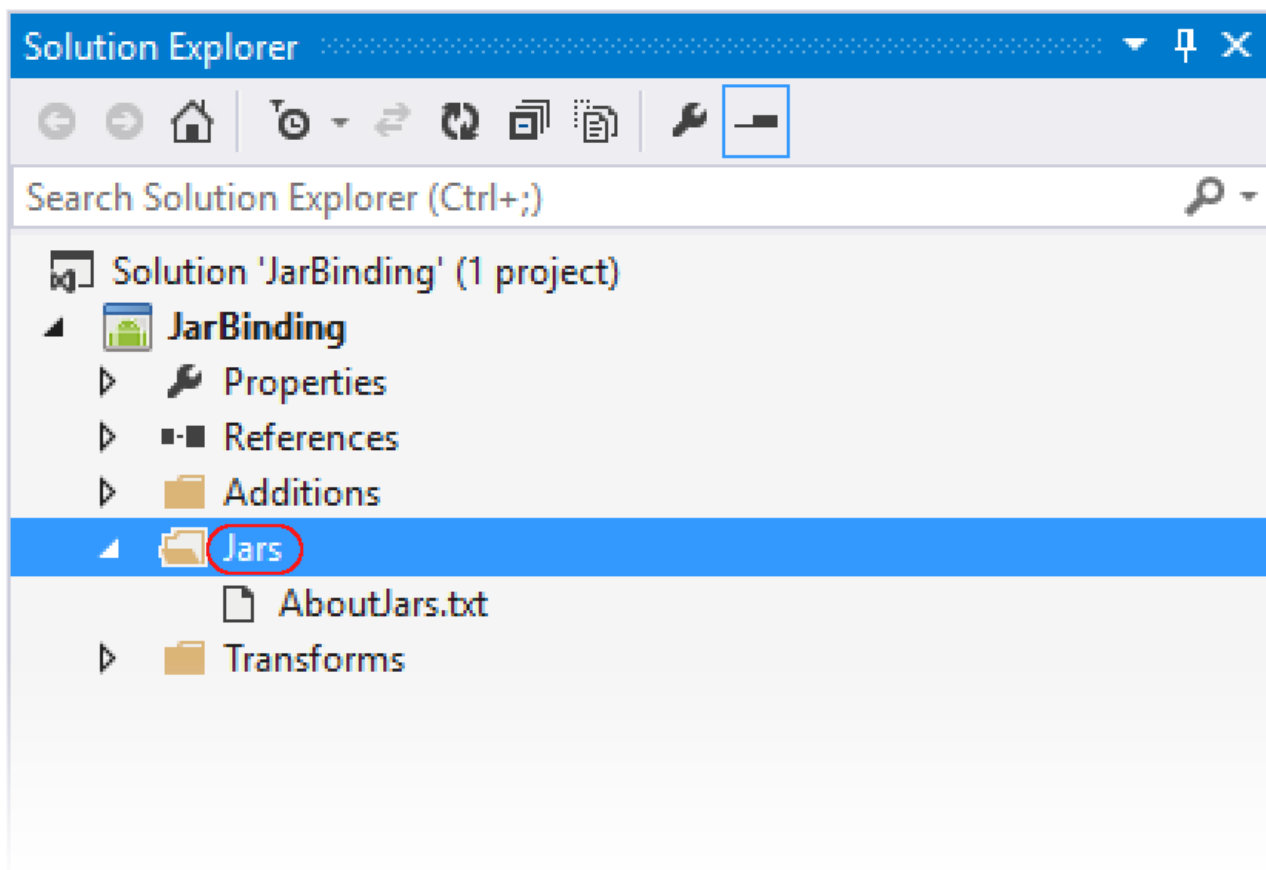
创建绑定库

在开始执行以下步骤之前，请下载[看毕加索会如何 2.x.x.jar](#)。

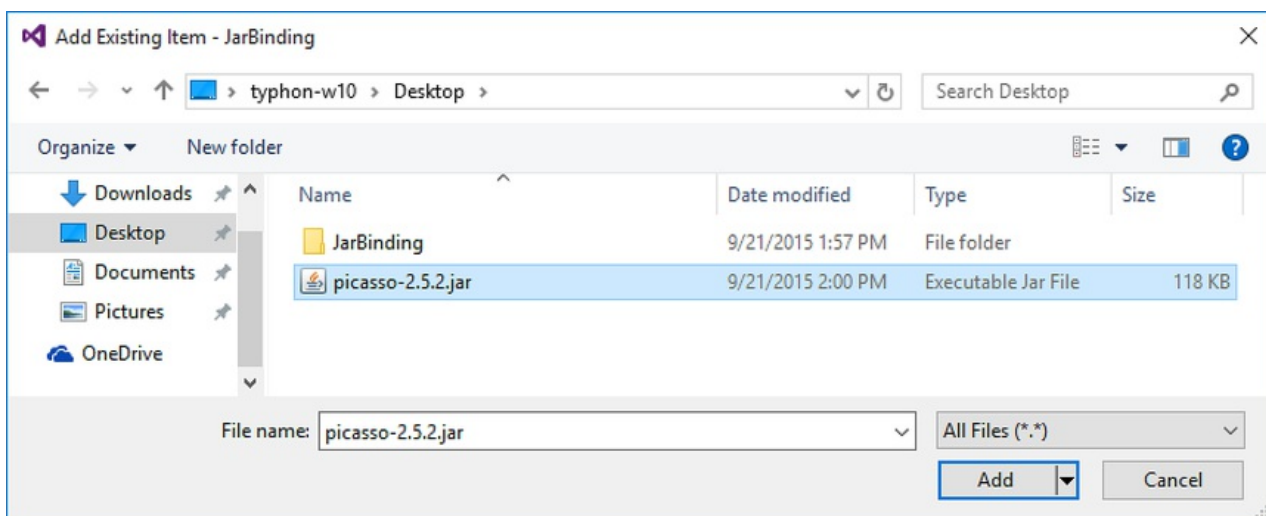
首先，创建一个新的绑定库项目。在 Visual Studio for Mac 或 Visual Studio 中，创建一个新的解决方案并选择 *Android 绑定库* 模板。（在本演练中的屏幕截图使用 Visual Studio 中，但 Visual Studio for Mac 是非常相似。）将解决方案命名 **JarBinding**：



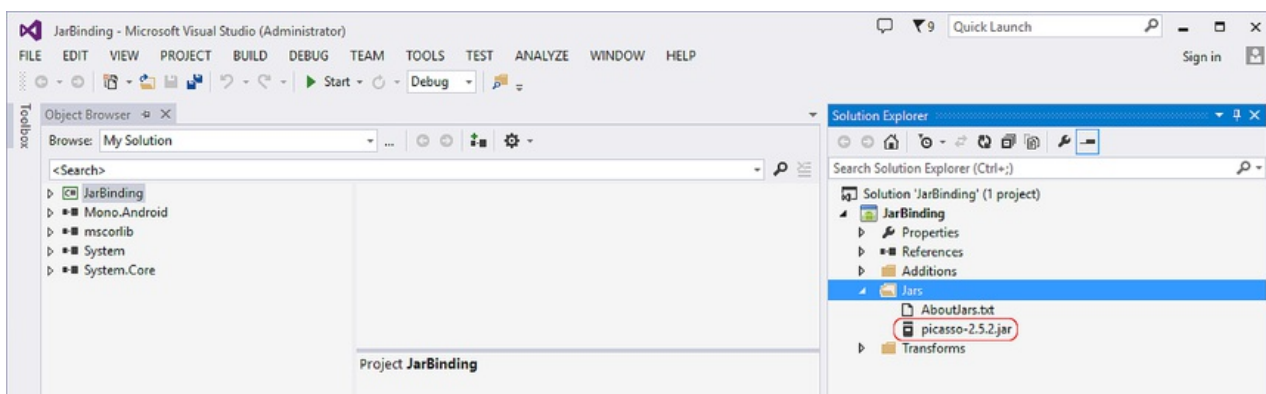
该模板包含 **Jar** 文件夹在其中添加你。JAR(s) 到绑定库项目。右键单击 **Jar** 文件夹，然后选择 **添加 > 现有项**：



导航到看毕加索会如何 2.x.x.jar 前面下载的文件，选择它并单击添加：



确认看毕加索会如何 2.x.x.jar 文件已成功添加到项目：

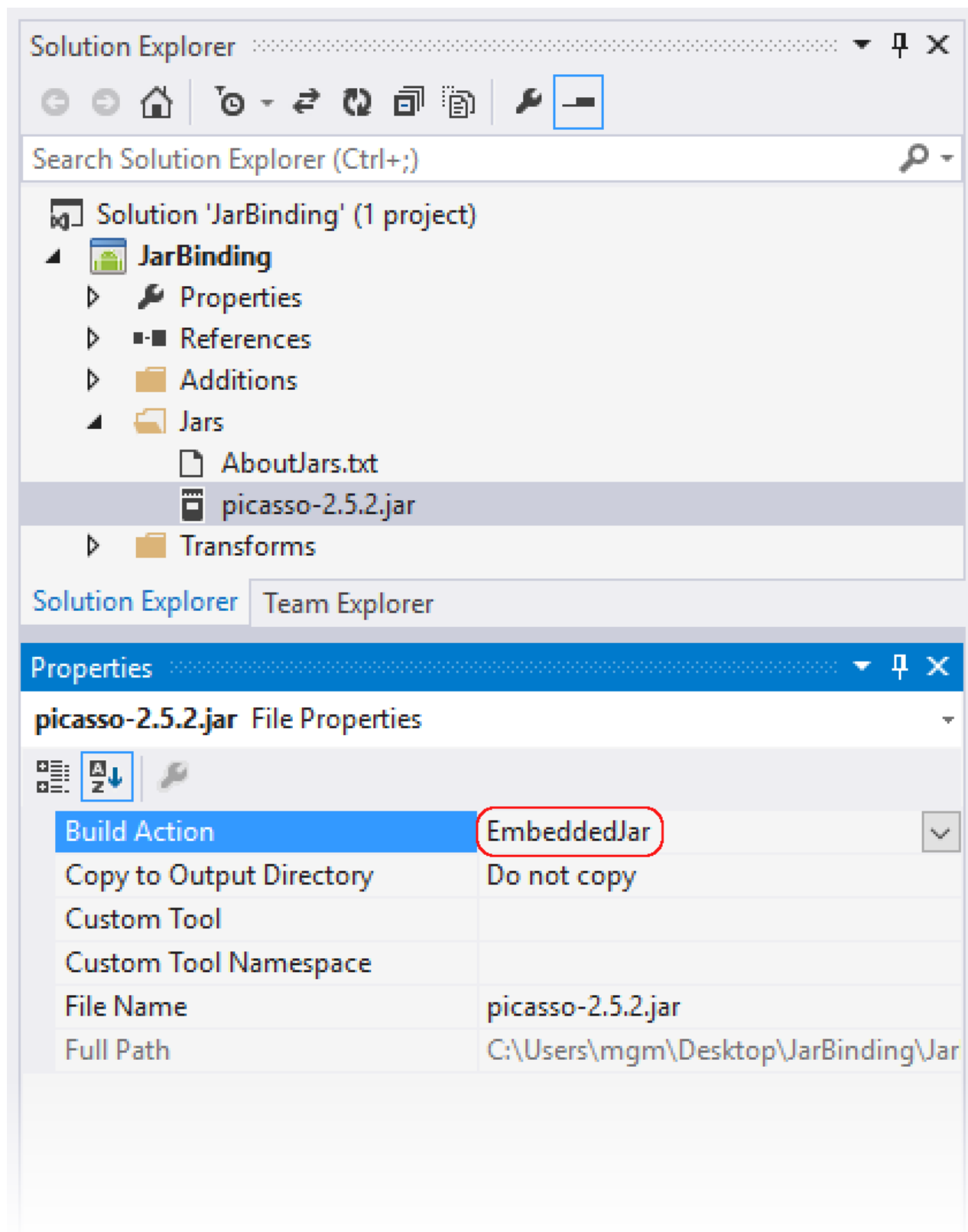


创建 Java 绑定库项目时，必须指定是否。JAR 是嵌入到绑定库中或单独打包。若要执行此操作，您指定以下值之一生成操作。

- **EmbeddedJar** –。绑定库中，将嵌入 JAR。
- **InputJar** –。将独立于绑定库保留 JAR。

通常情况下，使用**EmbeddedJar**生成操作，以便。JAR 自动打包到绑定库中。这是最简单的选项-中的 Java 字节码。JAR 转换为 Dex 字节码，以及（以及托管的可调用包装）嵌入到 APK。如果你想要保留。JAR 独立于绑定库，则可以使用**InputJar**选项；但是，您必须确保。JAR 文件位于运行您的应用程序的设备上。

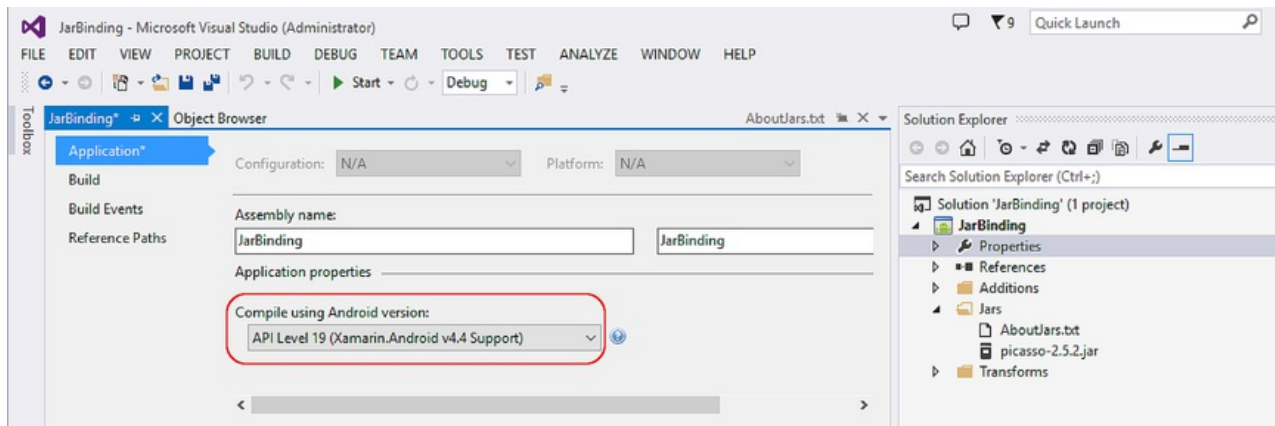
生成操作设置为**EmbeddedJar**：



接下来，打开项目属性来配置**目标框架**。如果。JAR 使用任何 Android Api，将目标框架设置为 API 级别。需要 JAR。通常情况下，开发人员的。JAR 文件将指示哪个 API 级别（或级别）的。与兼容 JAR。（有关目标框架设置和

在常规的 Android API 级别的详细信息, 请参阅[了解 Android API 级别](#)。)

设置目标 API 级别绑定库 (在此示例中, 我们将使用 API 级别 19):



最后, 生成绑定库。尽管可能会显示某些警告消息, 但绑定库项目应能成功生成, 并生成输出。在以下位置的 DLL:
JarBinding/bin/Debug/JarBinding.dll

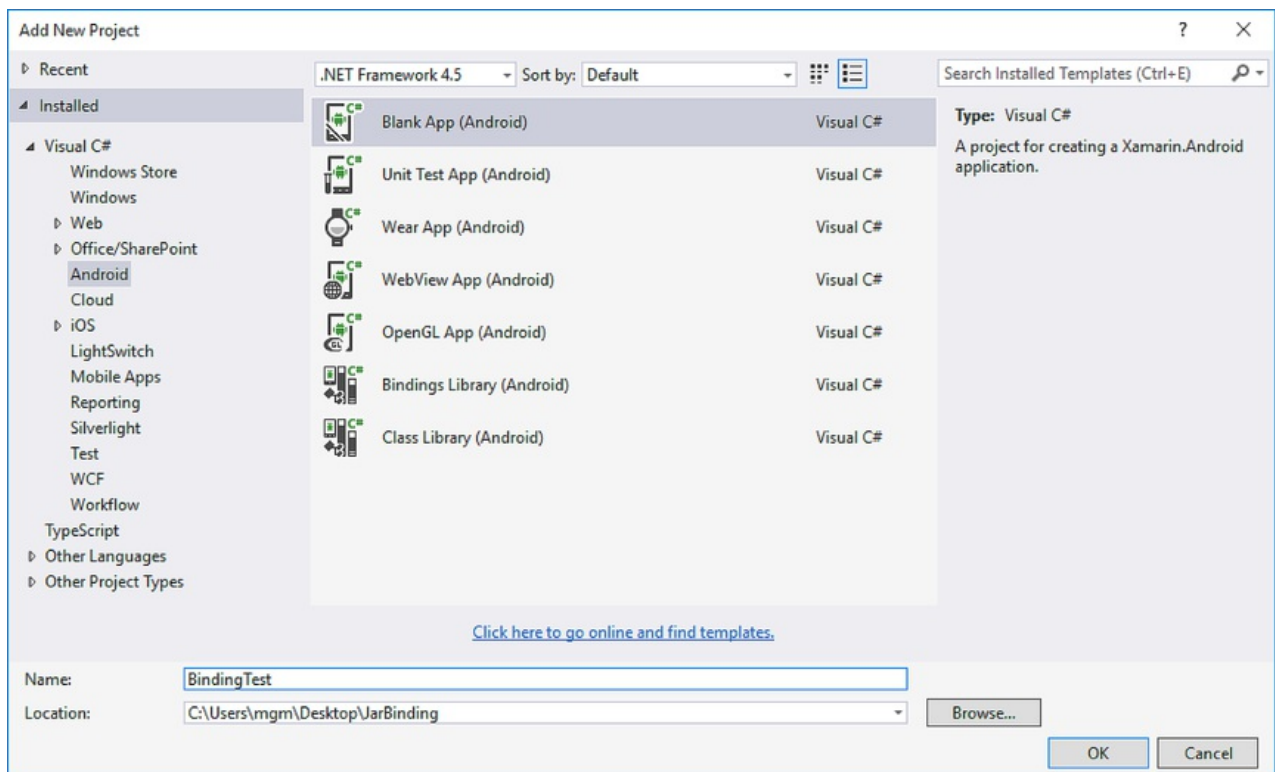
使用绑定库

若要使用这个。DLL 在 Xamarin.Android 应用中, 执行以下步骤:

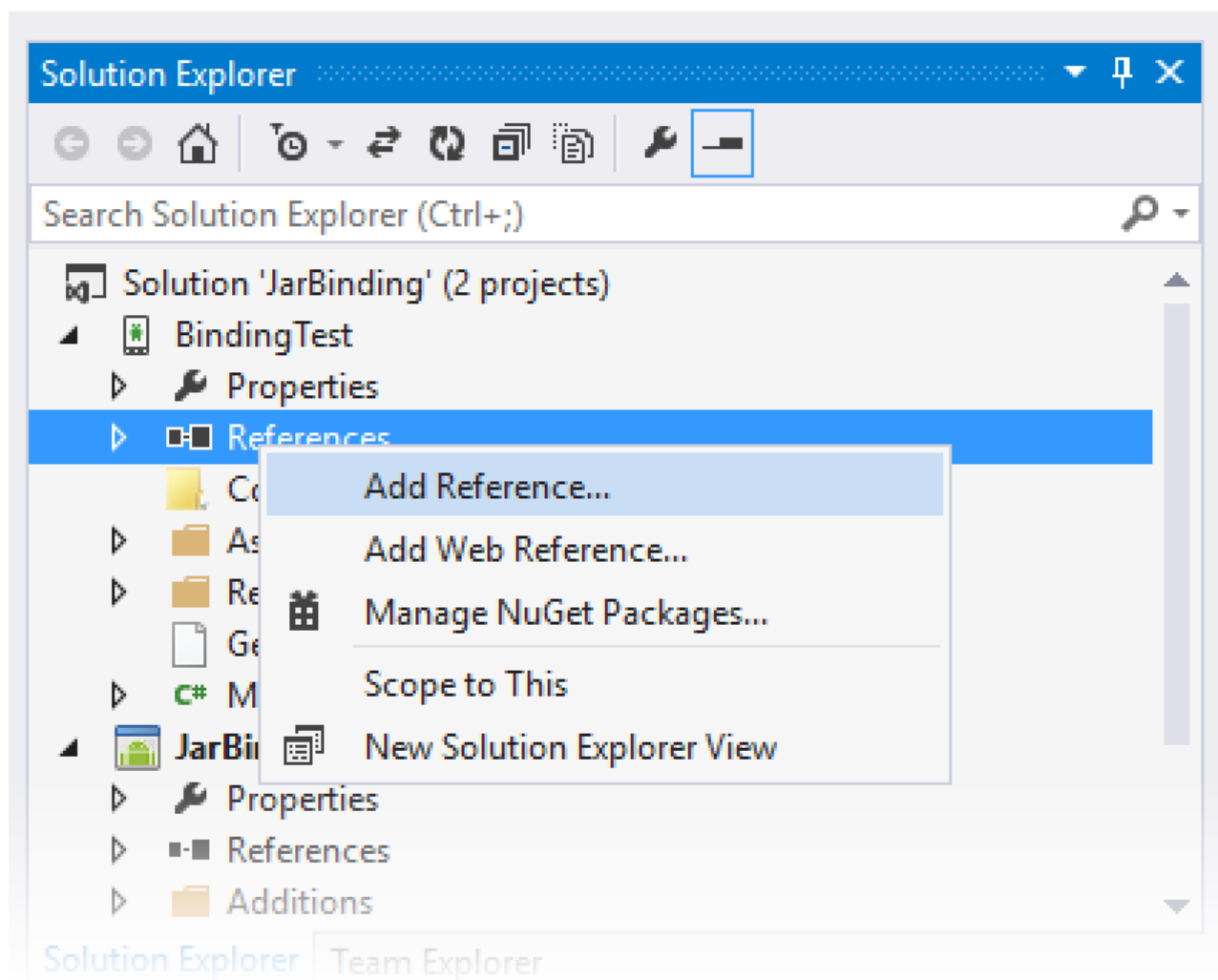
1. 添加到绑定库的引用。
2. 对进行调用。JAR 通过托管的可调用包装器。

在以下步骤中, 我们将创建一个使用绑定库下载并显示的图像的最小应用 `ImageView`; "繁重的工作", 可以驻留在的代码。JAR 文件。

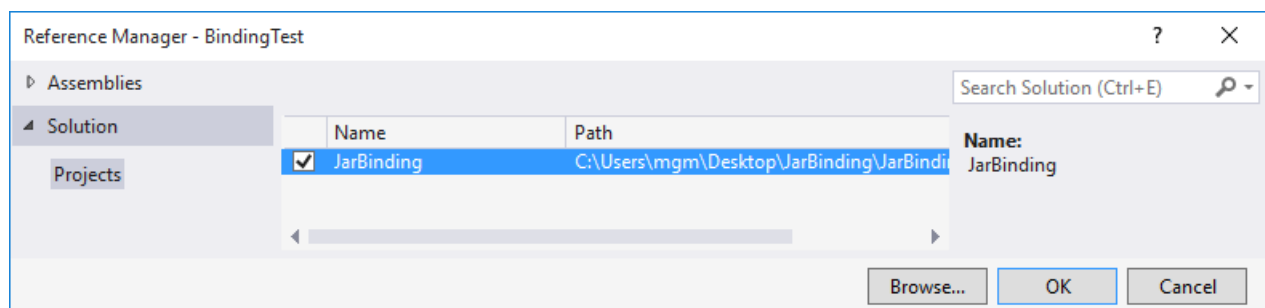
首先, 创建使用绑定库的新 Xamarin.Android 应用程序。右键单击解决方案并选择**添加新项目**; 将新项目命名 **BindingTest**。我们要在与绑定库相同的解决方案中创建此应用程序, 为了简化本演练中; 但是, 使用绑定库应用程序, 而是驻留在另一种解决方案:



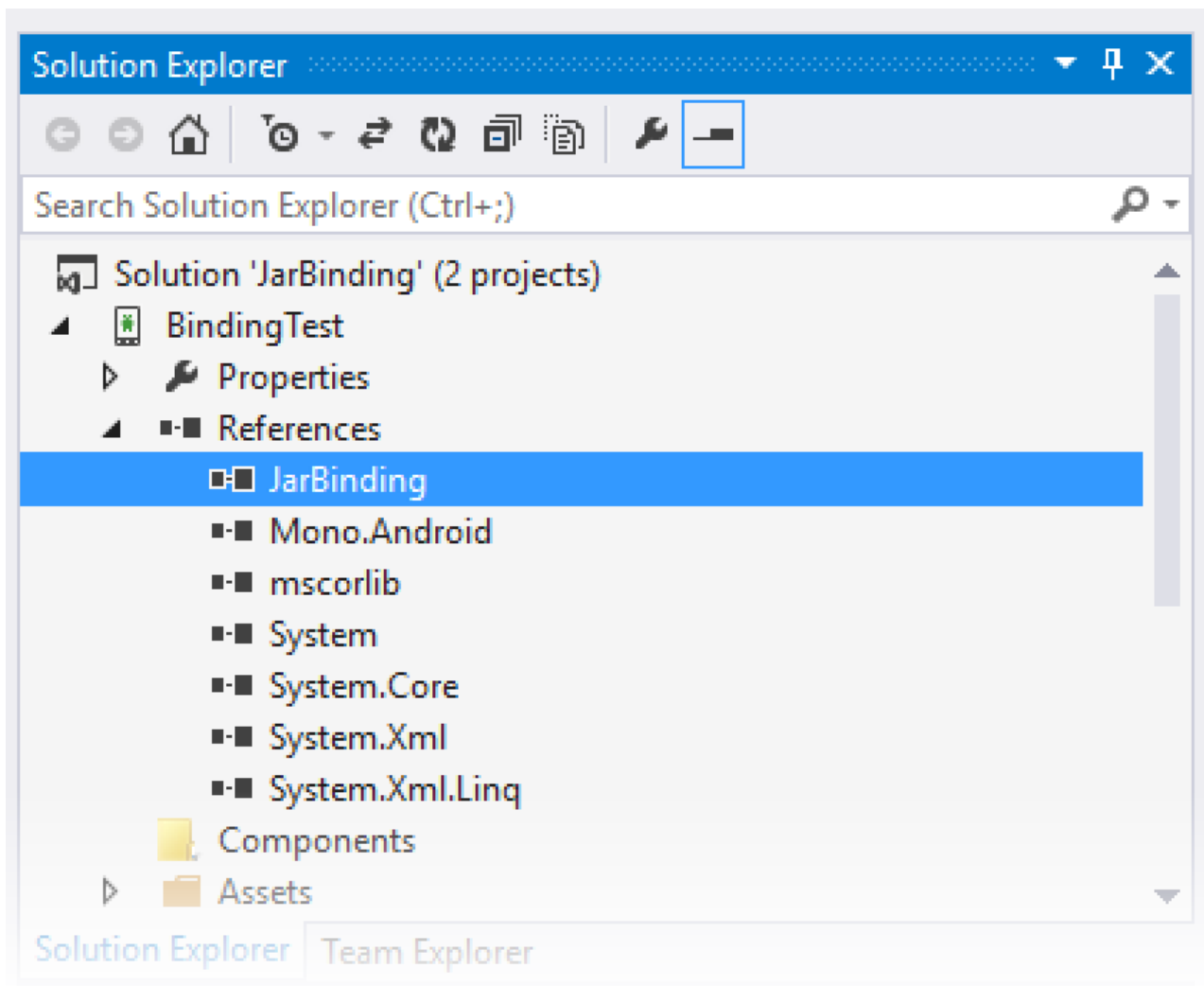
右键单击引用的节点 **BindingTest** 项目, 然后选择**添加引用...**:



检查**JarBinding**前面创建的项目并单击**确定**：



打开引用的节点**BindingTest**项目，然后确认**JarBinding**引用不存在：



修改 **BindingTest** 布局 (**Main.axml**), 以便它具有单个 `ImageView` :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:minWidth="25px"
    android:minHeight="25px">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/imageView" />
</LinearLayout>
```

添加以下 `using` 语句 **MainActivity.cs** - 这样就可以轻松地访问的方法的基于 Java 的 `Picasso` 驻留在绑定库中的类:

```
using Com.Squareup.Picasso;
```

修改 `OnCreate` 方法, 以便使用 `Picasso` 类来从 URL 加载图像并将其显示在 `ImageView` :

```
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);
        ImageView imageView = FindViewById<ImageView>(Resource.Id.imageView);

        // Use the Picasso jar library to load and display this image:
        Picasso.With (this)
            .Load ("http://i.imgur.com/DvpvklR.jpg")
            .Into (imageView);
    }
}
```

编译并运行 **BindingTest** 项目。应用程序将启动，并后一个短暂的延迟（具体取决于网络条件），它应下载并显示类似于以下屏幕截图的映像：



祝贺你！您已成功绑定 Java 库。JAR 并将 Xamarin.Android 应用中使用。

总结

在本演练中，我们将创建第三方绑定库。JAR 文件，添加到最小化测试应用，绑定库，然后运行应用程序以验证我们 C# 代码可以调用 Java 代码驻留在中。JAR 文件。

相关链接

- [生成 Java 绑定库 \(视频\)](#)
- [绑定 Java 库](#)

绑定。获取 AAR

2018/10/26 • [Edit Online](#)

本演练提供了从 Android 创建 Xamarin.Android Java 绑定库的分步说明。AAR 文件。

概述

Android 存档 (.AAR) 文件是 Android 库的文件格式。一个 .AAR 文件。ZIP 存档包含以下各项：

- 已编译的 Java 代码
- 资源 Id
- 资源
- 元数据（例如，活动声明，权限）

在本指南中，我们将逐步完成创建一个绑定库的基础知识。AAR 文件。Java 库绑定，一般情况下（其中一个基本代码示例）的概述，请参阅[绑定 Java 库](#)。

IMPORTANT

绑定项目只能包含一个 .AAR 文件。如果。在其他 AAR 依赖项。AAR，则这些依赖项应包含在其自己绑定项目和引用。请参阅[Bug 44573](#)。

演练

有关在 Android Studio 中创建 Android 存档文件的示例，我们将创建一个绑定库 `textanalyzer.aar`。这 .AAR 包含 `TextCounter` 包含元音和字符串中的辅音字母数目进行计数的静态方法的类。此外，`textanalyzer.aar` 包含的图像资源可帮助将显示计数的结果。

我们将使用以下步骤创建从绑定库。AAR 文件：

1. 创建新的 Java 绑定库项目。
2. 添加一个 .AAR 文件复制到项目。绑定项目只能包含一个 .AAR。
3. 设置为相应的生成操作。AAR 文件。
4. 选择目标框架。AAR 支持。
5. 生成绑定库。

一旦我们创建绑定库，我们将开发提示用户输入文本字符串，调用一个小的 Android 应用。AAR 方法分析文本，检索中的映像。AAR，并显示结果与图像。

示例应用程序将访问 `TextCounter` 的类 `textanalyzer.aar`。

```
package com.xamarin.textcounter;

public class TextCounter
{
    ...
    public static int numVowels (String text) { ... };
    ...
    public static int numConsonants (String text) { ... };
    ...
}
```

此外，此示例应用将检索并显示在打包的图像资源`textanalyzer.aar`：

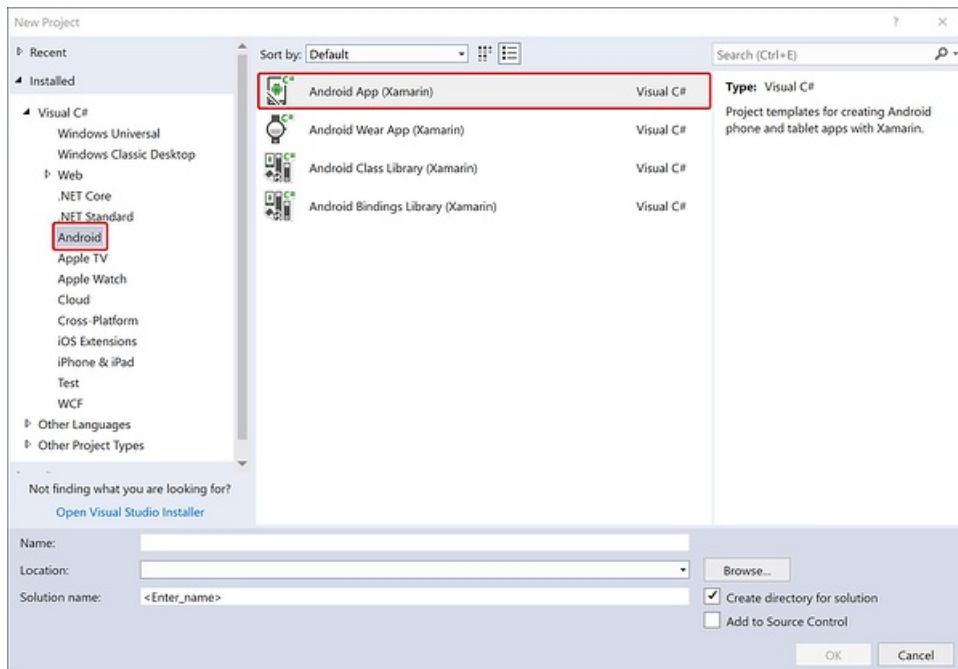


此映像资源驻留在`res/drawable/monkey.png`中`textanalyzer.aar`。

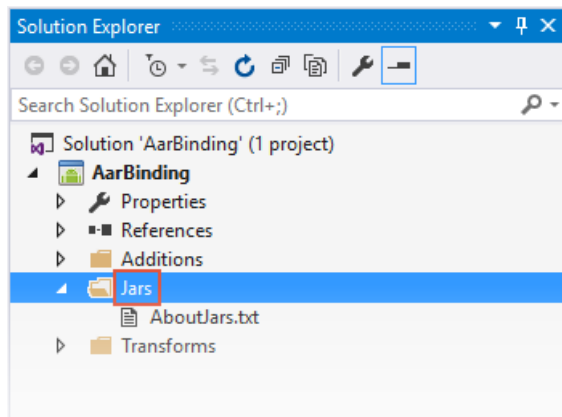
创建绑定库

在开始执行以下步骤之前，请下载该示例[textanalyzer.aar](#) Android 存档文件：

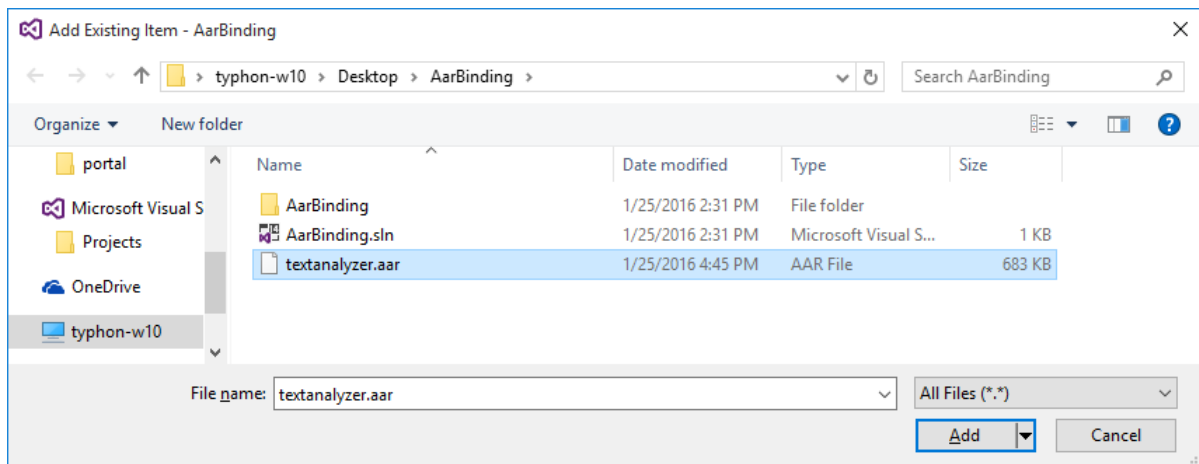
1. 创建从 Android 绑定库模板开始一个新绑定库项目。可以使用 Visual Studio for Mac 或 Visual Studio（下面的屏幕截图显示 Visual Studio 中，但 Visual Studio for Mac 是非常相似）。将解决方案命名 **AarBinding**：



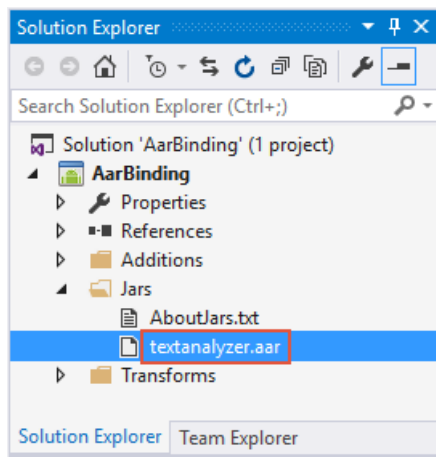
2. 该模板包含Jar文件夹在其中添加你。AAR(s) 到绑定库项目。右键单击Jar文件夹，然后选择添加 > 现有项:



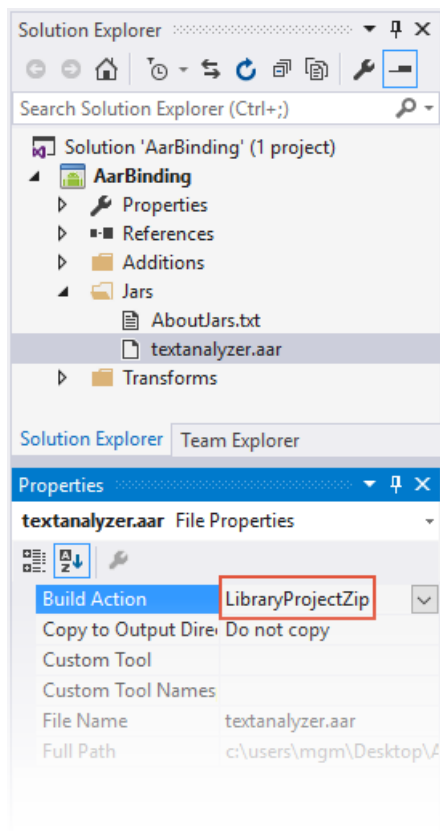
3. 导航到textanalyzer.aar前面下载的文件，选择它，然后单击添加:



4. 确认textanalyzer.aar文件已成功添加到项目:

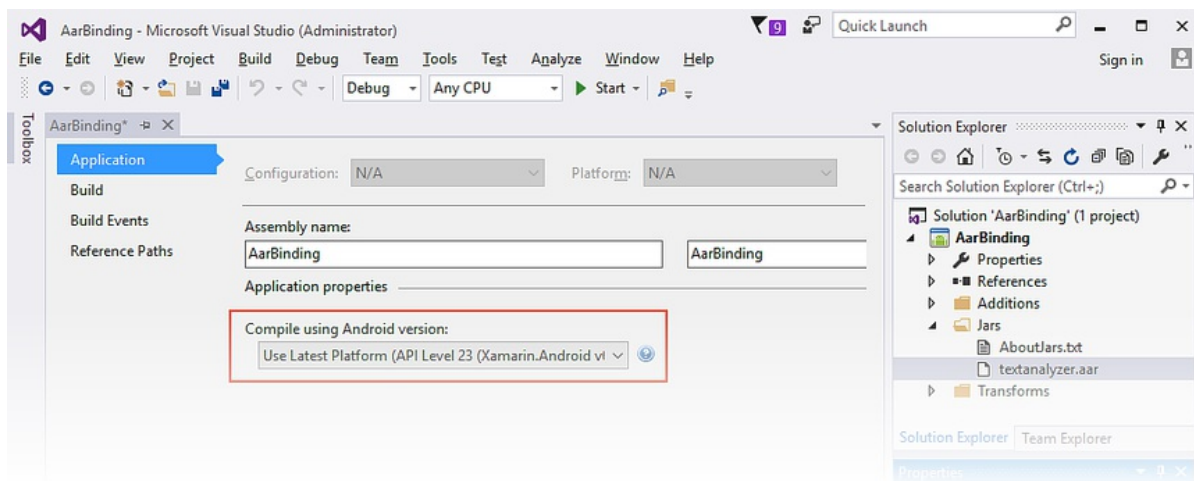


5. 设置的生成操作textanalyzer.aar到 LibraryProjectZip。在 Visual Studio for Mac 中, 右键单击textanalyzer.aar设置生成操作。在 Visual Studio 中, 可以设置生成操作属性窗格):



6. 打开项目属性来配置目标框架。如果。AAR 使用任何 Android Api, 将目标框架设置为 API 级别。AAR 需要。(有关目标框架设置和在常规的 Android API 级别的详细信息, 请参阅[了解 Android API 级别](#)。)

为绑定库设置目标 API 级别。在此示例中, 我们可以自由使用最新的平台 API 级别 (API 级别 23), 因为我们textanalyzer Android Api 上没有依赖项:



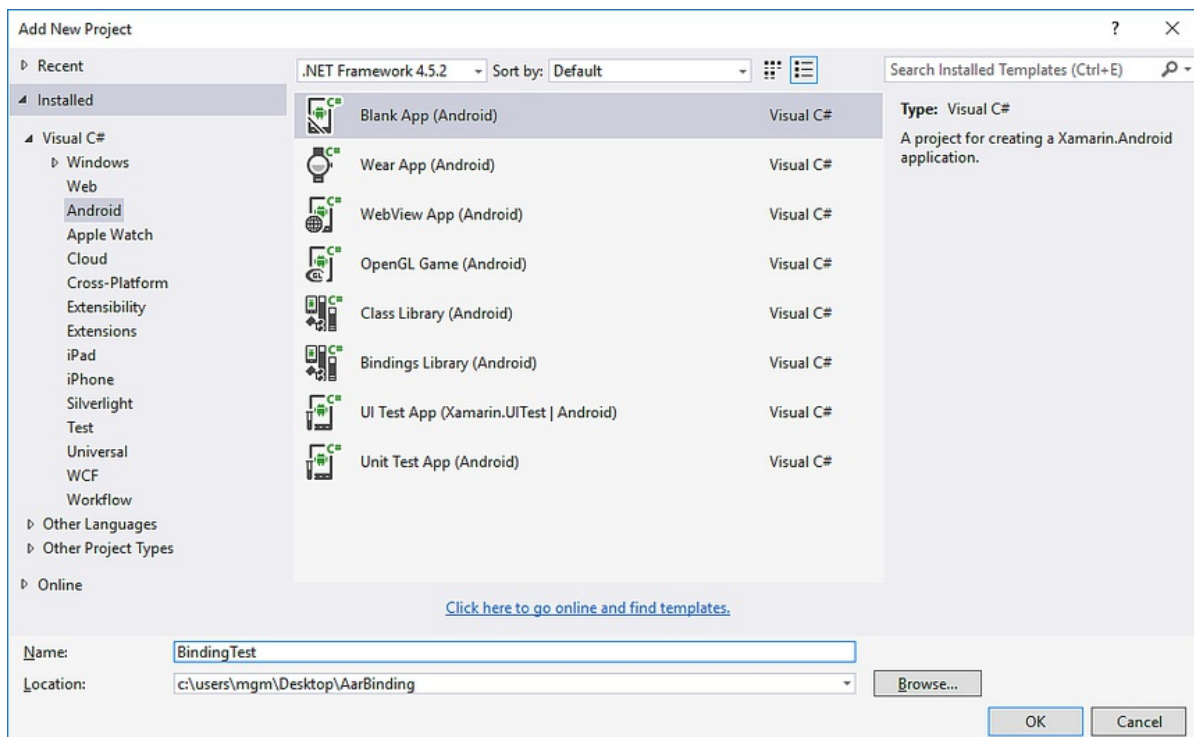
7. 生成绑定库。绑定库项目应能成功生成，并生成输出。在以下位置的 DLL：

AarBinding/bin/Debug/AarBinding.dll

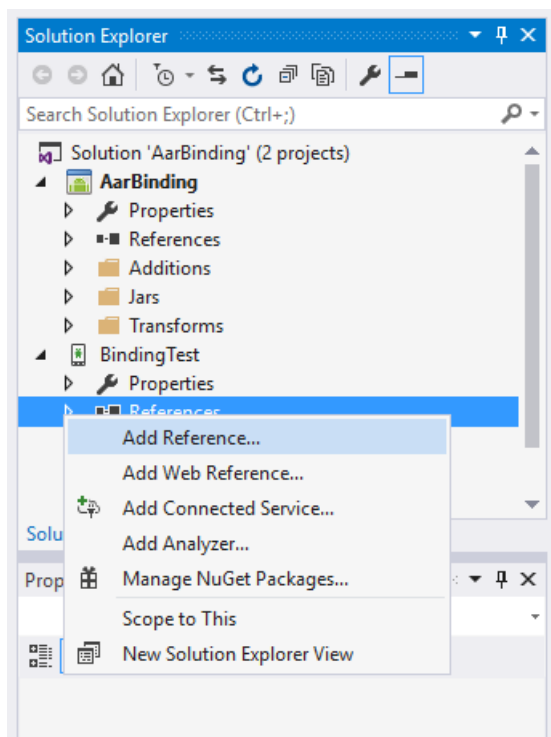
使用绑定库

若要使用这个。Xamarin.Android 应用程序中的 DLL，必须首先添加到绑定库的引用。使用以下步骤：

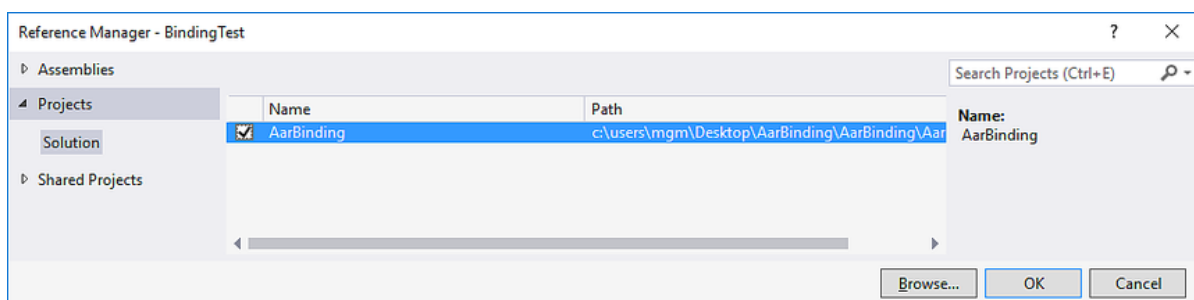
- 我们将在绑定库来简化本演练中的同一个解决方案中创建此应用。（使用绑定库应用程序也可以驻留在另一种解决方案。）创建新的 Xamarin.Android 应用程序：右键单击解决方案并选择**添加新项目**。将新项目命名**BindingTest**：



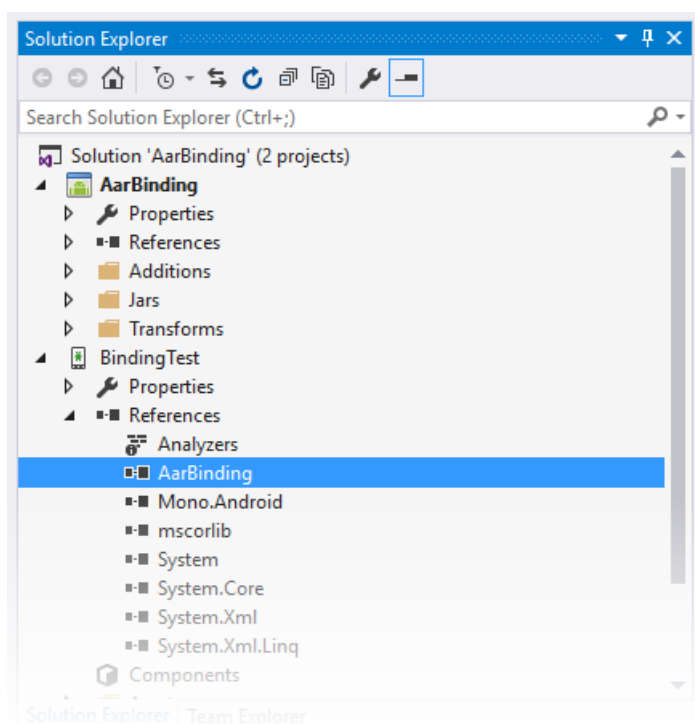
- 右键单击引用的节点**BindingTest**项目，然后选择**添加引用**....



3. 选择**AarBinding**前面创建的项目并单击**确定**：

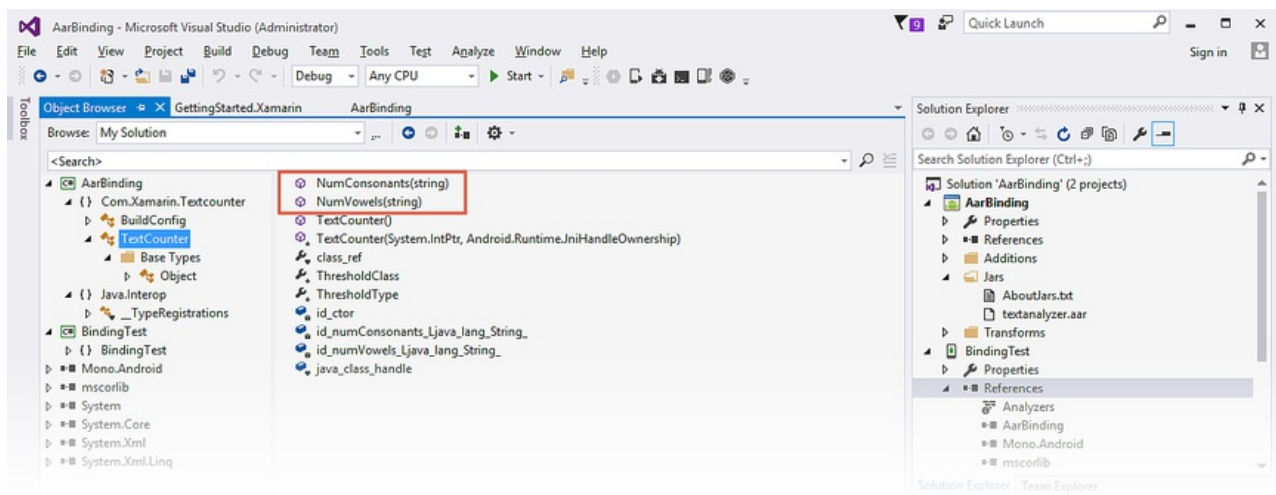


4. 打开引用的节点**BindingTest**项目，以验证**AarBinding**引用不存在：



如果你想要查看绑定库项目的内容，则可以双击要打开该文件的引用对象浏览器。您所见的内容

`Com.Xamarin.Textcounter` 命名空间 (通过 Java 映射 `com.xamarin.textanalyzezr` 包)，可以查看的成员 `TextCounter` 类：



上面的屏幕截图突出显示了这两个 `TextAnalyzer` 示例应用将调用的方法：`NumConsonants`（其包装基础 Java `numConsonants` 方法），和 `NumVowels`（其包装基础 Java `numVowels` 方法）。

访问 **AAR** 类型

添加对您的应用程序指向绑定库的引用后，可以访问中的 Java 类型。作为你的 AAR 可访问 C# 类型（感谢到 C# 包装）。C# 应用程序代码可以调用 `TextAnalyzer` 方法在此示例中所示：

```
using Com.Xamarin.Textcounter;
...
int numVowels = TextCounter.NumVowels (myText);
int numConsonants = TextCounter.NumConsonants (myText);
```

在上述示例中，我们调用静态方法 `TextCounter` 类。但是，您还可以实例化类并调用实例方法。例如，如果你 AAR 包装一个名为类 `Employee` 具有实例方法 `buildFullName`，可以实例化 `MyClass` 和此处使用它所示：

```
var employee = new Com.MyCompany.MyProject.Employee();
var name = employee.BuildFullName ();
```

以下步骤将代码添加到应用程序，以便它会提示用户输入文本，使用 `TextCounter` 要分析的文本，然后显示结果。

替换 **BindingTest** 布局 (**Main.axml**) 使用以下 XML。此布局具有 `EditText` 文本输入和两个按钮用于启动元音标记和辅音计数：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:text="Text to analyze:"
        android:textSize="24dp"
        android:layout_marginTop="30dp"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:id="@+id/input"
        android:text="I can use my .AAR file from C#!"
        android:layout_marginTop="10dp"
        android:layout_gravity="center"
        android:layout_width="300dp"
        android:layout_height="wrap_content"/>
    <Button
        android:id="@+id/vowels"
        android:layout_marginTop="30dp"
        android:layout_width="240dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Count Vowels" />
    <Button
        android:id="@+id/consonants"
        android:layout_width="240dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Count Consonants" />
</LinearLayout>

```

内容替换为**MainActivity.cs**用下面的代码。在此示例中所示，按钮事件处理程序调用包装 `TextCounter` 驻留在的方法。若要显示的结果的 AAR 和使用 toast。请注意 `using` 语句的命名空间的绑定库 (在这种情况下，`Com.Xamarin.Textcounter`)：

```

using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Android.Views.InputMethods;
using Com.Xamarin.Textcounter;

namespace BindingTest
{
    [Activity(Label = "BindingTest", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        InputMethodManager imm;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            SetContentView(Resource.Layout.Main);

            imm = (InputMethodManager)GetSystemService(Context.InputMethodService);

            var vowelsBtn = FindViewById<Button>(Resource.Id.vowels);
            var consonBtn = FindViewById<Button>(Resource.Id.consonants);
            var edittext = FindViewById<EditText>(Resource.Id.input);
            edittext.InputType = Android.Text.InputTypes.TextVariationPassword;

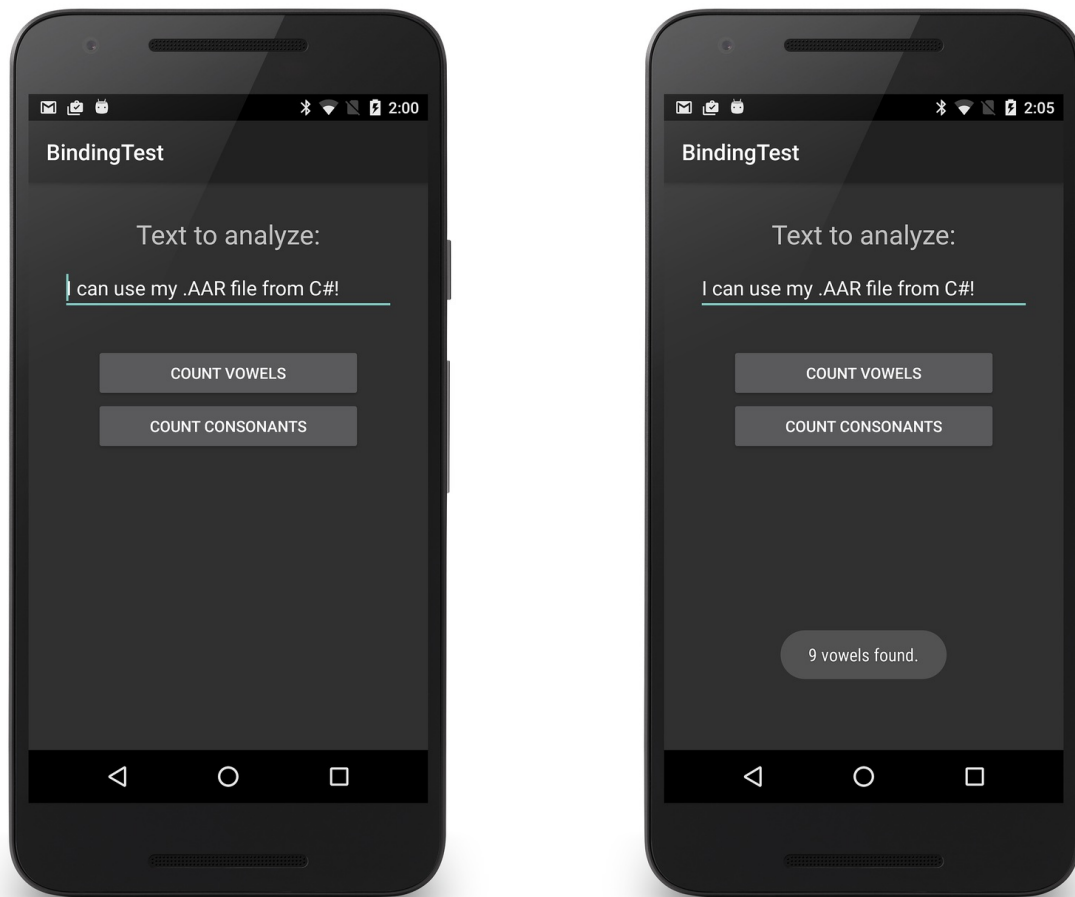
            edittext.KeyPress += (sender, e) =>
            {
                imm.HideSoftInputFromWindow(edittext.WindowToken, HideSoftInputFlags.NotAlways);
                e.Handled = true;
            };

            vowelsBtn.Click += (sender, e) =>
            {
                int count = TextCounter.NumVowels(edittext.Text);
                string msg = count + " vowels found.";
                Toast.MakeText (this, msg, ToastLength.Short).Show ();
            };

            consonBtn.Click += (sender, e) =>
            {
                int count = TextCounter.NumConsonants(edittext.Text);
                string msg = count + " consonants found.";
                Toast.MakeText (this, msg, ToastLength.Short).Show ();
            };
        }
    }
}

```

编译并运行**BindingTest**项目。应用程序将启动并显示在左侧的屏幕截图 (**EditText** 初始化与一些文本, 但你可以点击它可以对其进行更改)。当点击**计数元音**, toast 通知显示元音字母数, 如右侧所示:



请尝试点击计数辅音按钮。此外，可以修改一行文本并点击这些按钮再次以测试不同元音标记和辅音计数。

访问。AAR 资源

Xamarin 工具合并R中的数据。到你的应用的 AAR资源类。因此，您可以访问。AAR 资源从你的布局（和代码隐藏）中的相同方式将访问中的资源资源项目的路径。

若要访问的图像资源，请使用**Resource.Drawable**映像打包内部名称。AAR。例如，可以引用**image.png**中。通过使用 AAR 文件 `@drawable/image`：

```
<ImageView android:src="@drawable/image" ... />
```

您还可以访问位于资源布局。AAR。若要执行此操作，您可以使用**Resource.Layout**打包到内的布局的名称。AAR。例如：

```
var a = new ArrayAdapter<string>(this, Resource.Layout.row_layout, ...);
```

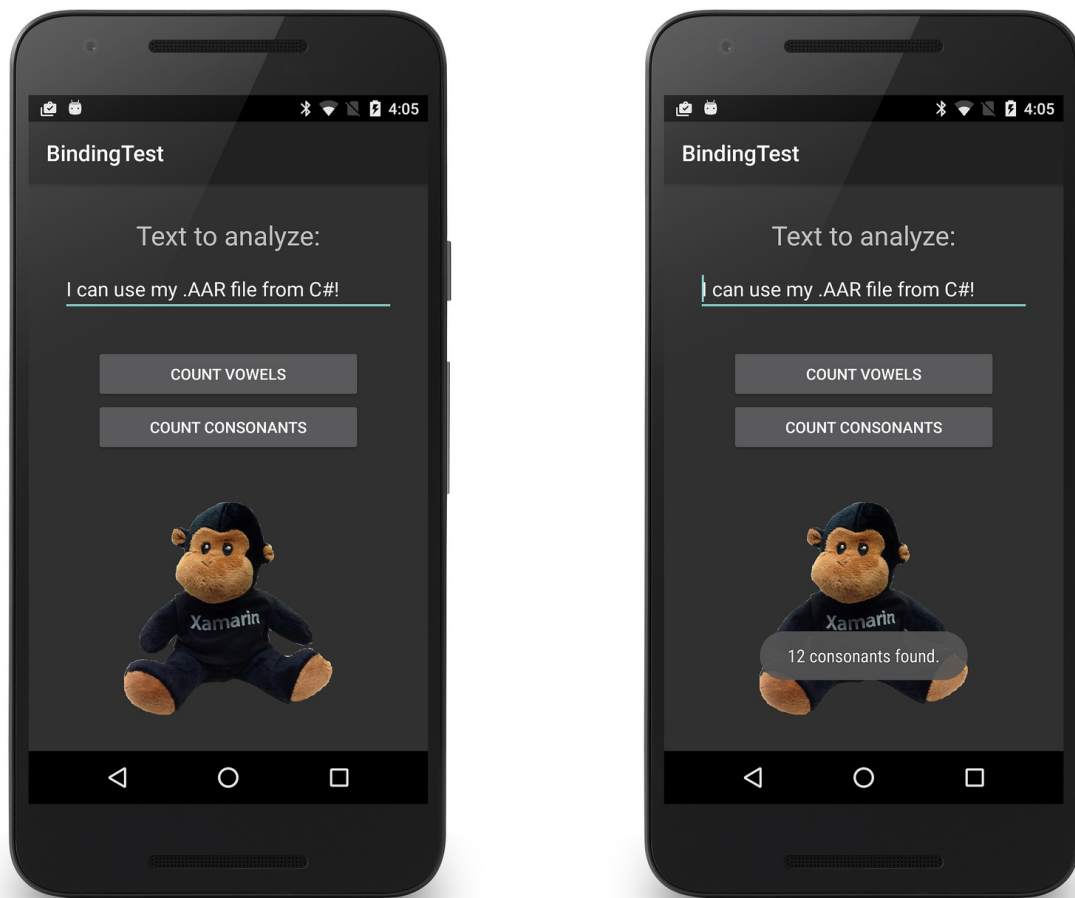
Textanalyzer.aar示例中包含的图像文件，它位于**res/drawable/monkey.png**。让我们访问此映像资源并在我们的示例应用程序中使用它：

编辑**BindingTest**布局 (**Main.axml**)，并添加 `ImageView` 末尾 `LinearLayout` 容器。这 `ImageView` 显示的图像位于 **@drawable/monkey****；此映像将加载的资源部分从**textanalyzer.aar****：

```
...
<ImageView
    android:src               ="@drawable/monkey"
    android:layout_marginTop   ="40dp"
    android:layout_width       ="200dp"
    android:layout_height      ="200dp"
    android:layout_gravity     ="center" />

</LinearLayout>
```

编译并运行 **BindingTest** 项目。应用程序将启动并显示在左侧的屏幕截图-，点击计数辅音，结果将显示在右侧所示：



祝贺你！您已成功绑定 Java 库。AAR ！

总结

在本演练中，我们创建了一个绑定库。AAR 文件，将绑定库添加到最小化测试应用，并运行应用程序以验证我们 C# 代码可以调用 Java 代码驻留在中。AAR 文件。此外，我们扩展此应用访问和显示驻留在的图像资源。AAR 文件。

相关链接

- [生成 Java 绑定库（视频）](#)
- [绑定 JAR](#)
- [绑定 Java 库](#)
- [AarBinding（示例）](#)

- [Bug 44573](#) 一项目无法绑定多个.aar 文件

绑定 Eclipse 库项目

2018/10/26 • [Edit Online](#)

此演练说明了如何使用 *Xamarin.Android* 项目模板绑定一个 *Eclipse Android* 库项目。

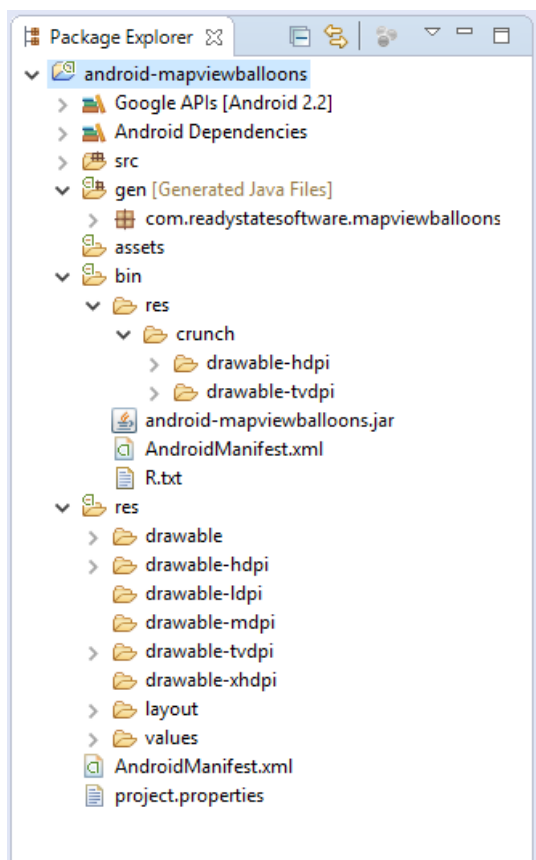
概述

尽管。AAR 文件正在逐渐成为 Android 库分发的标准，在某些情况下它为创建的绑定所需 *Android* 库项目。Android 库项目都包含可共享代码的特殊 Android 项目和 Android 应用程序项目，可以引用的资源。通常情况下，您将绑定到一个 Android 库项目时在 Eclipse IDE 中创建库。本演练介绍如何创建一个 Android 库项目的示例。从 Eclipse 项目的目录结构压缩。

Android 库项目的不同于常规的 Android 项目在于这些不编译到一个 APK，不，代表自身，可部署到设备。而是意味着一个 Android 库项目引用的 Android 应用程序项目。生成 Android 应用程序项目时，会首先编译 Android 库项目。Android 应用程序项目将会吸收到已编译的 Android 库项目，然后将代码和资源包含到分发的 APK。由于这种差异，创建一个 Android 库项目的绑定是创建 Java 绑定稍有不同。JAR 或。AAR 文件。

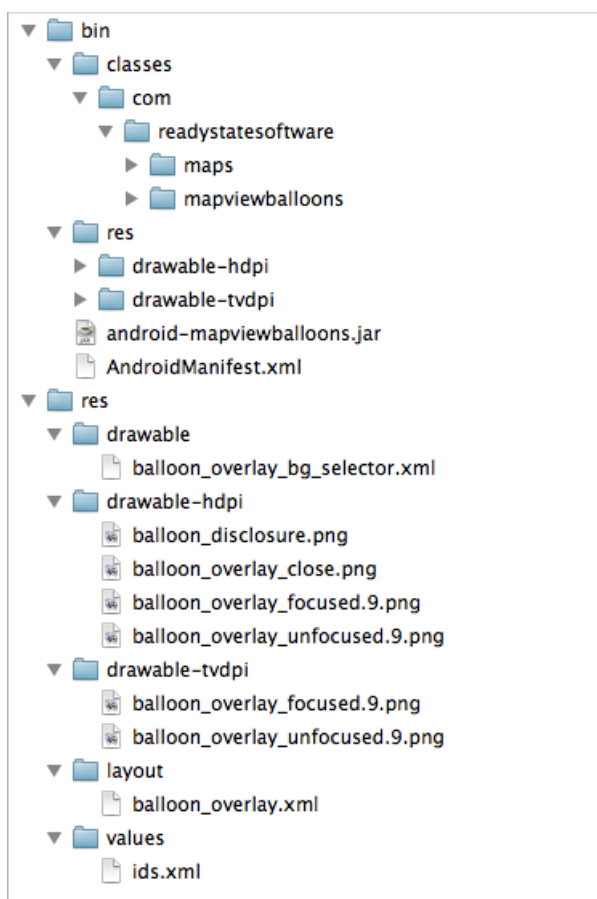
演练

在 Xamarin.Android Java 绑定项目中使用的 Android 库项目它时，第一个构建在 Eclipse 中的 Android 库项目。编译完成后，下面的屏幕截图显示一个 Android 库项目的示例：

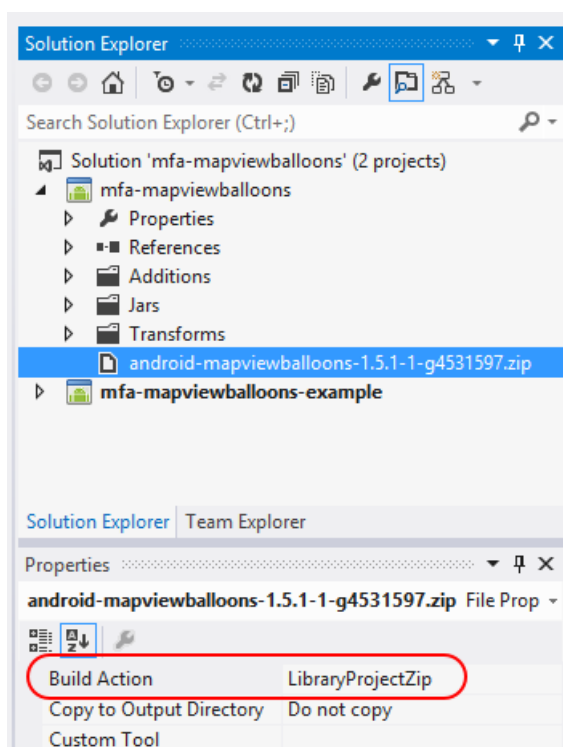


请注意，已从 Android 库项目的源代码编译到一个临时。名为 JAR 文件 **android mapviewballoons.jar**，和资源复制到 **bin/res/危机** 文件夹。

在 Eclipse 中编译之后的 Android 库项目，它然后可将绑定使用 Xamarin.Android Java 绑定项目。第一个。必须创建 ZIP 文件，其中包含 **bin** 并 **res** Android 库项目的文件夹。务必删除中间 **危机** 子目录，以便资源位于 **bin/res**。下面的屏幕截图显示了内容的其中一个此类。ZIP 文件：

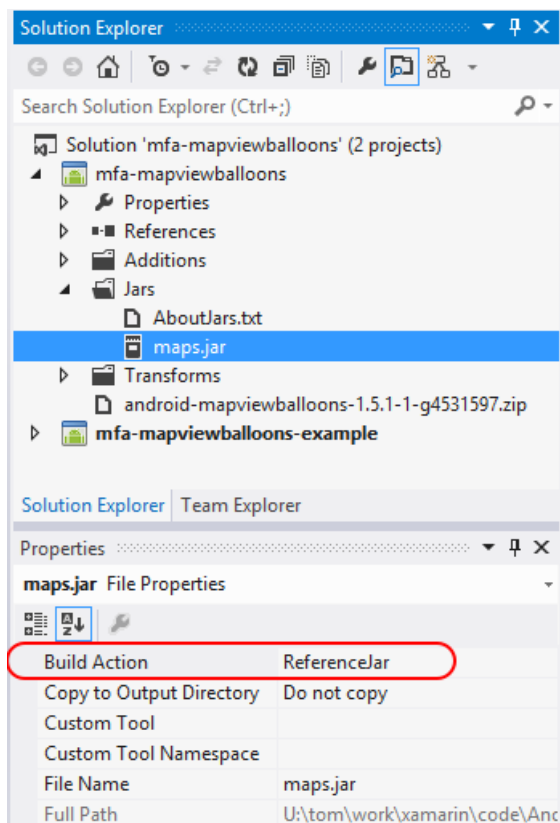


这。ZIP 文件然后添加到 Xamarin.Android Java 绑定项目中，如以下屏幕截图中所示：



请注意，生成操作的。ZIP 文件已自动设置为**LibraryProjectZip**。

如果有的话。所需的 Android 库项目的 JAR 文件，应将它们添加到**Jar** Java 绑定库项目的文件夹和**生成操作**设置为**ReferenceJar**。此示例所示的屏幕截图：



完成这些步骤后，如前面在本文档中所述，可以使用 Xamarin.Android Java 绑定项目。

NOTE

目前不支持在其他 Ide 中编译的 Android 库项目。相同的目录结构或中的文件不能创建其他 Ide **bin** Eclipse 的文件夹。

总结

在本文中，我们介绍了通过绑定一个 Android 库项目的过程。我们构建 Android 库项目在 Eclipse 中，然后我们创建了一个 zip 文件 **bin** 并 **res** Android 库项目的文件夹。接下来，我们使用此 zip 创建 Xamarin.Android Java 绑定项目。

自定义绑定

2018/10/26 • [Edit Online](#)

通过编辑控制绑定过程的元数据, 可以自定义的 Xamarin.Android 绑定。这些手动修改通常是必需的解决生成错误的定型所获得的 API, 这样就与更加一致 C#/.NET。这些指南介绍了此元数据的结构、如何修改的元数据, 以及如何使用 Javadoc 恢复方法参数的名称。

概述

Xamarin.Android 可自动执行大部分绑定过程中;但是, 在某些情况下手动修改被需要解决以下问题:

- 解决生成错误引起的缺少类型、经过模糊处理的类型、重复的名称、类的可见性问题, 以及其他无法解析的情况下由 Xamarin.Android 工具。
- 更改 Xamarin.Android 使用要绑定到中的不同类型的 Android API 映射 C#(例如, 许多开发人员更喜欢映射 Java `int` 常量 C# `enum` 常量)。
- 正在删除未使用不需要绑定的类型。
- 添加基础 Java API 中有任何相对应的类型。

可以通过修改控制绑定过程的元数据进行部分或所有这些更改。

参考线

以下指南描述了控制绑定过程的元数据, 并解释如何修改此元数据来解决这些问题:

- [Java 绑定元数据](#)提供进入 Java 绑定元数据的概述。它介绍了完成 Java 绑定库, 有时需要的各种手动步骤, 并介绍如何来调整一个绑定以更紧密地遵循.NET 设计指南所公开的 API。
- [使用 Javadoc 命名参数](#)介绍了如何通过使用从绑定的 Java 项目生成的 Javadoc 恢复 Java 绑定项目中的参数名称。

Java 绑定元数据

2018/11/2 • [Edit Online](#)

C#在 Xamarin.Android 中的代码是一种机制, 用于提取指定了在 Java 本机接口 (JNI) 的低级别详细信息的绑定通过调用 Java 库。Xamarin.Android 提供了一个工具, 将生成这些绑定。此工具可让开发人员控件如何通过使用元数据, 这允许如修改命名空间和重命名成员的过程创建一个绑定。本文档介绍元数据的工作原理, 汇总了的属性的元数据支持, 并说明如何通过修改此元数据来纠正绑定问题。

概述

Xamarin.Android **Java** 绑定库尝试自动执行所需绑定有时称为的工具的帮助的现有 Android 库的工作大部分_绑定生成器_。在绑定 Java 库时, Xamarin.Android 将检查的 Java 类, 并生成所有包、类型和成员的列表的绑定。Api 的此列表存储在一个 XML 文件, 请参阅 {项目 **directory**}\obj\Release\api.xml 有关 发布 生成而在 {项目 **directory**}\obj\Debug\api.xml 有关 调试 生成。

| Name | ^ | Date Modified | Size | Kind |
|--|---|------------------------|--------|-------------------|
| ▶ Additions | | Mar 31, 2016, 2:36 PM | -- | Folder |
| ▶ bin | | Apr 6, 2016, 10:47 AM | -- | Folder |
| evernote-android-job.csproj | | Jun 23, 2016, 12:37 PM | 4 KB | Xamari...Project |
| ▶ Jars | | Apr 1, 2016, 10:55 AM | -- | Folder |
| ▼ obj | | Jun 23, 2016, 12:55 PM | -- | Folder |
| ▼ Debug | | Jun 23, 2016, 12:44 PM | -- | Folder |
| _AndroidLibraryProjects_.zip | | Jun 23, 2016, 12:44 PM | 84 KB | ZIP archive |
| ▶ _library_projects_ | | Jun 23, 2016, 12:44 PM | -- | Folder |
| api.xml | | Jun 23, 2016, 12:44 PM | 103 KB | XML Document |
| evernote-android-job.csproj.FilesWrittenAbsolute.txt | | Jun 23, 2016, 12:44 PM | 1 KB | Plain Text |
| evernote-android-job.dll | | Jun 23, 2016, 12:44 PM | 176 KB | Microso...library |
| evernote-android-job.dll.mdb | | Jun 23, 2016, 12:44 PM | 23 KB | Document |
| evernoteandroidjob.Jars.cat-1.0.3.jar | | Feb 22, 2016, 12:33 PM | 12 KB | Java JAR file |
| evernoteandroidjob.obj.Debug_..._AndroidLibraryProjects_.zip | | Jun 23, 2016, 12:44 PM | 84 KB | ZIP archive |

将使用绑定生成器api.xml指导原则是用于生成所需的文件C#的包装类。此 XML 文件的内容是 Google 的一种变体_Android Open Source Project_格式。以下代码片段示范了的内容api.xml:

```
<api>
  <package name="android">
    <class abstract="false" deprecated="not deprecated" extends="java.lang.Object"
      extends-generic-aware="java.lang.Object"
      final="true"
      name="Manifest"
      static="false"
      visibility="public">
      <constructor deprecated="not deprecated" final="false"
        name="Manifest" static="false" type="android.Manifest"
        visibility="public">
      </constructor>
    </class>
  </package>
  ...
</api>
```

在此示例中, **api.xml**声明中的一个类 `android` 名为包 `Manifest` 扩展 `java.lang.Object`。

在许多情况下, 人工协助都需要使 Java API 感觉更多"等.NET"或以更正问题导致从编译的绑定程序集。例如, 它可能需要将 Java 包名称更改为.NET 命名空间、重命名一个类, 或更改一种方法的返回类型。

这些更改不通过修改api.xml直接。相反, Java 绑定库模板提供的特殊 XML 文件中记录的更改。在编译 Xamarin.Android 绑定程序集时, 绑定生成器创建的绑定程序集时将受这些映射文件

这些 XML 映射文件，可以在下文转换项目文件夹：

- **MetaData.xml** –允许更改不会对最终的 API，例如，更改所生成的绑定的命名空间。
- **EnumFields.xml** –包含 Java 之间的映射 `int` 常量和 C# `enums`。
- **EnumMethods.xml** –允许通过 Java 更改方法参数和返回类型 `int` 常量到 C# `enums`。

MetaData.xml文件是最导入这些文件，因为它允许对等绑定的常规用途更改：

- 因此，它们遵循.NET 约定重命名命名空间、类、方法或字段。
- 正在删除命名空间、类、方法或不需要的字段。
- 将类移到不同的命名空间。
- 添加其他支持类，以使绑定的设计遵循.NET framework 模式。

允许继续讨论**Metadata.xml**中更多详细信息。

Metadata.xml 转换文件

正如我们已经获知，该文件**Metadata.xml**绑定生成器用于影响绑定程序集创建。使用元数据格式XPath语法并且几乎完全相同GAPI 元数据中所述GAPI 元数据指南。此实现是几乎 XPath 1.0 的完整实现，因此支持标准 1.0 中的项。此文件是强大的基于 XPath 机制来更改、添加、隐藏或移动 API 文件中的任何元素或属性。所有元数据规范中的规则元素都包括一个路径属性来确定的规则是要应用的节点。按以下顺序应用规则：

- **添加节点**–将子节点追加到指定的路径属性的节点。
- **attr**–设置路径属性指定的元素的属性的值。
- **删除节点**–匹配指定的 XPath 节点中删除。

以下是一种**Metadata.xml**文件：

```
<metadata>
  <!-- Normalize the namespace for .NET -->
  <attr path="/api/package[@name='com.evernote.android.job']"
        name="managedName">Evernote.AndroidJob</attr>

  <!-- Don't need these packages for the Xamarin binding/public API -->
  <remove-node path="/api/package[@name='com.evernote.android.job.v14']" />
  <remove-node path="/api/package[@name='com.evernote.android.job.v21']" />

  <!-- Change a parameter name from the generic p0 to a more meaningful one. -->
  <attr
    path="/api/package[@name='com.evernote.android.job']/class[@name='JobManager']/method[@name='forceApi']/parameter[@name='p0']"
    name="name">api</attr>
</metadata>
```

下面列出了一些更常使用的 XPath 元素的 Java api:

- `interface` – 用于查找 Java 接口。例如 `/interface[@name='AuthListener']`。
- `class` – 用于查找一个类。例如 `/class[@name='MapView']`。
- `method` – 用于查找上一个 Java 类或接口的方法。例如 `/class[@name='MapView']/method[@name='setTitleSource']`。
- `parameter` – 标识一个方法的参数。例如 `/parameter[@name='p0']`。

添加类型

`add-node` 元素将告知 Xamarin.Android 绑定项目添加到新的包装器类 **api.xml**。例如，以下代码片段将指示要创建的类型具有一个构造函数和单个字段的绑定生成器：

```
<add-node path="/api/package[@name='org.alljoyn.bus']">
  <class abstract="false" deprecated="not deprecated" final="false" name="AuthListener.AuthRequest"
    static="true" visibility="public" extends="java.lang.Object">
    <constructor deprecated="not deprecated" final="false" name="AuthListener.AuthRequest" static="false"
      type="org.alljoyn.bus.AuthListener.AuthRequest" visibility="public" />
    <field name="p0" type="org.alljoyn.bus.AuthListener.Credentials" />
  </class>
</add-node>
```

删除类型

很可能会指示要忽略的 Java 类型并不将其绑定的 Xamarin.Android 绑定生成器。这是通过添加 `remove-node` XML 元素 **metadata.xml** 文件：

```
<remove-node path="/api/package[@name='{package_name}']/class[@name='{name}']" />
```

重命名成员

重命名成员不能通过直接编辑 **api.xml** 文件，因为 Xamarin.Android 需要原始 Java 本机接口 (JNI) 名称。因此，`//class/@name` 属性不能进行更改；如果是，绑定将不起作用。

我们想要重命名类型，这种情况，请考虑 `android.Manifest`。若要实现此目的，我们可能会尝试直接编辑 **api.xml** 和重命名一个类如下所示：

```
<attr path="/api/package[@name='android']/class[@name='Manifest']"
  name="name">NewName</attr>
```

这将导致绑定生成器创建以下 C# 包装器类的代码：

```
[Register ("android/NewName")]
public class NewName : Java.Lang.Object { ... }
```

请注意，包装器类已重命名为 `NewName`，而原始 Java 类型仍为 `Manifest`。不再可以访问任意方法上的 Xamarin.Android 绑定类 `android.Manifest`；包装类绑定到不存在的 Java 类型。

若要正确更改托管包装类型（或方法）的名称，是需要设置 `managedName` 特性，如本示例中所示：

```
<attr path="/api/package[@name='android']/class[@name='Manifest']"
  name="managedName">NewName</attr>
```

重命名 `EventArgs` 包装类

Xamarin.Android 绑定生成器时标识 `onXXX` setter 方法_侦听器类型_、C#事件和 `EventArgs` 将生成子类以支持.NET flavoured API 的基于 Java 的侦听器模式。作为示例，请考虑下面的 Java 类和方法：

```
com.someapp.android.mpa.guidance.NavigationManager.on2DSignNextManuever(NextManueverListener listener);
```

Xamarin.Android 将删除该前缀 `on` 从资源库方法，改为使用 `2DSignNextManuever` 作为名称的基础 `EventArgs` 子类。子类将被命名为类似于：

```
NavigationManager.2DSignNextManueverEventArgs
```

这不是合法C#类名。若要更正此问题，绑定作者必须使用 `argsType` 特性，并提供有效C#的名称 `EventArgs` 子类：

```
<attr path="/api/package[@name='com.someapp.android.mpa.guidance']/
  interface[@name='NavigationManager.Listener']/
  method[@name='on2DSignNextManeuver']"
  name="argsType">NavigationManager.TwoDSignNextManueverEventArgs</attr>
```

支持的属性

以下各节介绍了一些用于转换 Java Api 的属性。

argsType

此特性置于 setter 方法来命名 `EventArgs` 将生成以支持 Java 侦听器的子类。这更详细地下面部分中所述 [重命名 EventArgs 包装类](#) 稍后在本指南中。

事件名称

指定事件的名称。如果为空，它会抑制事件生成。这部分标题中的更多详细信息中所述 [重命名 EventArgs 包装类](#)。

managedName

这用于更改包、类、方法或参数的名称。例如，若要更改的 Java 类的名称 `MyClass` 到 `NewClassName`：

```
<attr path="/api/package[@name='com.my.application']/class[@name='MyClass']"
  name="managedName">NewClassName</attr>
```

下一个示例说明了重命名该方法的 XPath 表达式 `java.lang.object.toString` 到 `Java.Lang.Object.NewManagedName`：

```
<attr path="/api/package[@name='java.lang']/class[@name='Object']/method[@name='toString']"
  name="managedName">NewMethodName</attr>
```

managedType

`managedType` 用于更改一种方法的返回类型。在某些情况下绑定生成器会错误地推断返回类型的 Java 方法，这将导致编译时错误。在此情况下一个可能的解决方案是更改该方法的返回类型。

例如，绑定生成器认为的 Java 方法 `de.neom.neoreadersdk.resolution.compareTo()` 应返回 `int`，这会导致错误消息错误 **CS0535: DE. Neom.Neoreadersdk.Resolution 不实现接口成员**

Java.Lang.IComparable.CompareTo(Java.Lang.Object)。以下代码片段演示如何更改所生成的返回类型C#方法从 `int` 到 `Java.Lang.Object`：

```
<attr path="/api/package[@name='de.neom.neoreadersdk']/
  class[@name='Resolution']/
  method[@name='compareTo' and count(parameter)=1 and
  parameter[1][@type='de.neom.neoreadersdk.Resolution']]/
  parameter[1]"name="managedType">Java.Lang.Object</attr>
```

managedReturn

更改方法的返回类型。这不会更改返回值属性（作为有所变化，返回属性可能会导致不兼容的更改到的 JNI 签名）。在下面的示例中的返回类型的 `append` 方法更改从 `SpannableStringBuilder` 到 `IAppendable`（回想一下，C#不支持协变返回类型）：


```
<attr path="/api/package[@name='android.text']/
    class[@name='SpannableStringBuilder']/
    method[@name='append']"
    name="managedReturn">Java.Lang.IAppendable</attr>
```

经过模糊处理

模糊处理 Java 库的工具可能会影响 Xamarin.Android 绑定生成器，并生成其功能与 C# 包装器类。经过模糊处理类的特征包括：* 类名称中包含 \$，即 **\$class** * 类名完全破坏的小写字符，即 **a.class**

此代码片段示范了如何生成一个“未经过模糊处理”C# 类型：

```
<attr path="/api/package[@name='{package_name}']/class[@name='{name}']"
    name="obfuscated">false</attr>
```

propertyName

此属性可以用于更改托管属性的名称。

使用的专用的化 `propertyName` 涉及这种情况，其中一个 Java 类具有仅一个字段的 getter 方法。在这种情况下绑定生成器会想要创建一个只写属性，不建议在 .NET 中的内容。以下代码片段演示如何通过设置“删除”的 .NET 属性 `propertyName` 为空字符串：

```
<attr
path="/api/package[@name='org.java_websocket.handshake']/class[@name='HandshakeImpl1Client']/method[@name='setResourceDescriptor'
    and count(parameter)=1
    and parameter[1][@type='java.lang.String']]"
    name="propertyName"></attr>
<attr
path="/api/package[@name='org.java_websocket.handshake']/class[@name='HandshakeImpl1Client']/method[@name='getResourceDescriptor'
    and count(parameter)=0]"
    name="propertyName"></attr>
```

请注意仍由绑定生成器将创建的 setter 和 getter 方法。

sender

指定一种方法的参数应为 `sender` 时该方法映射到一个事件参数。值可以是 `true` 或 `false`。例如：

```
<attr path="/api/package[@name='android.app']/
    interface[@name='TimePickerDialog.OnTimeSetListener']/
    method[@name='onTimeSet']/
    parameter[@name='view']"
    name="sender">true</ attr>
```

可见性

此属性用于类、方法或属性的可见性更改。例如，可能有必要将提升 `protected` Java 方法，以便它的相应 C# 包装是否 `public`：

```
<!-- Change the visibility of a class -->
<attr path="/api/package[@name='namespace']/class[@name='ClassName']" name="visibility">public</attr>

<!-- Change the visibility of a method -->
<attr path="/api/package[@name='namespace']/class[@name='ClassName']/method[@name='MethodName']"
    name="visibility">public</attr>
```

EnumFields.xml 和 EnumMethods.xml

有些情况下，Android 库使用整数常量表示传递到库的属性或方法的状态。在许多情况下，最好先绑定到枚举中的这些整数常量C#。若要简化此映射，请使用**EnumFields.xml**并**EnumMethods.xml**绑定项目中的文件。

定义使用 EnumFields.xml 枚举

EnumFields.xml文件包含 Java 之间的映射 `int` 常量和C# `enums`。让我们看下面的示例对C#正在创建的一组用于枚举 `int` 常量：

```
<mapping jni-class="com/skobbler/ngx/map/realreach/SKRealReachSettings" clr-enum-type="Skobbler.Ngx.Map.RealReach.SKMeasurementUnit">
  <field jni-name="UNIT_SECOND" clr-name="Second" value="0" />
  <field jni-name="UNIT_METER" clr-name="Meter" value="1" />
  <field jni-name="UNIT_MILLIWATT_HOURS" clr-name="MilliwattHour" value="2" />
</mapping>
```

此处我们已采取的 Java 类 `SKRealReachSettings` 并定义C#称作 `SKMeasurementUnit` 命名空间中 `Skobbler.Ngx.Map.RealReach`。 `field` 条目定义的 Java 常量的名称 (示例 `UNIT_SECOND`)，枚举项的名称 (示例 `Second`)，并且这两个实体所表示的整数值 (示例 `0`)。

定义使用 EnumMethods.xml 的 Getter/Setter 方法

EnumMethods.xml文件允许从 Java 更改方法参数和返回类型 `int` 常量到C# `enums`。换言之，它将映射的读取和写入的C#枚举 (中定义**EnumFields.xml**文件) 到 Java `int` 常量 `get` 并 `set` 方法。

给定 `SKRealReachSettings` 枚举定义更高版本，以下**EnumMethods.xml**文件会定义 `getter/setter` 此枚举：

```
<mapping jni-class="com/skobbler/ngx/map/realreach/SKRealReachSettings">
  <method jni-name="getMeasurementUnit" parameter="return" clr-enum-type="Skobbler.Ngx.Map.RealReach.SKMeasurementUnit" />
  <method jni-name="setMeasurementUnit" parameter="measurementUnit" clr-enum-type="Skobbler.Ngx.Map.RealReach.SKMeasurementUnit" />
</mapping>
```

第一个 `method` 行映射 Java 的返回值 `getMeasurementUnit` 方法 `SKMeasurementUnit` 枚举。第二个 `method` 行映射的第一个参数 `setMeasurementUnit` 到相同的枚举。

所有这些更改后，你可以使用下面的代码在 Xamarin.Android 中设置 `MeasurementUnit`：

```
realReachSettings.MeasurementUnit = SKMeasurementUnit.Second;
```

总结

本文讨论了如何 Xamarin.Android 使用元数据转换中的 API 定义Google AOSP 格式。介绍可能的更改后使用**Metadata.xml**，它会检查重命名成员时遇到的限制和显示受支持的 XML 属性，描述应在何时使用每个属性的列表。

相关链接

- [使用 JNI](#)
- [绑定 Java 库](#)
- [GAPI 元数据](#)

使用 Javadoc 命名参数

2018/10/26 • [Edit Online](#)

此文章介绍了如何使用生成 Java 项目中的 Javadoc 恢复 Java 绑定项目中的参数名称。

概述

绑定现有的 Java 库，会丢失一些有关绑定 API 元数据。特别是对方法的参数的名称。参数名称将显示为 `p0`，`p1`，等等。这是因为 Java `.class` 文件不会保留在 Java 源代码中使用的参数名称。

如果它有权访问 Javadoc HTML 从原始库，Xamarin.Android Java 绑定项目可以提供参数名称。

将 Javadoc HTML 集成到 Java 绑定项目

将 Javadoc HTML 集成到 Java 绑定项目是一个手动过程包括以下步骤：

1. 下载库 Javadoc
2. 编辑 `.csproj` 文件，并添加 `<JavaDocPaths>` 属性：
3. 清除并重新生成项目

完成此操作后，原始 Java 参数名称应会出现在由 Java 绑定项目绑定的 Api。

NOTE

没有大量的 Javadoc 输出中的变体。。JAR 绑定工具链不支持每个单个可能的排列，因此某些参数可能未正确命名。

总结

本文介绍如何使用 Java 绑定项目中的 Javadoc 提供的绑定 Api 的含义参数名称。

绑定疑难解答

2018/10/26 • [Edit Online](#)

本文汇总了生成绑定, 以及可能的原因和解决这些问题的建议的方法时可能发生的多种常见错误。

概述

绑定 Android 库 (.aar或.jar) 文件很少是简单的会议时间; 它通常需要进行其他工作来缓解问题而导致的 Java 和.NET 之间的差异。这些问题将阻止 Xamarin.Android 绑定 Android 库和它们自身显示为生成日志中的错误消息。本指南将提供一些提示, 用于解决问题, 列出了一些较为常见的问题/情景, 并提供可能的解决方案成功绑定到 Android 库。

绑定现有 Android 库时, 必须要时刻牢记以下几点:

- **库的外部依赖关系**—必须为 Xamarin.Android 项目中包含所需的 Android 库的任何 Java 依赖项 **ReferenceJar**或是 **EmbeddedReferenceJar**。
- **Android 库所面向的 Android API 级别**—它无法"降级"的 Android API 级别; 确保, Xamarin.Android 绑定项目面向的相同的 API 级别 (或更高) 为 Android 库。
- **用于打包 Android 库的 Android jdk 版本**—绑定错误可能会发生, 如果使用不同版本的 JDK 比使用 xamarin.android 生成 Android 库。如果可能, 请重新编译使用相同的 Xamarin.Android 安装使用的 jdk 版本的 Android 库。

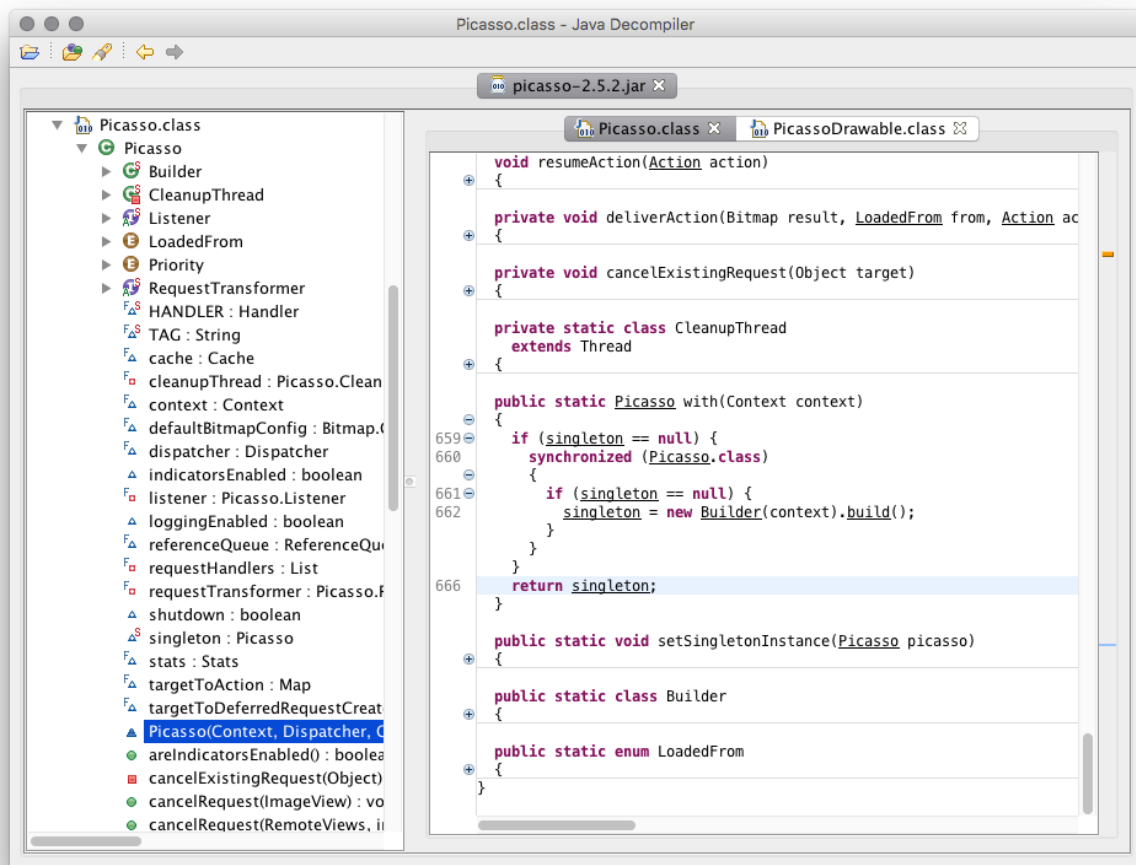
解决问题的绑定 Xamarin.Android 库的第一步是能够[诊断 MSBuild 输出](#)。启用诊断输出后, 重新生成 Xamarin.Android 绑定项目并检查生成日志, 以找到有关问题的原因的线索。

它可以证明反编译 Android 库并检查的类型和 Xamarin.Android 尝试绑定的方法很有帮助。这一点在稍后在本指南中的更多详细信息。

反编译的 Android 库

检查的类和方法的 Java 类可以提供有价值的信息会帮助你绑定库。[JD GUI](#)是一个图形实用工具, 可以显示从 Java 源代码类 JAR 中包含的文件。它可以作为独立应用程序或插件为 IntelliJ 或运行 Eclipse。

反编译一个 Android 库打开 **JAR** Java 反编译器使用的文件。如果库 **AAR**文件, 但有必要从中提取文件 **classes.jar**从存档文件。以下是使用 JD GUI 来分析的示例屏幕截图[看毕加索会如何JAR](#):



一旦具有反编译程序 Android 库，检查源代码。通常情况下，查找：

- 具有模糊处理的特征类—经过模糊处理类的特征包括：
 - Class 名称中包含**\$**，即 **\$.class**
 - 类名完全破坏的小写字符，即 **a.class**
- `import` 语句未引用的库—标识未引用的库并将这些依赖项添加到 Xamarin.Android 绑定项目与生成操作的 **ReferenceJar** 或 **EmbedddedReferenceJar**。

NOTE

反编译 Java 库可能会禁止或受到法律限制的约束根据当地的法律或在其下发布 Java 库的许可证。如有必要，然后再尝试反编译 Java 库和检查的源代码登记法律专业人士的服务。

检查 API.XML

生成一个绑定项目的一部分，Xamarin.Android 将生成 XML 文件名 **obj/Debug/api.xml**：

| Name | ^ | Date Modified | Size | Kind |
|--|---|------------------------|--------|-------------------|
| ▶ Additions | | Mar 31, 2016, 2:36 PM | -- | Folder |
| ▶ bin | | Apr 6, 2016, 10:47 AM | -- | Folder |
| ✖ evernote-android-job.csproj | | Jun 23, 2016, 12:37 PM | 4 KB | Xamarin...Project |
| ▶ Jars | | Apr 1, 2016, 10:55 AM | -- | Folder |
| ▼ obj | | Jun 23, 2016, 12:55 PM | -- | Folder |
| ▼ Debug | | Jun 23, 2016, 12:44 PM | -- | Folder |
| __AndroidLibraryProjects__.zip | | Jun 23, 2016, 12:44 PM | 84 KB | ZIP archive |
| ▶ __library_projects__ | | Jun 23, 2016, 12:44 PM | -- | Folder |
| api.xml | | Jun 23, 2016, 12:44 PM | 103 KB | XML Document |
| evernote-android-job.csproj.FilesWrittenAbsolute.txt | | Jun 23, 2016, 12:44 PM | 1 KB | Plain Text |
| evernote-android-job.dll | | Jun 23, 2016, 12:44 PM | 176 KB | Microso...library |
| evernote-android-job.dll.mdb | | Jun 23, 2016, 12:44 PM | 23 KB | Document |
| evernoteandroidjob.Jars.cat-1.0.3.jar | | Feb 22, 2016, 12:33 PM | 12 KB | Java JAR file |
| evernoteandroidjob.obj.Debug. __AndroidLibraryProjects__.zip | | Jun 23, 2016, 12:44 PM | 84 KB | ZIP archive |

此文件提供了 Xamarin.Android 正尝试绑定的所有 Java Api 的列表。此文件的内容可以帮助确定任何缺少的类型或方法，重复的绑定。尽管此文件进行检查是单调乏味并且耗时，但可提供用于在什么可能会导致任何绑定问题的线索。例如，**api.xml**属性返回一个不适当的类型，或有两个类型为该共享相同的托管名称可能会显示。

已知问题

本部分将列出的一些常见的错误消息或症状，我尝试绑定一个 Android 库时发生。

问题：Java 版本不匹配

有时不会生成类型或意外的故障可能是由于使用的 Java 库已与编译相比更高版本或较旧版本。重新编译使用相同版本的 Xamarin.Android 项目使用的 JDK 的 Android 库。

至少一个 Java 库是必需的问题：

您会收到错误“至少一个 Java 库是必需的”即使。已添加 JAR。

可能的原因：

请确保生成操作设置为 `EmbeddedJar`。由于没有对应的多个生成操作。JAR 文件（如 `InputJar`，`EmbeddedJar`，`ReferenceJar` 和 `EmbeddedReferenceJar`），不能自动在默认情况下使用哪一个猜出绑定生成器。有关生成操作的详细信息，请参阅[生成操作](#)。

问题：绑定工具无法加载。JAR 库

绑定库生成器无法加载。JAR 库。

可能的原因

一些。Java 工具不能加载使用（通过 Proguard 等工具）的代码混淆的 JAR 库。由于我们的工具可以利用 Java 反射和工程库的 ASM 字节代码，这些从属工具可能会拒绝的经过模糊处理的库，虽然 Android 运行时工具可能会将传递。对此解决方法是手动绑定而不是使用绑定生成器这些库。

问题：缺少 C# 中生成的输出类型。

绑定 .dll 生成，但未命中某些 Java 数据类型，或生成 C# 源未生成由于一个错误，指出缺少的类型。

可能的原因：

此错误可能有多种原因，如下所示：

- 要绑定的库可能引用第二个 Java 库。如果绑定库的公共 API 使用从第二个库类型，则必须引用第二个库的托管的绑定。
- 有可能库被注入由于 Java 反射，类似于上面，从而导致意外的元数据加载库加载错误的原因。Xamarin.Android 的工具当前不能解决此问题。在这种情况下，必须手动绑定库。
- 未能加载程序集时应具有的 .NET 4.0 运行时中出现 bug。在 .NET 4.5 运行时中已修复此问题。
- Java 允许从非公共类，派生的公共类，但这在 .NET 中不支持。因为绑定生成器不生成绑定的非公共类，派生类，如这些无法正确生成。若要解决此问题，请删除这些派生类使用中的删除节点的元数据条目 `Metadata.xml`，或修复公开非公共类的元数据。尽管后一种解决方案将创建绑定，以便 C# 将生成源，不

应使用非公共类。

例如：

```
<attr path="/api/package[@name='com.some.package']/class[@name='SomeClass']"
      name="visibility">public</attr>
```

- 模糊处理 Java 库的工具可能会影响 Xamarin.Android 绑定生成器，并生成其功能与 C# 包装器类。以下代码片段演示如何更新 **Metadata.xml** 以 unobfuscate 类名：

```
<attr path="/api/package[@name='{package_name}']/class[@name='{name}']"
      name="obfuscated">false</attr>
```

问题：生成 C# 源未生成由于参数类型不匹配

生成 C# 不生成源。重写方法的参数类型不匹配。

可能的原因：

Xamarin.Android 包含了多种映射到枚举中的 Java 字段的 C# 绑定。这些会导致生成的绑定中的类型不兼容问题。若要解决此问题，从绑定生成器创建的方法签名需要进行修改以使用枚举。有关详细信息，请参阅[更正枚举](#)。

包中的问题：NoClassDefFoundError

`java.lang.NoClassDefFoundError` 在打包步骤中，将引发。

可能的原因：

此错误最可能的原因是必需的 Java 库需要添加到应用程序项目 (**.csproj**)。JAR 文件不自动解决。针对目标设备或仿真程序中不存在的用户程序集始终不生成 Java 库绑定 (如 Google Maps **maps.jar**)。这是与库的不用于 Android 库项目支持这种情况。在类库 dll 中嵌入 JAR。例如：[Bug 4288](#)

问题：重复的自定义 EventArgs 类型

由于重复的自定义 EventArgs 类型，生成失败。发生如下错误：

```
error CS0102: The type `Com.Google.Ads.Mediation.DismissScreenEventArgs' already contains a definition for `p0'
```

可能的原因：

这是因为来自多个共享具有相同名称的方法的接口“侦听器”类型的事件类型之间的某些冲突。例如，如果在下面的示例所示，有两个 Java 接口，生成器将创建 `DismissScreenEventArgs` 同时 `MediationBannerListener` 和 `MediationInterstitialListener`，从而导致错误。

```
// Java:
public interface MediationBannerListener {
    void onDismissScreen(MediationBannerAdapter p0);
}
public interface MediationInterstitialListener {
    void onDismissScreen(MediationInterstitialAdapter p0);
}
```

这是特意设计，以便避免了事件自变量类型上的耗时较长名称。若要避免这些冲突，某些元数据转换是必需的。编辑 **Transforms\Metadata.xml**，并添加 `argType` 接口中的任何一个（或在接口方法）的属性：

```

<attr path="/api/package[@name='com.google.ads.mediation']/
    interface[@name='MediationBannerListener']/method[@name='onDismissScreen']"
    name="argsType">BannerDismissScreenEventArgs</attr>

<attr path="/api/package[@name='com.google.ads.mediation']/
    interface[@name='MediationInterstitialListener']/method[@name='onDismissScreen']"
    name="argsType">InterstitialDismissScreenEventArgs</attr>

<attr path="/api/package[@name='android.content']/
    interface[@name='DialogInterface.OnClickListener']"
    name="argsType">DialogClickEventArgs</attr>

```

问题：类不实现接口方法

指示生成的类不实现所需的生成的类实现的接口的方法来生成一条错误消息。但是，看一下生成的代码，您可以看到此方法实现。

下面是错误的示例：

```

obj\Debug\generated\src\Oauth.Signpost.Basic.HttpURLConnectionRequestAdapter.cs(8,23):
error CS0738: 'Oauth.Signpost.Basic.HttpURLConnectionRequestAdapter' does not
implement interface member 'Oauth.Signpost.Http.IHttpRequest.Unwrap()'.
'Oauth.Signpost.Basic.HttpURLConnectionRequestAdapter.Unwrap()' cannot implement
'Oauth.Signpost.Http.IHttpRequest.Unwrap()' because it does not have the matching
return type of 'Java.Lang.Object'

```

可能的原因：

这是协变返回类型与绑定 Java 方法会出现一个问题。在此示例中，该方法

`Oauth.Signpost.Http.IHttpRequest.Unwrap()` 需要返回 `Java.Lang.Object`。但是，该方法

`Oauth.Signpost.Basic.HttpURLConnectionRequestAdapter.Unwrap()` 的返回类型为 `HttpURLConnection`。有两种方法来解决此问题：

- 添加的分部类声明 `HttpURLConnectionRequestAdapter` 和显式实现 `IHttpRequest.Unwrap()`：

```

namespace Oauth.Signpost.Basic {
    partial class HttpURLConnectionRequestAdapter {
        Java.Lang.Object OauthSignpost.Http.IHttpRequest.Unwrap() {
            return Unwrap();
        }
    }
}

```

- 从生成中删除协方差 C# 代码。这涉及到添加到以下转换 `Transforms\Metadata.xml` 这将导致生成 C# 代码的返回类型 `Java.Lang.Object`：

```

<attr

path="/api/package[@name='oauth.signpost.basic']/class[@name='HttpURLConnectionRequestAdapter']/method[
@name='unwrap']"
    name="managedReturn">Java.Lang.Object
</attr>

```

问题：名称冲突上的内部类 / 属性

继承的对象的冲突可见性。

在 Java 中，这不被必需的派生的类具有相同的可见性作为其父级。Java 只需修复，为您。在 C#，具有可以明确，因此您需要确保层次结构中的所有类都有相应的可见性。下面的示例演示如何从 Java 包名称更改

`com.evernote.android.job` 到 `Evernote.AndroidJob`：


```
<!-- Change the visibility of a class -->
<attr path="/api/package[@name='namespace']/class[@name='ClassName']" name="visibility">public</attr>

<!-- Change the visibility of a method -->
<attr path="/api/package[@name='namespace']/class[@name='ClassName']/method[@name='MethodName']"
name="visibility">public</attr>
```

问题：一个 **.so** 由绑定所需的库是未加载

某些绑定项目还取决于其中的功能 **.so** 库。很可能不会自动加载 Xamarin.Android **.so** 库。Xamarin.Android 已包装的 Java 代码执行时，将无法进行 JNI 调用和错误消息 `java.lang.UnsatisfiedLinkError: 找不到的本机方法:_` 会在 logcat 出应用程序中显示。

此解决方法是手动加载 **.so** 库通过调用 `Java.Lang.JavaSystem.LoadLibrary`。例如假设 Xamarin.Android 项目已共享库 **libpocketsphinx_jni.so** 绑定项目的生成操作中包含 **EmbeddedNativeLibrary**，以下代码片段(使用共享的库之前执行) 将加载 **.so** 库：

```
Java.Lang.JavaSystem.LoadLibrary("pocketsphinx_jni");
```

总结

在本文中，我们将列出与 Java 绑定相关联的常见故障排除问题并说明了如何解决这些问题。

相关链接

- [库项目](#)
- [使用 JNI](#)
- [启用诊断输出](#)
- [适用于 Android 开发人员的 Xamarin](#)
- [JD-GUI](#)

使用本机库

2018/10/26 • [Edit Online](#)

Xamarin.Android 支持通过标准的 PInvoke 机制的本机库的使用。此外可以绑定其他本机库，后者不是你.apk 到操作系统的一部分。

若要部署 Xamarin.Android 应用程序使用的本机库，将二进制库添加到项目并设置其生成操作到**AndroidNativeLibrary**。

若要部署使用 Xamarin.Android 类库项目的本机库，将二进制库添加到项目并设置其生成操作到**EmbeddedNativeLibrary**。

请注意，由于 Android 支持多个应用程序二进制接口 (Abi)，则 Xamarin.Android 必须知道本机库专为哪个 ABI。可以通过两种方法完成：

1. 路径"探查"
2. 通过使用 `AndroidNativeLibrary/Abi` 位于项目文件中的元素

通过路径探查，本机库的父目录名称用于指定库的目标 ABI。因此，如果您将添加 `lib/armeabi/libfoo.so` 到项目中，然后 ABI 将被"探查"为 `armeabi`。

或者，可以编辑项目文件显式指定 ABI 使用：

```
<ItemGroup>
  <AndroidNativeLibrary Include="path/to/libfoo.so">
    <Abi>armeabi</Abi>
  </AndroidNativeLibrary>
</ItemGroup>
```

有关使用本机库的详细信息，请参阅[互操作使用本机库](#)。

使用 Visual Studio 2017 调试本机代码

如果您使用的 *Visual Studio 2017* 或更高版本，无需修改你的项目文件，如上文所述。您可以生成和调试 Xamarin.Android 解决方案内的 c + +，通过添加对 c + + 的项目引用动态共享库 (**Android**) 项目。

若要调试你的项目中的本机 c + + 代码，请按照下列步骤：

1. 双击项目属性，然后选择**Android** 选项页。
2. 向下滚动到调试选项。
3. 在中调试器下拉列表菜单中，选择c + + (而不是默认 **/.net (Xamarin)**)。

Visual Studio c + + 开发人员可以查看[SanAngeles_NativeDebug](#)示例尝试调试 c + + 中使用 Xamarin; Visual Studio 2017，请参考我们[博客文章](#)有关详细信息。

相关链接

- [SanAngeles_NativeDebug \(示例\)](#)
- [开发 Xamarin Android 本机应用程序](#)

Renderscript 简介

2018/10/26 • [Edit Online](#)

本指南介绍 *Renderscript* 并说明如何使用该目标 API 级别 17 或更高版本的内部函数 *Renderscript* 中的 API 的 *Xamarin.Android* 应用程序。

概述

Renderscript 是一个编程框架，由 Google 创建以便改进需要大量计算资源的 Android 应用程序的性能。它是基于 API 的较低级别、高性能 C99。因为它是低级别的 Cpu、Gpu 或 Dsp 将运行的 API，Renderscript 是非常适用于 Android 应用，可能需要执行以下任一项：

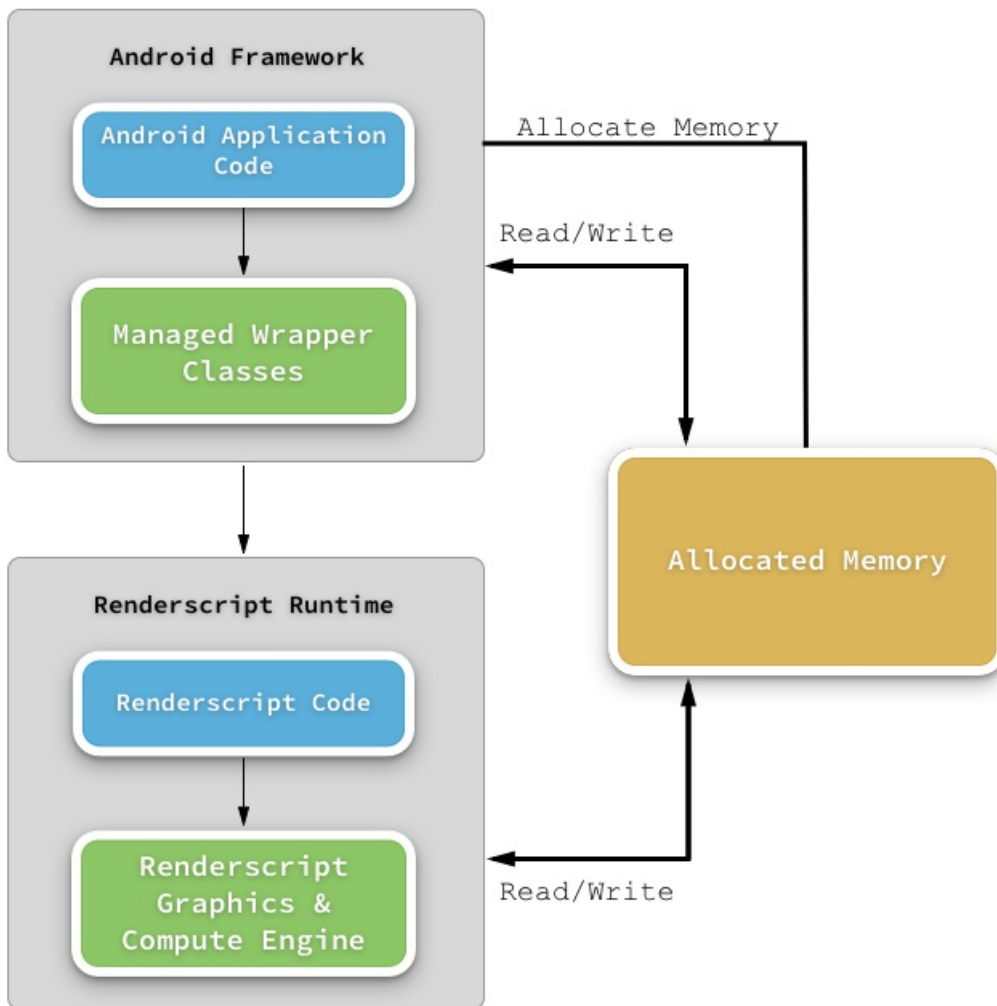
- 图形
- 图像处理
- 加密
- 信号处理
- 数学例程

将使用 Renderscript `clang` 并编译到 LLVM 字节代码捆绑到 apk 进行签名的脚本。第一次运行应用时，LLVM 字节代码将编译为机器代码，以在设备上的处理器。此体系结构允许的 Android 应用程序本身无需编写它为每个处理器在设备上自行利用的机器代码，而无需开发人员的优势。

有两个组件 Renderscript 例程：

1. **Renderscript 运行时**—这是负责执行 Renderscript 的本机 Api。这包括任何 Renderscripts 编写的应用程序。
2. **从 Android 框架托管包装**—托管允许的 Android 应用程序控制并 Renderscript 运行时和脚本与之交互的类。除了用于控制 Renderscript 运行时提供的框架类，Android 工具链将检查 Renderscript 源代码和生成 Android 应用程序使用的托管的包装类。

下图说明了这些组件之间的关系：



有三个 Android 应用程序中使用 Renderscripts 的重要概念：

1. **上下文**—一种托管的 Android SDK, 可将资源分配到 Renderscript 并允许 Android 应用程序传递和接收来自 Renderscript 数据提供的 API。
2. 一个 **计算内核**—也称为 **根内核** 或者 **内核**, 则此执行工作的例程。内核是非常类似于 C 函数;它是在分配的内存中的所有数据上运行的可并行化例程。
3. **分配的内存**—数据传递到和从通过内核 **分配**。内核可能有一个输入和/或一个输出分配。

`Android.Renderscripts`命名空间包含用来与 Renderscript 运行时交互的类。具体而言, `Renderscript` 类将管理的生命周期和 Renderscript 引擎的资源。一个或多个必须进行初始化的 Android 应用 `Android.Renderscripts.Allocation` 对象。分配是托管的 API, 它负责分配和访问的 Android 应用和 Renderscript 运行时之间共享的内存。通常情况下, 为输入, 创建一个资源分配, 并可以选择另一个分配创建用于保存该内核的输出。Renderscript 运行时引擎和关联的托管的包装类将管理对由分配的内存的访问, 则无需执行任何额外的工作应用的 Android 应用程序开发人员。

分配将包含一个或多个 `Android.Renderscripts.Elements`。元素是一种特殊的类型的描述中每个分配的数据。元素类型分配必须与输入元素的类型匹配的输。在执行时, 将循环并行情况下, 输入分配的每个元素并将结果写入到输出 Renderscript 分配。有两种类型的元素：

- **简单类型**—从概念上讲这是为 C 数据类型, 相同 `float` 或 `char`。
- **复杂类型**—此类型是类似于 C `struct`。

Renderscript 引擎将执行运行时检查以确保每个分配中的元素与所需的内核兼容。如果在分配的元素的数据类型与内核的期望的数据类型不匹配, 则将引发异常。

所有 Renderscript 内核将都包装的类型，是的后代 `Android.Renderscripts.Script` 类的新实例。`Script` 类用于为 Renderscript 设置参数，设置相应 `Allocations`，并运行 Renderscript。有两个 `Script` Android SDK 中的子类：

- `Android.Renderscripts.ScriptIntrinsic` – 一些常见 Renderscript 任务被捆绑在 Android SDK 中并且都可以访问的子类 `ScriptIntrinsic` 类。开发人员需要任何额外步骤以在其应用程序中使用这些脚本，因为它们已提供，没有必要。
- `ScriptC_XXXXX` – 也称为_用户脚本_，这些是由开发人员编写和打包在 APK 中的脚本。在编译时，Android 工具链将生成会允许脚本以在 Android 应用中使用的托管的包装类。这些生成的类的名称是带有前缀的 Renderscript 文件的名称 `ScriptC_`。编写并结合用户脚本不正式支持 xamarin.android 和超出了本指南的范围。

这两种类型，仅 `StringIntrinsic` Xamarin.Android 支持。本指南介绍了如何在 Xamarin.Android 应用程序中使用内部函数的脚本。

要求

本指南适用于 Xamarin.Android 应用程序的目标 API 级别 17 或更高版本。利用_用户脚本_本指南中未涉及。

[Xamarin.Android V8 支持库](#) backports 内部 Renderscript API 针对面向较旧版本的 Android SDK 的应用。将此包添加到 Xamarin.Android 项目应允许应用程序面向旧版本的 Android SDK，可利用内部函数的脚本。

在 Xamarin.Android 中使用内部函数 Renderscripts

内部函数的脚本是执行密集型计算任务进行少量的额外代码的好办法。它们已进行手动经过优化，可提供一大部分的设备上获得最佳性能。不常见的内部函数的脚本来运行 10 倍快于托管代码和 2-3 倍后比自定义的 C 实现的。内部函数的脚本的涵盖了许多典型处理方案。内部函数的脚本此列表描述了在 Xamarin.Android 中的当前脚本：

- [ScriptIntrinsic3DLUT](#) –将 RGB 将转换为 RGBA 使用 3D 查找表。
- [ScriptIntrinsicBLAS](#) – Provides high 性能 Renderscript Api 向 [BLAS](#)。BLAS（基本线性代数子程序）是提供用于执行基本的矢量和矩阵操作的标准构建基块的例程。
- [ScriptIntrinsicBlend](#) –将两个分配融合在一起。
- [ScriptIntrinsicBlur](#) –适用于分配的高斯模糊。
- [ScriptIntrinsicColorMatrix](#) –适用于分配（即更改色彩调整 hue）颜色矩阵。
- [ScriptIntrinsicConvolve3x3](#) –适用于分配的 3 x 3 个颜色矩阵。
- [ScriptIntrinsicConvolve5x5](#) –适用于分配的 5 x 5 颜色矩阵。
- [ScriptIntrinsicHistogram](#) –内部直方图筛选器。
- [ScriptIntrinsicLUT](#) –缓冲区应用于每个通道查找表。
- [ScriptIntrinsicResize](#) –执行调整大小的 2D 分配的脚本。
- [ScriptIntrinsicYuvToRGB](#) –将 YUV 缓冲区转换为 RGB。

请有关每个内部函数的脚本的详细信息，参阅 API 文档。

接下来介绍了在 Android 应用程序中使用 Renderscript 的基本步骤。

创建 Renderscript 上下文 – `Renderscript` 类是一个托管的包装 Renderscript 上下文并将控件初始化，资源管理和清理。使用创建 Renderscript 对象 `RenderScript.Create` 工厂方法，它使用 Android 上下文（如活动）作为参数。以下代码行演示如何初始化 Renderscript 上下文：

```
Android.Renderscripts.RenderScript renderScript = RenderScript.Create(this);
```

创建分配-具体取决于内部函数的脚本，可能有必要创建一个或两个 `Allocation` s。的

`Android.Renderscripts.Allocation` 类具有多个工厂方法，用于帮助进行实例化的内部函数的分配。例如，下面的代码段演示如何创建位图的分配。

```
Android.Graphics.Bitmap originalBitmap;  
Android.Renderscripts.Allocation inputAllocation = Allocation.CreateFromBitmap(renderScript,  
    originalBitmap,  
    Allocation.MipmapControl.MipmapFull,  
    AllocationUsage.Script);
```

通常情况下，它将需要创建 `Allocation` 以保存脚本的输出数据。此以下代码片段演示如何使用

`Allocation.CreateTyped` 帮助器实例化第二个 `Allocation`，相同的类型与原始：

```
Android.Renderscripts.Allocation outputAllocation = Allocation.CreateTyped(renderScript,  
    inputAllocation.Type);
```

实例化脚本包装-每个内部函数的脚本包装器类应具有帮助器方法（通常称为 `Create`）实例化该脚本的包装对象。以下代码片段示范了如何实例化 `ScriptIntrinsicBlur` 模糊对象。`Element.U8_4` 帮助器方法将创建一个描述是 8 位无符号整数值，适用于保存的数据的 4 个字段的数据类型的元素 `Bitmap` 对象：

```
Android.Renderscripts.ScriptIntrinsicBlur blurScript = ScriptIntrinsicBlur.Create(renderScript,  
    Element.U8_4(renderScript));
```

分配 `Allocation(s)`、设置参数和运行脚本 - `Script` 类提供了 `ForEach` 方法以实际运行 `RenderScript`。此方法将遍历每个 `Element` 在 `Allocation` 保存输入的数据。在某些情况下，可能有必要提供 `Allocation` 用于保存输出。`ForEach` 将覆盖输出的内容分配。若要执行与前面步骤中的代码段，此示例演示如何分配输入的分配，设置参数，并最后运行脚本（将结果复制到输出分配）：

```
blurScript.SetInput(inputAllocation);  
blurScript.SetRadius(25); // Set a parameter  
blurScript.ForEach(outputAllocation);
```

你可能想要签出模糊 `RenderScript` 映像方案，它是如何在 `Xamarin.Android` 中使用的内部函数的脚本的完整示例。

总结

本指南介绍了 `RenderScript` 以及如何在 `Xamarin.Android` 应用程序中使用它。它简要介绍了 `RenderScript` 是什么及其工作原理的 `Android` 应用程序中。它介绍了一些关键组件，`RenderScript` 和之间的差异_用户脚本_并_内部脚本_。最后，本指南介绍使用 `Xamarin.Android` 应用程序中的内部函数的脚本中的步骤。

相关链接

- [Android.Renderscripts 命名空间](#)
- [模糊 `RenderScript` 的映像](#)
- [RenderScript](#)
- [教程：开始使用 `RenderScript`](#)

Xamarin.Essentials

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Xamarin.Essentials 可为开发人员提供其移动应用程序的跨平台 API。

Android、iOS 和 UWP 提供了唯一的操作系统和平台 API，开发人员可以利用 Xamarin 访问 C# 中的所有 API。Xamarin.Essentials 提供了适用于任何 Xamarin.Forms、Android、iOS 或 UWP 应用程序的单个跨平台 API，不管如何创建用户界面，都可以通过共享代码进行访问。

Xamarin.Essentials 入门

按照[入门指南](#)将 **Xamarin.Essentials** NuGet 包安装到现有或新的 Xamarin.Forms、Android、iOS 或 UWP 项目中。

功能指南

按照指南将这些 Xamarin.Essentials 功能集成到你的应用程序：

- [加速计](#) – 检索设备在三个维空间中的加速数据。
- [应用信息](#) – 查找有关应用程序的信息。
- [气压计](#) – 监视气压计是否发生压力变化。
- [电池](#) – 轻松检测电池电量、源和状态。
- [剪贴板](#) – 快速方便地设置或读取剪贴板上的文本。
- [指南针](#) – 监视指南针是否发生变化。
- [连接性](#) – 检查连接状态并检测变化。
- [数据传输](#) – 将文本和网站 URI 发送到其他应用。
- [设备显示信息](#) – 获取设备的屏幕指标和方向。
- [设备信息](#) – 轻松查找有关设备的信息。
- [电子邮件](#) – 轻松发送电子邮件。
- [文件系统帮助程序](#) – 轻松地将文件保存到应用数据。
- [手电筒](#) – 打开/关闭手电筒的简单方法。
- [地理编码](#) – 地理编码和反向地理编码地址和坐标。
- [地理位置](#) – 检索设备的 GPS 位置。
- [陀螺仪](#) – 跟踪围绕设备的三个主轴的旋转。
- [启动器](#) – 使应用程序能够打开系统的 URI。
- [磁力计](#) – 检测设备相对于地球磁场的方向。
- [主线程](#) – 在应用程序的主线程上运行代码。
- [地图](#) – 将地图应用程序打开到特定位置。
- [打开浏览器](#) – 快速方便地将浏览器打开到特定网站。
- [方向传感器](#) – 检索设备在三个维空间中的方向。
- [电话拨号程序](#) – 打开电话拨号程序。
- [能源](#) – 获取设备的节能模式状态。

- [首选项](#) – 快速方便地添加永久首选项。
- [屏幕锁定](#) – 使设备屏幕保持唤醒状态。
- [安全存储](#) – 安全地存储数据。
- [SMS](#) – 创建要发送的短信。
- [文本到语音转换](#) – 在设备上读出文本。
- [版本跟踪](#) – 跟踪应用程序版本和内部版本号。
- [振动](#) – 使振动设备。

疑难解答

如果遇到问题, 请寻求帮助。

API 文档

浏览 [API 文档](#) 了解每个 Xamarin.Essentials 功能。

Xamarin.Essentials 入门

2018/11/2 • [Edit Online](#)



Pre-release NuGet

Xamarin.Essentials 提供了适用于任何 iOS、Android 或 UWP 应用程序的单一跨平台 API，不管如何创建用户界面，都可以通过共享代码进行访问。

平台支持

Xamarin.Essentials 支持以下平台和操作系统：

| 平台 | 版本 |
|---------|--------------------|
| Android | 4.4 (API 19) 或更高版本 |
| iOS | 10.0 或更高版本 |
| UWP | 10.0.16299.0 或更高版本 |

安装

Xamarin.Essentials 可用作 NuGet 包，可以通过使用 Visual Studio 将其添加到任何现有或新的项目。

1. 使用 [Visual Studio tools for Xamarin](#) 下载并安装 [Visual Studio](#)。
2. 使用 Visual Studio C# (Android、iPhone 和 iPad 或跨平台) 下的空白应用模板打开现有项目，或创建新项目。重要说明：如果添加到 UWP 项目，请确保在项目属性中设置内部版本 16299 或更高版本。
3. 将 Xamarin.Essentials NuGet 包添加到每个项目：
 - [Visual Studio](#)
 - [Visual Studio for Mac](#)

在“解决方案资源管理器”面板中，右键单击解决方案名称，然后选择“管理 NuGet 包”。搜索 Xamarin.Essentials 并将包安装到所有项目，包括 Android、iOS、UWP 和 .NET Standard 库。

TIP

[Xamarin.Essentials NuGet](#) 处于预览状态时，选中“包括预发行版”框。

4. 在任何 C# 类中添加对 Xamarin.Essentials 的引用以引用 API。

```
using Xamarin.Essentials;
```

5. Xamarin.Essentials 需要特定于平台的设置：

- [Android](#)
- [iOS](#)
- [UWP](#)

Xamarin.Essentials 支持最低 Android 版本 4.4 (对应于 API 级别 19)，但用于编译的目标 Android 版本必须为 8.1 (对应于 API 级别 27)。(在 Visual Studio 中，已在“Android 清单”选项卡中的 Android 项目的“项目属性”对话框中设置这两个版本。在 Visual Studio for Mac 中，已在“Android 应用程序”选项卡中的 Android 项目的“项目选项”对话框中设置这两个版本。)

Xamarin.Essentials 安装所需的 Xamarin.Android.Support 库的版本 27.0.2.1。应用程序所需的任何其他 Xamarin.Android.Support 库还应使用 NuGet 包管理器更新到版本 27.0.2.1。应用程序所使用的所有 Xamarin.Android.Support 库应相同，并且至少应为版本 27.0.2.1。如果在解决方案中添加 Xamarin.Essentials NuGet 或更新 Nuget 时遇到问题，请参阅[疑难解答页面](#)。

在 Android 项目的 `MainLauncher` 或任何启动的 `Activity` 中，必须在 `OnCreate` 方法中初始化 Xamarin.Essentials：

```
protected override void OnCreate(Bundle savedInstanceState) {  
    //...  
    base.OnCreate(savedInstanceState);  
    Xamarin.Essentials.Platform.Init(this, savedInstanceState); // add this line to your code  
    //...
```

若要处理 Android 上的运行时权限，Xamarin.Essentials 必须接收任何 `OnRequestPermissionsResult`。将以下代码添加到所有 `Activity` 类：

```
public override void OnRequestPermissionsResult(int requestCode, string[] permissions,  
    [GeneratedEnum] Android.Content.PM.Permission[] grantResults)  
{  
    Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, permissions,  
        grantResults);  
  
    base.OnRequestPermissionsResult(requestCode, permissions, grantResults);  
}
```

6. 按照 [Xamarin.Essentials 指南](#) 为每个功能复制并粘贴代码片段。

Xamarin.Essentials - 适用于移动应用的跨平台 API(视频)

其他资源

建议刚开始接触 Xamarin 的开发人员访问 [Xamarin 开发入门](#)。

访问 [Xamarin.Essentials GitHub 存储库](#) 查看当前源代码，接下来，运行示例，并克隆存储库。欢迎为社区做贡献！

浏览 [API 文档](#) 了解每个 Xamarin.Essentials 功能。

Xamarin.Essentials: Accelerometer

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Accelerometer 类可用于监视设备的加速计传感器，指示设备在三维空间内的加速度。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Accelerometer

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

Accelerometer 功能通过调用 `Start` 和 `Stop` 方法来侦听加速的变化。然后通过 `ReadingChanged` 事件反馈任何变化。示例用法如下：

```

public class AccelerometerTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public AccelerometerTest()
    {
        // Register for reading changes, be sure to unsubscribe when finished
        Accelerometer.ReadingChanged += Accelerometer_ReadingChanged;
    }

    void Accelerometer_ReadingChanged(object sender, AccelerometerChangedEventArgs e)
    {
        var data = e.Reading;
        Console.WriteLine($"Reading: X: {data.Acceleration.X}, Y: {data.Acceleration.Y}, Z: {data.Acceleration.Z}");
        // Process Acceleration X, Y, and Z
    }

    public void ToggleAccelerometer()
    {
        try
        {
            if (Accelerometer.IsMonitoring)
                Accelerometer.Stop();
            else
                Accelerometer.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}

```

Accelerometer 读数以 G 为单位反馈回来。G 是等于地球重力加速度 (9.81 m/s^2) 的重力单位。

相对于手机屏幕的默认方向定义坐标系。设备的屏幕方向更改时，不会交换轴。

X 轴水平向右，Y 轴垂直向上，Z 轴从屏幕正面指向外。在此坐标系中，屏幕后方的坐标具有负 Z 值。

示例：

- 当设备平放在桌面上，并从左侧向右推时，X 轴加速值为正。
- 当设备平放在桌面上，加速值为 $+1.00 \text{ G}$ ($+9.81 \text{ m/s}^2$)，对应于设备的加速度 (0 m/s^2) 减去重力加速度 (-9.81 m/s^2)，以 G 为单位规范化。
- 当设备平放在桌面上，并以 $A \text{ m/s}^2$ 的加速度向上推时，加速值等于 $A+9.81$ ，对应于设备的加速度 ($+A \text{ m/s}^2$) 减去重力加速度 (-9.81 m/s^2)，并以 G 为单位规范化。

传感器速度

- 最快 – 尽快获取传感器数据(不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度(不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

API

- [Accelerometer 源代码](#)
- [Accelerometer API 文档](#)

Xamarin.Essentials: 应用信息

2018/11/10 • [Edit Online](#)



Pre-release NuGet

AppInfo 类提供应用程序的相关信息。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 AppInfo

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

获取应用程序信息：

通过 API 公开了以下信息：

```
// Application Name
var appName = AppInfo.Name;

// Package Name/Application Identifier (com.microsoft.testapp)
var packageName = AppInfo.PackageName;

// Application Version (1.0.0)
var version = AppInfo.VersionString;

// Application Build Number (1)
var build = AppInfo.BuildString;
```

显示应用程序设置

AppInfo 类还可以显示由操作系统为应用程序维护的设置页面：

```
// Display settings page
AppInfo.ShowSettingsUI();
```

此设置页面使用户能够更改应用程序权限，并执行其他特定于平台的任务。

API

- [AppInfo 源代码](#)
- [AppInfo API 文档](#)

Xamarin.Essentials: Barometer

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Barometer 类可用于监视设备的气压计传感器，该传感器可测量压力。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Barometer

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

Barometer 功能通过调用 `Start` 和 `Stop` 方法来侦听气压计压力读数的变化（以千帕为单位）。然后通过 `ReadingChanged` 事件反馈任何变化。示例用法如下：

```

public class BarometerTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public BarometerTest()
    {
        // Register for reading changes.
        Barometer.ReadingChanged += Barometer_ReadingChanged;
    }

    void Barometer_ReadingChanged(object sender, BarometerChangedEventArgs e)
    {
        var data = e.Reading;
        // Process Pressure
        Console.WriteLine($"Reading: Pressure: {data.Pressure} kilopascals");
    }

    public void ToggleBarometer()
    {
        try
        {
            if (Barometer.IsMonitoring)
                Barometer.Stop();
            else
                Barometer.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}

```

传感器速度

- 最快 – 尽快获取传感器数据 (不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度 (不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

无特定于平台的实现细节。

API

- [Barometer 源代码](#)

- [Barometer API 文档](#)

Xamarin.Essentials: Battery

2018/11/10 • [Edit Online](#)



Pre-release NuGet

Battery 类使你能够查看设备的电池信息并监视更改。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Battery 功能，需要以下特定于平台的设置。

- [Android](#)
- [iOS](#)
- [UWP](#)

需要具有 `Battery` 权限，并且必须在 Android 项目中进行配置。可以通过以下方法添加此权限：

打开 Properties 文件夹下的 AssemblyInfo.cs 文件并添加：

```
[assembly: UsesPermission(Android.Manifest.Permission.BatteryStats)]
```

或更新 Android 清单：

打开 Properties 文件夹下的 AndroidManifest.xml 文件，并在“manifest”节点内添加以下代码。

```
<uses-permission android:name="android.permission.BATTERY_STATS" />
```

或右键单击 Android 项目并打开项目的属性。在“Android 清单”下找到“所需权限”区域，然后选中“Battery”权限。这样会自动更新 AndroidManifest.xml 文件。

使用 Battery

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

查看当前的电池信息：

```

var level = Battery.ChargeLevel; // returns 0.0 to 1.0 or -1.0 if unable to determine.

var state = Battery.State;

switch (state)
{
    case BatteryState.Charging:
        // Currently charging
        break;
    case BatteryState.Full:
        // Battery is full
        break;
    case BatteryState.Discharging:
    case BatteryState.NotCharging:
        // Currently discharging battery or not being charged
        break;
    case BatteryState.NotPresent:
        // Battery doesn't exist in device (desktop computer)
    case BatteryState.Unknown:
        // Unable to detect battery state
        break;
}

var source = Battery.PowerSource;

switch (source)
{
    case BatteryPowerSource.Battery:
        // Being powered by the battery
        break;
    case BatteryPowerSource.AC:
        // Being powered by A/C unit
        break;
    case BatteryPowerSource.Usb:
        // Being powered by USB cable
        break;
    case BatteryPowerSource.Wireless:
        // Powered via wireless charging
        break;
    case BatteryPowerSource.Unknown:
        // Unable to detect power source
        break;
}

```

每当电池的任一属性发生更改时，将触发一个事件：

```

public class BatteryTest
{
    public BatteryTest()
    {
        // Register for battery changes, be sure to unsubscribe when needed
        Battery.BatteryChanged += Battery_BatteryChanged;
    }

    void Battery_BatteryChanged(object sender, BatteryChangedEventArgs e)
    {
        var level = e.ChargeLevel;
        var state = e.State;
        var source = e.PowerSource;
        Console.WriteLine($"Reading: Level: {level}, State: {state}, Source: {source}");
    }
}

```

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)

无平台差异。

API

- [Battery 源代码](#)
- [Battery API 文档](#)

Xamarin.Essentials: Clipboard

2018/11/10 • [Edit Online](#)



Pre-release NuGet

Clipboard 类使你能够在应用程序之间将文本复制并粘贴到系统剪贴板。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Clipboard

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

检查 Clipboard 是否有当前已准备好要粘贴的文本：

```
var hasText = Clipboard.HasText;
```

将文本设置到 Clipboard：

```
Clipboard.SetText("Hello World");
```

从 Clipboard 读取文本：

```
var text = await Clipboard.GetTextAsync();
```

API

- [Clipboard 源代码](#)
- [Clipboard API 文档](#)

Xamarin.Essentials: Compass

2018/11/10 • [Edit Online](#)



Pre-release NuGet

Compass 类使你能够监视设备的磁北航向。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Compass

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用 `Start` 和 `Stop` 方法来使用 Compass 功能以侦听罗盘的变化。然后通过 `ReadingChanged` 事件反馈任何变化。下面是一个示例：

```

public class CompassTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public CompassTest()
    {
        // Register for reading changes, be sure to unsubscribe when finished
        Compass.ReadingChanged += Compass_ReadingChanged;
    }

    void Compass_ReadingChanged(object sender, CompassChangedEventArgs e)
    {
        var data = e.Reading;
        Console.WriteLine($"Reading: {data.HeadingMagneticNorth} degrees");
        // Process Heading Magnetic North
    }

    public void ToggleCompass()
    {
        try
        {
            if (Compass.IsMonitoring)
                Compass.Stop();
            else
                Compass.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Some other exception has occurred
        }
    }
}

```

传感器速度

- 最快 – 尽快获取传感器数据(不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度(不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

平台实现细节

- [Android](#)

Android 不提供用于检索罗盘航向的 API。我们使用 Google 推荐的方法，利用加速计和磁力计来计算磁北航向。

在极少数情况下，可能会看到不一致的结果，因为需要校准传感器，这就涉及到以 8 字形来移动设备。进行此操作的最佳方式是打开 Google 地图，点击你所在的位置点，然后选择“校准罗盘”。

请注意，同时从应用运行多个传感器可能需要调整传感器速度。

低通筛选器

根据 Android 罗盘值的更新和计算方式, 可能需要平滑处理这些值。可以应用一个低通筛选器来平均角度的正弦和余弦值, 并且可以通过在 `Compass` 类上设置 `ApplyLowPassFilter` 属性来启用此筛选器:

```
Compass.ApplyLowPassFilter = true;
```

这仅适用于 Android 平台。可以在[此处](#)查看详细信息。

API

- [Compass 源代码](#)
- [Campass API 文档](#)

Xamarin.Essentials: Connectivity

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Connectivity 类可用于监视设备的网络状况是否发生变化，检查当前的网络访问权限，以及当前的连接方式。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Connectivity 功能，需要以下特定于平台的设置。

- [Android](#)
- [iOS](#)
- [UWP](#)

`AccessNetworkState` 权限是必需的，且必须在 Android 项目中配置。可通过以下方法添加此权限：

打开 Properties 文件夹下的 AssemblyInfo.cs 文件并添加：

```
[assembly: UsesPermission(Android.Manifest.Permission.AccessNetworkState)]
```

或更新 Android 清单：

打开 Properties 文件夹下的 AndroidManifest.xml 文件，并在 manifest 节点内添加。

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

或右键单击 Android 项目并打开项目的属性。在“Android 清单”下查找“所需权限:”区域，然后选中“访问网络状态”权限。这样会自动更新 AndroidManifest.xml 文件。

使用 Connectivity

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

检查当前网络访问：

```
var current = Connectivity.NetworkAccess;  
  
if (current == NetworkAccess.Internet)  
{  
    // Connection to internet is available  
}
```

[网络访问](#)分为以下几类：

- Internet – 本地和 Internet 访问。
- ConstrainedInternet – 受限 Internet 访问。指示强制网络门户连接情况，其中可以本地访问 Web 门户，但需要通过门户提供特定凭据才能访问 Internet。
- 本地 – 仅限本地网络访问。
- 无 – 无可用连接。
- 未知 – 无法确定 Internet 连接。

你可以检查设备当前正在使用哪种[连接配置文件](#)：

```
var profiles = Connectivity.Profiles;
if (profiles.Contains(ConnectionProfile.WiFi))
{
    // Active Wi-Fi connection.
}
```

只要连接配置文件或网络访问发生变化，就可以接收已触发的事件：

```
public class ConnectivityTest
{
    public ConnectivityTest()
    {
        // Register for connectivity changes, be sure to unsubscribe when finished
        Connectivity.ConnectivityChanged += Connectivity_ConnectivityChanged;
    }

    void Connectivity_ConnectivityChanged(object sender, ConnectivityChangedEventArgs e)
    {
        var access = e.NetworkAccess;
        var profiles = e.Profiles;
    }
}
```

限制

需要注意的是 `Internet` 可能由 `NetworkAccess` 报告，但对 Web 的完全访问权限不可用。由于每个平台上的连接方式不同，因此只能保证连接可用。例如，设备可能会连接到 Wi-Fi 网络，但路由器与 Internet 断开连接。在此示例中，可能会报告 Internet，但活动连接不可用。

API

- [Connectivity 源代码](#)
- [Connectivity API 文档](#)

Xamarin.Essentials: 数据传输

2018/11/10 • [Edit Online](#)



Pre-release NuGet

DataTransfer 类使应用程序能够将数据(例如文本和 Web 链接)共享到设备上的其他应用程序。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用数据传输

在你的类中添加对 Xamarin.Essentials 的引用:

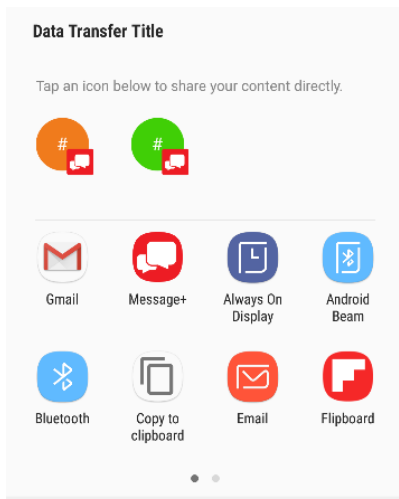
```
using Xamarin.Essentials;
```

通过调用具有数据请求有效负载的 `RequestAsync` 方法来使用数据传输功能, 此有效负载包括要共享到其他应用程序的信息。文本和 URI 可以混合使用, 每个平台都将根据内容进行筛选处理。

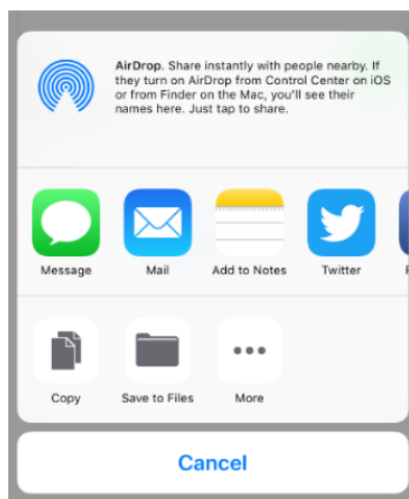
```
public class DataTransferTest
{
    public async Task ShareText(string text)
    {
        await DataTransfer.RequestAsync(new ShareTextRequest
        {
            Text = text,
            Title = "Share Text"
        });
    }

    public async Task ShareUri(string uri)
    {
        await DataTransfer.RequestAsync(new ShareTextRequest
        {
            Uri = uri,
            Title = "Share Web Link"
        });
    }
}
```

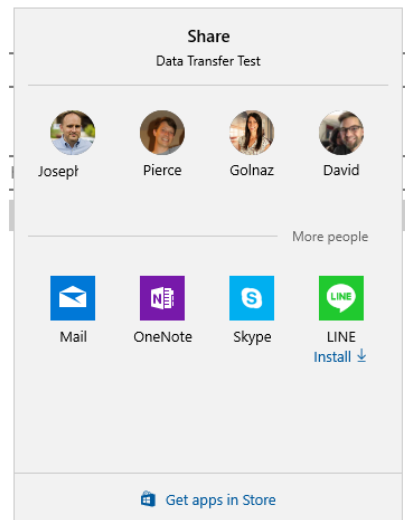
在进行请求时显示的共享到外部应用程序的用户界面:



Android



iOS



UWP

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)
- `Subject` 属性用于所需的消息主题。

API

- [数据传输源代码](#)
- [数据传输 API 文档](#)

Xamarin.Essentials: 设备显示信息

2018/11/10 • [Edit Online](#)



Pre-release NuGet

DeviceDisplay 类提供有关运行应用程序的设备的屏幕指标信息。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 DeviceDisplay

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

屏幕指标

除了基本的设备信息外，DeviceDisplay 类还包含有关设备的屏幕和方向信息。

```
// Get Metrics
var metrics = DeviceDisplay.ScreenMetrics;

// Orientation (Landscape, Portrait, Square, Unknown)
var orientation = metrics.Orientation;

// Rotation (0, 90, 180, 270)
var rotation = metrics.Rotation;

// Width (in pixels)
var width = metrics.Width;

// Height (in pixels)
var height = metrics.Height;

// Screen density
var density = metrics.Density;
```

DeviceDisplay 类还会公开可以订阅的一个事件，每当任何屏幕指标更改时就会触发此事件：

```
public class ScreenMetricsTest
{
    public ScreenMetricsTest()
    {
        // Subscribe to changes of screen metrics
        DeviceDisplay.ScreenMetricsChanged += OnScreenMetricsChanged;
    }

    void OnScreenMetricsChanged(ScreenMetricsChangedEventArgs e)
    {
        // Process changes
        var metrics = e.Metrics;
    }
}
```

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)

没有差异。

API

- [DeviceDisplay 源代码](#)
- [DeviceDisplay API 文档](#)

Xamarin.Essentials: 设备信息

2018/11/10 • [Edit Online](#)



Pre-release NuGet

DeviceInfo 类提供有关运行应用程序的设备的设备信息。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 DeviceInfo

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过 API 公开了以下信息：

```
// Device Model (SMG-950U, iPhone10,6)
var device = DeviceInfo.Model;

// Manufacturer (Samsung)
var manufacturer = DeviceInfo.Manufacturer;

// Device Name (Motz's iPhone)
var deviceName = DeviceInfo.Name;

// Operating System Version Number (7.0)
var version = DeviceInfo.VersionString;

// Platform (Android)
var platform = DeviceInfo.Platform;

// Idiom (Phone)
var idiom = DeviceInfo.Idiom;

// Device Type (Physical)
var deviceType = DeviceInfo.DeviceType;
```

平台

`DeviceInfo.Platform` 与映射到操作系统的一个常量字符串相关联。可以使用 `Platforms` 类检查以下值：

- **DeviceInfo.Platforms.iOS** - iOS
- **DeviceInfo.Platforms.Android** - Android
- **DeviceInfo.Platforms.UWP** - UWP
- **DeviceInfo.Platforms.Unsupported** - 不受支持

习惯用语

`DeviceInfo.Idiom` 与映射到运行应用程序的设备类型的一个常量字符串相关联。可以使用 `Idioms` 类检查以下值：

- **`DeviceInfo.Idioms.Phone`** - 手机
- **`DeviceInfo.Idioms.Tablet`** - 平板电脑
- **`DeviceInfo.Idioms.Desktop`** - 桌面
- **`DeviceInfo.Idioms.TV`** - 电视
- **`DeviceInfo.Idioms.Unsupported`** - 不受支持

设备类型

`DeviceInfo.DeviceType` 关联一个枚举以确定应用程序是在物理设备还是虚拟设备上运行。虚拟设备是指模拟器或仿真程序。

平台实现细节

- [iOS](#)

iOS 不会向开发人员公开一个 API 来获取特定 iOS 设备的名称。而是会返回一个硬件标识符，例如 iPhone10,6，这是指这些标识符的 iPhone X。A 映射不是由 Apple 提供的，但可以在 [The iPhone Wiki](#)（一个非官方来源）上找到。

API

- [DeviceInfo 源代码](#)
- [DeviceInfo API 文档](#)

Xamarin.Essentials: Email

2018/11/10 • [Edit Online](#)



Pre-release NuGet

Email 类使应用程序能够打开包含主题、正文和收件人 (TO、CC、BCC) 等指定信息的默认电子邮件应用程序。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Email

在你的类中添加对 Xamarin.Essentials 的引用:

```
using Xamarin.Essentials;
```

通过调用 `ComposeAsync` 方法和包含有关电子邮件信息的 `EmailMessage` 来使用 Email 功能:

```
public class EmailTest
{
    public async Task SendEmail(string subject, string body, List<string> recipients)
    {
        try
        {
            var message = new EmailMessage
            {
                Subject = subject,
                Body = body,
                To = recipients,
                //Cc = ccRecipients,
                //Bcc = bccRecipients
            };
            await Email.ComposeAsync(message);
        }
        catch (FeatureNotSupportedException fbsEx)
        {
            // Email is not supported on this device
        }
        catch (Exception ex)
        {
            // Some other exception occurred
        }
    }
}
```

API

- [Email 源代码](#)
- [Email API 文档](#)

Xamarin.Essentials: 文件系统帮助程序

2018/11/14 • [Edit Online](#)



Pre-release NuGet

FileSystem 类包含一系列帮助程序，用于查找应用程序的缓存和数据目录以及打开应用包内的文件。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用文件系统帮助程序

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

获取应用程序的目录以存储缓存数据。缓存数据可用于满足以下要求的任何数据：需要比临时数据持续更长时间，但不应是正确执行操作所需的数据。

```
var cacheDir = FileSystem.CacheDirectory;
```

为任何非用户数据文件的文件获取应用程序的顶级目录。这些文件是使用同步框架的操作系统进行备份的。查看下面的平台实现细节。

```
var mainDir = FileSystem.AppDataDirectory;
```

打开捆绑到应用程序包中的文件：

```
using (var stream = await FileSystem.OpenAppPackageFileAsync(templateFileName))
{
    using (var reader = new StreamReader(stream))
    {
        var fileContents = await reader.ReadToEndAsync();
    }
}
```

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

- **CacheDirectory** - 返回当前上下文的 [CacheDir](#)。
- **AppDataDirectory** - 返回当前上下文的 [FilesDir](#)，并且是使用 API 23 及更高版本的[自动备份](#)进行备份的。

将任何文件添加到 Android 项目中的 Assets 文件夹中，并将生成操作标记为 AndroidAsset 以将其与 `OpenAppPackageFileAsync` 一起使用。

API

- [文件系统帮助程序源代码](#)
- [文件系统 API 文档](#)

Xamarin.Essentials: Flashlight

2018/11/14 • [Edit Online](#)



Pre-release NuGet

Flashlight 类, 此类使你能够打开或关闭设备的照相机闪光灯, 将其转换为一个手电筒。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Flashlight 功能, 需要以下特定于平台的设置。

- [Android](#)
- [iOS](#)
- [UWP](#)

需要具有 Flashlight 和 Camera 权限, 并且必须在 Android 项目中进行配置。可以通过以下方法添加权限:

打开 Properties 文件夹下的 AssemblyInfo.cs 文件并添加:

```
[assembly: UsesPermission(Android.Manifest.Permission.Flashlight)]  
[assembly: UsesPermission(Android.Manifest.Permission.Camera)]
```

或更新 Android 清单:

打开 Properties 文件夹下的 AndroidManifest.xml 文件, 并在“manifest”节点内添加以下代码。

```
<uses-permission android:name="android.permission.FLASHLIGHT" />  
<uses-permission android:name="android.permission.CAMERA" />
```

或右键单击 Android 项目并打开项目的属性。在“Android 清单”下找到“所需权限”区域, 然后选中“FLASHLIGHT”和“CAMERA”权限。这样会自动更新 AndroidManifest.xml 文件。

通过添加这些权限, [Google Play](#) 将自动筛选出设备, 而无需任何特定硬件。可以通过将以下代码添加到 Android 项目中的 AssemblyInfo.cs 文件中来绕过此操作:

```
[assembly: UsesFeature("android.hardware.camera", Required = false)]  
[assembly: UsesFeature("android.hardware.camera.autofocus", Required = false)]
```

使用 Flashlight

在你的类中添加对 Xamarin.Essentials 的引用:

```
using Xamarin.Essentials;
```

可以通过 `TurnOnAsync` 和 `TurnOffAsync` 方法来打开或关闭手电筒:

```
try
{
    // Turn On
    await Flashlight.TurnOnAsync();

    // Turn Off
    await Flashlight.TurnOffAsync();
}
catch (FeatureNotSupportedException fnsEx)
{
    // Handle not supported on device exception
}
catch (PermissionException pEx)
{
    // Handle permission exception
}
catch (Exception ex)
{
    // Unable to turn on/off flashlight
}
```

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

Flashlight 类已根据设备的操作系统进行了优化。

API 级别 23 及更高版本

在更新的 API 级别上, [Torch 模式](#)将用于打开或关闭设备的闪光单元。

API 级别 22 及更高版本

创建一个相机表面纹理以打开或关闭相机单元的 `FlashMode`。

API

- [Flashlight 源代码](#)
- [Flashlight API 文档](#)

Xamarin.Essentials: Geocoding

2018/11/14 • [Edit Online](#)



Pre-release NuGet

Geocoding 类提供了 API, 既可以将地标地理编码为位置坐标, 又可以将坐标反向地理编码为地标。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Geocoding 功能, 需要以下特定于平台的设置。

- [Android](#)
- [iOS](#)
- [UWP](#)

无需其他设置。

使用 Geocoding

在你的类中添加对 Xamarin.Essentials 的引用:

```
using Xamarin.Essentials;
```

获取一个地址的[位置](#)坐标:

```
try
{
    var address = "Microsoft Building 25 Redmond WA USA";
    var locations = await Geocoding.GetLocationsAsync(address);

    var location = locations?.FirstOrDefault();
    if (location != null)
    {
        Console.WriteLine($"Latitude: {location.Latitude}, Longitude: {location.Longitude}, Altitude: {location.Altitude}");
    }
}
catch (FeatureNotSupportedException fnsEx)
{
    // Feature not supported on device
}
catch (Exception ex)
{
    // Handle exception that may have occurred in geocoding
}
```

高度并非总是可用的。如果不可用, `Altitude` 属性可能为 `null` 或值可能为零。如果高度可用, 此值为海拔高度 (以米为单位)。

为现有的一组坐标获取[地标](#):

```
try
{
    var lat = 47.673988;
    var lon = -122.121513;

    var placemarks = await Geocoding.GetPlacemarksAsync(lat, lon);

    var placemark = placemarks?.FirstOrDefault();
    if (placemark != null)
    {
        var geocodeAddress =
            $"AdminArea:      {placemark.AdminArea}\n" +
            $"CountryCode:    {placemark.CountryCode}\n" +
            $"CountryName:     {placemark.CountryName}\n" +
            $"FeatureName:      {placemark.FeatureName}\n" +
            $"Locality:         {placemark.Locality}\n" +
            $"PostalCode:       {placemark.PostalCode}\n" +
            $"SubAdminArea:     {placemark.SubAdminArea}\n" +
            $"SubLocality:      {placemark.SubLocality}\n" +
            $"SubThoroughfare: {placemark.SubThoroughfare}\n" +
            $"Thoroughfare:     {placemark.Thoroughfare}\n";

        Console.WriteLine(geocodeAddress);
    }
}
catch (FeatureNotSupportedException fnsEx)
{
    // Feature not supported on device
}
catch (Exception ex)
{
    // Handle exception that may have occurred in geocoding
}
```

两个位置之间的距离

[Location](#) 和 [LocationExtensions](#) 类定义了可用于计算两个位置之间的距离的方法。有关示例, 请参阅文章 [Xamarin.Essentials: Geolocation](#)。

API

- [Geocoding 源代码](#)
- [Geocoding API 文档](#)

Xamarin.Essentials: Geolocation

2018/11/14 • [Edit Online](#)



Pre-release NuGet

Geolocation 提供 API 以检索设备的当前地理位置坐标。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Geolocation 功能，需要以下特定于平台的设置：

- [Android](#)
- [iOS](#)
- [UWP](#)

需要具有 Coarse 和 Fine Location 权限，并且必须在 Android 项目中进行配置。此外，如果应用面向 Android 5.0(API 级别 21)或更高版本，则必须声明应用使用清单文件中的硬件功能。可以通过以下方法添加此声明：

打开 Properties 文件夹下的 AssemblyInfo.cs 文件并添加：

```
[assembly: UsesPermission(Android.Manifest.Permission.AccessCoarseLocation)]
[assembly: UsesPermission(Android.Manifest.Permission.AccessFineLocation)]
[assembly: UsesFeature("android.hardware.location", Required = false)]
[assembly: UsesFeature("android.hardware.location.gps", Required = false)]
[assembly: UsesFeature("android.hardware.location.network", Required = false)]
```

或更新 Android 清单：

打开 Properties 文件夹下的 AndroidManifest.xml 文件，并在“manifest”节点内添加以下代码：

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.location" android:required="false" />
<uses-feature android:name="android.hardware.location.gps" android:required="false" />
<uses-feature android:name="android.hardware.location.network" android:required="false" />
```

或右键单击 Android 项目并打开项目的属性。在“Android 清单”下找到“所需权限”区域，然后选中“ACCESS_COARSE_LOCATION”和“ACCESS_FINE_LOCATION”权限。这样会自动更新 AndroidManifest.xml 文件。

使用 Geolocation

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

Geolocation API 还将在必要时提醒用户所需的权限。

通过调用 `GetLastKnownLocationAsync` 方法获取设备上次的已知位置。这通常比执行完整的查询更快，但可能不太准确。

```
try
{
    var location = await Geolocation.GetLastKnownLocationAsync();

    if (location != null)
    {
        Console.WriteLine($"Latitude: {location.Latitude}, Longitude: {location.Longitude}, Altitude: {location.Altitude}");
    }
}
catch (FeatureNotSupportedException fnsEx)
{
    // Handle not supported on device exception
}
catch (PermissionException pEx)
{
    // Handle permission exception
}
catch (Exception ex)
{
    // Unable to get location
}
```

高度并非总是可用的。如果不可用，`Altitude` 属性可能为 `null` 或值可能为零。如果高度可用，此值为海拔高度（以米为单位）。

若要查询当前设备的位置坐标，可以使用 `GetLocationAsync`。最好传入一个完整的 `GeolocationRequest` 和 `CancellationToken`，因为获取设备的位置可能需要一些时间。

```
try
{
    var request = new GeolocationRequest(GeolocationAccuracy.Medium);
    var location = await Geolocation.GetLocationAsync(request);

    if (location != null)
    {
        Console.WriteLine($"Latitude: {location.Latitude}, Longitude: {location.Longitude}, Altitude: {location.Altitude}");
    }
}
catch (FeatureNotSupportedException fnsEx)
{
    // Handle not supported on device exception
}
catch (PermissionException pEx)
{
    // Handle permission exception
}
catch (Exception ex)
{
    // Unable to get location
}
```

地理位置的准确性

下表概述了每个平台的准确性：

最低

| 平台 | 距离(以米为单位) |
|---------|-------------|
| Android | 500 |
| iOS | 3000 |
| UWP | 1000 - 5000 |

低

| 平台 | 距离(以米为单位) |
|---------|------------|
| Android | 500 |
| iOS | 1000 |
| UWP | 300 - 3000 |

中等(默认值)

| 平台 | 距离(以米为单位) |
|---------|-----------|
| Android | 100 - 500 |
| iOS | 100 |
| UWP | 30-500 |

高

| 平台 | 距离(以米为单位) |
|---------|-----------|
| Android | 0 - 100 |
| iOS | 10 |
| UWP | <= 10 |

最佳

| 平台 | 距离(以米为单位) |
|---------|-----------|
| Android | 0 - 100 |
| iOS | ~0 |
| UWP | <= 10 |

两个位置之间的距离

`Location` 和 `LocationExtensions` 类定义了 `CalculateDistance` 方法, 可用于计算两个地理位置之间的距离。此计算得出的距离不考虑道路或其他路径, 仅仅是沿着地球表面的两点之间的最短距离, 也称为大圆距离或通俗地称为“直线距离”。

以下是一个示例：

```
Location boston = new Location(42.358056, -71.063611);
Location sanFrancisco = new Location(37.783333, -122.416667);
double miles = Location.CalculateDistance(boston, sanFrancisco, DistanceUnits.Miles);
```

`Location` 构造函数具有按该顺序排列的纬度和经度参数。正纬度值表示位于赤道以北，正经度值表示位于本初子午线以东。使用 `CalculateDistance` 的最后一个参数指定单位为英里还是公里。`Location` 类还定义了用于在两个单位之间进行转换的 `KilometersToMiles` 和 `MilesToKilometers` 方法。

API

- [Geolocation 源代码](#)
- [Geolocation API 文档](#)

Xamarin.Essentials: Gyroscope

2018/11/10 • • [Edit Online](#)



Pre-release NuGet

Gyroscope 类使你能够监控设备的陀螺仪传感器，此传感器测量围绕设备三个主轴的旋转。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Gyroscope

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用 `Start` 和 `Stop` 方法来使用 Gyroscope 功能以侦听陀螺仪的变化。然后通过 `ReadingChanged` 事件反馈任何变化。示例用法如下：

```

public class GyroscopeTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public GyroscopeTest()
    {
        // Register for reading changes.
        Gyroscope.ReadingChanged += Gyroscope_ReadingChanged;
    }

    void Gyroscope_ReadingChanged(object sender, GyroscopeChangedEventArgs e)
    {
        var data = e.Reading;
        // Process Angular Velocity X, Y, and Z
        Console.WriteLine($"Reading: X: {data.AngularVelocity.X}, Y: {data.AngularVelocity.Y}, Z: {data.AngularVelocity.Z}");
    }

    public void ToggleGyroscope()
    {
        try
        {
            if (Gyroscope.IsMonitoring)
                Gyroscope.Stop();
            else
                Gyroscope.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}

```

传感器速度

- 最快 – 尽快获取传感器数据(不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度(不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

API

- [Gyroscope 源代码](#)
- [Gyroscope API 文档](#)

Xamarin.Essentials: Launcher

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Launcher 类允许应用程序打开系统的 URI。通常在深入链接到另一个应用程序的自定义 URI 方案后使用此类。如果想要将浏览器打开到某个网站，则应引用[浏览器](#) API。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Launcher

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

若要使用 Launcher 功能，请调用 `OpenAsync` 方法并传入要打开的 `string` 或 `Uri`。（可选）`CanOpenAsync` 方法可用于检查是否可以由设备上的应用程序处理 URI 架构。

```
public class LauncherTest
{
    public async Task OpenRideShareAsync()
    {
        var supportsUri = await Launcher.CanOpenAsync("lyft://");
        if (supportsUri)
            await Launcher.OpenAsync("lyft://ridetype?id=lyft_line");
    }
}
```

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)

立即完成从 `CanOpenAsync` 返回的任务。

API

- [Launcher 源代码](#)
- [Launcher API 文档](#)

Xamarin.Essentials: Magnetometer

2018/11/10 • • [Edit Online](#)



Pre-release NuGet

Magnetometer 类使你能够监视设备的磁力计传感器，此传感器指示设备相对于地球磁场的方向。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Magnetometer

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用 `Start` 和 `Stop` 方法来使用 Magnetometer 功能以侦听磁力计的变化。然后通过 `ReadingChanged` 事件反馈任何变化。示例用法如下：

```

public class MagnetometerTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public MagnetometerTest()
    {
        // Register for reading changes.
        Magnetometer.ReadingChanged += Magnetometer_ReadingChanged;
    }

    void Magnetometer_ReadingChanged(object sender, MagnetometerChangedEventArgs e)
    {
        var data = e.Reading;
        // Process MagneticField X, Y, and Z
        Console.WriteLine($"Reading: X: {data.MagneticField.X}, Y: {data.MagneticField.Y}, Z:
{data.MagneticField.Z}");
    }

    public void ToggleMagnetometer()
    {
        try
        {
            if (Magnetometer.IsMonitoring)
                Magnetometer.Stop();
            else
                Magnetometer.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}

```

所有数据将以微特斯拉为单位返回。

传感器速度

- 最快 – 尽快获取传感器数据(不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度(不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

API

- [Magnetometer 源代码](#)
- [Magnetometer API 文档](#)

Xamarin.Essentials: MainThread

2018/11/1 • [Edit Online](#)



Pre-release NuGet

MainThread 类允许应用程序在主执行线程上运行代码，并确定当前是否在主线程上运行特定代码块。

背景

大多数操作系统(包括 iOS、Android 和通用 Windows 平台)对涉及用户界面的代码使用单线程模型。正确序列化用户界面事件(包括击键和触控输入)需要此模型。此线程通常称为“主线程”、“用户界面线程”或“UI 线程”。此模型的缺点是用于访问用户界面元素的所有代码必须在应用程序的主线程上运行。

应用程序有时需要使用在辅助执行线程上调用事件处理程序的事件。(Xamarin.Essentials 类 `Accelerometer`、`Compass`、`Gyroscope`、`Magnetometer` 和 `OrientationSensor` 以更快的速度使用时，可能会返回有关辅助线程的信息。)如果事件处理程序需要访问用户界面元素，则必须在主线程上运行该代码。MainThread 类允许应用程序在主线程上运行此代码。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

在主线程上运行代码

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

若要在主线程上运行代码，请调用静态 `MainThread.BeginInvokeOnMainThread` 方法。参数是 `Action` 对象，它只是没有参数且没有返回值的方法：

```
MainThread.BeginInvokeOnMainThread(() =>
{
    // Code to run on the main thread
});
```

也可以为必须在主线程上运行的代码定义单独的方法：

```
void MyMainThreadCode()
{
    // Code to run on the main thread
}
```

然后，可以通过在 `BeginInvokeOnMainThread` 方法中引用主线程，以便在主线程上运行此方法：

```
MainThread.BeginInvokeOnMainThread(MyMainThreadCode);
```

NOTE

Xamarin.Forms 具有名为 `Device.BeginInvokeOnMainThread(Action)` 的方法，该方法与 `MainThread.BeginInvokeOnMainThread(Action)` 执行相同操作。虽然可以在 Xamarin.Forms 应用中使用任何一种方法，但请考虑调用代码是否需要在 Xamarin.Forms 上有任何其他依赖项。如果不需要，则 `MainThread.BeginInvokeOnMainThread(Action)` 可能是更好的选择。

确定是否在主线程上运行代码

`MainThread` 类还允许应用程序确定是否在主线程上运行特定代码块。如果在主线程上运行调用 `IsMainThread` 属性的代码，则该属性会返回 `true`。程序可以使用此属性运行主线程或辅助线程的不同代码：

```
if (MainThread.IsMainThread)
{
    // Code to run if this is the main thread
}
else
{
    // Code to run if this is a secondary thread
}
```

在调用 `BeginInvokeOnMainThread` 之前，你可能想知道是否应检查代码是否在辅助线程上运行，如下所示：

```
if (MainThread.IsMainThread)
{
    MyMainThreadCode();
}
else
{
    MainThread.BeginInvokeOnMainThread(MyMainThreadCode);
}
```

如果已在主线程上运行代码块，你可能会猜想此检查可能会提高性能。

但是，不需要执行此检查。`BeginInvokeOnMainThread` 的平台实现本身会检查是否在主线程上调用代码。如果在并不需要时调用 `BeginInvokeOnMainThread`，则很少会有性能损失。

API

- [MainThread 源代码](#)
- [MainThread API 文档](#)

Xamarin.Essentials: Maps

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Maps 类允许应用程序将已安装的地图应用程序打开到特定位置或地标。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Maps

在你的类中添加对 Xamarin.Essentials 的引用:

```
using Xamarin.Essentials;
```

Maps 功能通过调用具有 `Location` 或 `Placemark` 的 `OpenAsync` 方法来使用可选的 `MapsLaunchOptions` 打开。

```
public class MapsTest
{
    public async Task NavigateToBuilding25()
    {
        var location = new Location(47.645160, -122.1306032);
        var options = new MapsLaunchOptions { Name = "Microsoft Building 25" };

        await Maps.OpenAsync(location, options);
    }
}
```

使用 `Placemark` 打开时, 需要以下信息:

- `CountryName`
- `AdminArea`
- `Thoroughfare`
- `Locality`

```
public class MapsTest
{
    public async Task NavigateToBuilding25()
    {
        var placemark = new Placemark
        {
            CountryName = "United States",
            AdminArea = "WA",
            Thoroughfare = "Microsoft Building 25",
            Locality = "Redmond"
        };
        var options = new MapsLaunchOptions { Name = "Microsoft Building 25" };

        await Maps.OpenAsync(placemark, options);
    }
}
```

扩展方法

如果已有对 `Location` 或 `Placemark` 的引用，则可以使用具有可选的 `MapsLaunchOptions` 的内置扩展方法

`OpenMapsAsync`：

```
public class MapsTest
{
    public async Task OpenPlacemarkOnMaps(Placemark placemark)
    {
        await placemark.OpenMapsAsync();
    }
}
```

方向模式

如果调用不带任何 `MapsLaunchOptions` 的 `OpenMapsAsync`，则地图将启动到指定位置。（可选）可以有从设备的当前位置开始计算的导航路线。这是通过设置 `MapsLaunchOptions` 上的 `MapDirectionsMode` 来完成的：

```
public class MapsTest
{
    public async Task NavigateToBuilding25()
    {
        var location = new Location(47.645160, -122.1306032);
        var options = new MapsLaunchOptions { MapDirectionsMode = MapDirectionsMode.Driving };

        await Maps.OpenAsync(location, options);
    }
}
```

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)
- `MapDirectionsMode` 支持骑行、驾车和步行。

平台实现细节

- [Android](#)

- [iOS](#)
- [UWP](#)

Android 使用 `geo:` URI 方案启动设备上的地图应用程序。这可能会提示用户从支持此 URI 方案的现有应用程序中进行选择。Xamarin.Essentials 使用支持此方案的 Google 地图进行测试。

API

- [Maps 源代码](#)
- [Maps API 文档](#)

Xamarin.Essentials: Browser

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Browser 类允许应用程序在优化的系统首选浏览器或外部浏览器中打开 Web 链接。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Browser

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

Browser 功能通过调用具有 `Uri` 和 `BrowserLaunchMode` 的 `OpenAsync` 方法工作。

```
public class BrowserTest
{
    public async Task OpenBrowser(Uri uri)
    {
        await Browser.OpenAsync(uri, BrowserLaunchMode.SystemPreferred);
    }
}
```

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

启动模式确定浏览器的启动方式：

系统首选

Chrome 自定义选项卡将尝试用于加载 URI 并保持导航识别。

外部

`Intent` 将用于请求通过系统常规浏览器打开的 URI。

API

- [Browser 源代码](#)

- [Browser API 文档](#)

Xamarin.Essentials: OrientationSensor

2018/11/1 • [Edit Online](#)



Pre-release NuGet

OrientationSensor 类可让你监视设备在三维空间中的方向。

NOTE

此类用于确定设备在三维空间中的方向。如果需要确定设备的视频显示器是处于纵向模式还是横向模式，请使用可从 `DeviceDisplay` 类获得的 `ScreenMetrics` 对象的 `Orientation` 属性。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 OrientationSensor

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用 `Start` 方法启用 `OrientationSensor` 以便监视对设备方向所做的更改，并通过调用 `Stop` 方法进行禁用。然后通过 `ReadingChanged` 事件反馈任何变化。示例用法如下：


```

public class OrientationSensorTest
{
    // Set speed delay for monitoring changes.
    SensorSpeed speed = SensorSpeed.UI;

    public OrientationSensorTest()
    {
        // Register for reading changes, be sure to unsubscribe when finished
        OrientationSensor.ReadingChanged += OrientationSensor_ReadingChanged;
    }

    void OrientationSensor_ReadingChanged(object sender, OrientationSensorChangedEventArgs e)
    {
        var data = e.Reading;
        Console.WriteLine($"Reading: X: {data.Orientation.X}, Y: {data.Orientation.Y}, Z: {data.Orientation.Z}, W: {data.Orientation.W}");
        // Process Orientation quaternion (X, Y, Z, and W)
    }

    public void ToggleOrientationSensor()
    {
        try
        {
            if (OrientationSensor.IsMonitoring)
                OrientationSensor.Stop();
            else
                OrientationSensor.Start(speed);
        }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Feature not supported on device
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}

```

`OrientationSensor` 读数以 `Quaternion` 的形式返回报告，描述了设备基于两个三维坐标系的方向：

设备(通常为手机或平板电脑)的三维坐标系具有以下轴：

- X 轴正方向指向显示器右侧(在纵向模式下)。
- Y 轴正方向指向显示器顶部(在纵向模式下)。
- Z 轴正方向指向屏幕外侧。

地球的三维坐标系具有以下轴：

- X 轴正方向与地球表面相切并指向东边。
- Y 轴正方向也与地球表面相切并指向北边。
- Z 轴正方向与地球表面垂直并指向上边。

`Quaternion` 描述了设备坐标系相对于地球坐标系的旋转。

`Quaternion` 值与绕轴旋转密切相关。如果旋转的轴是规范化矢量 (a_x, a_y, a_z)，并且旋转角度为 Θ ，则四元数的 (X, Y, Z, W) 分量是：

$(a_x \sin(\Theta/2), a_y \sin(\Theta/2), a_z \sin(\Theta/2), \cos(\Theta/2))$

这些是右手坐标系，因此右手的拇指指向旋转轴的正方向，手指弯曲表示正角的旋转方向。

示例：

- 如果设备平躺在桌子上，使其屏幕面朝上，设备顶部(在纵向模式下)指向北边，则会对齐这两个坐标系。
`Quaternion` 值表示标识四元数 (0, 0, 0, 1)。可以相对于此位置分析所有旋转。
- 如果设备平躺在桌子上，使其屏幕面朝上，设备顶部(在纵向模式下)指向西边，则 `Quaternion` 值为 (0, 0, 0.707, 0.707)。设备已绕地球的 Z 轴旋转 90 度。
- 如果设备直立，设备顶部(在纵向模式下)指向天空，设备背面朝向北边，则设备已绕 X 轴旋转 90 度。
`Quaternion` 值为 (0.707, 0, 0, 0.707)。
- 如果设备的左边缘在桌子上，顶部指向北边，则设备已绕 Y 轴旋转 -90 度(或绕 Y 轴负方向旋转 90 度)。
`Quaternion` 值为 (0, -0.707, 0, 0.707)。

传感器速度

- 最快 – 尽快获取传感器数据(不保证在 UI 线程上返回)。
- 游戏 – 适合游戏的速度(不保证在 UI 线程上返回)。
- 正常 – 适合屏幕方向更改的默认速率。
- UI – 适合常规用户界面的速率。

如果事件处理程序不能保证在 UI 线程上运行，并且如果事件处理程序需要访问用户界面元素，请使用

`MainThread.BeginInvokeOnMainThread` 方法在 UI 线程上运行该代码。

API

- [OrientationSensor 源代码](#)
- [OrientationSensor API 文档](#)

Xamarin.Essentials: 电话拨号程序

2018/11/10 • [Edit Online](#)



Pre-release NuGet

PhoneDialer 类使应用程序能够在拨号程序中打开一个电话号码。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用电话拨号程序

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用具有要用于打开拨号程序的一个电话号码的 `Open` 方法来使用电话拨号程序功能。当请求 `Open` 时，API 将自动尝试根据国家/地区代码设置号码的格式(如果已指定)。

```
public class PhoneDialerTest
{
    public async Task PlacePhoneCall(string number)
    {
        try
        {
            PhoneDialer.Open(number);
        }
        catch (ArgumentNullException anEx)
        {
            // Number was null or white space
        }
        catch (FeatureNotSupportedException ex)
        {
            // Phone Dialer is not supported on this device.
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}
```

API

- [电话拨号程序源代码](#)
- [电话拨号程序 API 文档](#)

Xamarin.Essentials: 节能模式状态

2018/11/1 • [Edit Online](#)



Pre-release NuGet

Power 类提供有关设备的节能模式状态的信息，指示设备是否在低功耗模式下运行。如果设备的节能模式状态已打开，则应用程序应避免后台处理。

背景

使用电池运行的设备可以置于低功耗节能模式。有时，设备会自动切换到此模式，例如，当电池电量降到 20% 以下时。操作系统通过减少往往会消耗电池的活动来响应节能模式。打开节能模式时，应用程序有助于避免后台处理或其他高功率活动。

对于 Android 设备，Power 类仅为 Android 版本 5.0 (Lollipop) 及更高版本返回有意义的信息。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Power 类

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

使用静态 `Power.EnergySaverStatus` 属性获取设备的当前节能状态：

```
// Get energy saver status
var status = Power.EnergySaverStatus;
```

此属性会返回 `EnergySaverStatus` 枚举的成员，可以是 `On`、`Off` 或 `Unknown`。如果该属性返回 `On`，则应用程序应避免后台处理或可能会消耗大量电力的其他活动。

应用程序还应安装事件处理程序。Power 类会公开节能模式状态发生更改时触发的事件：

```
public class EnergySaverTest
{
    public EnergySaverTest()
    {
        // Subscribe to changes of energy-saver status
        Power.EnergySaverStatusChanged += OnEnergySaverStatusChanged;
    }

    private void OnEnergySaverStatusChanged(EnergySaverStatusChangedEventArgs e)
    {
        // Process change
        var status = e.EnergySaverStatus;
    }
}
```

如果节能模式状态更改为 `On`，则应用程序应停止执行后台处理。如果状态更改为 `Unknown` 或 `Off`，则应用程序可以继续执行后台处理。

API

- [Power 源代码](#)
- [Power API 文档](#)

Xamarin.Essentials: Preferences

2018/11/14 • [Edit Online](#)



Pre-release NuGet

Preferences 类帮助将应用程序首选项存储在键/值存储中。

入门

若要开始使用此 API, 请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Preferences

在你的类中添加对 Xamarin.Essentials 的引用:

```
using Xamarin.Essentials;
```

将给定密钥的值保存在首选项中:

```
Preferences.Set("my_key", "my_value");
```

从首选项检索值或检索默认值(如果未设置):

```
var myValue = Preferences.Get("my_key", "default_value");
```

从首选项删除密钥:

```
Preferences.Remove("my_key");
```

删除所有首选项:

```
Preferences.Clear();
```

除了这些方法外, 每个方法都采用一个可选的 `sharedName`, 可用于创建首选的其他容器。查看下面的平台实现细节。

支持的数据类型

以下数据类型在 Preferences 中受到支持:

- **bool**
- **double**
- **int**
- **float**

- **long**
- **string**
- **DateTime**

实现详细信息

`DateTime` 的值是使用 `DateTime` 类定义的两种方法以 64 位二进制(长整型)格式存储的: `ToBinary` 方法用于对 `DateTime` 值进行编码, `FromBinary` 方法对值进行解码。当存储的 `DateTime` 不是协调世界时 (UTC) 值时, 请参阅这些方法的相关文档以了解如何进行对值解码的调整。

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

所有数据都存储到[共享首选项](#)中。如果未指定 `sharedName`, 则使用默认的共享首选项, 否则此名称将用于获取具有指定名称的私有共享首选项。

持久性

卸载应用程序将导致所有首选项被删除。对此有一个例外, 即面向使用[自动备份](#)的 Android 6.0 (API 级别 23) 或更高版本并在其上运行的应用。此功能默认启用并且会保留应用数据, 包括共享首选项 (即 Preferences API 使用的内容)。可以遵循 Google 的[文档](#)禁用此功能。

限制

当存储一个字符串时, 此 API 用于存储少量文本。如果尝试将其用于存储大量文本, 则可能会导致性能欠佳。

API

- [Preferences 源代码](#)
- [Preferences API 文档](#)

Xamarin.Essentials: 屏幕锁定

2018/11/10 • [Edit Online](#)



Pre-release NuGet

ScreenLock 类可以请求在应用程序运行时防止屏幕进入睡眠状态。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 ScreenLock

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

通过调用 `RequestActive` 和 `RequestRelease` 方法使用屏幕锁定功能来防止屏幕关闭。

```
public class ScreenLockTest
{
    public void ToggleScreenLock()
    {
        if (!ScreenLock.IsActive)
            ScreenLock.RequestActive();
        else
            ScreenLock.RequestRelease();
    }
}
```

API

- [屏幕锁定源代码](#)
- [屏幕锁定 API 文档](#)

Xamarin.Essentials: Secure Storage

2018/11/1 • [Edit Online](#)



Pre-release NuGet

SecureStorage 类有助于安全地存储简单的键/值对。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 SecureStorage 功能，需要以下特定于平台的设置：

- [Android](#)
- [iOS](#)
- [UWP](#)

TIP

[应用的自动备份](#)是 Android 6.0 (API 级别 23) 及更高版本的功能，可备份用户的应用数据（共享首选项、应用的内部存储中的文件和其他特定文件）。在新设备上重新安装或安装应用时，会还原数据。这可能会影响使用共享首选项（已备份但在还原时无法解密）的 `SecureStorage`。Xamarin.Essentials 可通过删除键（以便可以进行重置）自动处理这种情况，但你可以通过禁用自动备份来采取其他步骤。

启用或禁用备份

可以选择通过在 `AndroidManifest.xml` 文件中将 `android:allowBackup` 设置设为 `false`，为整个应用程序禁用自动备份。仅当计划按另一种方式还原数据时才建议使用此方法。

```
<manifest ... >
  ...
  <application android:allowBackup="false" ... >
    ...
  </application>
</manifest>
```

选择性备份

可以将自动备份配置为禁止备份特定内容。可以创建自定义规则集以禁止备份 `SecureStore` 项。

1. 在你的 `AndroidManifest.xml` 中设置 `android:fullBackupContent` 属性：

```
<application ...
  android:fullBackupContent="@xml/auto_backup_rules">
</application>
```

2. 在 `Resources/xml` 目录中创建名为 `auto_backup_rules.xml` 的新 XML 文件。然后设置以下内容，包括除 `SecureStorage` 以外的所有共享首选项：

```
<?xml version="1.0" encoding="utf-8"?>
<full-backup-content>
  <include domain="sharedpref" path="."/>
  <exclude domain="sharedpref" path="{applicationId}.xamarinessentials.xml"/>
</full-backup-content>
```

使用 Secure Storage

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

将给定密钥的值保存在安全存储中：

```
try
{
    await SecureStorage.SetAsync("oauth_token", "secret-oauth-token-value");
}
catch (Exception ex)
{
    // Possible that device doesn't support secure storage on device.
}
```

从安全存储中检索值：

```
try
{
    var oauthToken = await SecureStorage.GetAsync("oauth_token");
}
catch (Exception ex)
{
    // Possible that device doesn't support secure storage on device.
}
```

NOTE

如果没有与所请求的密钥关联的值，则 `GetAsync` 将返回 `null`。

若要删除特定密钥，请调用：

```
SecureStorage.Remove("oauth_token");
```

若要删除所有密钥，请调用：

```
SecureStorage.RemoveAll();
```

平台实现细节

- [Android](#)
- [iOS](#)
- [UWP](#)

[Android 密钥存储](#)用来存储用于在将值保存到文件名为 **[你的应用包 ID].xamarinessentials** 的[共享首选项](#)中之前加密值的加密密钥。共享首选项文件中所使用的密钥是传递到 `SecureStorage` API 的密钥的 MD5 哈希。

API 级别 23 及更高版本

在较新的 API 级别上, AES 密钥是从 Android 密钥存储中获得的, 并与 AES/GCM/NoPadding 密码结合使用以在将值存储在共享首选项文件中之前加密值。

API 级别 22 及更高版本

在较旧的 API 级别上, Android 密钥存储仅支持存储 RSA 密钥, 这些密钥与 RSA/ECB/PKCS1Padding 密码结合使用以加密 AES 密钥(运行时随机生成), 并且存储在密钥 `SecureStorageKey` 下的共享首选项文件中(如果尚未生成一个)。

`SecureStorage` 使用[首选项](#) API, 并遵循[首选项](#)文档中所述的相同数据持久性。如果设备从 API 级别 22 或更低级别升级到 API 级别 23 及更高版本, 则将继续使用此类型的加密, 除非卸载该应用或调用 `RemoveAll`。

限制

此 API 用于存储少量文本。如果尝试将其用于存储大量文本, 则可能会降低性能。

API

- [SecureStorage 源代码](#)
- [SecureStorage API 文档](#)

Xamarin.Essentials: SMS

2018/11/1 • [Edit Online](#)



Pre-release NuGet

SMS 类允许应用程序使用要发送到收件人的指定消息打开默认短信应用程序。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 SMS

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

SMS 功能通过调用 `ComposeAsync` 方法工作，该方法是包含消息收件人和消息正文(两者都是可选的)的 `SmsMessage` 。

```
public class SmsTest
{
    public async Task SendSms(string messageText, string recipient)
    {
        try
        {
            var message = new SmsMessage(messageText, recipient);
            await Sms.ComposeAsync(message);
        }
        catch (FeatureNotSupportedException ex)
        {
            // Sms is not supported on this device.
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}
```

此外，还可以向 `SmsMessage` 传入多个收件人：

```
public class SmsTest
{
    public async Task SendSms(string messageText, string[] recipients)
    {
        try
        {
            var message = new SmsMessage(messageText, recipients);
            await Sms.ComposeAsync(message);
        }
        catch (FeatureNotSupportedException ex)
        {
            // Sms is not supported on this device.
        }
        catch (Exception ex)
        {
            // Other error has occurred.
        }
    }
}
```

API

- [SMS 源代码](#)
- [SMS API 文档](#)

Xamarin.Essentials: Text-to-Speech

2018/11/1 • [Edit Online](#)



Pre-release NuGet

TextToSpeech 类允许应用程序使用内置的文本到语音转换引擎回讲设备中的文本并查询引擎可以支持的可用语言。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用 Text-to-Speech

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

Text-to-Speech 通过调用具有文本和可选参数的 `SpeakAsync` 方法工作，并在完成语音样本后返回。

```
public async Task SpeakNowDefaultSettings()
{
    await TextToSpeech.SpeakAsync("Hello World");

    // This method will block until utterance finishes.
}

public void SpeakNowDefaultSettings2()
{
    TextToSpeech.SpeakAsync("Hello World").ContinueWith((t) =>
    {
        // Logic that will run after utterance finishes.

    }, TaskScheduler.FromCurrentSynchronizationContext());
}
```

此方法采用可选的 `CancellationToken`，以便在语音样本启动时将其停止。

```

CancellationTokenSource cts;
public async Task SpeakNowDefaultSettings()
{
    cts = new CancellationTokenSource();
    await TextToSpeech.SpeakAsync("Hello World", cancelToken: cts.Token);

    // This method will block until utterance finishes.
}

public void CancelSpeech()
{
    if (cts?.IsCancellationRequested ?? false)
        return;

    cts.Cancel();
}

```

Text-to-Speech 会自动将同一线程中的语音请求加入队列。

```

bool isBusy = false;
public void SpeakMultiple()
{
    isBusy = true;
    Task.Run(async () =>
    {
        await TextToSpeech.SpeakAsync("Hello World 1");
        await TextToSpeech.SpeakAsync("Hello World 2");
        await TextToSpeech.SpeakAsync("Hello World 3");
        isBusy = false;
    });

    // or you can query multiple without a Task:
    Task.WhenAll(
        TextToSpeech.SpeakAsync("Hello World 1"),
        TextToSpeech.SpeakAsync("Hello World 2"),
        TextToSpeech.SpeakAsync("Hello World 3"))
        .ContinueWith((t) => { isBusy = false; }, TaskScheduler.FromCurrentSynchronizationContext());
}

```

语音设置

为了更好地控制如何使用可用于设置音量、音调和区域设置的 `SpeakSettings` 回讲音频。

```

public async Task SpeakNow()
{
    var settings = new SpeakSettings()
    {
        Volume = .75,
        Pitch = 1.0
    };

    await TextToSpeech.SpeakAsync("Hello World", settings);
}

```

下面是这些参数的支持值：

| 参数 | 最低 | 最大值 |
|----|----|-----|
| 音调 | 0 | 2.0 |
| 音量 | 0 | 1.0 |

语音区域设置

每个平台支持不同的区域设置，以便使用不同语言和重音回讲文本。平台具有用于指定区域设置的不同代码和方法，这就是 Xamarin.Essentials 为何提供跨平台 `Locale` 类以及使用 `GetLocalesAsync` 查询区域设置的方法的原因。

```
public async Task SpeakNow()
{
    var locales = await TextToSpeech.GetLocalesAsync();

    // Grab the first locale
    var locale = locales.FirstOrDefault();

    var settings = new SpeakSettings()
    {
        Volume = .75,
        Pitch = 1.0,
        Locale = locale
    };

    await TextToSpeech.SpeakAsync("Hello World", settings);
}
```

限制

- 如果跨多个线程调用语音样本队列，则不能予以保证。
- 背景音频播放未受到官方支持。

API

- [TextToSpeech 源代码](#)
- [TextToSpeech API 文档](#)

Xamarin.Essentials: 版本跟踪

2018/11/10 • [Edit Online](#)



Pre-release NuGet

VersionTracking 类使你能够检查应用程序版本和内部版本号以及查看其他信息，例如，此应用程序是第一次启动还是当前版本的第一次启动，以及获取之前的内部版本信息等。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

使用版本跟踪

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

首次使用 VersionTracking 类时，它将开始跟踪当前版本。每次加载时，必须仅在应用程序中提前调用 `Track` 以确保跟踪当前版本信息：

```
VersionTracking.Track();
```

调用初始 `Track` 后，可以读取版本信息：

```
// First time ever launched application
var firstLaunch = VersionTracking.IsFirstLaunchEver;

// First time launching current version
var firstLaunchCurrent = VersionTracking.IsFirstLaunchForCurrentVersion;

// First time launching current build
var firstLaunchBuild = VersionTracking.IsFirstLaunchForCurrentBuild;

// Current app version (2.0.0)
var currentVersion = VersionTracking.CurrentVersion;

// Current build (2)
var currentBuild = VersionTracking.CurrentBuild;

// Previous app version (1.0.0)
var previousVersion = VersionTracking.PreviousVersion;

// Previous app build (1)
var previousBuild = VersionTracking.PreviousBuild;

// First version of app installed (1.0.0)
var firstVersion = VersionTracking.FirstInstalledVersion;

// First build of app installed (1)
var firstBuild = VersionTracking.FirstInstalledBuild;

// List of versions installed (1.0.0, 2.0.0)
var versionHistory = VersionTracking.VersionHistory;

// List of builds installed (1, 2)
var buildHistory = VersionTracking.BuildHistory;
```

平台实现细节

所有版本信息均是使用 Xamarin.Essentials 中的 [Preferences](#) API 存储的, 是以 [你的-应用-包-ID].xamarinessentials.versiontracking 为文件名存储的, 并且遵循 [Preferences](#) 文档中概述的同一数据持久性。

API

- [版本跟踪源代码](#)
- [版本跟踪 API 文档](#)

Xamarin.Essentials: Vibration

2018/11/10 • [Edit Online](#)



Pre-release NuGet

Vibration 类使你能够在所需的时间内启动和停止振动功能。

入门

若要开始使用此 API，请阅读 Xamarin.Essentials 的[入门](#)指南以确保在项目中正确安装和设置库。

若要访问 Vibration 功能，需要以下特定于平台的设置。

- [Android](#)
- [iOS](#)
- [UWP](#)

需要具有 Vibrate 权限，并且必须在 Android 项目中进行配置。可以通过以下方法添加此权限：

打开 Properties 文件夹下的 AssemblyInfo.cs 文件并添加：

```
[assembly: UsesPermission(Android.Manifest.Permission.Vibrate)]
```

或更新 Android 清单：

打开 Properties 文件夹下的 AndroidManifest.xml 文件，并在“manifest”节点内添加以下代码。

```
<uses-permission android:name="android.permission.VIBRATE" />
```

或右键单击 Android 项目并打开项目的属性。在“Android 清单”下找到“所需权限”区域，然后选中“VIBRATE”权限。这样会自动更新 AndroidManifest.xml 文件。

使用 Vibration

在你的类中添加对 Xamarin.Essentials 的引用：

```
using Xamarin.Essentials;
```

可以通过设置所需的时间量或使用默认 500 毫秒来使用 Vibration 功能。

```
try
{
    // Use default vibration length
    Vibration.Vibrate();

    // Or use specified time
    var duration = TimeSpan.FromSeconds(1);
    Vibration.Vibrate(duration);
}
catch (FeatureNotSupportedException ex)
{
    // Feature not supported on device
}
catch (Exception ex)
{
    // Other error has occurred.
}
```

可以使用 `Cancel` 方法来请求取消使用设备振动：

```
try
{
    Vibration.Cancel();
}
catch (FeatureNotSupportedException ex)
{
    // Feature not supported on device
}
catch (Exception ex)
{
    // Other error has occurred.
}
```

平台差异

- [Android](#)
- [iOS](#)
- [UWP](#)

无平台差异。

API

- [Vibration 源代码](#)
- [Vibration API 文档](#)

Xamarin.Essentials: 故障排除

2018/7/11 • [Edit Online](#)



Pre-release NuGet

错误：为 Xamarin.Android.Support.Compat 检测到的版本冲突

更新 NuGet 包时，可能出现以下错误（或添加新的包）与使用 Xamarin.Essentials 的 Xamarin.Forms 项目：

```
NU1107: Version conflict detected for Xamarin.Android.Support.Compat. Reference the package directly from the project to resolve this issue.
  MyApp -> Xamarin.Essentials 0.8.0-preview -> Xamarin.Android.Support.CustomTabs 27.0.2.1 ->
  Xamarin.Android.Support.Compat (= 27.0.2.1)
  MyApp -> Xamarin.Forms 3.1.0.583944 -> Xamarin.Android.Support.v4 25.4.0.2 -> Xamarin.Android.Support.Compat
  (= 25.4.0.2).
```

问题在于不匹配的两个 NuGet 依赖项。这可以通过手动添加特定版本的依赖项解析（在这种情况下 **Xamarin.Android.Support.Compat**），可以支持这两个。

若要执行此操作，添加 NuGet 的手动冲突的源，并使用版本列表以选择特定版本。当前版本的 Xamarin.Android.Support.Compat 和 Xamarin.Android.Support.Core.Util NuGet 27.0.2.1 将解决此错误。

请参阅[这篇博客文章](#)有关详细信息和如何解决此问题的视频。

如果遇到任何问题或 bug 请报告其查找[Xamarin.Essentials GitHub 存储库](#)。

数据和云服务

2018/10/26 • [Edit Online](#)

数据和云服务

Xamarin.Android 应用程序通常需要访问数据（从本地数据库或从云）和许多这些应用使用 web 服务使用各种技术来实现。在本部分中的指南了解如何访问数据，并且使云服务的使用。

数据访问

本部分讨论 Xamarin.Android 使用 SQLite 作为数据库引擎中的数据访问。

Google 消息传送

提供 Google 的 Firebase Cloud Messaging 和 Google Cloud Messaging 的旧服务之间移动应用和服务器应用程序之间的消息传送。本部分提供有关在 Xamarin.Android 应用程序中提供的分步说明如何使用这些服务来实现远程通知（也称为推送通知）的每个服务的概述。

Microsoft Azure Active Directory

2018/6/6 • [Edit Online](#)

Azure Active Directory 允许开发人员安全资源，例如文件、链接和 Web Api、Office 365 和员工用于登录到他们的系统或检查其电子邮件的同一个组织帐户的详细信息。

入门

请按照[入门说明](#)配置 Azure 门户，并将 Active Directory 身份验证添加到你的 Xamarin 应用程序。

1. [注册到 Azure Active Directory](#)上[windowsazure.com](#)门户，然后
2. [配置服务](#)。
3. 挂钩以下项之一：

Office 365

在已经将 Active Directory 身份验证添加到应用程序，你可以使用凭据以与 Office 365 交互。

Graph API

了解如何访问[Graph API](#)使用 Xamarin (中还涉及我们[博客](#))。

Azure Active Directory

2018/6/6 • • [Edit Online](#)

注册应用程序，以便使用 Azure Active Directory

Azure Active Directory 允许对安全资源，如文件、链接和使用相同的组织帐户登录到他们的系统或检查其电子邮件的员工使用的 Web Api 的开发人员。

开发移动应用程序可以对与 Azure Active Directory 进行身份验证涉及三个步骤。前两个步骤通常是相同的而不考虑你打算使用哪些服务。第三步是为每个服务类型不同：

1. [注册到 Azure Active Directory](#)上[windowsazure.com](#)门户，然后
2. [配置服务](#)。
3. 开发移动应用程序使用的服务。

你可以访问的不同服务的示例包括：

- [Graph API](#)
- Web API
- Office365

结束语

使用上面的步骤可以进行 Azure Active Directory 针对您的移动应用身份验证。Active Directory 身份验证库 (ADAL)，可以更轻松使用更少行代码，同时保持的大部分代码相同，并因此使其可共享跨平台。

相关链接

- [Microsoft NativeClient 示例](#)

步骤 1。注册应用程序，以便使用 Azure Active Directory

2018/6/6 • [Edit Online](#)

1. 导航到windowsazure.com并使用你的 Microsoft 帐户或在 Azure 门户中的组织帐户登录。如果你没有 Azure 订阅，你可以获取从试用版azure.com
2. 在登录后，请转到**Active Directory** (1) 部分，然后选择想要注册的应用程序 (2) 的目录

| NAME | STATUS | SUBSCRIPTION | DATACENTER REGION |
|------------------|--------|--|-----------------------------|
| Mayur Tendulkar | Active | Shared by all Mayur Tendulkar subscriptions | Asia, Europe, United States |
| Tendulkar's | Active | Shared by all Tendulkar's subscriptions | Asia, Europe, United States |
| Tendulkar | Active | Shared by all Tendulkar subscriptions | Asia, Europe, United States |
| MayurT Directory | Active | Shared by all MayurT Directory subscriptions | Asia, Europe, United States |
| zevenseas | Active | Shared by all zevenseas subscriptions | Europe, United States |

3. 单击**添加**若要创建新的应用程序，然后选择**添加我的组织正在开发的应用程序**

What do you want to do?

- ➔ Add an application my organization is developing
- ➔ Add an application from the gallery

ADD

4. 在下一个屏幕上，为你的应用程序指定名称（例如。XAM-DEMO）。请确保选择**本机客户端应用程序**作为应用程序的类型。

ADD APPLICATION

Tell us about your application

NAME

XAM-DEMO

Type

☐ WEB APPLICATION AND/OR WEB API

☒ NATIVE CLIENT APPLICATION PREVIEW

→ 2

5. 在最终屏幕上，提供 ***重定向 URI** 作为身份验证完成时它将返回到此 URI，这是唯一的应用程序。

ADD APPLICATION

Application information

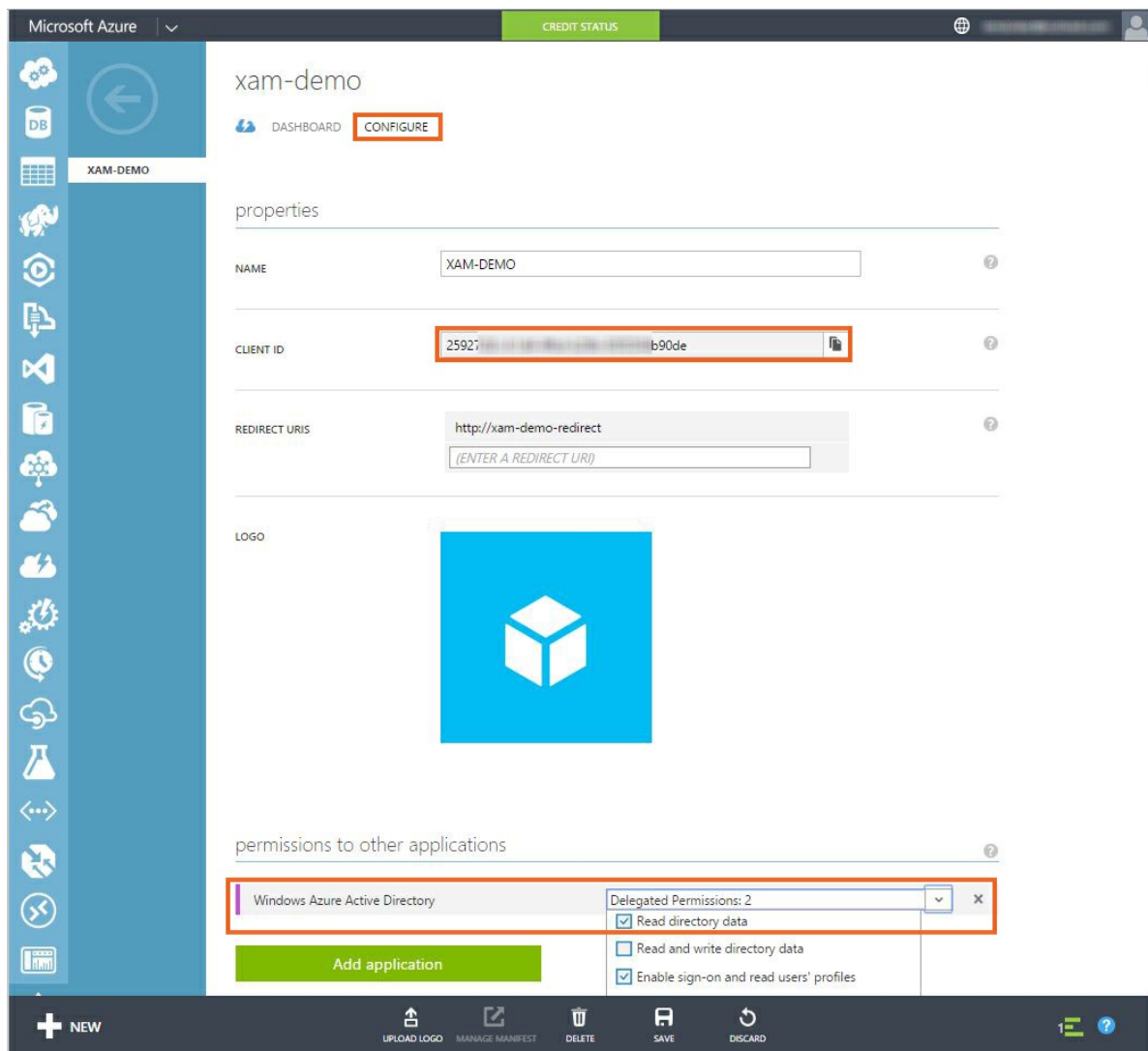
REDIRECT URI

http://xam-demo-redirect

1

← ✓

6. 创建应用程序后，导航到**配置**选项卡。记下**客户端 ID**我们将用在我们的应用程序更高版本。此外，在此屏幕上可以为 Active Directory 提供你的**移动应用程序**访问权限或添加另一个应用程序，如 Web API 或 Office365，身份验证完成后，**移动应用程序**可以使用。



相关链接

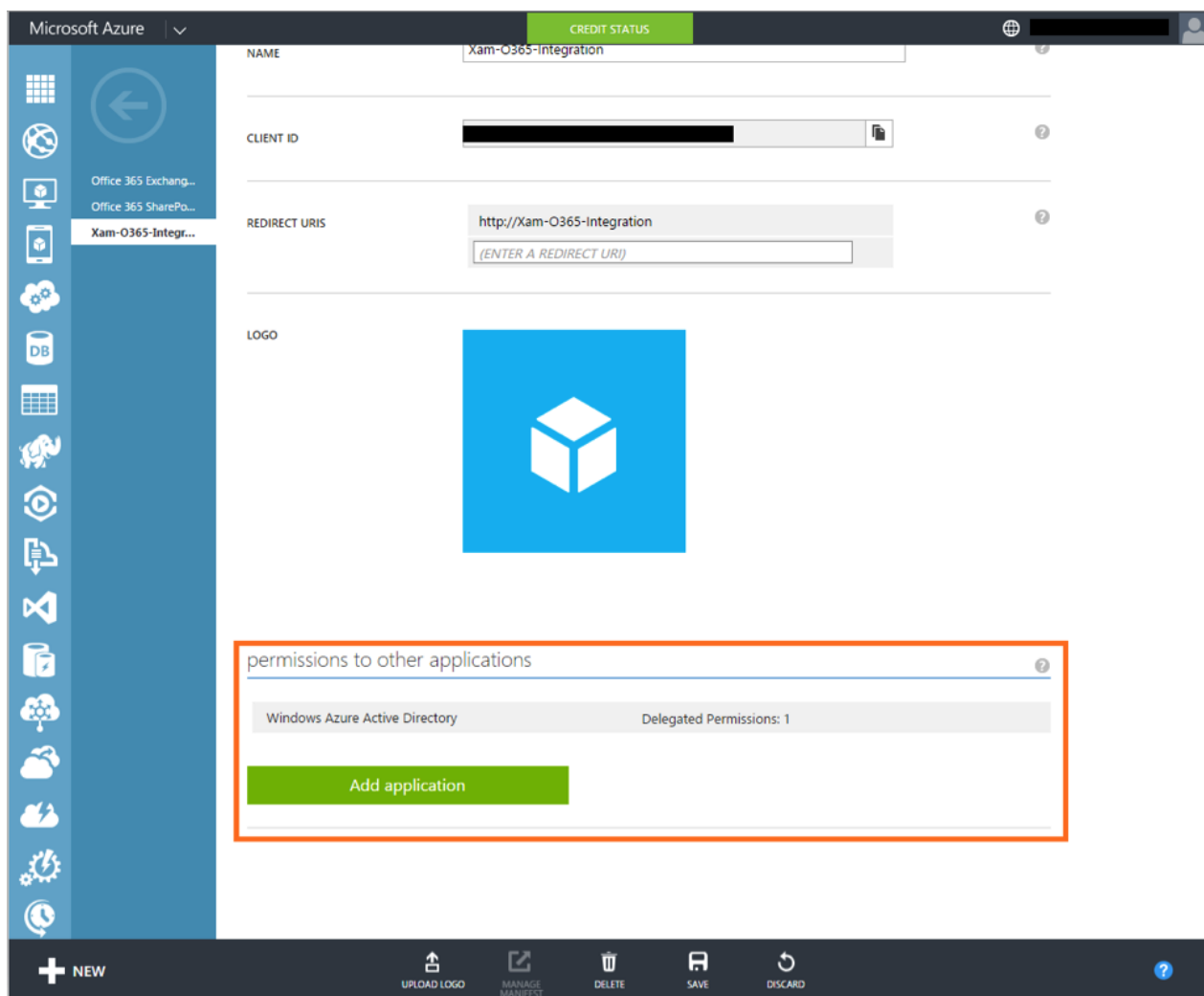
- [Microsoft NativeClient 示例](#)

步骤 2。为移动应用程序配置服务访问

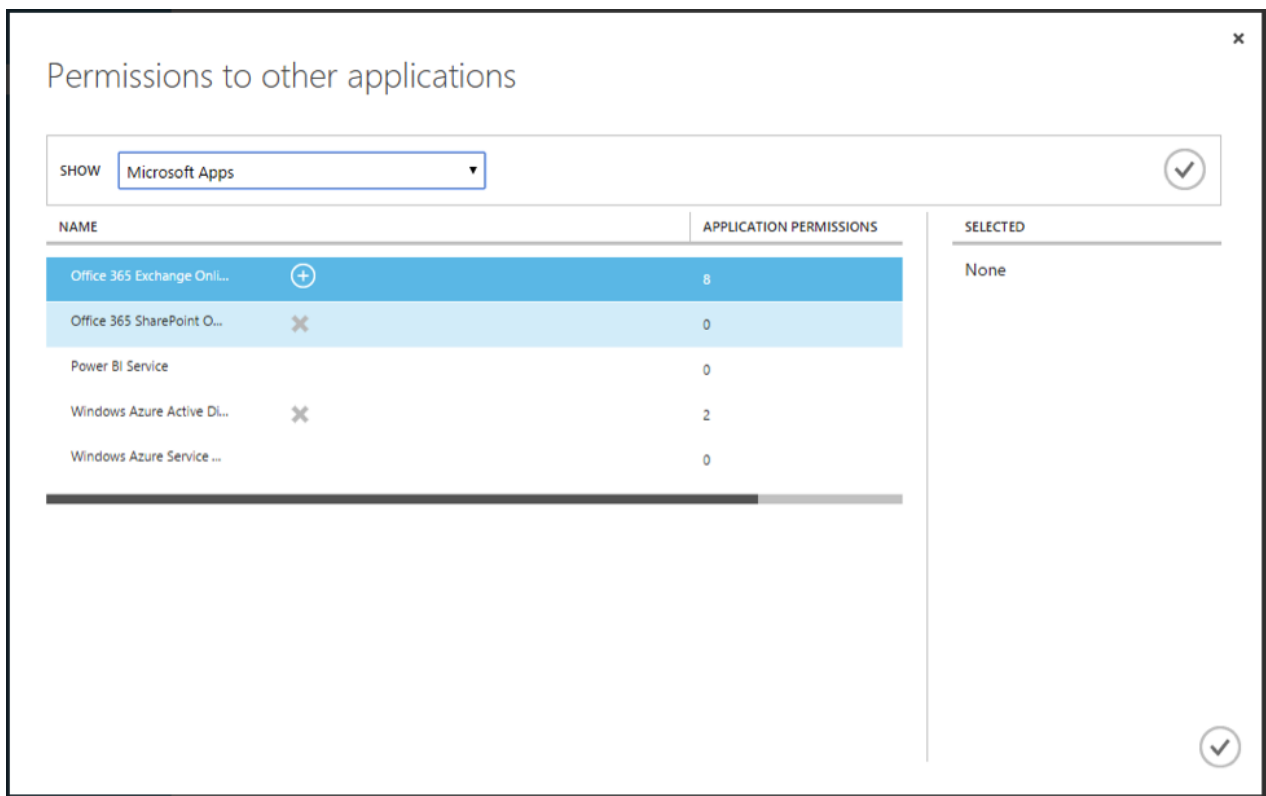
2018/6/6 • [Edit Online](#)

每当任何资源例如 web 应用程序、web 服务等需要由 Azure Active Directory 保护，它需要注册。所有安全应用程序或服务，可以查看下应用程序选项卡。在此处你可以选择从移动应用程序访问，并为其赋予访问所需要的应用程序。

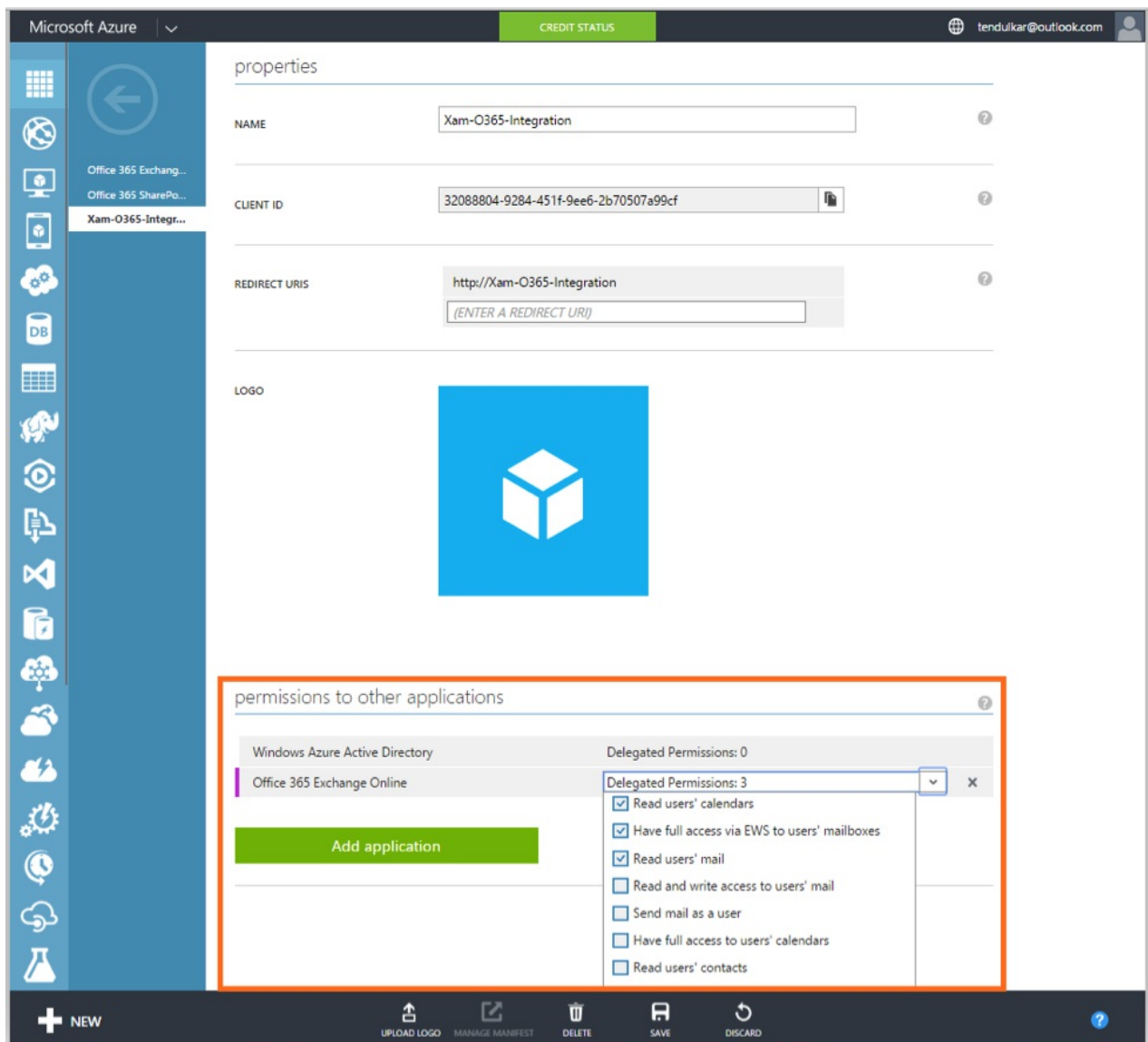
1. 上配置选项卡上，找到其他应用程序的权限部分：



2. 单击**添加应用程序**按钮。在弹出的下一个屏幕中，你会看到所有由 Azure Active Directory 保护的应用程序的列表。选择的应用程序需要从移动应用程序访问。



3. 选择应用程序之后, 再一次选择中的新增的应用程序其他应用程序的权限部分, 并提供适当的权限。



4. 最后, **保存配置**。这些服务应可在移动应用程序 !

相关链接

- [Microsoft NativeClient 示例](#)

访问 Graph API

2018/6/6 • [Edit Online](#)

按照这些步骤, 使用 Graph API 在 Xamarin 应用程序中:

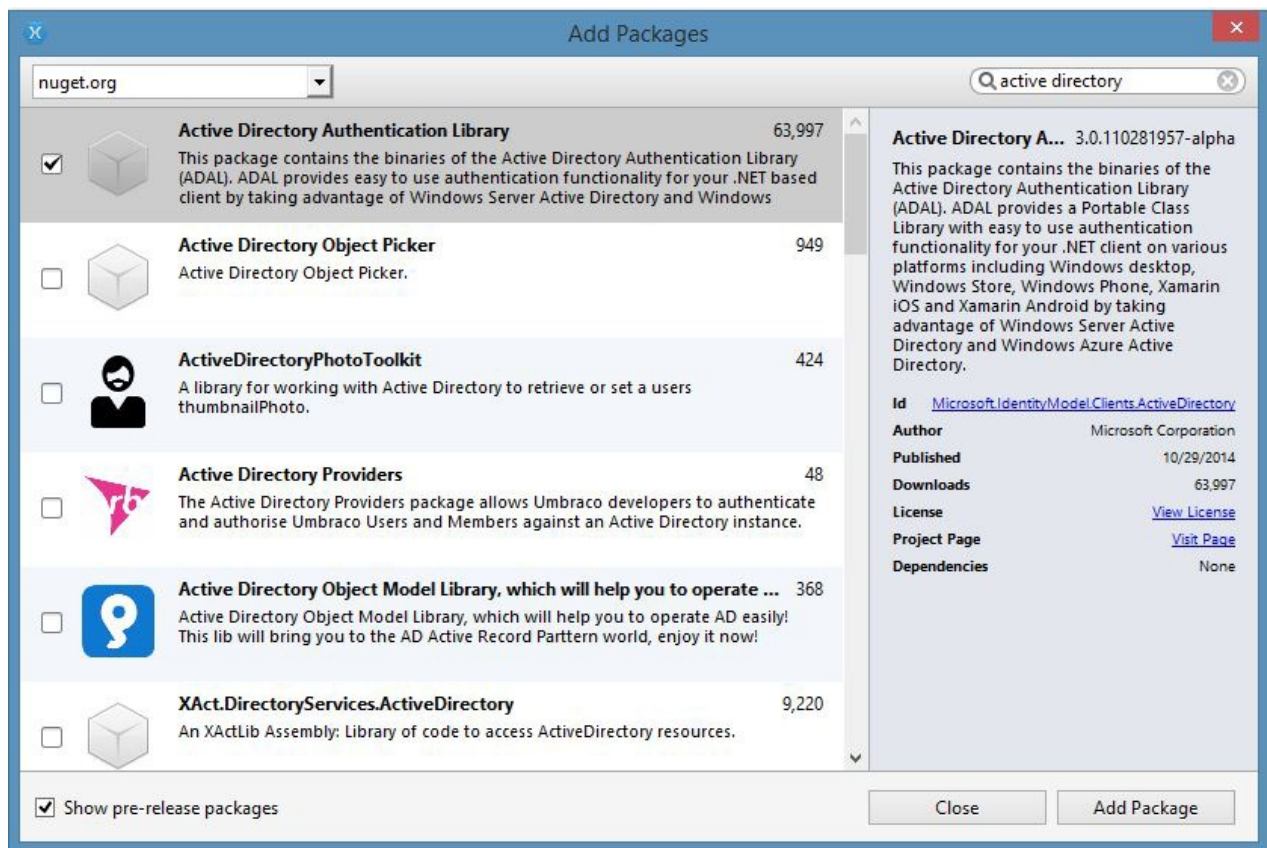
1. 注册到 [Azure Active Directory](#) 上 [windowsazure.com](#) 门户, 然后
2. 配置服务。

步骤 3。将 Active Directory 身份验证添加到应用程序

在你的应用程序添加到的引用 **Azure Active Directory 身份验证库 (Azure ADAL)** Visual Studio 或 Visual Studio 中的 NuGet 包管理器用于 mac。请确保选择显示预发行包以包括此包, 因为它仍处于预览阶段。

IMPORTANT

注意: Azure ADAL 3.0 目前预览和之前发布的最终版本可能是重大更改。



你的应用程序, 现在需要添加以下类级变量所需的身份验证流。

```
//Client ID
public static string clientId = "25927d3c-.....-63f2304b90de";
public static string commonAuthority = "https://login.windows.net/common"
//Redirect URI
public static Uri returnUrl = new Uri("http://xam-demo-redirect");
//Graph URI if you've given permission to Azure Active Directory
const string graphResourceUri = "https://graph.windows.net";
public static string graphApiVersion = "2013-11-08";
//AuthenticationResult will hold the result after authentication completes
AuthenticationResult authResult = null;
```

一个需要注意的事项还有 `commonAuthority`。身份验证终结点时 `common`，应用程序变得多租户，这意味着任何用户可以使用登录名使用其 Active Directory 凭据。身份验证后，该用户将在他们自己的 Active Directory 的上下文上工作-即，他们将看到其 Active Directory 与相关的详细信息。

编写方法以获取访问令牌

(对于 Android) 下面的代码将启动身份验证并将完成后将分配中的结果 `authResult`。iOS 和 Windows Phone 实现略有不同：第二个参数 (`Activity`) 在 iOS 上有不同，且不在 Windows Phone 上。

```
public static async Task<AuthenticationResult> GetAccessToken
    (string serviceResourceId, Activity activity)
{
    authContext = new AuthenticationContext(Authority);
    if (authContext.TokenCache.ReadItems().Count() > 0)
        authContext = new AuthenticationContext(authContext.TokenCache.ReadItems().First().Authority);
    var authResult = await authContext.AcquireTokenAsync(serviceResourceId, clientId, returnUrl, new
        AuthorizationParameters(activity));
    return authResult;
}
```

在上面的代码中，`AuthenticationContext` 负责使用 `commonAuthority` 进行身份验证。它具有 `AcquireTokenAsync` 方法，将参数作为需要访问，在这种情况下的资源 `graphResourceUri`，`clientId`，和 `returnUri`。应用程序将返回到 `returnUri` 身份验证完成时。此代码将保持不变，则对于所有平台，但是，最后一个参数，`AuthorizationParameters`，将在不同平台上有所不同，并且负责管理身份验证流。

对于 Android 或 iOS，我们传递 `this` 参数 `AuthorizationParameters(this)` 以共享上下文，而在 Windows 中它将不带任何参数作为传递新 `AuthorizationParameters()`。

适用于 Android 的句柄延续

身份验证完成后，流应返回到该应用。对于 Android 下面的代码处理此，应将其添加到 **MainActivity.cs**：

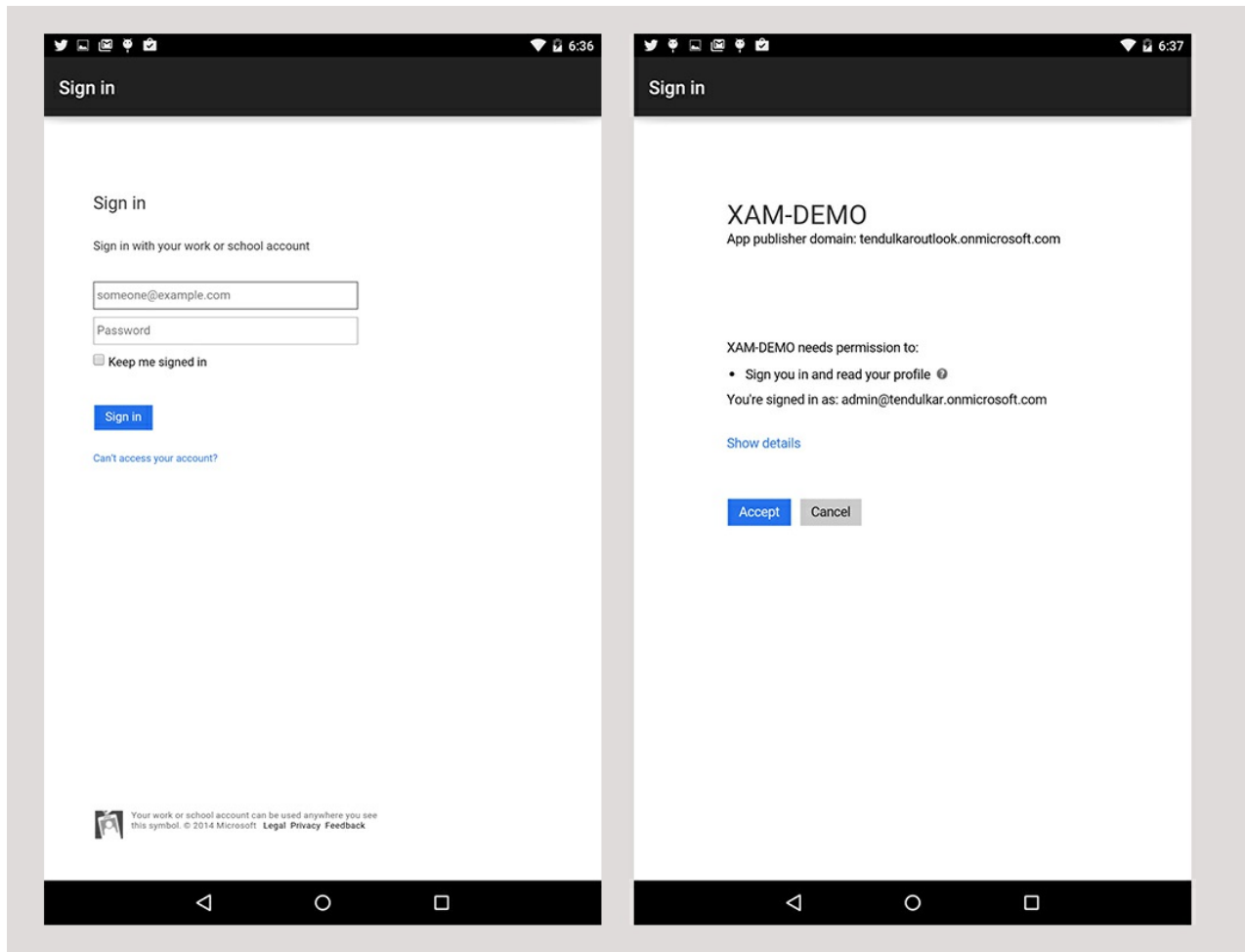
```
protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
{
    base.OnActivityResult(requestCode, resultCode, data);
    AuthenticationAgentContinuationHelper.SetAuthenticationAgentContinuationEventArgs(requestCode, resultCode, data);
}
```

Windows Phone 的句柄延续

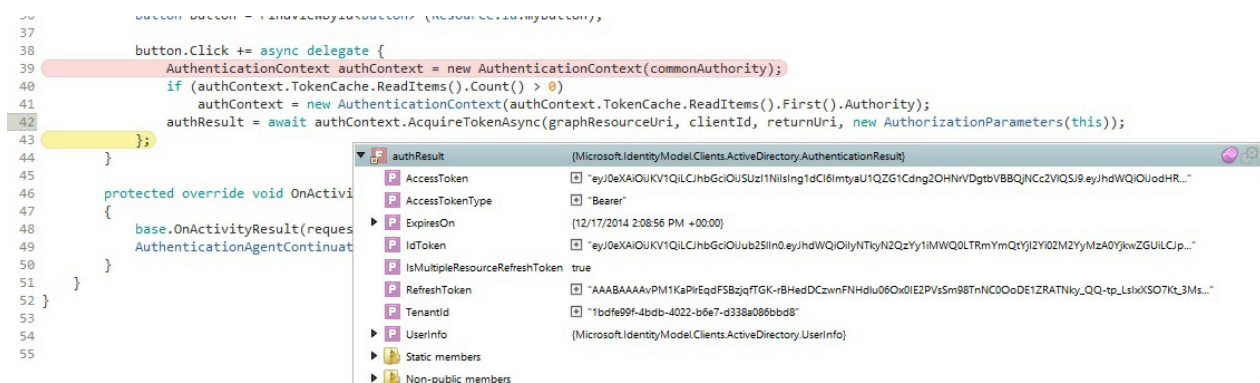
对于 Windows Phone 修改 `OnActivated` 中的方法 **App.xaml.cs** 文件与下面的代码：


```
protected override void OnActivated(IActivatedEventArgs args)
{
    #if WINDOWS_PHONE_APP
        if (args is IWebAuthenticationBrokerContinuationEventArgs)
        {
            WebAuthenticationBrokerContinuationHelper.SetWebAuthenticationBrokerContinuationEventArgs(args as
IWebAuthenticationBrokerContinuationEventArgs);
        }
    #endif
    base.OnActivated(args);
}
```

现在如果你运行应用程序，你将看到身份验证对话框。成功进行身份验证，它将要求您用于访问（在我们的示例 Graph API）的资源权限：



如果身份验证会成功，并且您已授权应用程序访问资源，你应获得 `AccessToken` 和 `RefreshToken` 中的组合 `authResult`。这些令牌是进一步 API 调用和与 Azure Active Directory 在幕后的授权需要。



例如, 下面的代码, 可从 Active Directory 中获取用户列表。你可以替换你的 Web API 由 Azure AD 保护 Web API URL。

```
var client = new HttpClient();
var request = new HttpRequestMessage(HttpMethod.Get,
    "https://graph.windows.net/tendulkar.onmicrosoft.com/users?api-version=2013-04-05");
request.Headers.Authorization =
    new AuthenticationHeaderValue("Bearer", authResult.AccessToken);
var response = await client.SendAsync(request);
var content = await response.Content.ReadAsStringAsync();
```

Microsoft Azure 移动应用

2018/6/6 • [Edit Online](#)

Azure 门户文档的下载示例和代码项目。

这些链接是有关 Xamarin 文档的信息可在上找到[Azure Mobile Apps](#)网站。将 Azure 功能添加到 Xamarin 应用程序, 通过下载[Azure 移动客户端](#)。

使用 Xamarin Azure 组件

常规文档使用 [Xamarin 客户端库 \(组件\)](#) 来完成与 Azure Mobile Apps 的各种任务。此页包含大量示例代码段, 而无需详细的说明和示例可在每个下面列出的演练文章。

入门

本文提供了分步说明, 以便启动并运行您的第一个 Xamarin Azure 应用程序。它介绍如何在门户中创建新的 Azure 移动应用程序然后下载并运行预配置的应用程序。

- [iOS](#)
- [Android](#)
- [Xamarin.Forms](#)

用户入门

对配置和代码使用 Azure 移动服务的登录屏幕中提供完整的说明。支持的身份验证提供程序包括 Microsoft、Google、Facebook 和 Twitter。

- [iOS](#)
- [Android](#)

在脚本中的用户授权

Javascript 后端某些示例代码

- [Todo.js](#)

推送通知入门

完成说明将推送通知配置上的 Apple 和 Google 的网站, 然后从 Azure 移动服务向设备发送推送通知。

- [iOS](#)
- [Android](#)

使用通知中心入门

若要在 Apple 和 Google 网站上配置推送通知配置 Azure 通知中心, 然后将生成的完整说明将通知推送到设备。

- [iOS](#)
- [Android](#)

相关链接

- [GettingStarted \(示例\)](#)
- [GetStartedWithData \(示例\)](#)
- [GetStartedWithUsers \(示例\)](#)
- [GetStartedWithPush \(示例\)](#)
- [NotificationHubs \(示例\)](#)
- [Azure 的移动客户端](#)
- [Azure Mobile Apps 的学习路径](#)

Xamarin.Android 数据访问

2018/11/1 • [Edit Online](#)

大多数应用程序具有一些要求将数据保存在本地设备上。除非数据量非常小，这通常需要一个数据库和管理数据库的访问权限的应用程序中的数据层。Android 包含 SQLite 数据库引擎内置和 Xamarin 的平台通过简化的存储和检索数据的访问权限。本文档演示如何跨平台的方式访问 SQLite 数据库。

数据访问概述

大多数应用程序具有一些要求将数据保存在本地设备上。除非数据量非常小，这通常需要一个数据库和管理数据库的访问权限的应用程序中的数据层。Android 这两个具有"内置"SQLite 数据库引擎和 Xamarin 的平台，其中随附了 SQLite 数据提供程序简化对数据的访问。

Xamarin.Android 支持数据库访问 Api，如：

- ADO.NET 框架。
- SQLite NET 第三方库。

在本部分中的代码大部分是完全跨平台，并将在 iOS 或 Android 上运行而无需修改。有两个示例应用介绍：

- [DataAccess_Basic](#) -简单的数据操作写入结果显示为文本显示的控件;
- [DataAccess_Advanced](#) -集成到一个小的工作应用程序，列出并编辑简单的数据结构的数据操作。

这两个示例解决方案包含 iOS 和 Android 示例应用程序项目。

有关 Xamarin.Forms 应用程序，请阅读[使用数据库](#)其中解释了如何使用 SQLite PCL 库中使用 Xamarin.Forms。

在本部分中的主题讨论 Xamarin.Android 使用 SQLite 作为数据库引擎中的数据访问。可以通过使用 ADO.NET 语法"直接"访问数据库或可包含 SQLite.NET ORM，并在 C# 中执行数据操作。

查看两个示例：一个包含非常简单的数据访问代码的输出到文本字段，并包括一个简单应用程序创建、读取、更新和删除的功能。线程处理以及如何播种使用预填充的 SQLite 数据库应用程序还讨论了。

有关跨平台数据访问的其他示例，请参阅我们[Tasky Pro](#)案例研究。

相关链接

- [DataAccess Basic \(示例\)](#)
- [DataAccess 高级 \(示例\)](#)
- [Android 数据方案](#)
- [Xamarin.Forms 数据访问](#)

何时使用数据库

而递增的移动设备的存储和处理功能，手机和平板电脑仍将滞后于其桌面和便携式计算机的对应项中。出于此原因值得花费一些时间来规划您的应用程序的数据存储体系结构而不只假定数据库正确的答案的时间。有多种不同的选项，满足不同的要求，如：

- **首选项**– Android 提供了内置的机制，用于存储数据的简单键 / 值对。如果要存储的少量数据（如个性化设置信息）或简单的用户设置然后使用此平台的本机功能来存储此类信息。
- **文本文件**– 用户输入或的缓存下载的内容（例如。可以直接在文件系统上存储 HTML）。使用适当的文件命名约定来帮助组织文件并查找数据。
- **序列化数据文件**– 对象可保留 XML 或 json 格式在文件系统。.NET framework 包括进行序列化和反序列化对象轻松的库。使用合适的名称来组织数据文件。
- **数据库**– SQLite 数据库引擎是适用于 Android 的平台，非常适合存储结构化查询、排序或以其他方式操作所需的数据。数据库存储适合于具有许多属性的数据的列表。
- **图像文件**– 尽管可以在移动设备上的数据库中存储二进制数据，但建议将其存储在文件系统中直接。如有必要您可以在数据库中以与图像关联与其他数据存储的文件名。处理大型映像或映像的很多，很好的做法规划删除不再需要避免使用用户的所有存储空间的文件缓存策略。

如果数据库是您的应用程序的适当的存储机制，此文档的其余部分将讨论如何在 Xamarin 平台上使用 SQLite。

使用数据库的优点

有很多优点到移动应用程序中使用的 SQL 数据库：

- SQL 数据库允许高效地存储结构化数据。
- 可以使用复杂查询提取特定的数据。
- 可以对查询结果进行排序。
- 查询结果可进行聚合。
- 使用现有数据库技术的开发人员可以利用他们的知识来设计数据库和数据访问代码。
- 从连接的应用程序的服务器组件的数据模型可能会重新使用（整体或部分）中的移动应用程序。

SQLite 数据库引擎

SQLite 是一种开放源代码数据库引擎，为其移动平台采用的 Google。SQLite 数据库引擎是内置于这两种操作系统，因此开发人员能够利用它的任何额外工作。SQLite 是非常适用于跨平台移动开发的因为：

- 数据库引擎是小型、快速且可轻松地移植。
- 数据库存储在单个文件，这是在移动设备上轻松管理。
- 跨平台可以方便使用的文件格式是：是否 32 位或 64 位和大或小-端系统。
- 它实现大部分标准 sql92 功能。

SQLite 旨在作为小巧快捷，因为它的使用有一些需要注意的问题：

- 不支持某些外部联接语法。
- 仅限于表重命名和 ADD COLUMN 受支持。无法执行对您的架构进行其他修改。
- 视图是只读的。

您可以了解有关 SQLite 的网站- [SQLite.org](https://sqlite.org) -但是, 您需要通过 Xamarin 使用 SQLite 的所有信息包含在本文档和相关示例。自 Android 2, 在 Android 中支持 SQLite 数据库引擎。尽管这一章中未涉及, SQLite 也是可在 Windows Phone 和 Windows 应用程序使用。

Windows 和 Windows Phone

尽管本文档未涵盖这些平台, 还可以在 Windows 平台上使用 SQLite。更多信息, 请参阅[Tasky](#)并[Tasky Pro](#)案例研究, 并查看[Tim Heuer](#) 的博客。

相关链接

- [DataAccess Basic \(示例\)](#)
- [DataAccess 高级 \(示例\)](#)
- [Android 数据方案](#)
- [Xamarin.Forms 数据访问](#)

配置

2018/11/1 • [Edit Online](#)

若要将需要确定您的数据库文件的正确的文件位置在 Xamarin.Android 应用程序中使用 SQLite。

数据库文件路径

无论使用哪种数据访问方法，可以使用 SQLite 存储数据之前，必须创建数据库文件。根据您面向的哪种平台的文件位置将有所不同。适用于 Android 可用于环境类构造有效的路径，如下面的代码段中所示：

```
string dbPath = Path.Combine (
    Environment.GetFolderPath (Environment.SpecialFolder.Personal),
    "database.db3");
// dbPath contains a valid file path for the database file to be stored
```

没有要确定用于存储数据库文件的位置时需要考虑的其他事项。例如，在 Android 上你可以选择是否使用内部或外部存储。

如果你想要在跨平台应用程序中的每个平台上使用的其他位置，可用的编译器指令所示来生成每个平台的不同路径：

```
var sqliteFilename = "MyDatabase.db3";
#if __ANDROID__
// Just use whatever directory SpecialFolder.Personal returns
string libraryPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal); ;
#else
// we need to put in /Library/ on iOS5.1 to meet Apple's iCloud terms
// (they don't want non-user-generated data in Documents)
string documentsPath = Environment.GetFolderPath (Environment.SpecialFolder.Personal); // Documents folder
string libraryPath = Path.Combine (documentsPath, "..", "Library"); // Library folder instead
#endif
var path = Path.Combine (libraryPath, sqliteFilename);
```

有关在 Android 中使用文件系统上的提示，请参阅[浏览文件](#)方案。请参阅[生成跨平台应用程序](#)使用编译器指令将写入每个平台特定代码的详细信息文档。

线程

不应跨多个线程使用相同的 SQLite 数据库连接。请注意打开、使用，然后关闭任何同一线程创建的连接。

若要确保你的代码不尝试从多个线程同时访问 SQLite 数据库，手动采用的锁时想要访问数据库时，此类：

```
object locker = new object(); // class level private field
// rest of class code
lock (locker){
    // Do your query or insert here
}
```

应使用同一个锁包装所有数据库访问权限（读取、写入、更新等）。必须格外小心，以避免死锁情况下，通过确保锁子句中的工作保持简单并不会不调用其他方法的可能还需要一个锁！

相关链接

- [DataAccess Basic \(示例\)](#)
- [DataAccess 高级 \(示例\)](#)
- [Android 数据方案](#)
- [Xamarin.Forms 数据访问](#)

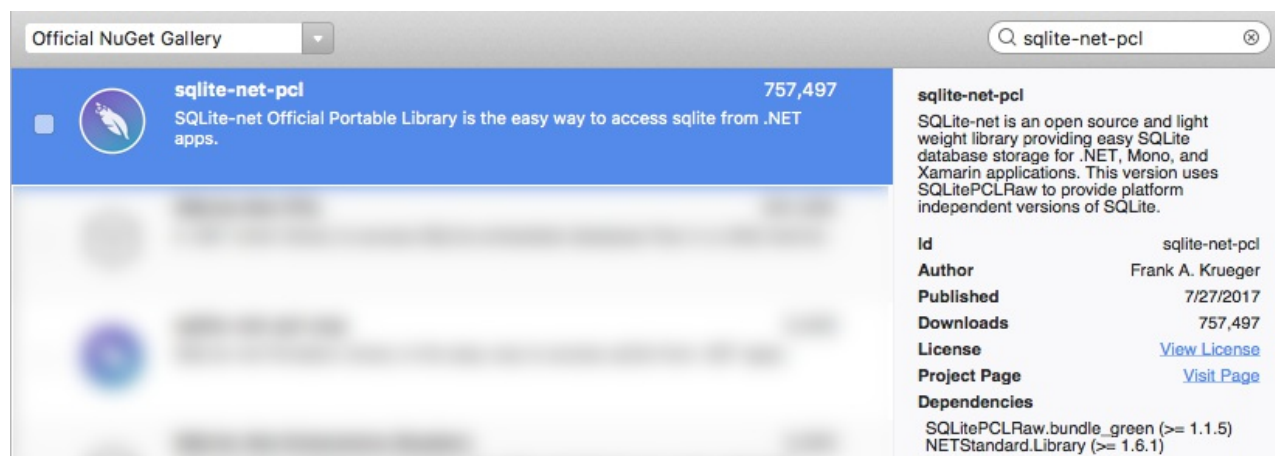
与 Android 一起使用 SQLite.NET

2018/10/26 • [Edit Online](#)

Xamarin 建议 SQLite.NET 库是非常基本的 ORM，你可以轻松地存储和检索 Android 设备上的本地 SQLite 数据库中的对象。ORM 代表对象关系映射-API，你可以保存和检索数据库中的“对象”，而无需编写 SQL 语句。

若要在 Xamarin 应用中包含 SQLite.NET 库，请向项目添加以下 NuGet 包：

- 包名称：sqlite net pcl
- 作者：Frank A.Krueger
- Id: sqlite net pcl
- Url: nuget.org/packages/sqlite-net-pcl



Official NuGet Gallery

Search:

sqlite-net-pcl 757,497

SQLite-net Official Portable Library is the easy way to access sqlite from .NET apps.

sqlite-net-pcl

SQLite-net is an open source and light weight library providing easy SQLite database storage for .NET, Mono, and Xamarin applications. This version uses SQLitePCLRaw to provide platform independent versions of SQLite.

Id sqlite-net-pcl

Author Frank A. Krueger

Published 7/27/2017

Downloads 757,497

License [View License](#)

Project Page [Visit Page](#)

Dependencies

SQLitePCLRaw.bundle_green (>= 1.1.5)

NETStandard.Library (>= 1.6.1)

TIP

有多种不同的 SQLite 包-请确保选择正确的订阅（它可能不是在搜索结果）。

可用的 SQLite.NET 库后，请执行以下三个步骤来使用它来访问数据库：

1. **添加 using 语句**-添加以下语句到C#数据访问是必需的文件：

```
using SQLite;
```

2. **创建空数据库**-可以通过将文件路径传递 SQLiteConnection 类构造函数创建的数据库引用。不需要检查文件是否已存在-它会自动创建必要，否则将打开现有的数据库文件。`dbPath` 变量，必须根据本文档前面所述的规则来确定：

```
var db = new SQLiteConnection (dbPath);
```

3. **将数据保存**-创建 SQLiteConnection 对象后，通过调用其方法，例如 CreateTable 和插入此类执行数据库命令：

```
db.CreateTable<Stock> ();  
db.Insert (newStock); // after creating the newStock object
```

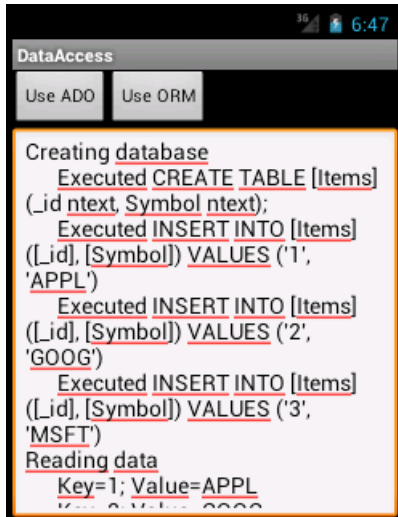
4. **检索数据**-若要检索一个对象（或一系列对象）使用以下语法：

```
var stock = db.Get<Stock>(5); // primary key id of 5
var stockList = db.Table<Stock>();
```

基本数据访问示例

DataAccess_Basic 本文档的示例代码如下所示，在 Android 上运行时。该代码演示了如何执行简单的 SQLite.NET 操作，然后为应用程序的主窗口中的文本显示中的结果。

Android



下面的代码示例显示了使用 SQLite.NET 库封装基础数据库访问权限的整个数据库交互。显示：

1. 创建数据库文件
2. 通过创建对象，然后保存它们插入一些数据
3. 查询数据

你将需要包含这些命名空间：

```
using SQLite; // from the github SQLite.cs class
```

最后一个需要向项目添加了 SQLite。请注意，通过将特性添加到类定义的 SQLite 数据库表 (`Stock` 类) 而不是 CREATE TABLE 命令。

```

[Table("Items")]
public class Stock {
    [PrimaryKey, AutoIncrement, Column("_id")]
    public int Id { get; set; }
    [MaxLength(8)]
    public string Symbol { get; set; }
}
public static void DoSomeDataAccess () {
    Console.WriteLine ("Creating database, if it doesn't already exist");
    string dbPath = Path.Combine (
        Environment.GetFolderPath (Environment.SpecialFolder.Personal),
        "ormdemo.db3");
    var db = new SQLiteConnection (dbPath);
    db.CreateTable<Stock> ();
    if (db.Table<Stock> ().Count() == 0) {
        // only insert the data if it doesn't already exist
        var newStock = new Stock ();
        newStock.Symbol = "AAPL";
        db.Insert (newStock);
        newStock = new Stock ();
        newStock.Symbol = "GOOG";
        db.Insert (newStock);
        newStock = new Stock ();
        newStock.Symbol = "MSFT";
        db.Insert (newStock);
    }
    Console.WriteLine("Reading data");
    var table = db.Table<Stock> ();
    foreach (var s in table) {
        Console.WriteLine (s.Id + " " + s.Symbol);
    }
}

```

使用 `[Table]` 属性而无需指定表名称参数将导致基础数据库表具有与类相同的名称（在本例中为"Stock"）。如果编写直接针对数据库的 SQL 查询，而不是使用 ORM 数据访问方法，是重要的实际表名称。同样 `[Column("_id")]` 属性是可选的并且如果不存在列将添加到具有与类中的属性相同的名称的表。

SQLite 属性

您可以将应用于您的类来控制如何在基础数据库中存储的常见属性包括：

- **[PrimaryKey]** –此特性可以应用于一个整数属性，以强制其为基础表的主键。不支持复合主键。
- **[自动增量]** –此属性会导致一个整数属性的值是自动递增每个新对象插入到数据库
- **[Column(name)]** –提供可选 `name` 参数将替代基础数据库列的名称（这是与属性相同）的默认值。
- **[Table(name)]** –将标记为无法存储在基础的 SQLite 表中的类。指定可选的 `name` 参数将覆盖基础数据库表的名称（这是与类名称相同）的默认值。
- **[MaxLength(value)]** –将 text 属性的长度限制时尝试执行数据库插入。使用代码应验证这一点在为此属性时才检查的数据库插入或更新操作尝试插入对象之前。
- **[忽略]** –导致 SQLite.NET 以忽略此属性。这是对于具有无法在数据库中存储的类型的属性或属性不能自动解决的模型集合是 SQLite 特别有用。
- **[Unique]** –可以确保基础数据库列中的值的唯一性。

大多数这些属性是可选的 SQLite 将使用默认值表和列的名称。应始终指定整数主键，以便可以对数据有效地执行所选内容和删除查询。

更复杂的查询

上的以下方法 `SQLiteConnection` 可用于执行其他数据操作：

- **插入**—向数据库添加一个新的对象。
- **获取 <T>** —尝试检索使用为主键的对象。
- **表 <T>** —返回表中的所有对象。
- **删除**—删除对象使用它的主键。
- **查询 <T>** —执行 SQL 查询返回的（作为对象）的行数。
- **执行**—使用此方法（而不 `Query`）时不希望从 SQL（如 INSERT、UPDATE 和 DELETE 的说明）返回的行。

通过主键获取对象

SQLite.Net 提供 `Get` 方法来检索单个对象基于其主键。

```
var existingItem = db.Get<Stock>(3);
```

选择使用 Linq 对象

返回集合的方法支持 `IEnumerable<T>` 以便您可以使用 Linq 来查询或对表的内容进行排序。下面的代码演示如何使用 Linq 来筛选出以字母"A"开头的所有条目：

```
var apple = from s in db.Table<Stock>()
            where s.Symbol.StartsWith ("A")
            select s;
Console.WriteLine ("-> " + apple.FirstOrDefault ().Symbol);
```

选择使用 SQL 对象

即使 SQLite.Net 可提供基于对象的访问数据，有时您可能需要执行更复杂的查询不是 Linq 允许（或可能需要更快的性能）。可以通过查询方法中，使用 SQL 命令，如下所示：

```
var stocksStartingWithA = db.Query<Stock>("SELECT * FROM Items WHERE Symbol = ?", "A");
foreach (var s in stocksStartingWithA) {
    Console.WriteLine ("a " + s.Symbol);
}
```

NOTE

直接编写 SQL 语句创建一个依赖项上的表和列在数据库中，其中已从您的类以及它们的属性生成的名称。如果在代码中更改这些名称必须记住要更新的任何手动编写的 SQL 语句。

删除对象

主键用于删除行，如下所示：

```
var rowcount = db.Delete<Stock>(someStock.Id); // Id is the primary key
```

你可以检查 `rowcount` 确认多少行受影响（在这种情况下删除）。

通过多个线程使用 SQLite.NET

SQLite 支持三种不同的线程模式：*单线程*，*多线程*，并*已序列化*。如果你想要从多个线程不受任何限制地访问数据库，可以配置要使用的 SQLite*序列化*线程处理模式。务必尽早在你的应用程序中设置此模式（例如，在开头

OnCreate 方法)。

若要更改线程的模式, 请调用 `SqlConnection.SetConfig`。例如, 这行代码配置适用于 SQLite 序列化模式:

```
SqlConnection.SetConfig(SQLiteConfig.Serialized);
```

SQLite 的 Android 版本具有一个限制, 即需要几个步骤。如果在调用 `SqlConnection.SetConfig` 如生成 SQLite 异常 `library used incorrectly`, 则必须使用以下解决方法:

1. 链接的本机 `libsqlite.so` 库, 以便 `sqlite3_shutdown` 和 `sqlite3_initialize` Api 都提供给应用程序:

```
[DllImport("libsqlite.so")]
internal static extern int sqlite3_shutdown();

[DllImport("libsqlite.so")]
internal static extern int sqlite3_initialize();
```

2. 最开头 `OnCreate` 方法, 将此代码添加到关闭 SQLite, 将其用于配置序列化模式, 并重新初始化 SQLite:

```
sqlite3_shutdown();
SqlConnection.SetConfig(SQLiteConfig.Serialized);
sqlite3_initialize();
```

此解决方法也适用于 `Mono.Data.Sqlite` 库。有关 SQLite 和多线程处理的详细信息, 请参阅 [SQLite](#) 和 [多个线程](#)。

相关链接

- [DataAccess Basic \(示例\)](#)
- [DataAccess 高级 \(示例\)](#)
- [Xamarin.Forms 数据访问](#)

使用 ADO.NET 和 Android

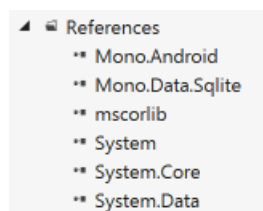
2018/11/1 • [Edit Online](#)

Xamarin 提供内置支持是可在 Android 上, 可以使用熟悉的类似于 ADO.NET 的语法公开的 SQLite 数据库。使用这些 Api 需要你编写 SQL 语句处理的 SQLite, 如 `CREATE TABLE`, `INSERT` 和 `SELECT` 语句。

程序集引用

若要使用的访问必须添加的 ADO.NET 通过 SQLite `System.Data` 和 `Mono.Data.Sqlite` 引用到 Android 项目, 如下所示:

- [Visual Studio](#)
- [Visual Studio for Mac](#)



右键单击引用 > 编辑引用..., 然后单击以选择所需的程序集。

有关 Mono.Data.Sqlite

我们将使用 `Mono.Data.Sqlite.SqliteConnection` 类, 以创建空数据库文件, 然后实例化 `SQLiteCommand` 对象, 我们可以使用执行 SQL 对数据库的说明。

创建空数据库–调用 `CreateFile` 方法包含有效的 (即可写入) 的文件路径。您应检查是否调用此方法之前已存在该文件, 否则将旧的基础上创建新的 (空白) 数据库和旧的文件中的数据都将丢失。

`Mono.Data.Sqlite.SqliteConnection.CreateFile (dbPath);` `dbPath` 变量, 必须根据本文档前面所述的规则来确定。

创建数据库连接–SQLite 数据库文件创建后可以创建连接对象, 以访问数据。使用连接字符串, 其形式的构造连接 `Data Source=file_path`, 如下所示:

```
var connection = new SqliteConnection ("Data Source=" + dbPath);
connection.Open();
// do stuff
connection.Close();
```

前面曾提到, 连接永远不应重复使用跨不同的线程。如果有疑问, 创建所需的连接并将其关闭时完成;但要注意的执行详细通常不太必要。

创建和执行数据库命令–我们有一个连接后我们可以执行对其的任意 SQL 命令。下面的代码显示 `CREATE TABLE` 正在执行的语句。

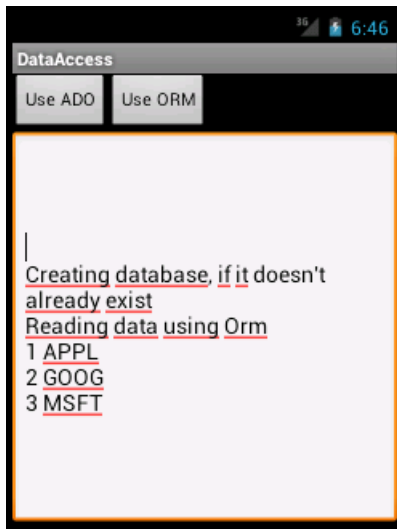
```
using (var command = connection.CreateCommand ()) {
    command.CommandText = "CREATE TABLE [Items] ([_id] int, [Symbol] ntext, [Name] ntext);";
    var rowcount = command.ExecuteNonQuery ();
}
```

直接对数据库执行 SQL 时, 应执行正常的预防措施不以使无效请求, 例如尝试创建已存在的表。跟踪的数据库的结

构, 以便不会造成 `SqliteException` 如**SQLite 错误表 [项目] 已存在**。

基本数据访问

`.DataAccess_Basic`在 Android 上运行时, 本文档的示例代码如下所示:



下面的代码说明了如何执行简单的 SQLite 操作并为应用程序的主窗口中的文本显示中的结果。

你将需要包含这些命名空间:

```
using System;
using System.IO;
using Mono.Data.Sqlite;
```

下面的代码示例显示了整个数据库进行交互:

1. 创建数据库文件
2. 插入一些数据
3. 查询数据

这些操作将通常会出现在多个位置在整个代码, 例如可能你的应用程序首次启动时创建的数据库文件和表, 并在单个屏幕中执行数据读取和写入应用程序中。在下面的示例已被分组到单个方法对于此示例:


```

public static SQLiteConnection connection;
public static string DoSomeDataAccess ()
{
    // determine the path for the database file
    string dbPath = Path.Combine (
        Environment.GetFolderPath (Environment.SpecialFolder.Personal),
        "adodemo.db3");

    bool exists = File.Exists (dbPath);

    if (!exists) {
        Console.WriteLine("Creating database");
        // Need to create the database before seeding it with some data
        Mono.Data.Sqlite.SQLiteConnection.CreateFile (dbPath);
        connection = new SQLiteConnection ("Data Source=" + dbPath);

        var commands = new[] {
            "CREATE TABLE [Items] (_id ntext, Symbol ntext);",
            "INSERT INTO [Items] ([_id], [Symbol]) VALUES ('1', 'AAPL')",
            "INSERT INTO [Items] ([_id], [Symbol]) VALUES ('2', 'GOOG')",
            "INSERT INTO [Items] ([_id], [Symbol]) VALUES ('3', 'MSFT')"
        };
        // Open the database connection and create table with data
        connection.Open ();
        foreach (var command in commands) {
            using (var c = connection.CreateCommand ()) {
                c.CommandText = command;
                var rowcount = c.ExecuteNonQuery ();
                Console.WriteLine("\tExecuted " + command);
            }
        }
    } else {
        Console.WriteLine("Database already exists");
        // Open connection to existing database file
        connection = new SQLiteConnection ("Data Source=" + dbPath);
        connection.Open ();
    }

    // query the database to prove data was inserted!
    using (var contents = connection.CreateCommand ()) {
        contents.CommandText = "SELECT [_id], [Symbol] from [Items]";
        var r = contents.ExecuteReader ();
        Console.WriteLine("Reading data");
        while (r.Read ())
            Console.WriteLine("\tKey={0}; Value={1}",
                r ["_id"].ToString (),
                r ["Symbol"].ToString ());
    }
    connection.Close ();
}

```

更复杂的查询

由于 SQLite 允许对数据运行任意 SQL 命令，因此可以执行任何 `CREATE`，`INSERT`，`UPDATE`，`DELETE`，或 `SELECT` 您喜欢的语句。您可以阅读 SQLite 支持 SQLite 网站上的 SQL 命令。使用三种方法之一上运行 SQL 语句 `SQLiteCommand` 对象：

- **ExecuteNonQuery** –通常用于表创建或数据期间插入操作。对于某些操作的返回值是受影响的行数，否则为-1。
- **ExecuteReader** –时应作为返回的行集合使用 `SqlDataReader`。
- **ExecuteScalar** –检索单个值（例如聚合）。

EXECUTENONQUERY

`INSERT`、`UPDATE`，和 `DELETE` 语句将返回受影响的行数。所有其他 SQL 语句将返回-1。

```
using (var c = connection.CreateCommand ()) {
    c.CommandText = "INSERT INTO [Items] ([_id], [Symbol]) VALUES ('1', 'APPL')";
    var rowcount = c.ExecuteNonQuery (); // rowcount will be 1
}
```

EXECUTEREADER

下面的方法演示 `WHERE` 子句中的 `SELECT` 语句。因为代码精心编制一个完整的 SQL 语句必须请注意它转义保留的字符，如字符串周围的引号 (')。

```
public static string MoreComplexQuery ()
{
    var output = "";
    output += "\nComplex query example: ";
    string dbPath = Path.Combine (
        Environment.GetFolderPath (Environment.SpecialFolder.Personal), "ormdemo.db3");

    connection = new SqlConnection ("Data Source=" + dbPath);
    connection.Open ();
    using (var contents = connection.CreateCommand ()) {
        contents.CommandText = "SELECT * FROM [Items] WHERE Symbol = 'MSFT'";
        var r = contents.ExecuteReader ();
        output += "\nReading data";
        while (r.Read ())
            output += String.Format ("\n\tKey={0}; Value={1}",
                r ["_id"].ToString (),
                r ["Symbol"].ToString ());
    }
    connection.Close ();

    return output;
}
```

`ExecuteReader` 方法返回 `SqlDataReader` 对象。除了 `Read` 方法中所示示例中，其他有用属性包括：

- **RowsAffected** –受查询影响的行的计数。
- **HasRows** –是否没有返回任何行。

EXECUTESCALAR

使用此窗体 `SELECT` 返回单个值（例如聚合）的语句。

```
using (var contents = connection.CreateCommand ()) {
    contents.CommandText = "SELECT COUNT(*) FROM [Items] WHERE Symbol <> 'MSFT'";
    var i = contents.ExecuteScalar ();
}
```

`ExecuteScalar` 方法的返回类型是 `object` –应具体取决于数据库查询结果强制转换。结果可能是一个整数 `COUNT` 查询或单个列中的字符串 `SELECT` 查询。请注意，这是与其他不同 `Execute` 方法返回一个读取器对象或受影响的行数。

相关链接

- [DataAccess Basic（示例）](#)
- [DataAccess 高级（示例）](#)
- [Android 数据方案](#)

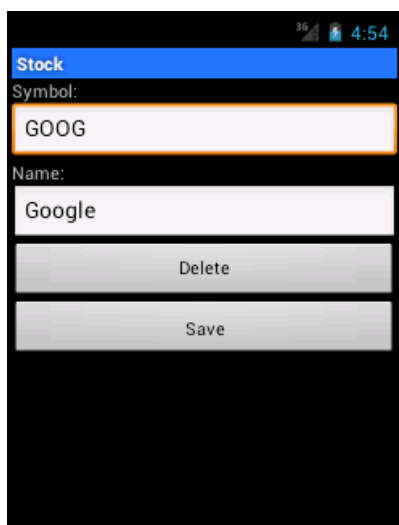
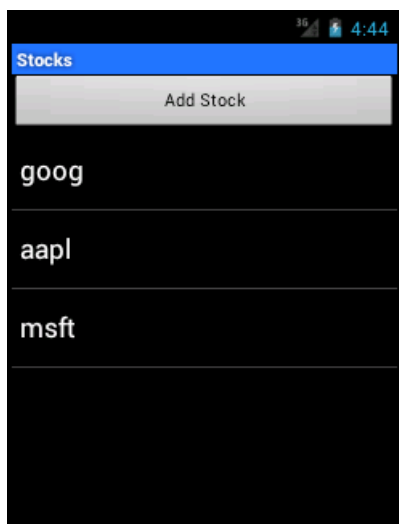
- [Xamarin.Forms 数据访问](#)

在应用中使用数据

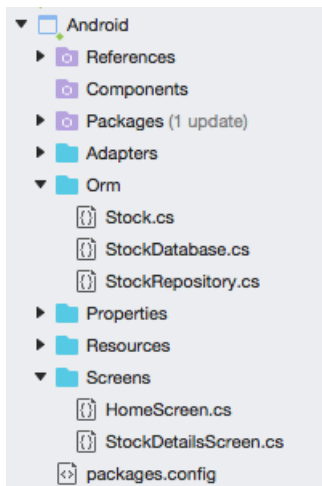
2018/10/26 • [Edit Online](#)

DataAccess_Adv示例显示允许用户输入和 CRUD（创建、读取、更新和删除）的数据库功能的工作应用程序。此应用程序包含两个屏幕：列表和数据输入窗体。所有数据访问代码都是在 iOS 和 Android 中无需修改即可重复使用。

添加一些数据后的应用程序屏幕如下所示在 Android 上：



Android 项目如下所示–在本部分中所示的代码包含在**Orm**目录：



在 Android 中的活动的本机 UI 代码不在本文的范围之内。请参阅[Android Listview](#) 和[适配器](#)指南以获取对 UI 控件的详细信息。

读取

有几个示例中的读取操作：

- 读取列表
- 读取单个记录

中的两个方法 `StockDatabase` 类：

```
public IEnumerable<Stock> GetStocks ()
{
    lock (locker) {
        return (from i in Table<Stock> () select i).ToList ();
    }
}
public Stock GetStock (int id)
{
    lock (locker) {
        return Table<Stock>().FirstOrDefault(x => x.Id == id);
    }
}
```

Android 将为数据呈现 `ListView`。

创建和更新

为了简化应用程序代码，一条存储方法是提供执行插入或更新取决于是否已设置 `PrimaryKey`。因为 `Id` 属性是否标记有 `[PrimaryKey]` 属性不应在代码中设置它。此方法将检测是否值已被以前保存（通过检查主键属性），并插入或相应地更新该对象：

```
public int SaveStock (Stock item)
{
    lock (locker) {
        if (item.Id != 0) {
            Update (item);
            return item.Id;
        } else {
            return Insert (item);
        }
    }
}
```

实际应用程序通常需要一些（如必填的字段，最小长度或其他业务规则）的验证。尽可能多的验证逻辑作为在共享的代码，传递验证错误，备份到平台的功能根据显示的 UI，可以实现很好的跨平台应用程序。

删除

与不同 `Insert` 并 `Update` 方法，`Delete<T>` 方法可接受只是主密钥值而不是一个完整 `Stock` 对象。在此示例中 `Stock` 对象传递给该方法，但仅 `Id` 属性传递给 `Delete<T>` 方法。

```
public int DeleteStock(Stock stock)
{
    lock (locker) {
        return Delete<Stock> (stock.Id);
    }
}
```

使用预填充的 SQLite 数据库文件

某些应用程序附带已用数据填充数据库。您可以轻松地完成此操作在移动应用程序中传送你的应用与现有的 SQLite 数据库文件并对其进行访问之前将其复制到可写目录。由于 SQLite 是多个平台使用的标准文件格式，有许多工具可用于创建一个 SQLite 数据库文件：

- **SQLite Manager Firefox 扩展**—适用于 Mac 和 Windows，并生成与 iOS 和 Android 兼容的文件。
- **命令行**—请参阅www.sqlite.org/sqlite.html。

时使用您的应用程序创建分发数据库文件，请注意使用命名的表和列，确保它们与您的代码的预期，尤其是如果您使用 SQLite.NET 这将要求要与 C# 类和属性匹配的名称（或关联的自定义属性）。

若要确保某些代码运行在 Android 应用中的其他部分之前，你可以将它放在要加载的第一个活动，也可以创建 `Application` 加载之前的任何活动的子类。下面的代码显示 `Application` 会将现有的数据库文件复制的子类 `data.sqlite` 共 `/Resources/Raw/` 目录。

```

[Application]
public class YourAndroidApp : Application {
    public override void OnCreate ()
    {
        base.OnCreate ();
        var docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        Console.WriteLine ("Data path:" + Database.DatabaseFilePath);
        var dbFile = Path.Combine(docFolder, "data.sqlite"); // FILE NAME TO USE WHEN COPIED
        if (!System.IO.File.Exists(dbFile)) {
            var s = Resources.OpenRawResource(Resource.Raw.data); // DATA FILE RESOURCE ID
            FileStream writeStream = new FileStream(dbFile, FileMode.OpenOrCreate, FileAccess.Write);
            ReadWriteStream(s, writeStream);
        }
    }
    // readStream is the stream you need to read
    // writeStream is the stream you want to write to
    private void ReadWriteStream(Stream readStream, Stream writeStream)
    {
        int Length = 256;
        Byte[] buffer = new Byte[Length];
        int bytesRead = readStream.Read(buffer, 0, Length);
        // write the required bytes
        while (bytesRead > 0)
        {
            writeStream.Write(buffer, 0, bytesRead);
            bytesRead = readStream.Read(buffer, 0, Length);
        }
        readStream.Close();
        writeStream.Close();
    }
}

```

相关链接

- [DataAccess Basic（示例）](#)
- [DataAccess 高级（示例）](#)
- [Android 数据方案](#)
- [Xamarin.Forms 数据访问](#)

Google 消息传送

2018/10/26 • [Edit Online](#)

本部分包含介绍如何实现使用 Google 消息传送服务的 Xamarin.Android 应用的指南。

Firebase 云消息传送

Firebase Cloud Messaging (FCM) 是一种便于移动应用和服务器应用程序之间的消息传送的服务。FCM 是到 Google Cloud Messaging 的 Google 的后续版本。本文提供 FCM 的工作原理的概述和它提供的分步过程，以便您的应用程序可以使用 FCM 服务获取的凭据。

远程通知使用 Firebase Cloud Messaging

本演练介绍如何使用 Firebase Cloud Messaging 实现远程通知（也称为推送通知）的分步说明，Xamarin.Android 应用程序中。它演示如何实现的各种类所需的通信使用 Firebase Cloud Messaging (FCM)，提供有关如何配置用于访问 FCM，Android 清单的示例并演示如何使用 Firebase 下游消息传递控制台。

Google 云消息传送

本部分提供 Google Cloud Messaging (GCM) 将您的应用程序和应用程序服务器之间的消息的路由的高级概述，并获取凭据，以便您的应用程序可以使用 GCM 服务提供的分步过程。（请注意，已被 FCM 取代 GCM）。

NOTE

已被取代 GCM [Firebase Cloud Messaging](#) (FCM)。GCM 服务器和客户端 Api [已弃用](#) 将不再提供为 2019 年 4 月 11 日推出。

Google 云消息传送与远程通知

本部分提供如何在 Xamarin.Android 中实现远程通知的分步说明使用 Google Cloud Messaging。它还说明了必须用于启用 Google Cloud Messaging 的 Android 应用程序中的各种组件。

Firebase 云消息传送

2018/10/26 • [Edit Online](#)

Firebase Cloud Messaging (FCM) 是一种便于移动应用和服务器应用程序之间的消息传送的服务。本文提供的 FCM 的工作原理, 概述并介绍如何配置 Google 服务, 以便您的应用程序可以使用 FCM。

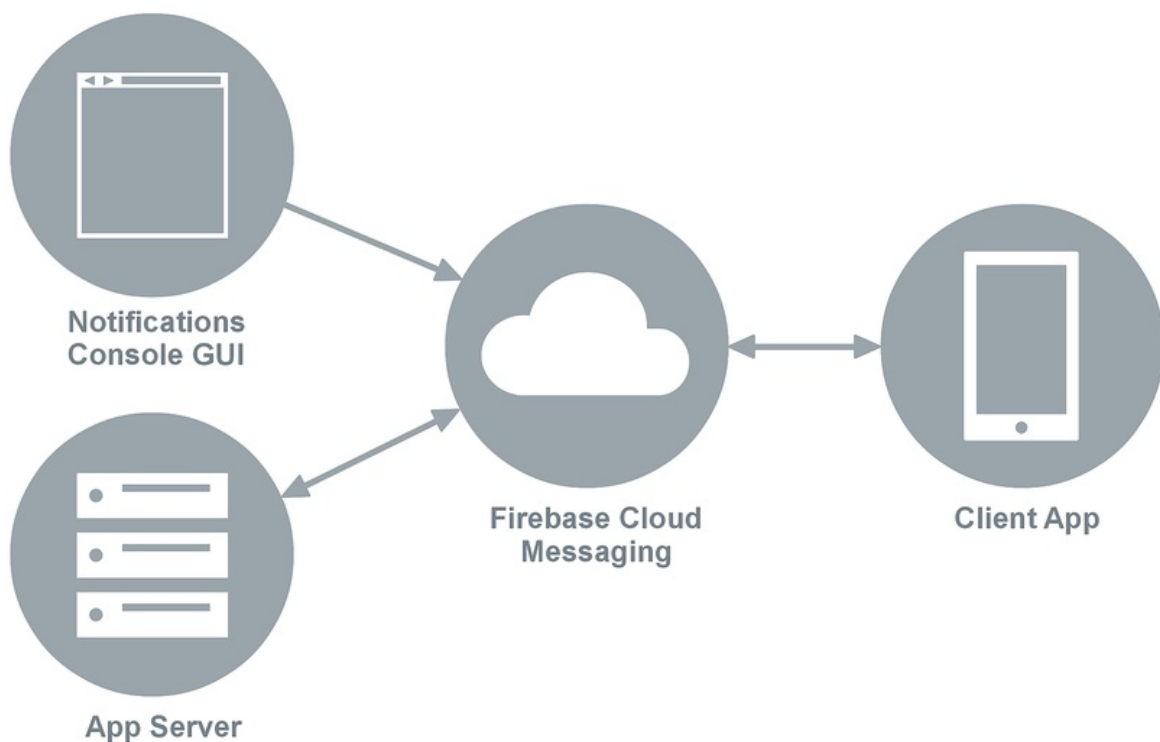


本主题提供的 Firebase Cloud Messaging 将 Xamarin.Android 应用程序和应用程序服务器之间的消息的路由概述和它提供的分步过程, 以便您的应用程序可以使用 FCM 服务获取的凭据。

概述

Firebase Cloud Messaging (FCM) 是一个跨平台服务, 处理发送、路由和服务器应用程序和移动客户端应用程序之间的消息将排入队列。FCM 是后续任务到 Google Cloud Messaging (GCM), 并基于 Google Play Services。

下图所示, FCM 充当消息的发件人和客户端之间的中介。一个 **客户端应用** 是设备运行的已启用 FCM 的应用。**应用程序服务器** (由你或贵公司提供) 是客户端应用进行通信, 与通过 FCM 的 FCM 启用服务器。与不同的 GCM, FCM 使你 **可以将消息发送到客户端应用程序直接通过 Firebase 控制台通知 GUI**:



使用 FCM, 应用程序服务器可发送消息到单个设备、一组设备, 或多个订阅到主题的设备。客户端应用程序可

以使用 FCM 订阅下游消息从应用程序服务器（例如，若要接收远程通知）。有关不同类型的 Firebase 消息的详细信息，请参阅[关于 FCM 消息](#)。

Firebase Cloud Messaging 操作中

时从应用程序服务器的下游消息发送到客户端应用，应用程序服务器将发送到的消息 FCM 连接服务器 Google；提供 FCM 连接服务器，反过来，将消息转发到正在运行的设备客户端应用程序。可以通过 HTTP 发送消息或 XMPP（可扩展消息和状态显示协议）。因为客户端应用程序不始终连接或运行的 FCM 连接服务器排入队列，并将存储消息，将它们发送到客户端应用程序重新连接并变得可用。同样，FCM 排入队列上游消息从客户端应用程序到应用服务器，如果将应用程序服务器不可用。有关 FCM 连接服务器的详细信息，请参阅[有关 Firebase 云消息传送服务器](#)。

FCM 使用下面的凭据来确定应用程序服务器和客户端应用程序，并使用这些凭据授权通过 FCM 消息事务：

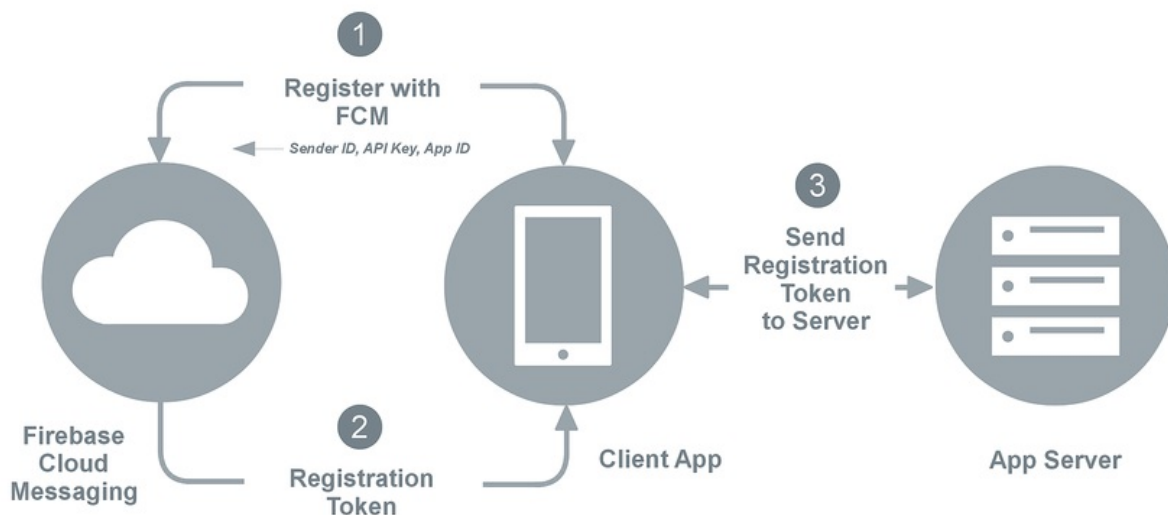
- **发件人 ID** – 发件人 ID 是创建 Firebase 项目分配一个唯一数字值。发件人 ID 用于标识可以将消息发送到客户端应用程序的每个应用程序服务器。发件人 ID 也是你的项目编号；注册你的项目时，可以从 Firebase 控制台获取的发件人 ID。发件人 ID 的一个示例是 `496915549731`。
- **API 密钥** – API 密钥授予应用程序服务器访问 Firebase 服务；FCM 使用此密钥进行身份验证的应用程序服务器。此凭据也称为 **服务器密钥** 或 **Web API 密钥**。API 密钥的一个示例是 `AJzbSyCTcpfRT1YRqbz-jIwp1h06YdauvewGDzk`。
- **应用程序 ID** – 的客户端应用（独立于任何给定的设备）注册以接收来自 FCM 消息的标识。应用程序 ID 的一个示例是 `1:415712510732:android:0e1eb7a661af2460`。
- **注册令牌** – 注册令牌（也称为 **实例 ID**）是在客户端应用程序上的给定设备的 FCM 标识。在运行时生成的注册令牌–时它首先会向注册 FCM 的设备上运行时，您的应用程序接收的注册令牌。注册令牌授权客户端应用程序（在该特定设备上运行）的实例来接收来自 FCM 的消息。注册令牌的一个示例是 `fkBQTHxKKhs:AP91bHuEedxM4xFAUn0z ... JKZS`（非常长的字符串）。

设置了 [Firebase Cloud Messaging](#)（在本指南的后面介绍）提供了用于创建项目并生成这些凭据的详细的说明。当创建新的项目中 [Firebase 控制台](#)，调用凭据文件 `google-services.json` 创建–中所述，将此文件添加到你的 Xamarin.Android 项目 [使用 FCM 远程通知](#)。

以下部分介绍当客户端应用程序与通过 FCM 的应用程序服务器通信时，如何使用这些凭据。

使用 FCM 的注册

消息传送进行之前，首先必须通过 FCM 注册客户端应用程序。客户端应用程序必须完成下面的关系图中显示的注册步骤：



1. 客户端应用程序与 FCM 以获取将发件人 ID、API 密钥和应用程序 ID 传递到 FCM 的注册令牌。

2. FCM 返回到客户端应用程序注册令牌。

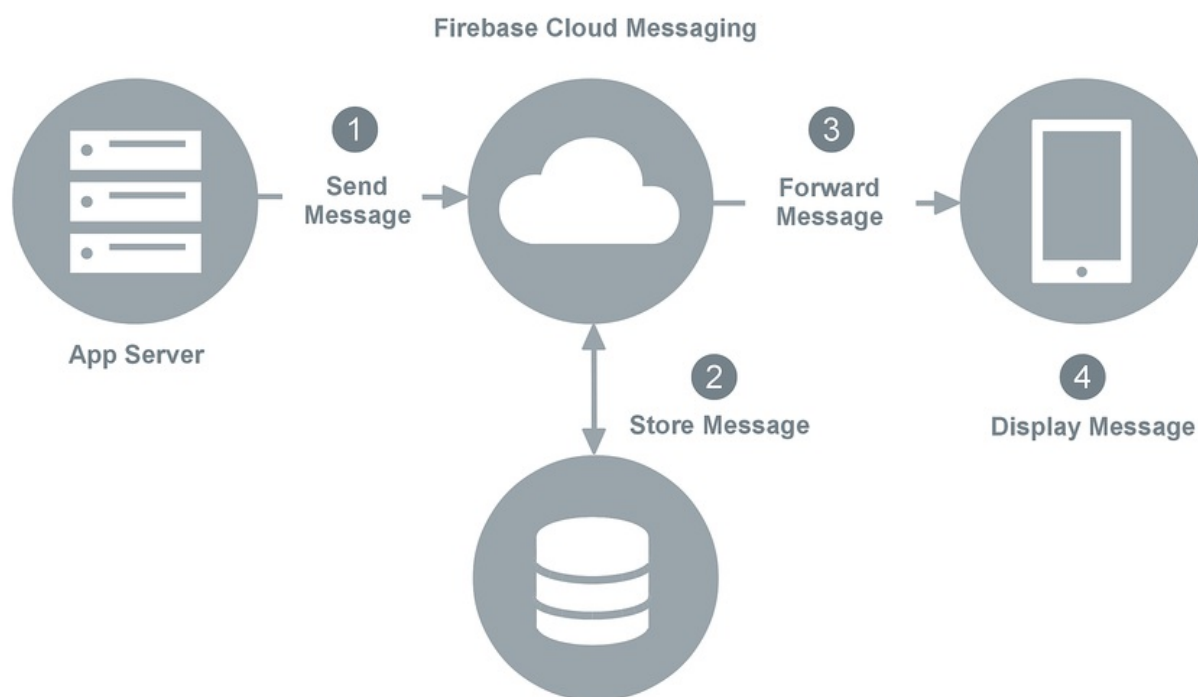
3. 客户端应用程序（可选）将转发到应用服务器的注册令牌。

应用程序服务器将缓存与客户端应用程序的后续通信的注册令牌。应用程序服务器可以将确认发送回客户端应用程序，以指示接收到的注册令牌。发生此握手后，客户端应用程序可以接收消息（或将消息发送到）的应用程序服务器。如果旧令牌已泄露，客户端应用程序可能会收到新的注册令牌（请参阅[远程通知使用 FCM](#)以举例说明如何应用接收注册令牌更新）。

当客户端应用程序不再想要从应用服务器接收消息时，它可以为要删除的注册令牌的应用程序服务器发送请求。从设备上卸载客户端应用程序，FCM 会检测到这，自动通知要删除的注册令牌的应用程序服务器。

下游消息传送

下图说明了如何 Firebase Cloud Messaging 将存储和转发下游消息：



当应用程序服务器将下游消息发送到客户端应用程序时，它作为枚举上图中使用以下步骤：

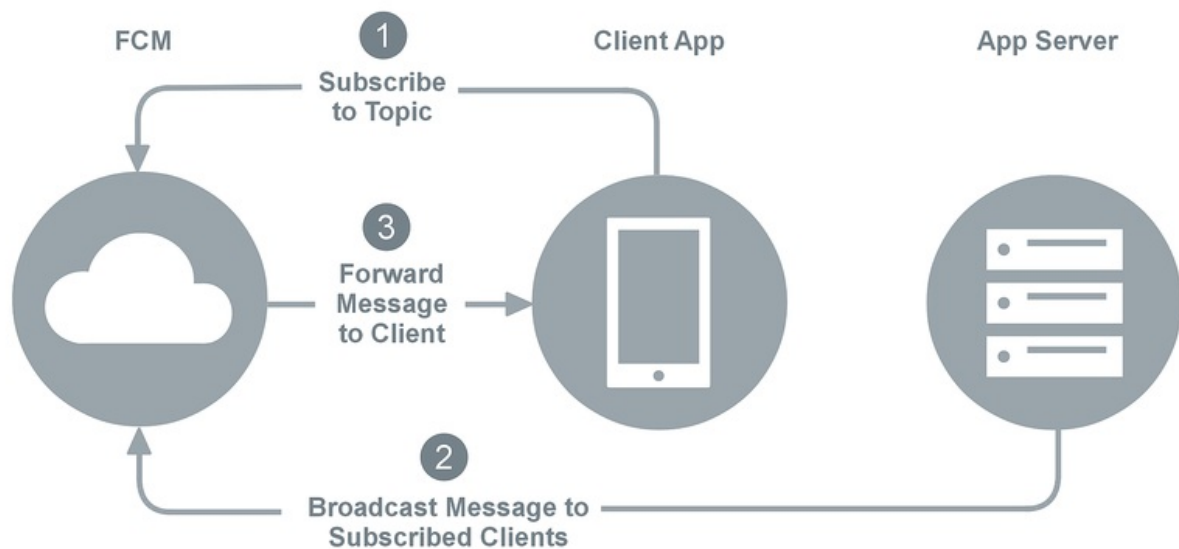
1. 应用程序服务器将消息发送到 FCM。
2. 如果客户端设备不可用，FCM 服务器以在稍后传输队列中存储消息。最多 4 周的 FCM 存储中保留消息（有关详细信息，请参阅[设置一条消息的生命期](#)）。
3. 当客户端设备可用时，FCM 将转发到该设备上的客户端应用的消息。
4. 客户端应用程序接收来自 FCM 的消息、处理，并为用户显示。例如，如果消息是远程通知，则将它显示给通知区域中的用户。

在此消息传递方案中（其中应用服务器发送一条消息到单个客户端应用），消息可以是长度最多为 4 kB。

有关接收在 Android 上的下游 FCM 消息的详细信息，请参阅[远程通知使用 FCM](#)。

主题的消息传送

主题消息传送使得应用程序服务器将消息发送到已选择加入某个特定主题的多个设备。您还可以撰写并发送 Firebase 控制台通知 GUI 中主题的消息。FCM 处理的路由和主题消息传送到已订阅的客户端。此功能可用于消息，如天气警报、股票报价和标题新闻。



(在客户端应用程序获取注册令牌，如前面所述)后，将主题消息传送中使用以下步骤：

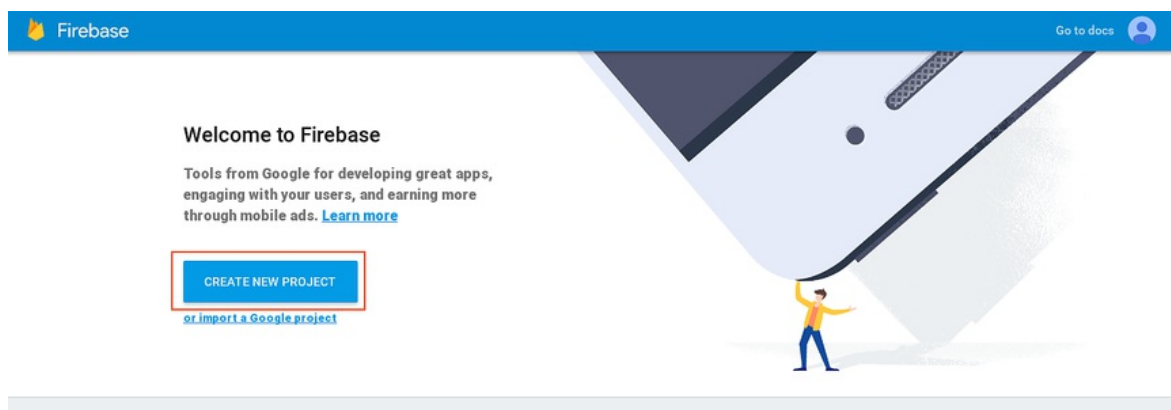
1. 客户端应用程序通过将订阅消息发送到 FCM 订阅主题。
2. 应用程序服务器分发给 FCM 发送主题消息。
3. FCM 将主题消息转发到已订阅该主题的客户端。

Firebase 主题消息传送的详细信息，请参阅 [Google 主题的消息传送在 Android 上](#)。

Firebase Cloud Messaging 设置

应用程序中使用 FCM 服务之前，你必须创建一个新项目（或导入现有项目）通过 [Firebase 控制台](#)。使用以下步骤创建您的应用程序的 Firebase Cloud Messaging 项目：

1. 登录到 [Firebase 控制台](#) 使用 Google 帐户（即，你的 Gmail 地址），单击 **创建新项目**：



如果你有现有的项目，单击导入 **Google 项目**。

2. 在中 **创建项目** 对话框中，输入你的项目的名称，然后单击 **创建项目**。在以下示例中，新的项目称为 **XamarinFCM** 创建：

Create a project

Project name

XamarinFCM

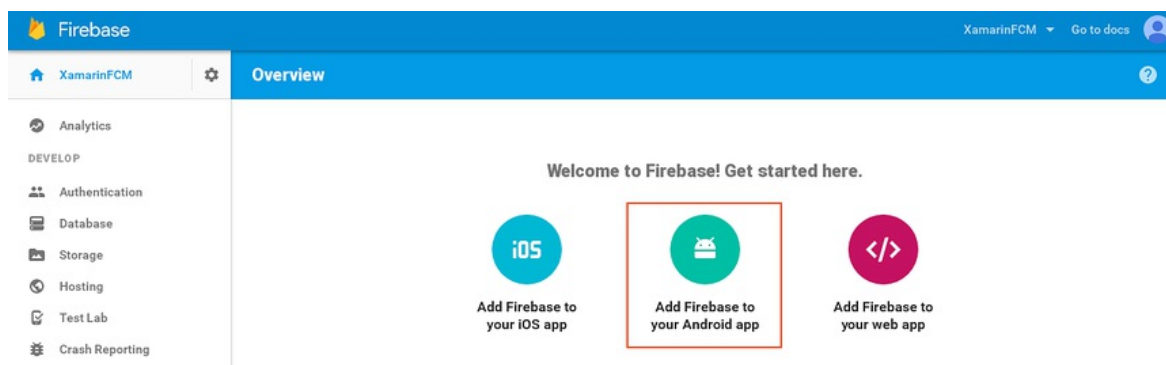
Country/region ?

United States

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at anytime. [Learn more](#)

CANCELCREATE PROJECT

- 在 Firebase 控制台概述，单击将 **Firebase 添加到 Android 应用**：



- 在下一个屏幕中，输入您的应用程序的包名称。在此示例中，包名称是`com.xamarin.fcmexample`。此值必须匹配你的 Android 应用包名称。也可以在输入应用昵称应用昵称字段：

Add Firebase to your Android app

1 Enter app details 2 Copy config file 3 Add to build.gradle

Package name [?]

com.xamarin.fcmexample

App nickname (optional) [?]

FCM Example

Debug signing certificate SHA-1 (optional) [?]

00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

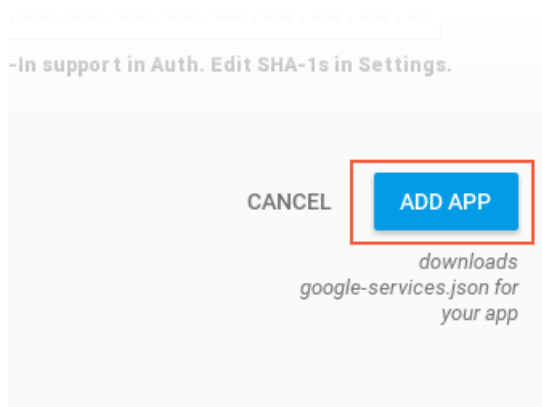
Required for Dynamic Links, Invites, and Google Sign-In support in Auth. Edit SHA-1s in Settings.

CANCEL ADD APP

downloads
google-services.json for
your app

5. 如果您的应用程序使用动态链接，邀请或 Google 身份验证，还必须输入签名证书在调试。查找您的签名证书的详细信息，请参阅[查找密钥存储的 MD5 或 SHA1 签名](#)。在此示例中，为空的签名证书。

6. 单击**添加应用**：



应用程序自动生成服务器 API 密钥和客户端 ID。此信息打包在 **google-services.json** 当您单击时自动下载的文件**添加应用**。请务必将此文件保存在安全的位置。

有关如何添加的详细示例 **google-services.json** 应用程序项目，以接收在 Android 上的 FCM 推送通知消息，请参阅[远程通知使用 FCM](#)。

有关其他参考资料

- Google [Firebase Cloud Messaging](#) Firebase Cloud Messaging 的关键功能概述、其工作方式，以及安装说明进行操作的说明。

- [Google生成应用程序服务器发送请求](#)介绍了如何使用你的应用程序服务器发送的消息。
- [RFC 6120](#)和[RFC 6121](#)解释, 并定义可扩展消息传送和协议 (XMPP)。
- [关于 FCM 消息](#)介绍不同类型的使用 Firebase Cloud Messaging 可以发送的消息。

总结

本文章提供 Firebase Cloud Messaging (FCM) 的概述。它介绍了各种用于标识和授权应用程序服务器和客户端应用程序之间的消息传送的凭据。它所示的注册和下游消息传送方案, 并详细介绍如何通过 FCM 以使用 FCM 服务注册您的应用程序。

相关链接

- [Firebase 云消息传送](#)

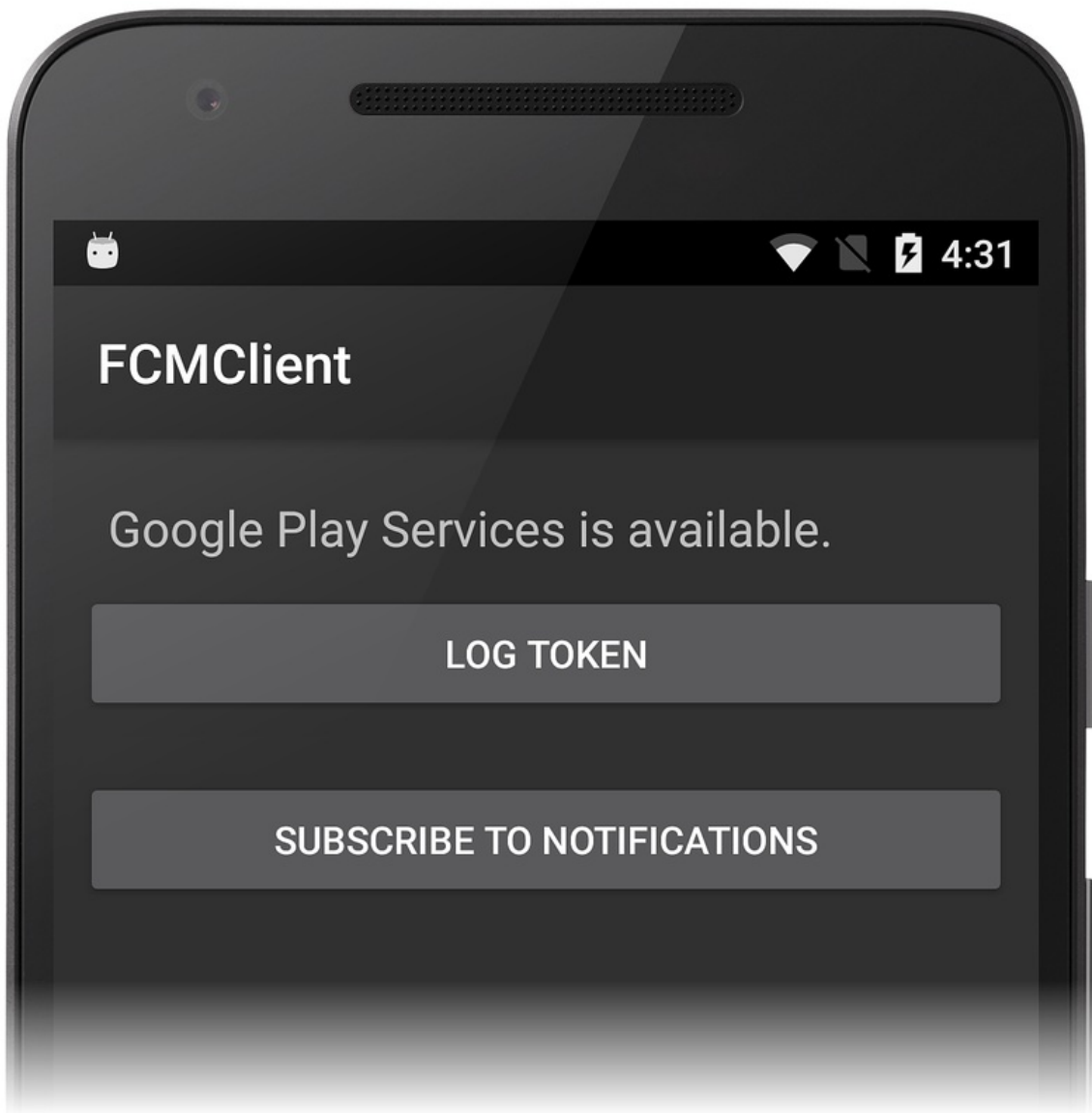
远程通知使用 Firebase Cloud Messaging

2018/10/26 • [Edit Online](#)

本演练介绍如何使用 Firebase Cloud Messaging 实现远程通知（也称为推送通知）的分步说明, Xamarin.Android 应用程序中。它演示如何实现的各种类所需的通信使用 Firebase Cloud Messaging (FCM), 提供有关如何配置用于访问 FCM, Android 清单的示例并演示如何使用 Firebase 下游消息传递控制台。

FCM 通知概述

在本演练中, 一个基本的应用中调用 **FCMClient** 将创建用于说明的 FCM 消息传送基础知识。**FCMClient** 检查 Google Play 服务是否存在、接收来自 FCM 注册令牌, 显示远程用户从 Firebase 控制台中, 发送的通知和订阅主题的消息:



将探讨下列主题领域:

1. 背景通知
2. 主题的消息
3. 前景色通知

在本演练中，您将以增量方式将功能添加到**FCMClient**并运行设备或仿真程序以了解它如何与 FCM 交互上。将使用日志记录以实时应用事务 FCM 服务器与见证服务器，您将观察到如何从输入到 Firebase 控制台通知 GUI 的 FCM 消息生成的通知。

要求

这会有助于自己应熟悉[不同类型的消息](#)，可以通过 Firebase Cloud Messaging 发送。消息的负载将确定如何客户端应用程序将接收和处理消息。

您可以继续执行本演练之前，必须获取所需的凭据以使用 Google 的 FCM 服务器;此过程所述[Firebase Cloud Messaging](#)。具体而言，必须下载[google-services.json](#)要用于此演练中介绍的示例代码文件。如果你尚未创建一个项目在 Firebase 控制台中 (或者如果尚未下载[google-services.json](#)文件)，请参阅[Firebase Cloud Messaging](#)。

若要运行示例应用程序中，将需要 Android 测试设备或仿真程序是使用 Firebase 兼容。Firebase Cloud Messaging 支持运行 Android 4.0 或更高版本，客户端和这些设备还必须安装 Google Play 应用商店应用 (Google Play Services 9.2.1 或更高版本)。如果你还没有在设备上安装 Google Play 应用商店应用程序，请访问[Google Play](#)网站上下载并安装它。或者，可以使用 Google Play Services 安装而不是测试设备 (您无需安装 Google Play 商店，如果你使用 Android SDK 仿真器) 使用 Android SDK 仿真器。

启动应用程序项目

若要开始，创建一个名为的新空 Xamarin.Android 项目**FCMClient**。如果您不熟悉创建 Xamarin.Android 项目，请参阅[Hello, Android](#)。创建新应用后下，一步是设置包名称，并安装将用于与 FCM 进行通信的多个 NuGet 包。

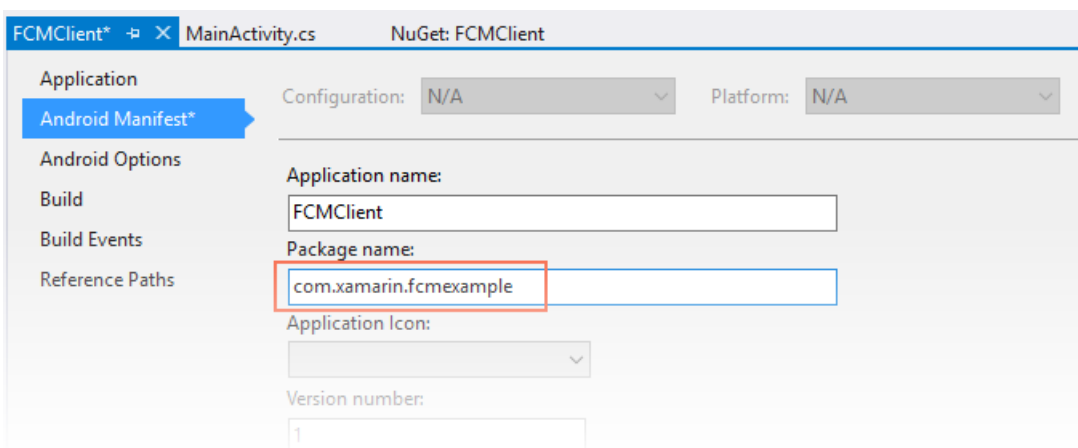
设置包名称

在中[Firebase Cloud Messaging](#)，指定 FCM 启用应用的包名称。此包名称也可作为[应用程序 ID](#) 关联[API 密钥](#)。将应用配置为使用此包名称：

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 打开的属性**FCMClient**项目。
2. 在中**Android** 清单页上，将包名称。

在以下示例中，包名称设置为 `com.xamarin.fcmexample`：



更新时**Android** 清单，还检查，以确保 `Internet` 启用权限。

IMPORTANT

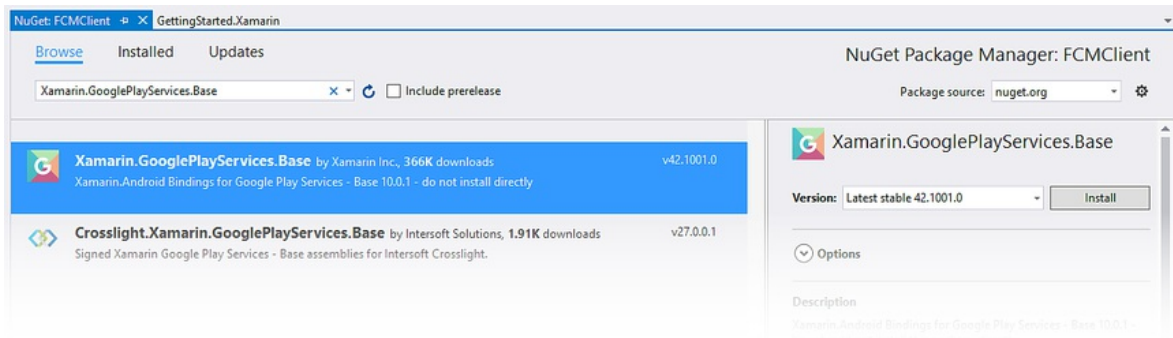
客户端应用程序将不接收来自 FCM 注册令牌，如果此包名称不能完全与已输入到 Firebase 控制台中的包名称匹配。

添加 Xamarin Google Play 服务基本包

Firebase Cloud Messaging 取决于 Google Play 服务，因为Xamarin Google Play 服务的基本必须将 NuGet 包添加到 Xamarin.Android 项目。需要版本 29.0.0.2 或更高版本。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 在 Visual Studio 中，右键单击引用 > 管理 NuGet 包....
2. 单击浏览选项卡并搜索Xamarin.GooglePlayServices.Base。
3. 安装到此包FCMClient项目：



如果在安装 NuGet 的过程中遇到错误，关闭FCMClient项目，重新打开，然后重试 NuGet 安装。

当你安装Xamarin.GooglePlayServices.Base，还安装所有必要的依赖项。编辑MainActivity.cs并添加以下 using 语句：

```
using Android.Gms.Common;
```

此语句使 GoogleApiAvailability 类中Xamarin.GooglePlayServices.Base供FCMClient代码。

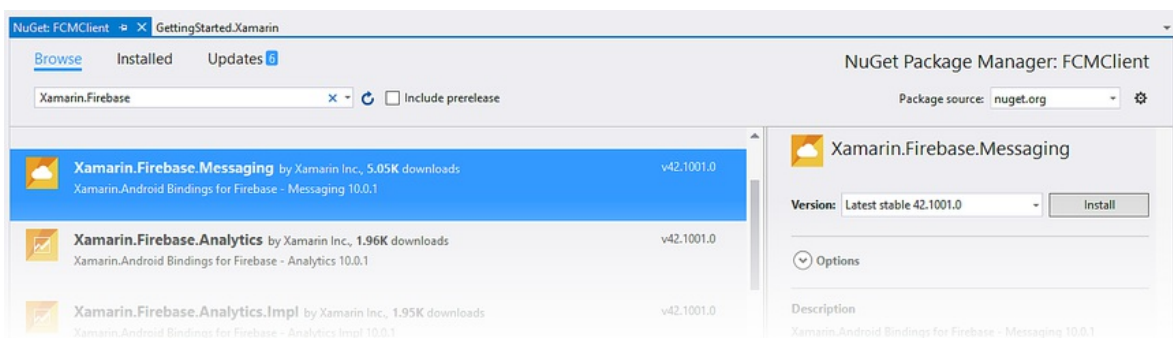
GoogleApiAvailability 用于检查存在的 Google Play 服务。

添加 Xamarin Firebase Messaging 包

若要从 FCM，接收消息Xamarin Firebase-消息传送必须将 NuGet 包添加到应用程序项目。无此包的 Android 应用程序无法接收来自 FCM 服务器的消息。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 在 Visual Studio 中，右键单击引用 > 管理 NuGet 包....
2. 搜索Xamarin.Firebase.Messaging。
3. 安装到此包FCMClient项目：



当你安装Xamarin.Firebase.Messaging，还安装所有必要的依赖项。

接下来，编辑**MainActivity.cs**并添加以下 `using` 语句：

```
using Firebase.Messaging;
using Firebase.Iid;
using Android.Util;
```

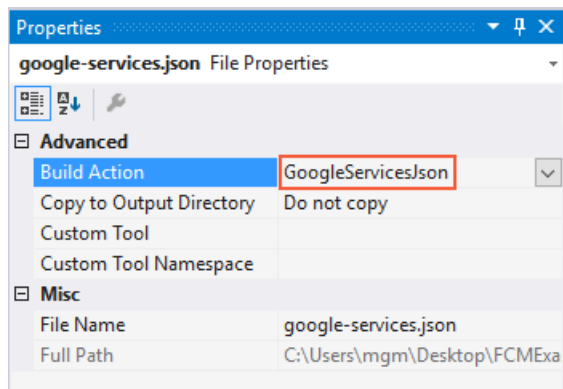
前两个语句进行中的类型**Xamarin.Firebase.Messaging** NuGet 包可供**FCMClient**代码。**Android.Util**添加将用于观察 FMS 的事务的日志记录功能。

添加 Google 服务 JSON 文件

下一步是添加**google-services.json**到你的项目的根目录的文件：

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 复制**google-services.json**到项目文件夹。
2. 添加**google-services.json**向应用程序项目（单击显示所有文件中解决方案资源管理器，右键单击**google-services.json**，然后选择包括在项目）。
3. 选择**google-services.json**中解决方案资源管理器窗口。
4. 在中属性窗格中，设置生成操作到**GoogleServicesJson**：



NOTE

如果**GoogleServicesJson**生成操作不会显示，保存并关闭解决方案，然后重新打开它。

当**google-services.json**添加到项目（并**GoogleServicesJson**生成操作设置），生成过程中提取客户端 ID 和**API 密钥**，然后将这些凭据添加到合并/生成**AndroidManifest.xml**驻留在**obj/Debug/android/AndroidManifest.xml**。此合并进程会自动添加的任何权限和其他 FCM 元素所需的 FCM 服务器的连接。

检查 Google Play Services 并创建通知通道

Google 建议访问 Google Play Services 功能之前的 Android 应用检查是否存在的 Google Play Services APK（有关详细信息，请参阅[检查 Google Play services](#)）。

将首先创建应用程序的 UI 的初始布局。编辑**Resources/layout/Main.axml**并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp">
    <TextView
        android:text=" "
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/msgText"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:padding="10dp" />
</LinearLayout>
```

这 `TextView` 将用于显示指明是否已安装 Google Play 服务的消息。保存对更改 **Main.xml**。

编辑 **MainActivity.cs** 并添加以下实例变量添加到 `MainActivity` 类：

```
public class MainActivity : AppCompatActivity
{
    static readonly string TAG = "MainActivity";

    internal static readonly string CHANNEL_ID = "my_notification_channel";
    internal static readonly int NOTIFICATION_ID = 100;

    TextView msgText;
```

变量 `CHANNEL_ID` 并 `NOTIFICATION_ID` 将在方法中使用 `CreateNotificationChannel`，将添加到 `MainActivity` 稍后在本演练中。

在以下示例中，`OnCreate` 方法将验证 Google Play Services 之前是应用程序尝试使用 FCM 服务是否可用。添加以下方法 `MainActivity` 类：

```
public bool IsPlayServicesAvailable ()
{
    int resultCode = GoogleApiAvailability.Instance.IsGooglePlayServicesAvailable (this);
    if (resultCode != ConnectionResult.Success)
    {
        if (GoogleApiAvailability.Instance.IsUserResolvableError (resultCode))
            msgText.Text = GoogleApiAvailability.Instance.GetErrorString (resultCode);
        else
        {
            msgText.Text = "This device is not supported";
            Finish ();
        }
        return false;
    }
    else
    {
        msgText.Text = "Google Play Services is available.";
        return true;
    }
}
```

此代码检查设备以查看是否已安装 Google Play Services APK。如果未安装，在显示一条消息 `TextBox`，指示用户可以从 Google Play 商店下载 APK（或设备的系统设置中启用它）。

Android 8.0（API 级别 26）或更高版本运行的应用程序必须创建 [通知通道](#) 发布其通知。添加以下方法 `MainActivity` 类将创建通知通道（如有必要）：

```

void CreateNotificationChannel()
{
    if (Build.VERSION.SdkInt < BuildVersionCodes.O)
    {
        // Notification channels are new in API 26 (and not a part of the
        // support library). There is no need to create a notification
        // channel on older versions of Android.
        return;
    }

    var channel = new NotificationChannel(CHANNEL_ID,
                                         "FCM Notifications",
                                         NotificationImportance.Default)
    {
        Description = "Firebase Cloud Messages appear in this channel"
    };

    var notificationManager =
(NotificationManager)GetSystemService(Android.Content.Context.NotificationService);
    notificationManager.CreateNotificationChannel(channel);
}

```

将 `OnCreate` 方法替换为以下代码：

```

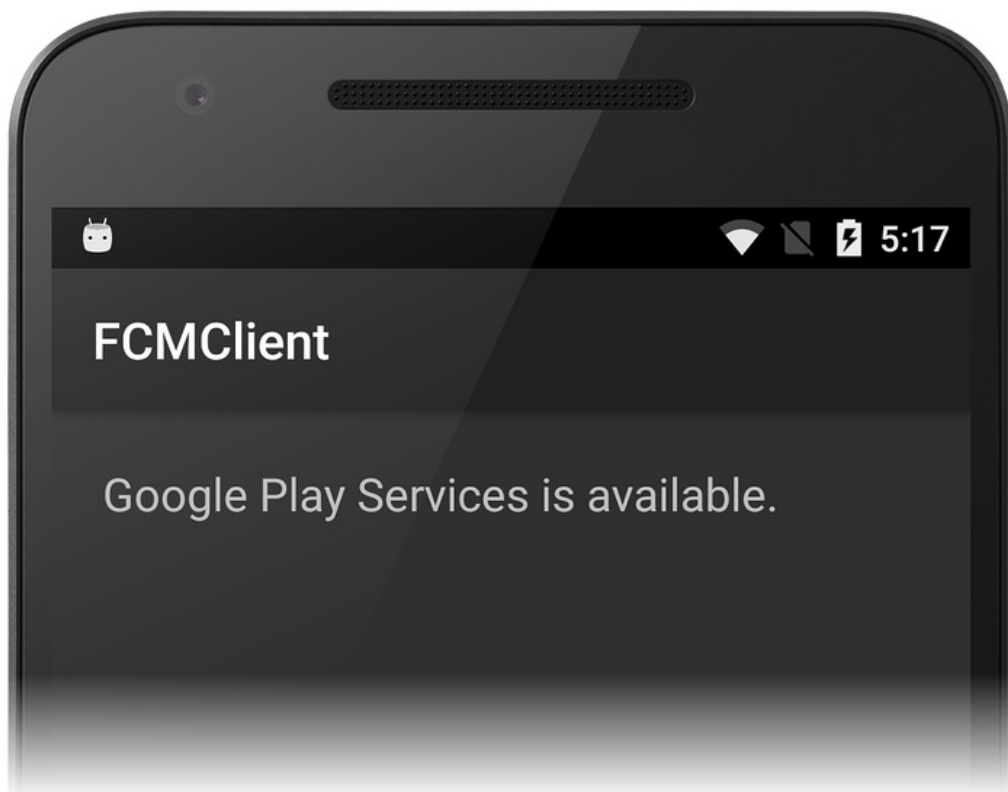
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
    SetContentView (Resource.Layout.Main);
    msgText = FindViewById<TextView> (Resource.Id.msgText);

    IsPlayServicesAvailable ();

    CreateNotificationChannel();
}

```

`IsPlayServicesAvailable` 结束时调用 `OnCreate`，以便 Google Play Services 检查运行每次启动应用。该方法 `CreateNotificationChannel` 被调用，以确保通知通道存在于运行 Android 8 的设备或更高版本。如果你的应用程序 `OnResume` 方法，则应调用 `IsPlayServicesAvailable` 从 `OnResume` 也。完全重新生成并运行应用程序。如果所有配置正确，应看到一个屏幕，如以下屏幕截图所示：



如果没有获得此结果，请验证你的设备上是否安装了 Google Play Services APK (有关详细信息，请参阅[设置 Google Play 服务](#))。此外验证是否已添加 `Xamarin.Google.Play.Services.Base` 打包到你 `FCMClient` 项目，如前面所述。

添加实例 ID 接收器

下一步是添加一项服务，扩展了 `FirebaseInstanceIdService` 来处理旋转、创建和更新 [Firebase 注册令牌](#)。`FirebaseInstanceIdService` 所必需的 FCM，以便能够将消息发送到设备服务。当 `FirebaseInstanceIdService` 服务添加到客户端应用程序中，应用将自动接收 FCM 消息并将其显示为通知，每当应用程序在后台运行。

声明在 Android 清单中的接收方

编辑 `AndroidManifest.xml`，并插入以下 `<receiver>` 元素插入 `<application>` 部分：

```
<receiver
    android:name="com.google.firebase.iid.FirebaseInstanceIdInternalReceiver"
    android:exported="false" />
<receiver
    android:name="com.google.firebase.iid.FirebaseInstanceIdReceiver"
    android:exported="true"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
        <category android:name="${applicationId}" />
    </intent-filter>
</receiver>
```

此 XML 将执行以下操作：

- 声明 `FirebaseInstanceIdReceiver` 实现，提供[唯一标识符](#)针对每个应用实例。此接收器还进行身份验证和授权操作。
- 声明的内部 `FirebaseInstanceIdInternalReceiver` 用于安全地启动服务的实现。

- **应用程序 ID** 存储在 **google-services.json** 已[添加到项目](#)。Xamarin.Android Firebase 绑定将替换令牌 `{{applicationId}}` 应用 id 中; 无需其他代码的客户端应用提供应用程序 id。

`FirebaseInstanceIdReceiver` 是 `WakefulBroadcastReceiver` 接收 `FirebaseInstanceId` 并 `FirebaseMessaging` 事件并将其传递到从派生类 `FirebaseInstanceIdService`。

实现 **Firebase** 实例 ID 服务

由自定义处理向 FCM 注册应用程序的工作 `FirebaseInstanceIdService` 提供服务。 `FirebaseInstanceIdService` 执行以下步骤:

1. 使用**实例 ID API**生成授权客户端应用程序访问 FCM 和应用程序服务器的安全令牌。反过来, 应用将获取**注册令牌**来自 FCM。
2. 如果应用程序服务器有此要求, 将转发到应用服务器的注册令牌。

添加名为的新文件**MyFirebaseIIDService.cs**和其模板代码替换为以下代码:

```
using System;
using Android.App;
using Firebase.Iid;
using Android.Util;

namespace FCMClient
{
    [Service]
    [IntentFilter(new[] { "com.google.firebase.INSTANCE_ID_EVENT" })]
    public class MyFirebaseIIDService : FirebaseInstanceIdService
    {
        const string TAG = "MyFirebaseIIDService";
        public override void OnTokenRefresh()
        {
            var refreshedToken = FirebaseInstanceId.Instance.Token;
            Log.Debug(TAG, "Refreshed token: " + refreshedToken);
            SendRegistrationToServer(refreshedToken);
        }
        void SendRegistrationToServer(string token)
        {
            // Add custom implementation, as needed.
        }
    }
}
```

此服务实现 `OnTokenRefresh` 注册令牌最初创建或更改时调用的方法。当 `OnTokenRefresh` 运行时, 它检索的最新令牌从 `FirebaseInstanceId.Instance.Token` 属性 (它通过 FCM 以异步方式更新)。在此示例中, 以便可以在输出窗口中查看该记录的刷新的令牌:

```
var refreshedToken = FirebaseInstanceId.Instance.Token;
Log.Debug(TAG, "Refreshed token: " + refreshedToken);
```

`OnTokenRefresh` 不经常调用: 它用于更新在下列情况下的令牌:

- 安装或卸载应用程序时。
- 当用户删除应用程序数据。
- 应用程序清除实例 id。
- 当令牌的安全已遭到破坏。

根据 Google**实例 ID**文档, FCM 实例 ID 服务将请求该应用程序刷新其令牌定期 (通常情况下, 每 6 个月)。

`OnTokenRefresh` 此外会调用 `SendRegistrationToAppServer` 关联用户的注册令牌与服务器端的帐户（如果有），由应用程序维护：

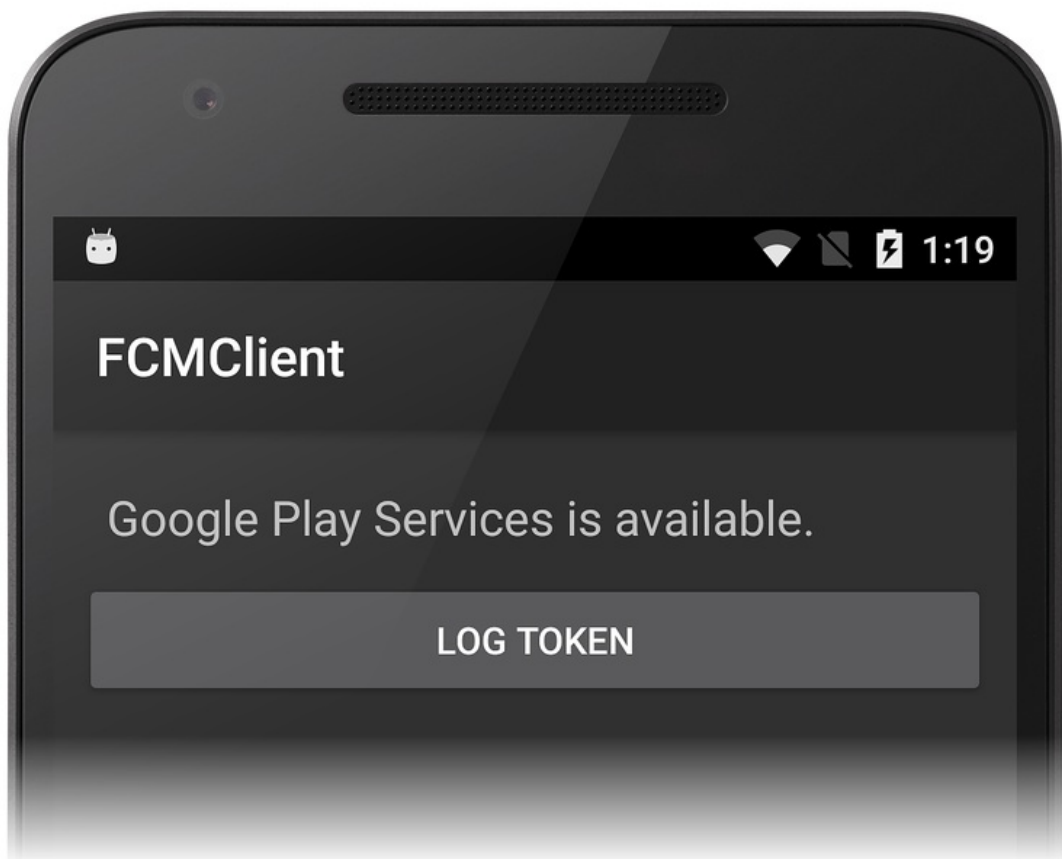
```
void SendRegistrationToAppServer (string token)
{
    // Add custom implementation here as needed.
}
```

因为此实现取决于应用程序服务器的设计，在此示例中提供空的方法体。如果你的应用程序服务器要求 FCM 注册信息，修改 `SendRegistrationToAppServer` 若要用户的 FCM 实例 ID 令牌由您的应用程序维护任何服务器端的帐户相关联。（请注意，该令牌是不透明的客户端应用程序。）

在将令牌发送到应用服务器 `SendRegistrationToAppServer` 应保持指示令牌是否已发送到服务器的布尔值。如果此布尔值为 false，`SendRegistrationToAppServer` 将该令牌发送到应用服务器—否则，该令牌已发送到上一次调用中的应用程序服务器。在某些情况下（如这 `FCMClient` 示例），应用程序服务器不需要令牌；因此，此方法不是此示例所必需的。

实现客户端应用程序代码

现在，接收方服务位于适当位置，可以编写客户端应用程序代码以利用这些服务。在以下部分中，一个按钮添加到 UI，以记录注册令牌（也称为实例 ID 令牌），并且更多的代码添加到 `MainActivity` 若要查看 `Intent` 时从启动应用程序的信息通知：



日志标记

在此步骤中添加的代码仅供演示之—生产客户端应用程序必须无需记录注册令牌。编辑 `Resources/layout/Main.xml` 并添加以下 `Button` 声明后立即 `TextView` 元素：


```
<Button
    android:id="@+id/logTokenButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Log Token" />
```

在 `MainActivity.OnCreate` 方法的末尾添加以下代码：

```
var logTokenButton = FindViewById<Button>(Resource.Id.logTokenButton);
logTokenButton.Click += delegate {
    Log.Debug(TAG, "InstanceID token: " + FirebaseInstanceId.Instance.Token);
};
```

此代码会在当前令牌记录到输出窗口时日志令牌点击按钮。

处理通知意向

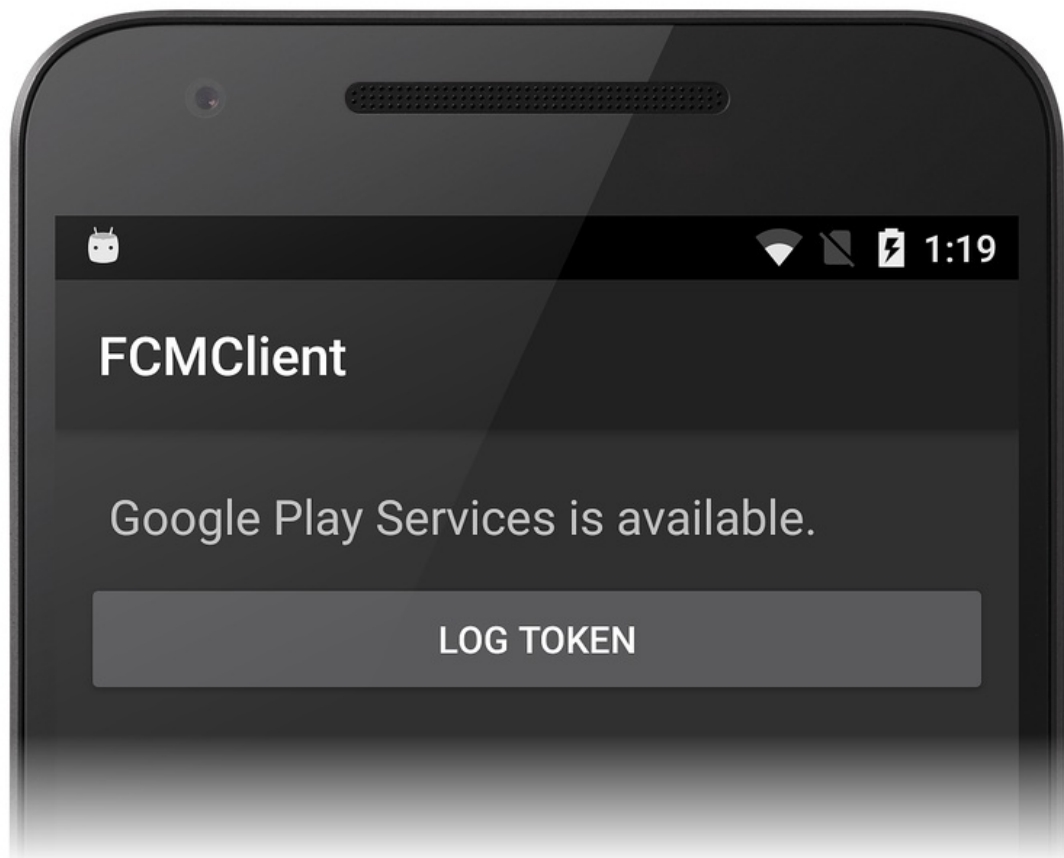
当用户点击通知从颁发 **FCMClient** 随附该通知的任何数据、消息可在 `Intent` 其他功能。编辑 **MainActivity.cs** 并将以下代码添加到顶部 `OnCreate` 方法（在对调用之前 `IsPlayServicesAvailable`）：

```
if (Intent.Extras != null)
{
    foreach (var key in Intent.Extras.KeySet())
    {
        var value = Intent.Extras.GetString(key);
        Log.Debug(TAG, "Key: {0} Value: {1}", key, value);
    }
}
```

应用程序的启动器 `Intent` 当用户点击其通知消息，因此此代码将记录中的任何随附的数据时触发 `Intent` 到输出窗口。具有不同 `Intent` 必须在激发 `click_action` 通知消息的字段必须设置为该 `Intent`（启动器 `Intent` 时不使用 `click_action` 指定）。

背景通知

生成并运行 **FCMClient** 应用。日志令牌显示按钮：



点击日志令牌按钮。IDE 输出窗口中，应显示如下所示的消息：

```
Output
Show output from: Debug
11-18 12:21:34.694 D/OpenGLRenderer(17289): Use EGL_SWAP_BEHAVIOR_PRESERVED: true
11-18 12:21:34.754 I/Adreno-EGL(17289): <qeglDrvAPI_eglInitialize:379>: QUALCOMM Build: 10/21/15, 369a2ea, I96aee987eb
11-18 12:21:34.759 I/OpenGLRenderer(17289): Initialized EGL, version 1.4
11-18 12:21:41.092 D/Mono (17289): Assembly Ref addrf FCMClient[0xab4bdea0] -> Xamarin.Firebase.Iid[0xab4be320]: 2
11-18 12:21:41.098 D/Mono (17289): DllImport searching in: '__Internal' ('(null)').
11-18 12:21:41.098 D/Mono (17289): Searching for 'java_interop_jnienv_call_static_int_method_a'.
11-18 12:21:41.098 D/Mono (17289): Probing 'java_interop_jnienv_call_static_int_method_a'.
11-18 12:21:41.098 D/Mono (17289): Found as 'java_interop_jnienv_call_static_int_method_a'.
11-18 12:21:41.099 D/MainActivity(17289): InstanceID token: {"token": "cFypG01m80s:APA91bEETmrwFT#kpscX3_qpYXo3NE_DunTB8csHnuDqqMowL"}
```

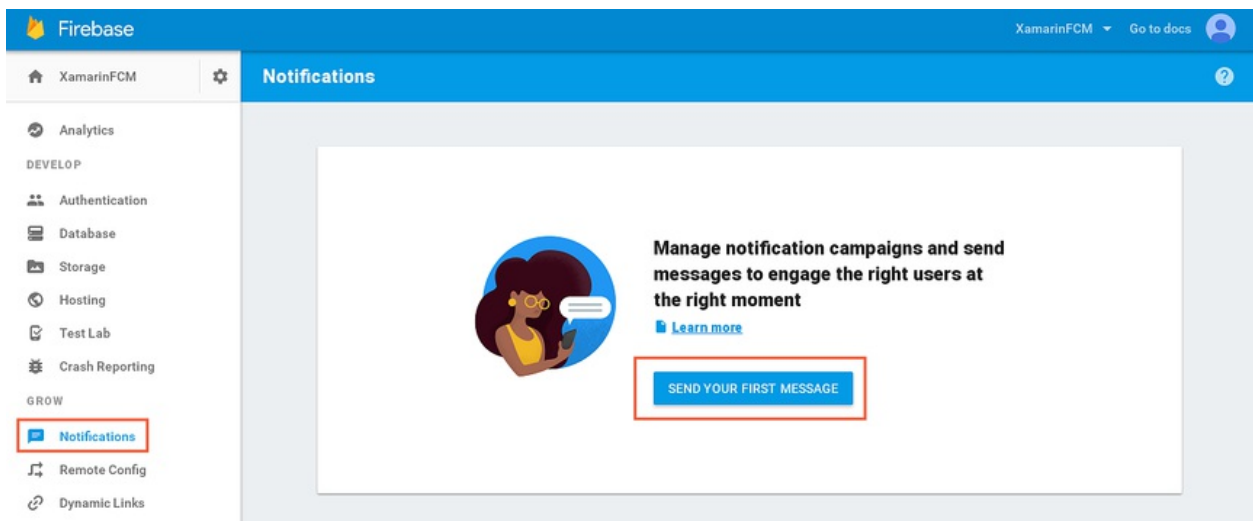
带标记的一长串令牌是实例 ID 令牌，然后将其粘贴到 Firebase 控制台-选择，然后将此字符串复制到剪贴板。如果看不到的实例 ID 令牌，将以下行添加到顶部 `OnCreate` 方法以验证 `google-services.json` 已正确分析：

```
Log.Debug(TAG, "google app id: " + GetString(Resource.String.google_app_id));
```

`google_app_id` 记录到输出窗口的值应与匹配 `mobilesdk_app_id` 中记录的值 `google-services.json`。

发送一条消息

登录到 [Firebase 控制台](#)，选择你的项目，单击 **通知**，然后单击 **发送第一条消息**：



上 **Compose 消息** 页上, 输入消息文本, 然后选择单个设备。从 IDE 输出窗口复制的实例 ID 令牌并将其粘贴到 **FCM 注册令牌** Firebase 控制台中的字段:

Message text

Message label (optional) ?

Delivery date ?

Send Now ▾

Target

☐ User segment ☐ Topic ☒ Single device

FCM registration token ?

Conversion events ?

▾

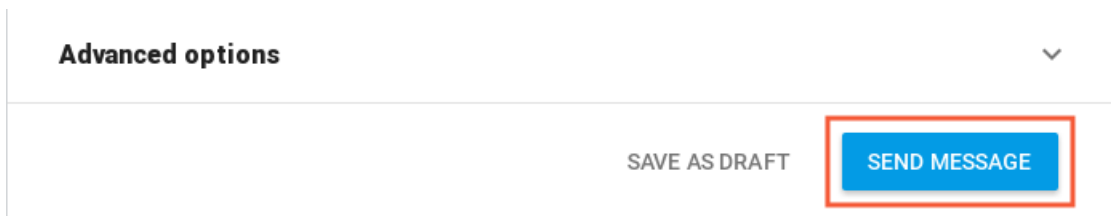
Advanced options

▾

SAVE AS DRAFT

SEND MESSAGE

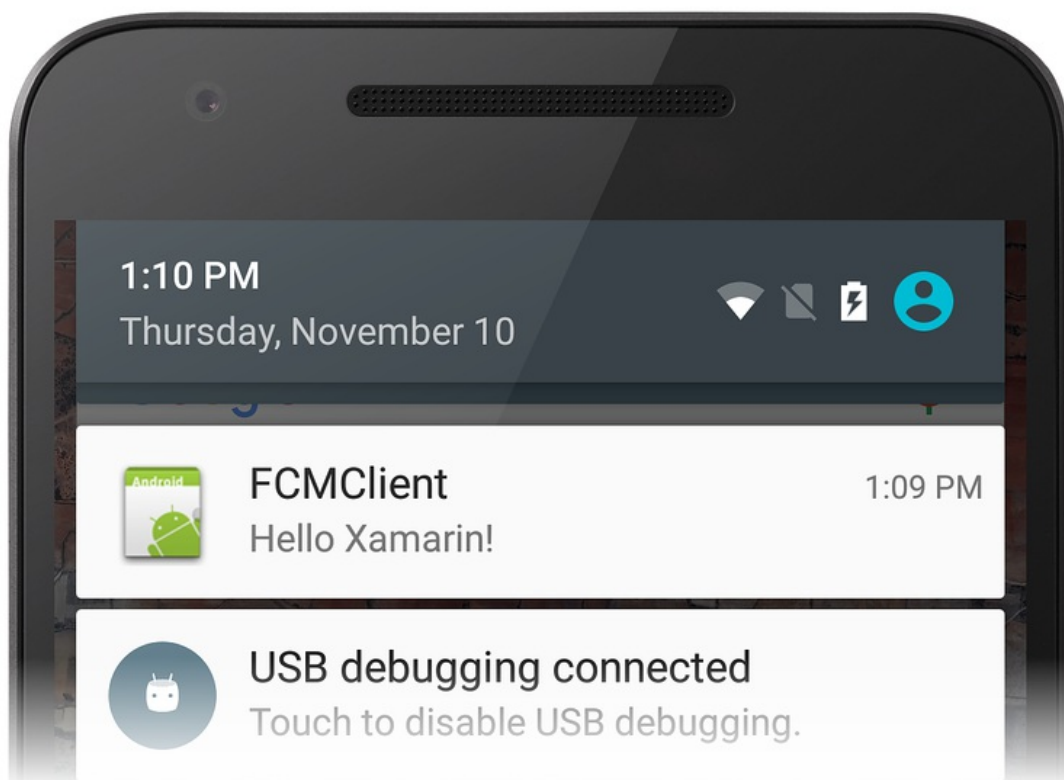
Android 设备 (或模拟器) 上, 通过点击 Android 后台应用程序概述按钮和触摸主屏幕。在设备准备就绪后, 单击 **发送消息** Firebase 控制台中:



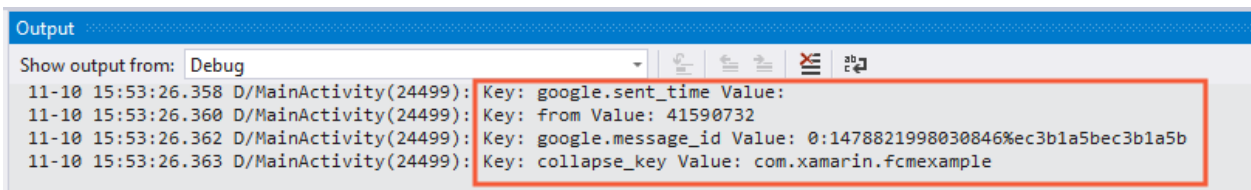
当评审消息显示对话框中，单击发送。通知图标应出现在通知区域中的设备（或仿真器）：



打开通知图标以查看该消息。通知消息应为完全什么已键入消息正文 Firebase 控制台中的字段：



点击该通知图标即可启动**FCMClient**应用。Intent 发送到 extras **FCMClient** IDE 输出窗口中列出：



```
Output
Show output from: Debug
11-10 15:53:26.358 D/MainActivity(24499): Key: google.sent_time Value:
11-10 15:53:26.360 D/MainActivity(24499): Key: from Value: 41590732
11-10 15:53:26.362 D/MainActivity(24499): Key: google.message_id Value: 0:1478821998030846%ec3b1a5bec3b1a5b
11-10 15:53:26.363 D/MainActivity(24499): Key: collapse_key Value: com.xamarin.fcmexample
```

在此示例中，`from`键设置为应用程序的 Firebase 项目编号（在此示例中，`41590732`），和`collapse_key`设置为其包名称（`com.xamarin.fcmexample`）。如果您不会收到一条消息，请尝试删除FCMClient设备（或仿真器）上的应用，然后重复上述步骤。

NOTE

如果强制关闭应用，FCM 将停止将通知传递。Android 可阻止后台服务广播无意中或不必要地启动已停止的应用程序的组件。（有关此行为的详细信息，请参阅[启动已停止的应用程序上的控件](#)。）出于此原因，则务必手动卸载该应用每次运行它，并在调试会话中停止—这会强制 FCM，生成新令牌，以便将继续接收消息。

添加自定义默认通知图标

在上一示例中，通知图标设置为应用程序图标。下面的 XML 配置通知的自定义的默认图标。Android 显示所有通知消息，其中的通知图标未显式设置此自定义的默认图标。

若要添加自定义默认通知图标，添加到应用图标资源/`drawable`目录中，编辑`AndroidManifest.xml`，并插入以下 `<meta-data>` 元素插入 `<application>` 部分：

```
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_stat_ic_notification" />
```

在此示例中，通知图标，位于资源/`drawable/ic_stat_ic_notification.png`将用作自定义默认通知图标。如果未在中配置自定义的默认图标`AndroidManifest.xml`和无图标设置通知有效负载中，Android 作为通知图标（如上面的通知图标屏幕截图中所示）将使用应用程序图标。

处理主题的消息

到目前为止编写的代码处理注册令牌，并将远程通知功能添加到应用程序。下一步的示例将添加代码，可侦听主题消息并将其转发到该用户为远程通知。主题的消息是发送到某个特定主题订阅的一个或多个设备的 FCM 消息。有关主题的消息的详细信息，请参阅[消息传递主题](#)。

订阅主题

编辑`Resources/layout/Main.axml`并添加以下 `Button` 紧前面的声明 `Button` 元素：

```
<Button
    android:id="@+id/subscribeButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp"
    android:text="Subscribe to Notifications" />
```

此 XML 将添加通知订阅到布局的按钮。编辑`MainActivity.cs`并将以下代码添加到末尾 `OnCreate` 方法：

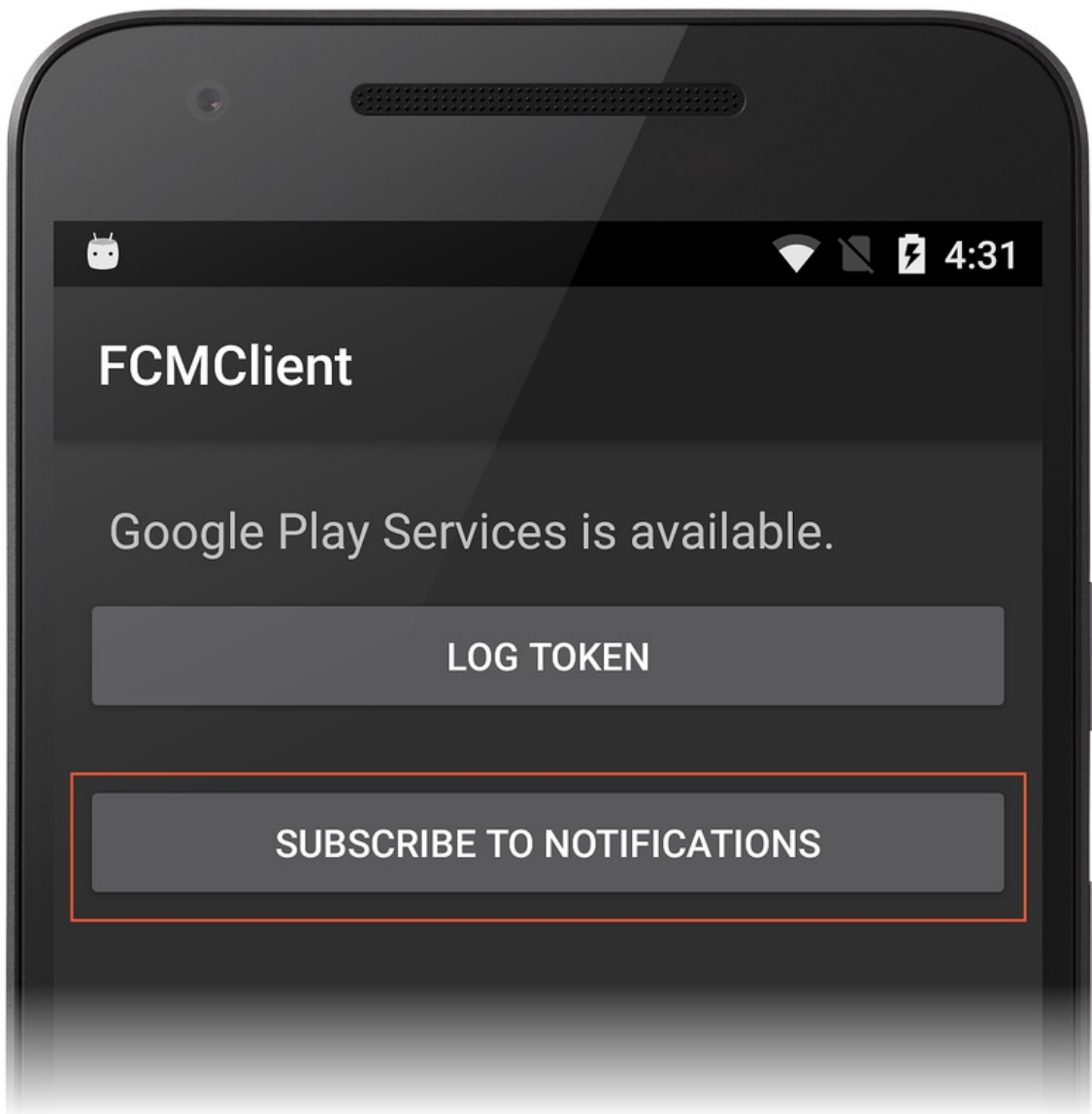
```
var subscribeButton = FindViewById<Button>(Resource.Id.subscribeButton);
subscribeButton.Click += delegate {
    FirebaseMessaging.Instance.SubscribeToTopic("news");
    Log.Debug(TAG, "Subscribed to remote notifications");
};
```

此代码定位通知订阅布局中的按钮并将其 click 处理程序分配给调用的代码

`FirebaseMessaging.Instance.SubscribeToTopic`，并在已订阅的主题中，传递 `新闻`。当用户点击 **Subscribe** 按钮，该应用程序订阅 `新闻` 主题。在以下部分中，`新闻` 将从 Firebase 控制台通知 GUI 发送主题消息。

将发送主题消息

卸载应用、重新生成，并再次运行。单击订阅通知按钮：

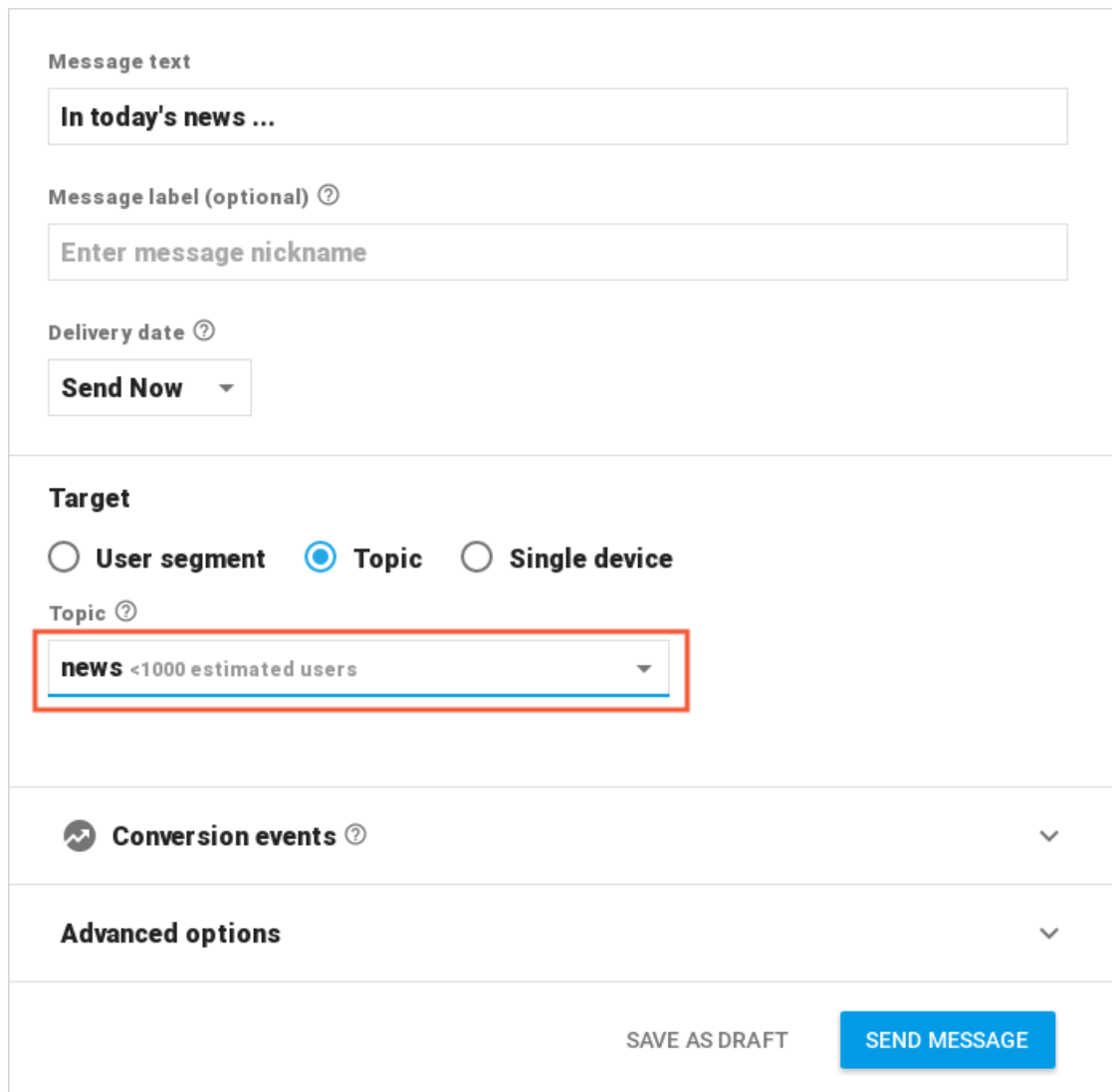


如果已成功订阅了该应用程序后，应会看到主题同步成功在 IDE 中输出窗口：

```
Output
Show output from: Debug
11-10 14:13:21.382 D/Mono (20411): Searching for 'java_interop_jnienv_call_static_int_method_a'.
11-10 14:13:21.382 D/Mono (20411): Probing 'java_interop_jnienv_call_static_int_method_a'.
11-10 14:13:21.382 D/Mono (20411): Found as 'java_interop_jnienv_call_static_int_method_a'.
11-10 14:13:21.383 D/MyFirebaseIIDService(20411): Refreshed token: ebpPYcl76rQ:APA91bG_CnlhMm42zeb_Hh4UwZ1R7ih6xqkMQCoFsQ8V0gF5G1Fq4nAazD3x6pgMRU2cgyFQ857K0SLr0YKvkv
11-10 14:13:22.435 D/MainActivity(20411): InstanceID token: ebpPYcl76rQ:APA91bG_CnlhMm42zeb_Hh4UwZ1R7ih6xqkMQCoFsQ8V0gF5G1Fq4nAazD3x6pgMRU2cgyFQ857K0SLr0YKvkv_T9G7
11-10 14:13:27.779 D/Mono (20411): Assembly Ref addrf FCMClient[0xab4bdd20] -> Xamarin.Firebase.Messaging[0xab4be200]: 2
11-10 14:13:27.786 D/MainActivity(20411): Subscribed to remote notifications
11-10 14:13:28.219 D/FirebaseInstanceId(20411): topic sync succeeded
11-10 14:13:40.511 D/Mono (20411): Assembly Ref addrf Xamarin.Firebase.Messaging[0xab4be200] -> Xamarin.Firebase.Iid[0xab4be1a0]: 3
```

使用以下步骤将发送主题消息：

1. 在 Firebase 控制台中，单击**新消息**。
2. 上**Compose 消息**页上，输入消息文本，然后选择主题。
3. 在中主题下拉菜单，选择内置主题**新闻**：



Message text

In today's news ...

Message label (optional) ?

Enter message nickname

Delivery date ?

Send Now

Target

☐ User segment ☒ Topic ☐ Single device

Topic ?

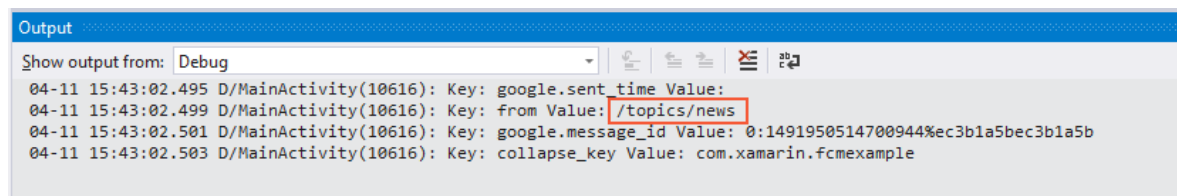
news <1000 estimated users

Conversion events ?

Advanced options

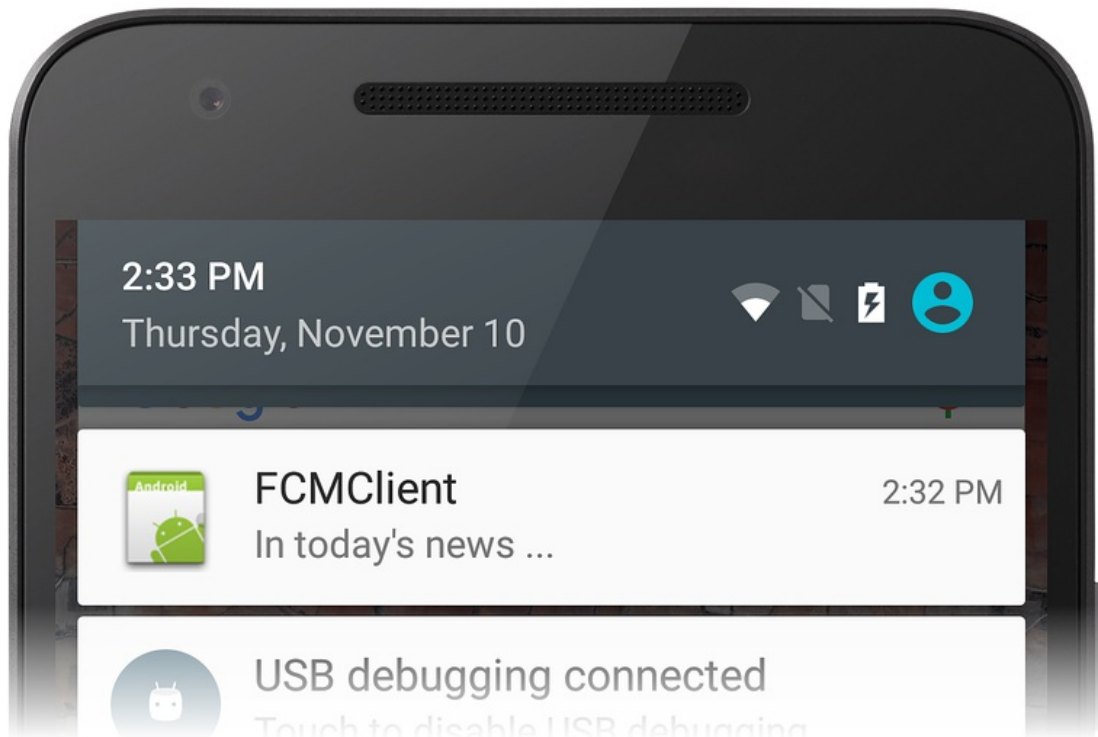
SAVE AS DRAFT SEND MESSAGE

4. Android 设备（或模拟器）上，通过点击 Android 后台**应用程序概述**按钮和触摸主屏幕。
5. 在设备准备就绪后，单击**发送消息** Firebase 控制台中。
6. 检查 IDE 输出窗口以查看 **/主题/新闻**的日志输出：



```
04-11 15:43:02.495 D/MainActivity(10616): Key: google.sent_time Value:
04-11 15:43:02.499 D/MainActivity(10616): Key: from Value: /topics/news
04-11 15:43:02.501 D/MainActivity(10616): Key: google.message_id Value: 0:1491950514700944%ec3b1a5bec3b1a5b
04-11 15:43:02.503 D/MainActivity(10616): Key: collapse_key Value: com.xamarin.fcmexample
```

在输出窗口中会出现此消息，通知图标应也出现在 Android 设备上的通知区域中。打开通知图标以查看该主题的消息：



如果您不会收到一条消息，请尝试删除**FCMClient**设备（或仿真器）上的应用，然后重复上述步骤。

前景色通知

若要在 foregrounded 应用中接收通知，必须实现 `FirebaseMessagingService`。此服务，还需要用于接收数据负载和发送上游消息。下面的示例演示了如何以扩展的服务 `FirebaseMessagingService` 生成的应用将能够在前台运行时处理远程通知。

实现 `FirebaseMessagingService`

`FirebaseMessagingService` 服务负责接收和处理从 Firebase 消息。每个应用必须子类，此类型和重写 `onMessageReceived` 以处理传入的消息。当应用位于前台，`onMessageReceived` 回调将始终处理该消息。

NOTE

应用程序只能有 10 秒，用来处理传入 Firebase 云消息。所花时间超过这应计划使用一个库，类似于后台执行任何工作 [Android 作业计划程序](#) 或 [Firebase 作业调度程序](#)。

添加名为的新文件 **MyFirebaseMessagingService.cs** 和其模板代码替换为以下代码：


```

using System;
using Android.App;
using Android.Content;
using Android.Media;
using Android.Util;
using Firebase.Messaging;

namespace FCMClient
{
    [Service]
    [IntentFilter(new[] { "com.google.firebase.MESSAGING_EVENT" })]
    public class MyFirebaseMessagingService : FirebaseMessagingService
    {
        const string TAG = "MyFirebaseMsgService";
        public override void OnMessageReceived(RemoteMessage message)
        {
            Log.Debug(TAG, "From: " + message.From);
            Log.Debug(TAG, "Notification Message Body: " + message.GetNotification().Body);
        }
    }
}

```

请注意，`MESSAGING_EVENT` 必须声明意向筛选器，以便新 FCM 消息定向到 `MyFirebaseMessagingService`：

```
[IntentFilter(new[] { "com.google.firebase.MESSAGING_EVENT" })]
```

当客户端应用程序收到来自 FCM，一条消息时 `OnMessageReceived` 提取从传入的消息内容 `RemoteMessage` 对象通过调用其 `GetNotification` 方法。接下来，以便可以在 IDE 输出窗口中查看该日志的消息内容：

```

var body = message.GetNotification().Body;
Log.Debug(TAG, "Notification Message Body: " + body);

```

NOTE

如果在 IDE 中设置断点 `FirebaseMessagingService`，你在调试会话可能或不可能由于 FCM 消息的传递会命中这些断点。

发送另一条消息

卸载该应用、重新生成，它再次运行，并请执行以下步骤将发送另一条消息：

1. 在 Firebase 控制台中，单击**新消息**。
2. 上**Compose 消息**页上，输入消息文本，然后选择单个设备。
3. 从 IDE 输出窗口复制的标记字符串并将其粘贴到**FCM 注册令牌** Firebase 控制台与以前的字段。
4. 请确保在应用运行在前台，然后单击**发送消息** Firebase 控制台中：

Message text

Hello Again!

Message label (optional) ?

Enter message nickname

Delivery date ?

Send Now

Target

☐ User segment
 ☐ Topic
 ☒ Single device

FCM registration token ?

:hyPZ8lsIs_NKEX832ZkVRRhMpX6rO4NHXAuAKh

Conversion events ?

Advanced options

SAVE AS DRAFT

SEND MESSAGE

5. 当评审消息显示对话框中，单击发送。

6. 传入的消息记录到 IDE 输出窗口中：

```

Output
Show output from: Debug
11-18 14:54:15.991 D/MyFirebaseMsgService(27478): From: 415913540732
11-18 14:54:15.994 D/MyFirebaseMsgService(27478): Notification Message Body: Hello Again!
  
```

添加本地通知发件人

在剩余的示例中，传入的 FCM 消息将转换为本地应用程序在前台运行时启动的通知。编辑 **MyFirebaseMessageService.cs** 并添加以下 `using` 语句：

```

using FCMClient;
using System.Collections.Generic;
  
```

添加以下方法 `MyFirebaseMessagingService`：

```

void SendNotification(string messageBody, IDictionary<string, string> data)
{
    var intent = new Intent(this, typeof(MainActivity));
    intent.AddFlags(ActivityFlags.ClearTop);
    foreach (var key in data.Keys)
    {
        intent.PutExtra(key, data[key]);
    }

    var pendingIntent = PendingIntent.GetActivity(this,
                                                MainActivity.NOTIFICATION_ID,
                                                intent,
                                                PendingIntentFlags.OneShot);

    var notificationBuilder = new NotificationCompat.Builder(this, MainActivity.CHANNEL_ID)
        .SetSmallIcon(Resource.Drawable.ic_stat_ic_notification)
        .SetContentTitle("FCM Message")
        .SetContentText(messageBody)
        .SetAutoCancel(true)
        .SetContentIntent(pendingIntent);

    var notificationManager = NotificationManagerCompat.From(this);
    notificationManager.Notify(MainActivity.NOTIFICATION_ID, notificationBuilder.Build());
}

```

为了区分从后台通知此通知，此代码将标记通知图标不同于应用程序图标。将文件添加 `ic_stat_ic_notification.png` 到资源/`drawable` 并将其包含在 `FCMClient` 项目。

`SendNotification` 方法使用 `NotificationCompat.Builder` 若要创建通知，和 `NotificationManagerCompat` 用于启动该通知。通知持有 `PendingIntent`，将允许用户打开应用并查看传入的字符串的内容 `messageBody`。有关详细信息 `NotificationCompat.Builder`，请参阅[本地通知](#)。

调用 `SendNotification` 方法末尾处 `OnMessageReceived` 方法：

```

public override void OnMessageReceived(RemoteMessage message)
{
    Log.Debug(TAG, "From: " + message.From);

    var body = message.GetNotification().Body;
    Log.Debug(TAG, "Notification Message Body: " + body);
    SendNotification(body, message.Data);
}

```

这些更改会导致 `SendNotification` 将运行该应用位于前台，而该通知会在通知区域中收到通知。

应用时在后台消息的负载将确定如何处理该消息：

- **通知**—消息将发送到系统托盘。将其中显示本地通知。当用户点击通知时将启动该应用程序。
- **数据**—消息将由处理 `OnMessageReceived`。
- **这两**—将直接发送到系统托盘中的通知和数据负载的消息。数据有效负载应用启动时，将出现在 `Extras` 的 `Intent` 用于启动应用程序。

在此示例中，如果应用在后台运行 `SendNotification` 如果消息具有数据负载将运行。否则，将启动后台通知（在本演练前面所示）。

发送最后一条消息

卸载应用程序，重新生成它，它再次运行，然后使用以下步骤将发送最后一条消息：

1. 在 Firebase 控制台中，单击**新消息**。
2. 上**Compose 消息**页上，输入消息文本，然后选择单个设备。

3. 从 IDE 输出窗口复制的标记字符串并将其粘贴到**FCM 注册令牌** Firebase 控制台与以前的字段。
4. 请确保在应用运行在前台，然后单击**发送消息** Firebase 控制台中：

Message text

Message label (optional) ?


Delivery date ?

Send Now ▾

Target

☐ User segment ☐ Topic ☒ Single device

FCM registration token ?

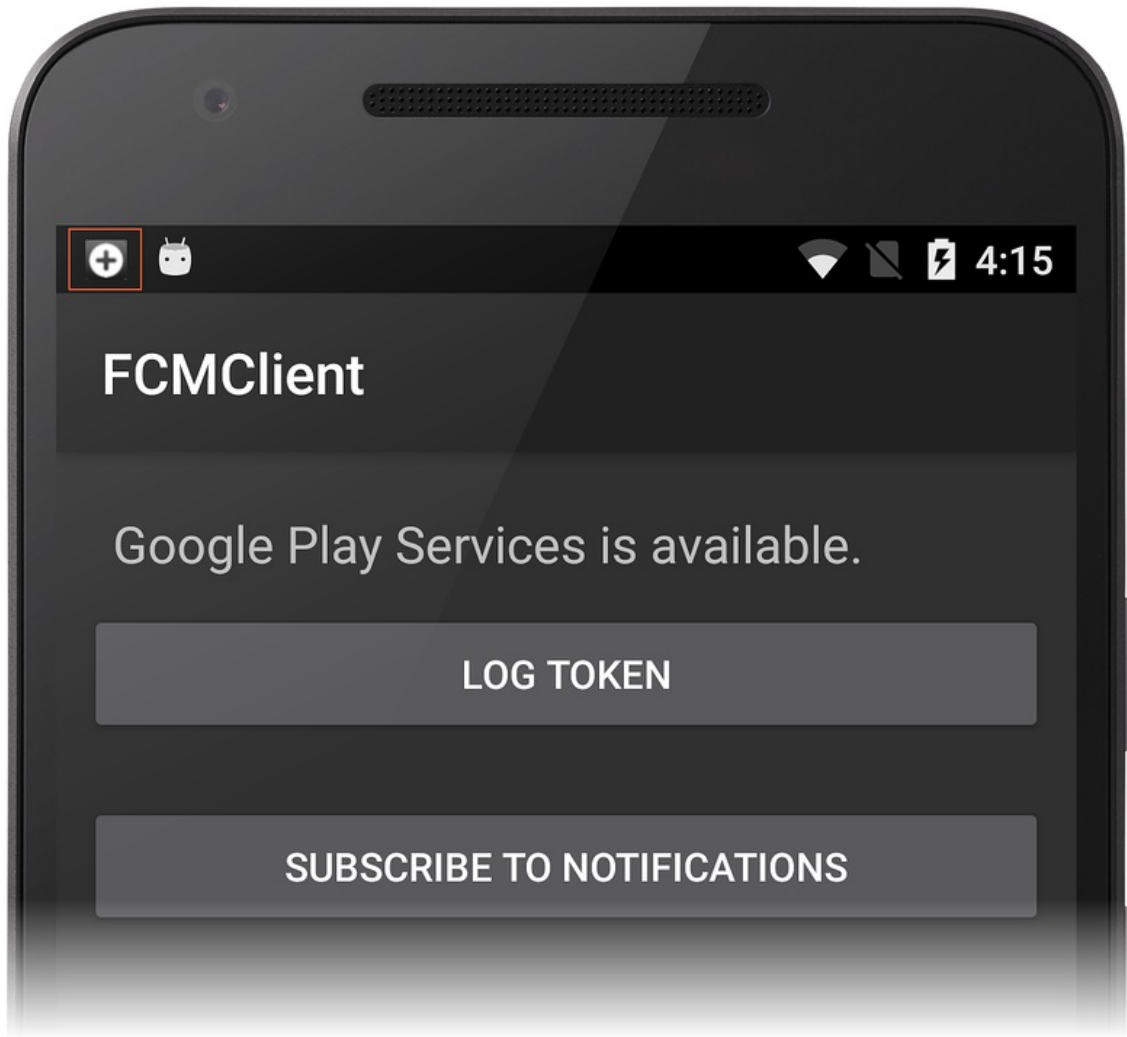
 **Conversion events ?** ▾

Advanced options ▾

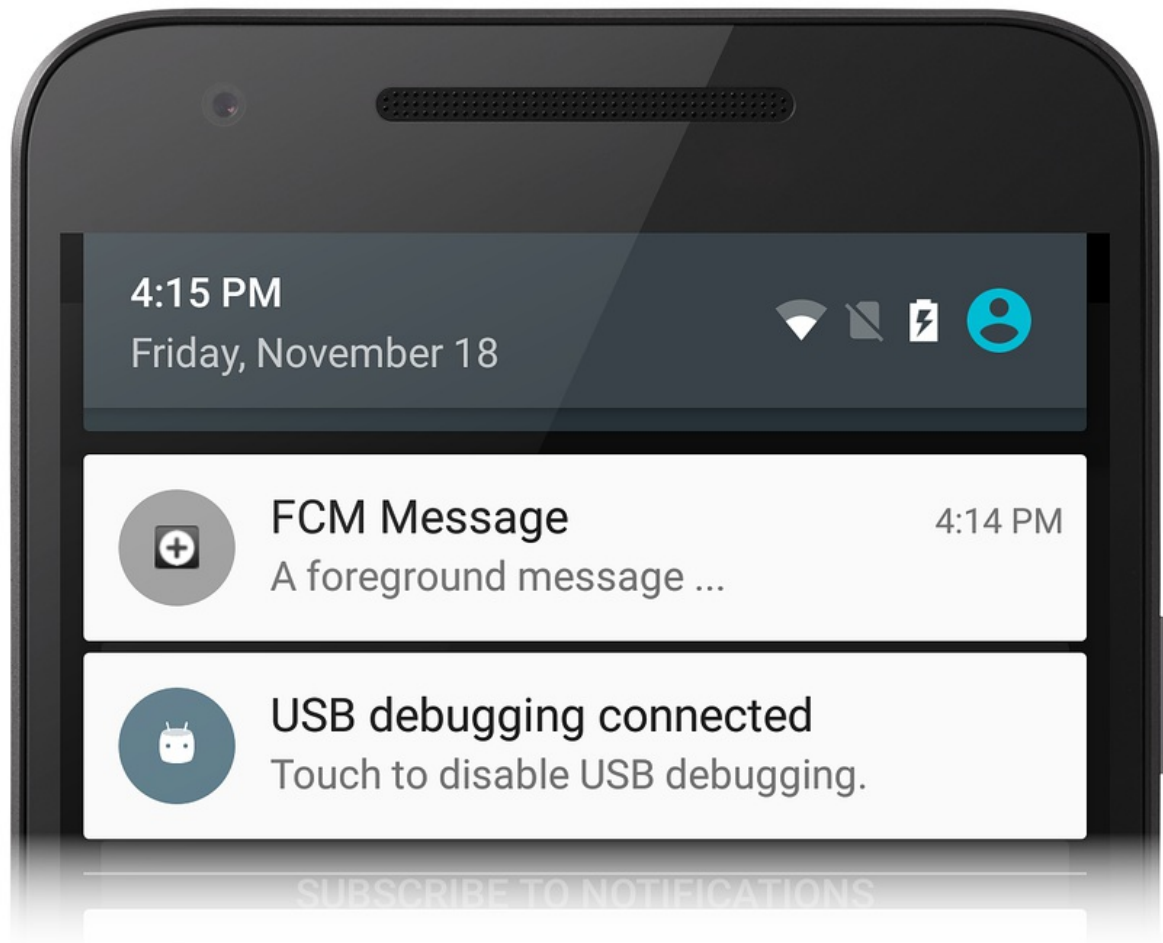
SAVE AS DRAFT

SEND MESSAGE

这一次，在输出窗口中记录了该消息还打包在一个新的通知-通知图标出现在通知栏中，应用程序在前台运行时：



当您打开的通知时，应看到从 Firebase 控制台通知 GUI 发送最后一条消息：



来自 FCM 断开连接

若要取消订阅从主题，请调用[UnsubscribeFromTopic](#)方法[FirebaseMessaging](#)类。例如，若要取消订阅_新闻_主题，若要更早版本，**Unsubscribe**按钮可以添加到具有以下处理程序代码的布局：

```
var unsubscribeButton = FindViewById<Button>(Resource.Id.unsubscribeButton);
unsubscribeButton.Click += delegate {
    FirebaseMessaging.Instance.UnsubscribeFromTopic("news");
    Log.Debug(TAG, "Unsubscribed from remote notifications");
};
```

若要取消注册从 FCM 完全设备，删除的实例 ID，通过调用[DeleteInstanceId](#)方法[FirebaseInstanceId](#)类。例如：

```
FirebaseInstanceId.Instance.DeleteInstanceId();
```

此方法调用将删除实例 ID 和与之关联的数据。因此，将停止定期发送 FCM 数据复制到设备。

疑难解答

以下描述问题和解决方法使用 Xamarin.Android 使用 Firebase Cloud Messaging 时可能出现。

未初始化 **FirebaseApp**

在某些情况下，可能会看到此错误消息：

```
Java.Lang.IllegalStateException: Default FirebaseApp is not initialized in this process  
Make sure to call FirebaseApp.initializeApp(Context) first.
```

这是一个可以通过清除解决方案并重新生成项目解决的已知的问题 ([生成 > 清理解决方案](#), [生成 > 重新生成解决方案](#))。有关详细信息, 请参阅此[论坛讨论](#)。

总结

本演练详细 Xamarin.Android 应用程序中实现 Firebase Cloud Messaging 远程通知的步骤。介绍了如何安装所需的包所需的 FCM 的通信, 并介绍了如何配置 Android 清单的 FCM 服务器访问。它提供了示例代码, 说明了如何检查存在的 Google Play 服务。它演示了如何实现与 FCM 注册令牌, 协商实例 ID 侦听器服务, 并且它说明了此代码将应用程序在后台运行时如何创建背景通知。本文介绍了如何订阅主题消息, 并提供了用于接收和应用程序在前台运行时显示远程通知的消息侦听器服务的实现示例。

相关链接

- [FCMNotifications \(示例\)](#)
- [Firebase 云消息传送](#)
- [关于 FCM 消息](#)

Google 云消息传送

2018/10/26 • • [Edit Online](#)

Google Cloud Messaging (GCM) 是一种便于移动应用和服务器应用程序之间的消息传送的服务。此文章概述的 GCM 的工作原理, 并介绍如何配置 Google 服务, 使您的应用程序可以使用 GCM。



本主题提供的 Google Cloud Messaging 将您的应用程序和应用程序服务器之间的消息的路由概述和它提供的分步过程, 以便您的应用程序可以使用 GCM 服务获取的凭据。

NOTE

已被取代 GCM [Firebase Cloud Messaging \(FCM\)](#)。GCM 服务器和客户端 Api [已弃用](#) 将不再提供为 2019 年 4 月 11 日推出。

概述

Google Cloud Messaging (GCM) 是处理发送、路由和服务器应用程序和移动客户端应用程序之间的消息队列服务。一个 *客户端应用* 是一种启用 GCM 的应用, 设备上运行。*应用程序服务器* (由你或贵公司提供) 是客户端应用程序通过 GCM 与通信的启用 GCM 的服务器:



使用 GCM, 应用程序服务器可以向单个设备、一组设备或多个订阅到主题的设备发送消息。客户端应用程序可以使用 GCM 来订阅到下游消息从应用程序服务器 (例如, 若要接收远程通知)。此外, GCM 使得客户端应用以将上游消息发送回应用程序服务器。

有关为 GCM 实现应用程序服务器的信息, 请参阅[关于 GCM 连接服务器](#)。

Google 云消息传送在操作中

当下游消息从应用程序服务器发送到客户端应用程序时, 应用程序服务器将发送到消息 *GCM 连接服务器*, GCM 连接服务器, 反过来, 将消息转发到正在运行客户端应用程序的设备。可以通过 HTTP 发送消息或 [XMPP](#) (可扩展消息和状态显示协议)。因为客户端应用程序不始终连接或运行的 GCM 连接服务器排入队列, 并将存储消息, 将它们发送到客户端应用程序重新连接并变得可用。同样, GCM 排入队列上游消息从客户端应用程序到应用服务器, 如果将应用程序服务器不可用。

GCM 使用下面的凭据来标识应用程序服务器和客户端应用程序, 并使用这些凭据授权通过 GCM 消息事务:

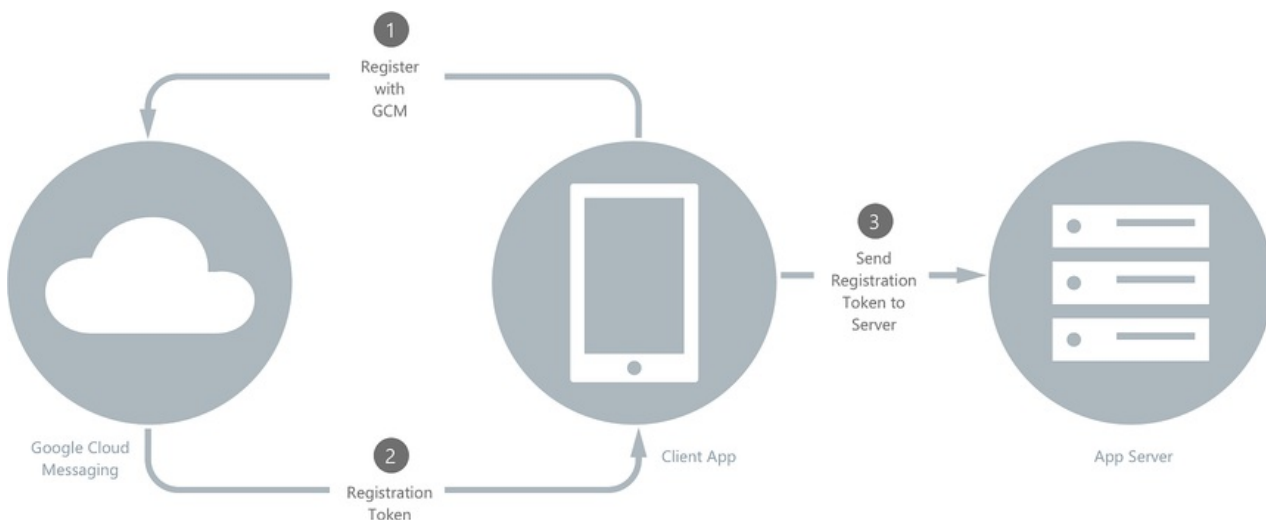
- **API 密钥** – API 密钥允许应用程序服务器访问的 Google 服务;GCM 使用此密钥进行身份验证应用程序服务器。可以使用 GCM 服务之前,你必须首先获取的 API 密钥[Google Developer Console](#)通过创建项目。API 密钥应保持安全;有关如何保护你的 API 密钥的详细信息,请参阅[的最佳做法使用 API 密钥安全地](#)。
- **发件人 ID** – 发件人 ID 授权客户端应用程序到应用服务器-它是标识允许将消息发送到客户端应用程序的应用程序服务器的唯一编号。发件人 ID 也是你的项目编号;注册你的项目时,可以从 Google 开发人员控制台获取的发件人 ID。
- **注册令牌** – 注册令牌是在客户端应用程序上的给定设备的 GCM 标识。在运行时生成的注册令牌-时它首先向 GCM 注册的设备上运行时,您的应用程序接收的注册令牌。注册令牌授权客户端应用程序(在该特定设备上运行)的实例以从 GCM 中接收消息。
- **应用程序 ID** – 的客户端应用(独立于任何给定的设备)注册以从 GCM 中接收消息的标识。在 Android 上,应用程序 ID 是在中记录的包名称 **AndroidManifest.xml**, 如 `com.xamarin.gcmexample`。

设置了 [Google Cloud Messaging](#) (在本指南的后面介绍) 提供了用于创建项目并生成这些凭据的详细的说明。

以下部分介绍当客户端应用与应用程序服务器通过 GCM 通信时,如何使用这些凭据。

注册到 GCM

消息传送进行之前,首先必须向 GCM 注册设备上安装的客户端应用程序。客户端应用程序必须完成下面的关系图中显示的注册步骤:



1. 客户端应用程序与 GCM 获取注册令牌,将发件人 ID 传递到 GCM。
2. GCM 返回到客户端应用程序注册令牌。
3. 客户端应用程序将转发到应用服务器的注册令牌。

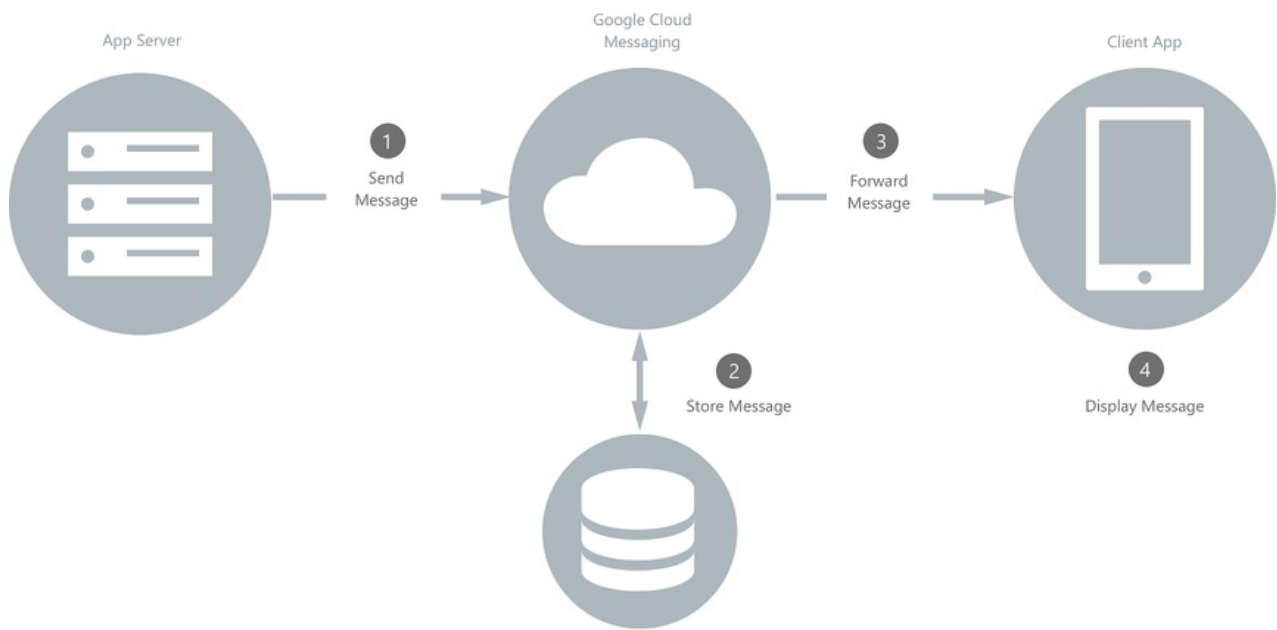
应用程序服务器将缓存与客户端应用程序的后续通信的注册令牌。(可选)在应用程序服务器可以将确认发送回客户端应用程序,以指示接收到的注册令牌。发生此握手后,客户端应用程序可以接收消息(或将消息发送到)的应用程序服务器。

当客户端应用程序不再想要从应用服务器接收消息时,它可以为要删除的注册令牌的应用程序服务器发送请求。如果客户端应用程序接收主题消息(本文稍后所述),它可以取消订阅从主题。从设备上卸载客户端应用程序,GCM 会检测到这,自动通知要删除的注册令牌的应用程序服务器。

Google[注册客户端应用](#)介绍了注册过程中更多详细信息;其中介绍了取消注册和取消订阅,并且卸载客户端应用程序时,它描述的注销过程。

下游消息传送

当应用程序服务器将下游消息发送到客户端应用程序时,它遵循下图所示的步骤:



1. 应用程序服务器将消息发送到 GCM。
2. 如果客户端设备不可用，在 GCM 服务器以在稍后传输队列中存储消息。
3. 当客户端设备可用时，GCM 将消息发送到该设备上的客户端应用。
4. 客户端应用程序从 GCM 中接收消息，并相应地对其进行处理。例如，如果消息是远程通知，它是向用户显示。

在此消息传递方案中（其中应用服务器发送一条消息到单个客户端应用），消息可以是长度最多为 4 kB。

有关详细信息（包括代码示例）接收在 Android 上的下游 GCM 消息，请参阅[远程通知](#)。

主题的消息传送

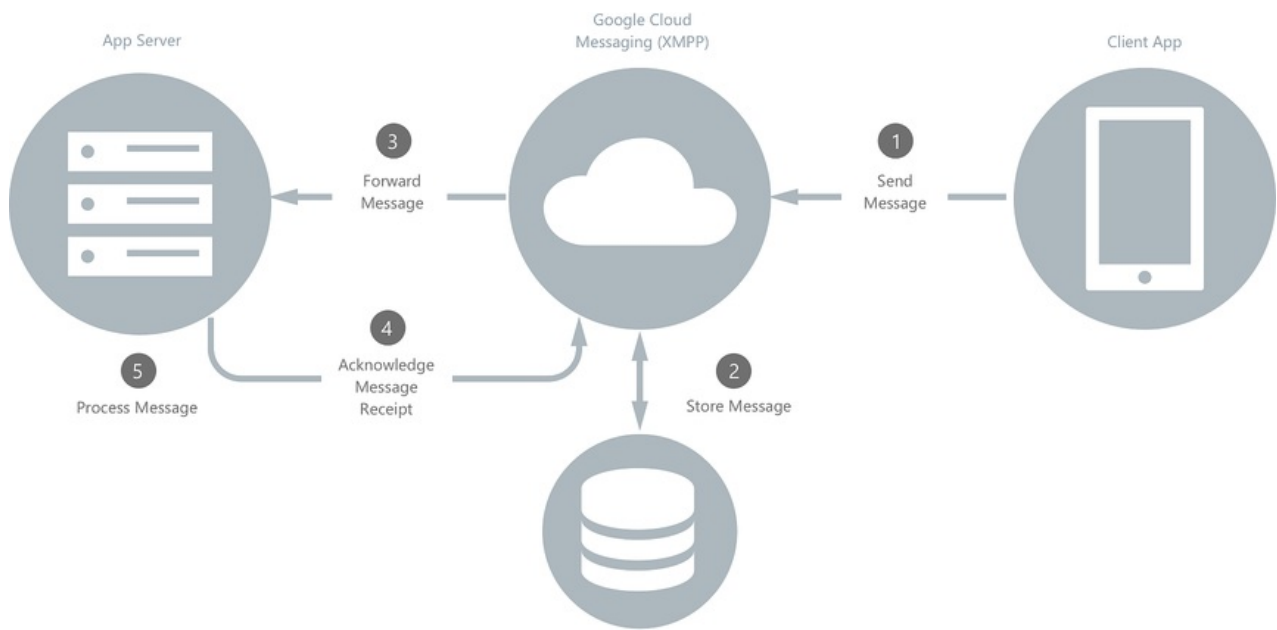
主题消息传送是一种类型的下游消息传送应用程序服务器到订阅（如天气预报）主题的多个客户端应用程序设备发送一条消息的位置。主题的消息可以是最大为 2 KB 的长度，并且消息传递主题支持最多 100 万个订阅每个应用。如果 GCM 要使用仅进行消息传送的主题，客户端应用程序不需要注册令牌发送到应用服务器。Google[实现主题的消息传送](#)介绍了如何将消息从应用程序服务器发送到某个特定主题订阅的多个设备。

组消息传递

组消息传递是一种类型的下游消息传送应用服务器到属于某个组（例如，一组属于单个用户的设备）的多个客户端应用程序设备发送一条消息的位置。组的消息可以是适用于 iOS 设备的长度最多为 2 KB 和高达 4 KB 的适用于 Android 设备的长度。一组被限制为最多 20 个成员。Google[设备组的消息传送](#)介绍了如何应用服务器可向属于某个组的设备上运行的多个客户端应用程序实例发送一条消息。

上游消息传送

如果客户端应用程序连接到支持的服务器XMPP，它可以将消息发送回应用程序服务器，如以下关系图中所示：



1. 客户端应用程序将消息发送到 GCM XMPP 连接服务器。
2. 如果应用程序服务器断开连接，GCM 服务器会将消息存储到队列中以更高版本转发。
3. 当应用程序服务器重新连接时，GCM 将转发到应用服务器的消息。
4. 应用服务器分析消息来验证身份的客户端应用程序，然后将"确认"发送到 GCM 确认消息接收。
5. 应用程序服务器处理的消息。

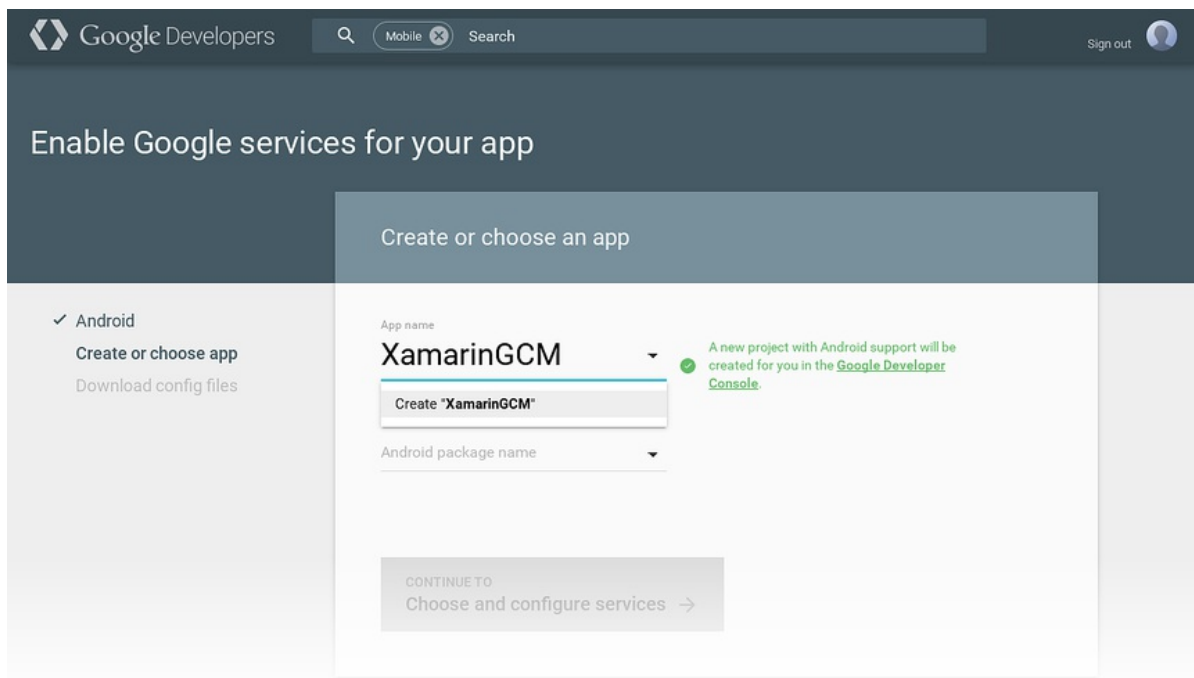
Google[上游消息](#)介绍了如何构建 JSON 编码的消息并将其发送到运行 Google 的基于 XMPP 的云连接服务器的应用程序服务器。

Google Cloud Messaging 设置

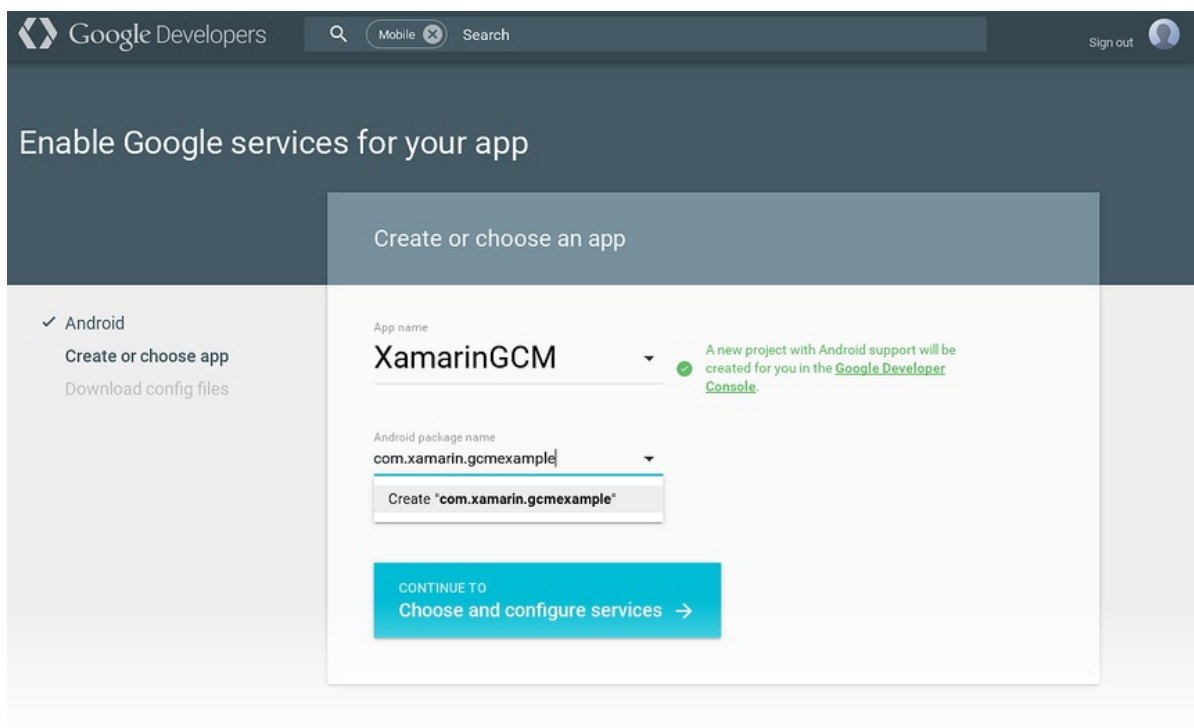
应用程序中使用 GCM 服务之前，必须首先获取对 Google 的 GCM 服务器的访问的凭据。以下部分介绍了完成此过程所需的步骤：

启用适用于应用的 Google 服务

1. 登录到[Google 开发人员控制台](#)使用 Google 帐户（即，你的 gmail 地址），并创建一个新项目。如果你有现有的项目，选择想要成为启用 GCM 的项目。在以下示例中，新的项目称为**XamarinGCM**创建：

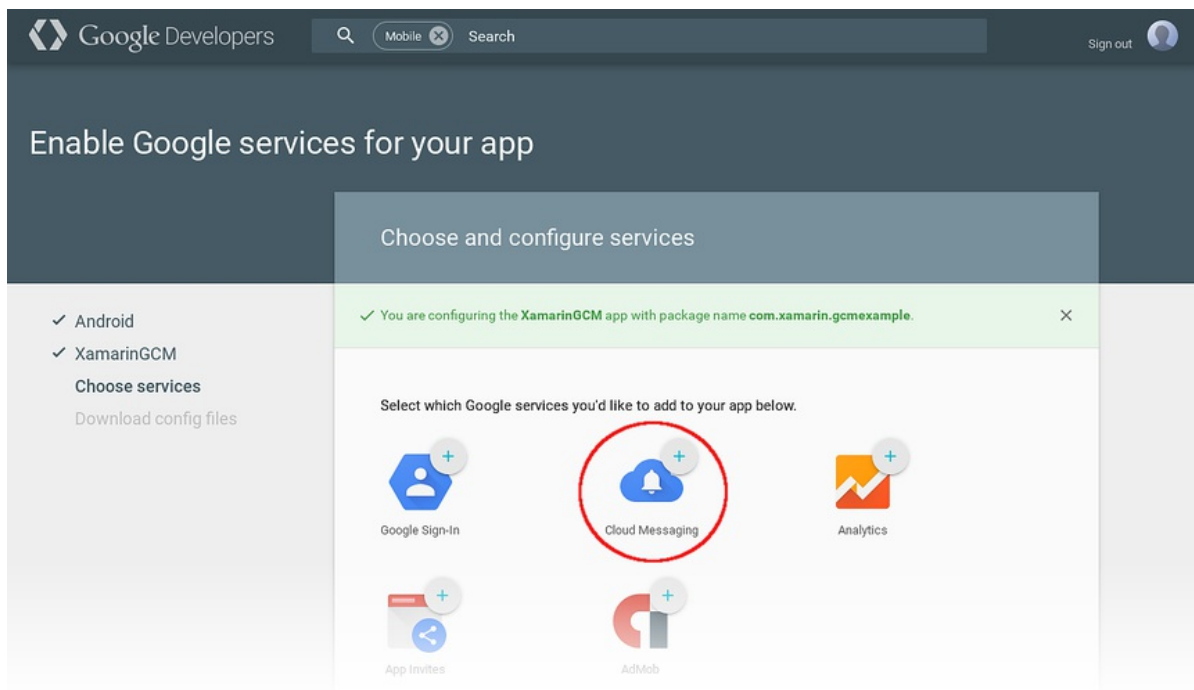


2. 接下来，输入您的应用程序的包名称 (在此示例中，包名称是`com.xamarin.gcmexample`)，单击继续选择和配置服务：

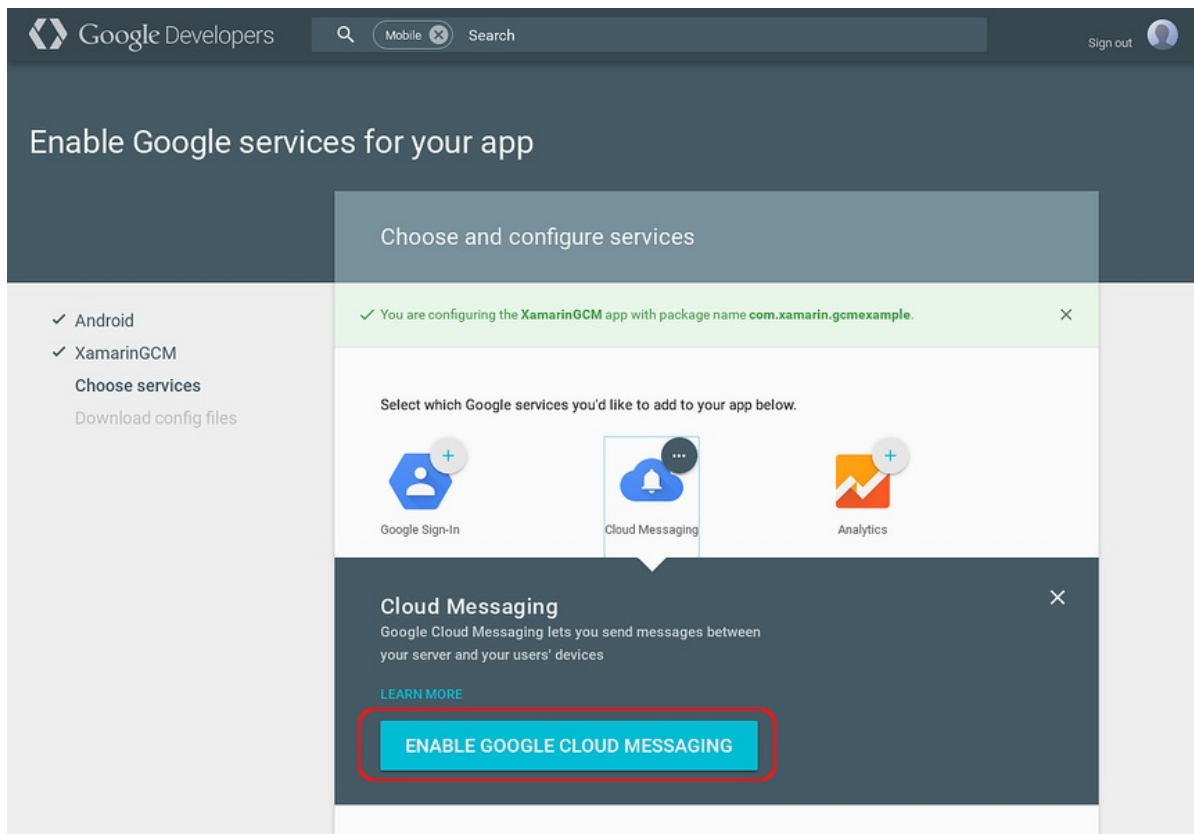


请注意，此包名称也是您的应用程序的应用程序 ID。

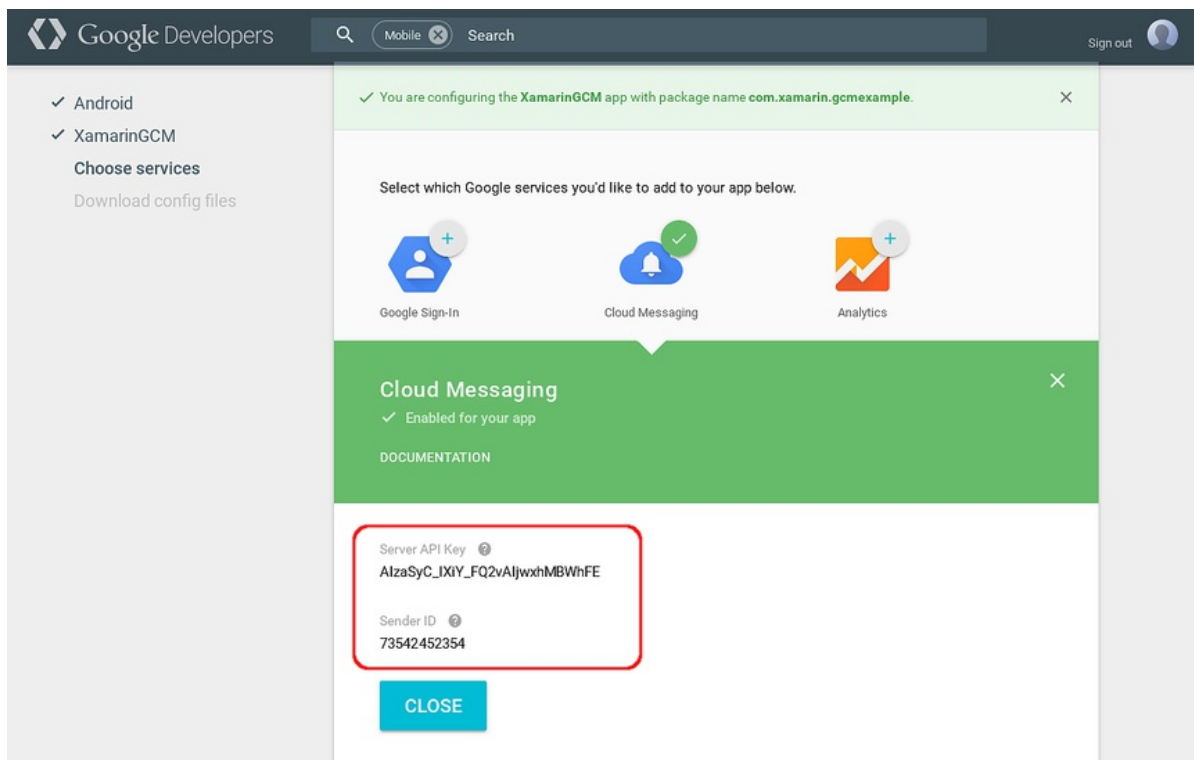
3. 选择并配置服务部分列出了可以添加到你的应用的 Google 服务。单击云消息传送：



4. 接下来, 单击启用 **GOOGLE CLOUD MESSAGING**:



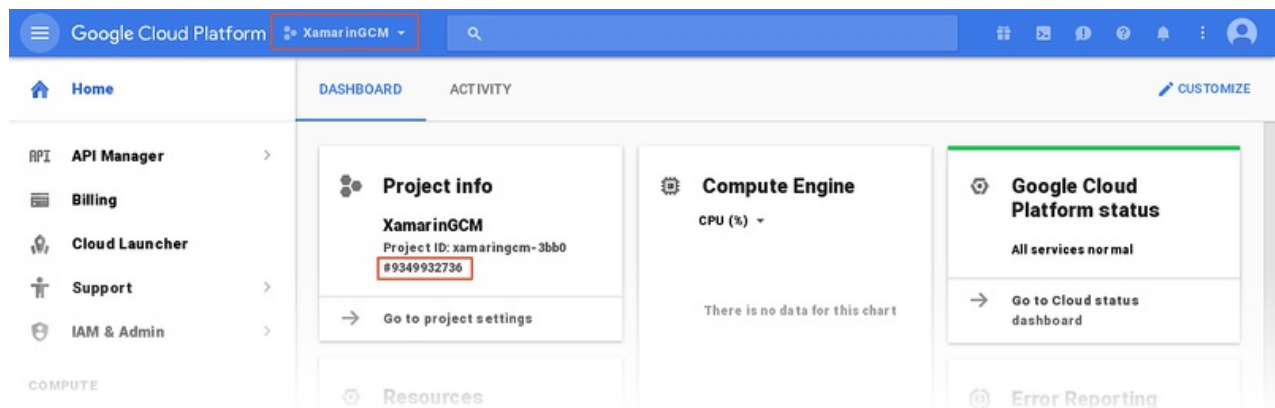
5. 一个服务器 **API** 密钥和一个发件人 **ID** 为应用生成。记录这些值, 并且单击关闭:



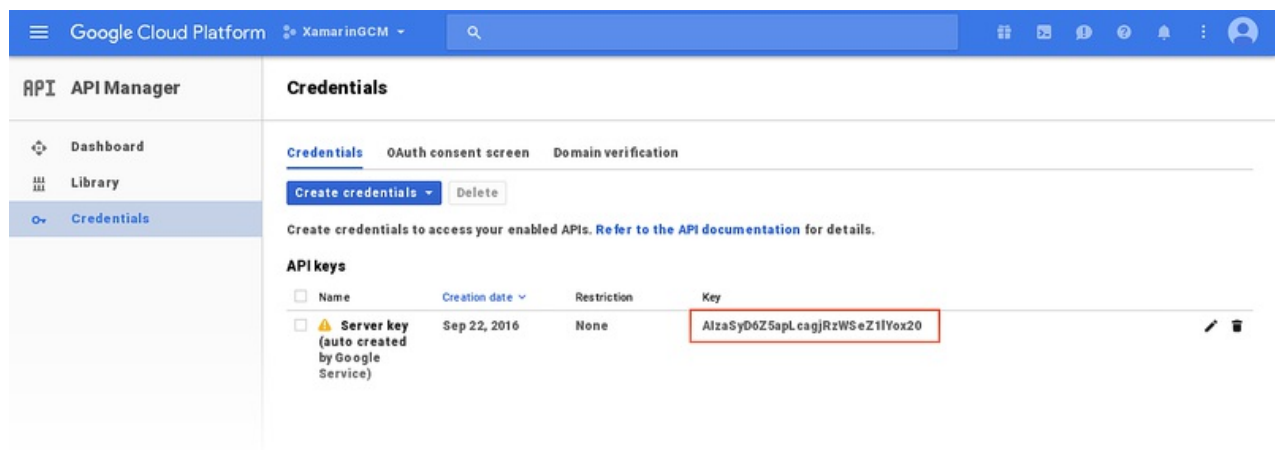
保护 API 密钥-它不是以公共方式使用。如果 API 密钥已泄漏, 未经授权的服务器无法将消息发布到客户端应用程序。[有关使用 API 密钥安全最佳实践](#)提供有用的指导原则来保护你的 API 密钥。

查看你的项目设置

可以通过登录到在任何时候查看你的项目设置[Google Cloud Console](#)并选择你的项目。例如, 可以查看发件人 ID 通过在页面顶部的下拉列表中选择你的项目 (在此示例中, 该项目称为XamarinGCM)。发件人 ID 是此屏幕截图中所示的项目编号 (此处的发件人 ID 是9349932736):



若要查看API 密钥, 单击API 管理器, 然后单击凭据:



其他阅读材料

- Google[注册客户端应用](#)介绍更多信息中的客户端注册过程，并提供有关配置自动重试和保持注册状态的同步信息。
- [RFC 6120](#)和[RFC 6121](#)解释，并定义可扩展消息传送和协议 (XMPP)。

总结

本文章提供 Google Cloud Messaging (GCM) 的概述。它介绍了各种用于标识和授权应用程序服务器和客户端应用程序之间的消息传送的凭据。所示的最常见的消息传送方案，并详细为您的应用程序注册到 GCM 使用 GCM 服务的步骤。

相关链接

- [云消息传送](#)

Google 云消息传送与远程通知

2018/11/13 • [Edit Online](#)

本演练介绍如何使用 Google Cloud Messaging 实现远程通知（也称为推送通知）的分步说明，Xamarin.Android 应用程序中。它描述了为通信使用 Google Cloud Messaging (GCM) 而必须实现的各种类，它还说明了如何访问 GCM、在 Android 清单中设置权限和它演示了端到端消息传送和示例测试程序。

NOTE

已被取代 GCM [Firebase Cloud Messaging](#) (FCM)。GCM 服务器和客户端 Api 已弃用将不再提供为 2019 年 4 月 11 日推出。

GCM 通知概述

在本演练中，我们将创建一个使用 Google Cloud Messaging (GCM) 来实现远程通知的 Xamarin.Android 应用程序（也称为推送通知）。我们将实现将 GCM 用于远程消息的各种意向和侦听器服务，我们将测试我们的实现与命令行程序，用于模拟的应用程序服务器。

请注意，Firebase Cloud Messaging (FCM) 是新版本的 GCM – Google 强烈建议使用 FCM，而不是 GCM。如果你当前使用 GCM，建议升级到 FCM。有关 FCM 的详细信息，请参阅[Firebase Cloud Messaging](#)。

您可以继续执行本演练之前，必须获取所需的凭据以使用 Google 的 GCM 服务器；此过程所述[Google Cloud Messaging](#)。具体而言，将需要 API 密钥和一个发件人 ID 要插入到在本演练中提供的示例代码。

我们将使用以下步骤创建一个启用 GCM 的 Xamarin.Android 客户端应用程序：

1. 安装与 GCM 服务器通信所需的其他包。
2. 配置应用程序对 GCM 服务器的访问权限。
3. 实现代码来检查 Google Play 服务存在。
4. 实现与 GCM 注册令牌协商的注册意向服务。
5. 实现实例 ID 侦听器服务，可侦听从 GCM 注册令牌更新。
6. 实现应用服务器通过 GCM 中接收远程消息的 GCM 侦听器服务。

此应用程序将使用名为的新 GCM 功能消息传递主题。在主题消息，应用服务器发送一条消息到一个主题，而不是单个设备的列表。订阅该主题的设备可以接收作为推送通知的主题的消息。GCM 主题消息传送的详细信息，请参阅 Google [实现主题的消息传送](#)。

客户端应用程序准备就绪后，我们将实现的命令行 C# 向我们的客户端应用程序通过 GCM 发送推送通知的应用程序。

演练

若要开始，让我们创建一个新的空解决方案，称为远程通知。接下来，让我们将新的 Android 项目添加到此解决方案基于 Android 应用模板。让我们称此项目 ClientApp。（如果您不熟悉创建 Xamarin.Android 项目，请参阅[Hello, Android](#)。）ClientApp 项目将包含接收远程通知通过 GCM Xamarin.Android 客户端应用程序的代码。

添加所需的包

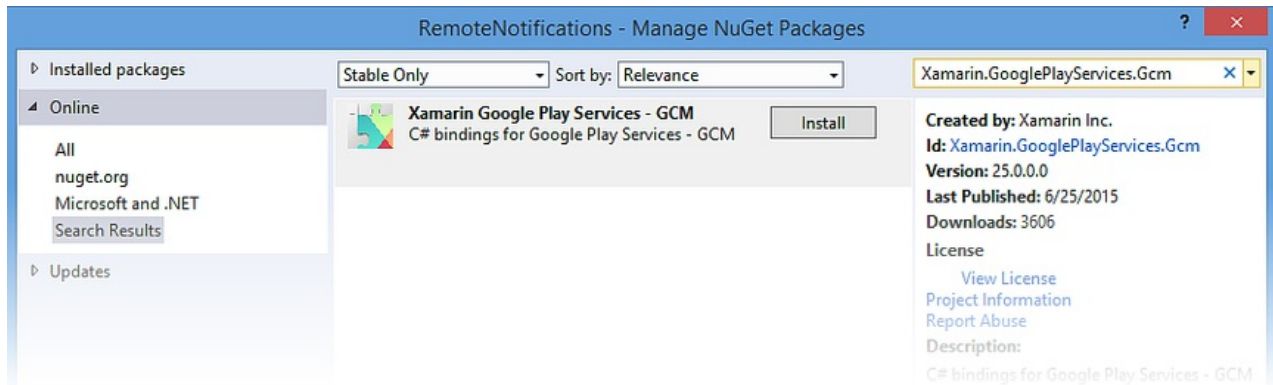
我们可以实现我们的客户端应用程序代码之前，必须安装多个包，我们将使用与 GCM 通信。此外，我们必须添加到我们的设备的 Google Play 商店应用程序，如果尚未安装。

添加 Xamarin Google Play Services GCM 包

若要从 Google Cloud Messaging，接收消息[Google Play Services](#) 框架必须在设备上存在。缺少这种框架，Android

应用程序不能从 GCM 服务器接收消息。Google Play Services 在后台运行 Android 设备处于打开状态，而安静地侦听来自 GCM 的消息。这些消息到达时，Google Play 服务将消息转换意向，然后广播到已注册为它们的应用程序这些意图。

在 Visual Studio 中，右键单击引用 > 管理 NuGet 包...; 在 Visual Studio for Mac 中，右键单击包 > 添加包...搜索 **Xamarin Google Play 服务-GCM** 并安装到此包 **ClientApp** 项目：



当你安装 **Xamarin Google Play 服务-GCM**，**Xamarin Google Play 服务** 的基本自动安装。如果遇到错误时，更改项目的 **最低 Android 目标** 而不设置为值编译使用 **SDK 版本**，然后重试 NuGet 安装。

接下来，编辑 **MainActivity.cs** 并添加以下 `using` 语句：

```
using Android.Gms.Common;
using Android.Util;
```

这使得类型在 Google Play Services GMS 包中提供给我们的代码，并添加日志记录功能，我们会将跟踪与 GMS 我们事务。

Google Play 商店

若要从 GCM 中接收消息，必须在设备上安装 Google Play 商店应用程序。（每次在设备上安装 Google Play 应用程序时，Google Play 商店也会安装，因此很可能它已经安装在测试设备上。）没有 Google Play 的 Android 应用程序不能从 GCM 中接收消息。如果你还没有在设备上安装 Google Play 应用商店应用程序，请访问 [Google Play](#) 网站上下载并安装 Google Play。

或者，可以使用 Android 模拟器运行 Android 2.2 或更高版本，而不是测试设备（您无需在 Android 仿真程序上安装 Google Play 商店）。但是，如果使用仿真程序，您必须使用 Wi-fi 连接到 GCM，并且稍后在本演练中所述，必须要在你的 Wi-fi 的防火墙中打开多个端口。

设置包名称

在中 [Google Cloud Messaging](#)，我们指定我们 GCM 启用的应用程序的包名称（此包名称也可作为 *应用程序 ID* 与我们的 API 密钥和发件人 ID 相关联）。让我们打开的属性 **ClientApp** 项目和包名称设置为此字符串。在此示例中，我们可以将包名称设置为 `com.xamarin.gcmexample`：

The screenshot shows the 'Android Manifest' configuration screen in Android Studio. The left sidebar has 'Android Manifest' selected. The main area shows configuration fields: 'Application name' (RemoteNotifications), 'Package name' (com.xamarin.gcmexample, highlighted with a red circle), 'Application Icon' (@drawable/icon), 'Version number' (1), and 'Version name' (1.0). The 'Configuration' and 'Platform' dropdowns at the top are set to 'N/A'.

请注意，客户端应用程序将无法从 GCM 中接收的注册令牌，如果此包名称不完全我们进入 Google 开发人员控制台的包名称匹配。

将权限添加到 Android 清单

Android 应用程序必须具有配置才能从 Google Cloud Messaging 接收通知的以下权限：

- `com.google.android.c2dm.permission.RECEIVE` – 向我们注册并接收来自 Google Cloud Messaging 的消息的应用程序授予权限。(用途 `c2dm` 意味着？这代表_云到设备_的消息传送，这是到 GCM 现已弃用前置任务。GCM 仍使用 `c2dm` 许多及其权限的字符串中。)
- `android.permission.WAKE_LOCK` – (可选)防止设备 CPU 进入睡眠状态时侦听的一条消息。
- `android.permission.INTERNET` – 授予 internet 访问权限，以便客户端应用程序能够与 GCM 通信。
- `package_name.permission.C2D_MESSAGE` –与 Android 一起注册该应用程序并请求权限以独占方式接收所有 C2D (云到设备) 的消息。`Package_name`前缀等同于应用程序 id。

我们将在 Android 清单中设置这些权限。让我们编辑**AndroidManifest.xml**并将内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="YOUR_PACKAGE_NAME"
    android:versionCode="1"
    android:versionName="1.0"
    android:installLocation="auto">
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="YOUR_PACKAGE_NAME.permission.C2D_MESSAGE" />
    <permission android:name="YOUR_PACKAGE_NAME.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <application android:label="ClientApp" android:icon="@drawable/icon">
    </application>
</manifest>
```

在上述 XML 中，更改您的包名为你的客户端应用程序项目的包名称。例如 `com.xamarin.gcmexample`。

检查 Google Play 服务

对于本演练，我们要创建一个基本应用程序使用单个 `TextView` 在 UI 中。此应用程序并不直接指示 GCM 与的交互。相反，我们会监视输出窗口以查看如何使用 GCM，我们应用握手到达，我们将检查新的通知的通知栏。

首先，让我们创建适用于在消息区域的布局。编辑**Resources.layout.Main.axml**并将内容替换为以下 XML：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp">
    <TextView
        android:text=" "
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/msgText"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:padding="10dp" />
</LinearLayout>
```

保存**Main.xml**并将其关闭。

客户端应用程序启动时，我们想让它验证之前我们尝试联系 GCM Google Play 服务可用。编辑**MainActivity.cs**，并将为 `count` 实例使用以下实例变量声明的变量声明：

```
TextView msgText;
```

接下来，添加以下方法**MainActivity**类：

```
public bool IsPlayServicesAvailable ()
{
    int resultCode = GoogleApiAvailability.Instance.IsGooglePlayServicesAvailable (this);
    if (resultCode != ConnectionResult.Success)
    {
        if (GoogleApiAvailability.Instance.IsUserResolvableError (resultCode))
            msgText.Text = GoogleApiAvailability.Instance.GetErrorString (resultCode);
        else
        {
            msgText.Text = "Sorry, this device is not supported";
            Finish ();
        }
        return false;
    }
    else
    {
        msgText.Text = "Google Play Services is available.";
        return true;
    }
}
```

此代码检查设备以查看是否已安装 Google Play Services APK。如果未安装，是在消息区域中，指示用户从 Google Play 商店下载 APK（或设备的系统设置中启用它）中显示一条消息。因为我们想要运行此检查，客户端应用程序启动时，我们将在末尾添加对此方法的调用 `OnCreate`。

接下来，将 `OnCreate` 方法使用以下代码：

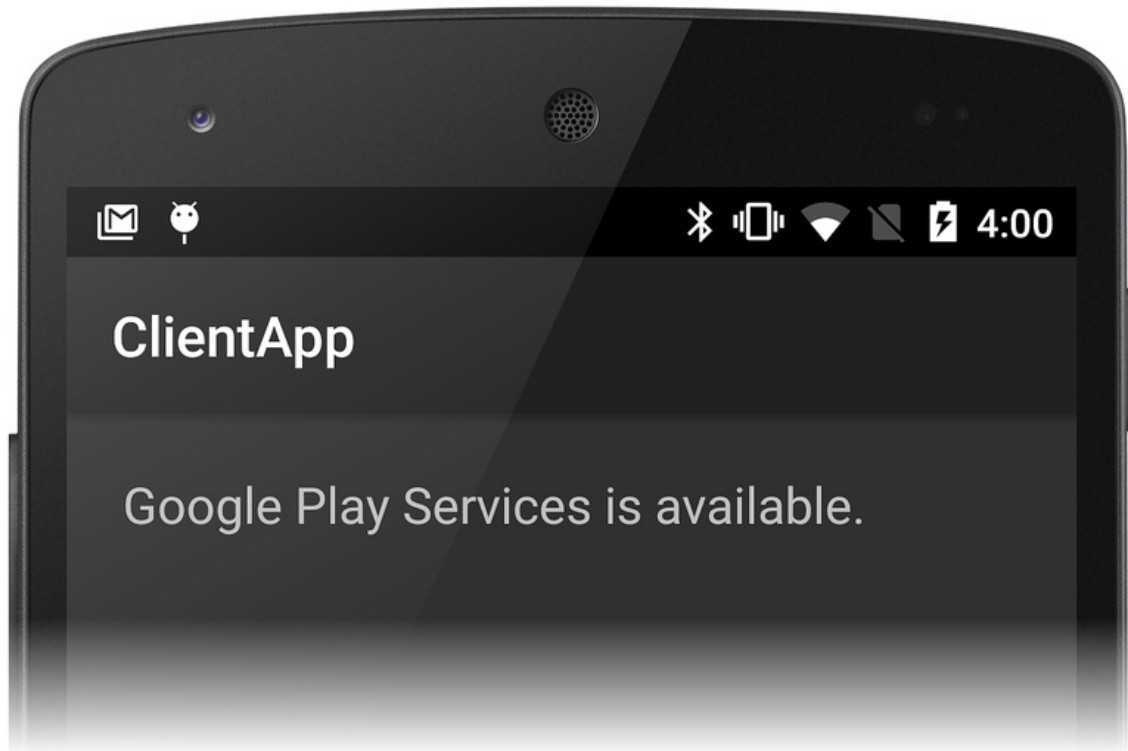
```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    SetContentView (Resource.Layout.Main);
    msgText = FindViewById<TextView> (Resource.Id.msgText);

    IsPlayServicesAvailable ();
}
```

此代码检查存在的 Google Play Services APK, 并将结果写入到消息区域中。

让我们完全重新生成并运行应用程序。应看到如以下屏幕截图所示的屏幕：



如果没有获得此结果, 请验证你的设备上是否安装了 Google Play Services APK **Xamarin Google Play 服务-GCM**包添加到你**ClientApp**项目如下所述前面。如果收到生成错误, 请尝试清除解决方案, 然后再次生成项目。

接下来, 我们将编写代码, 请联系 GCM 并重新注册令牌。

向 GCM 注册

应用可从应用服务器接收远程通知之前, 它必须向 GCM 注册并重新获取注册令牌。向 GCM 注册我们的应用程序的工作由 `IntentService` 我们创建的。我们 `IntentService` 将执行以下步骤：

1. 使用 `InstanceId` API 来生成授权我们的客户端应用程序来访问应用程序服务器的安全令牌。作为回报, 我们得到一个注册令牌从 GCM。
2. (如果该应用程序服务器都需要它), 将转发到应用服务器的注册令牌。
3. 订阅一个或多个通知主题通道。

我们实现这后 `IntentService`, 我们将测试以查看是否收到注册令牌从 GCM。

添加名为的新文件 **RegistrationIntentService.cs**和模板代码替换为以下：

```

using System;
using Android.App;
using Android.Content;
using Android.Util;
using Android.Gms.Gcm;
using Android.Gms.Gcm.Iid;

namespace ClientApp
{
    [Service(Exported = false)]
    class RegistrationIntentService : IntentService
    {
        static object locker = new object();

        public RegistrationIntentService() : base("RegistrationIntentService") { }

        protected override void OnHandleIntent (Intent intent)
        {
            try
            {
                Log.Info ("RegistrationIntentService", "Calling InstanceID.GetToken");
                lock (locker)
                {
                    var instanceID = InstanceID.GetInstance (this);
                    var token = instanceID.GetToken (
                        "YOUR_SENDER_ID", GoogleCloudMessaging.InstanceIdScope, null);

                    Log.Info ("RegistrationIntentService", "GCM Registration Token: " + token);
                    SendRegistrationToAppServer (token);
                    Subscribe (token);
                }
            }
            catch (Exception e)
            {
                Log.Debug("RegistrationIntentService", "Failed to get a registration token");
                return;
            }
        }

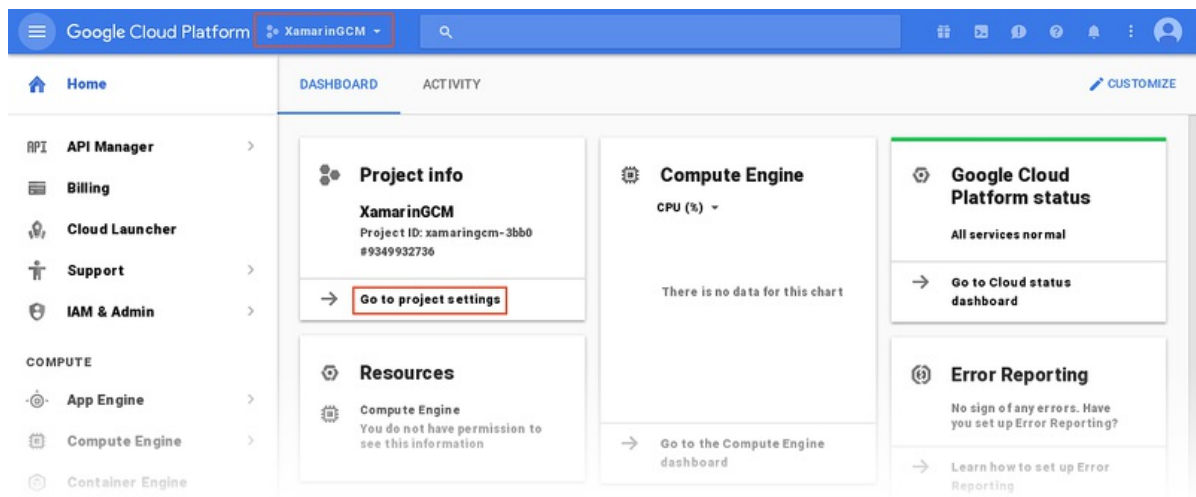
        void SendRegistrationToAppServer (string token)
        {
            // Add custom implementation here as needed.
        }

        void Subscribe (string token)
        {
            var pubSub = GcmPubSub.GetInstance(this);
            pubSub.Subscribe(token, "/topics/global", null);
        }
    }
}

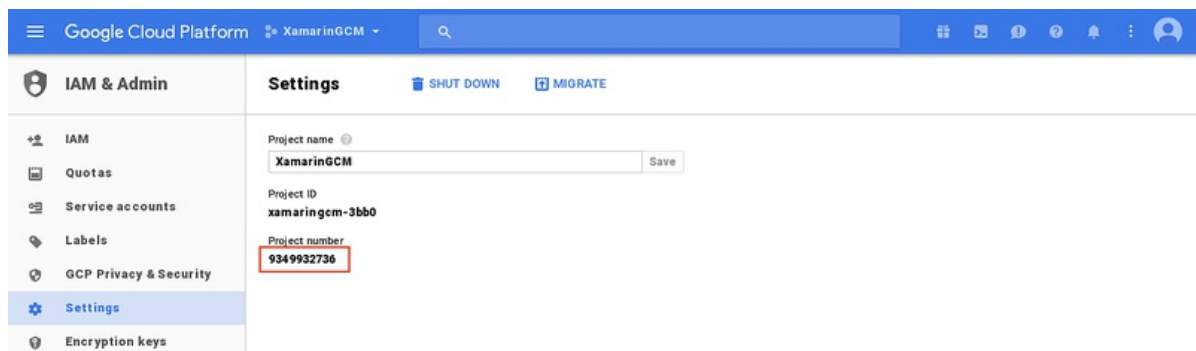
```

在上面的示例代码中，更改`YOUR_SENDER_ID`到你的客户端应用程序项目的发件人 ID 号。若要获取你的项目的发件人 ID：

1. 登录到[Google Cloud Console](#)，然后从下拉菜单中选择你的项目名称。在中项目信息窗格中显示的项目中，单击转到项目设置：



2. 上设置页上, 找到项目编号-这是你的项目的发件人 ID:



我们想要启动我们 `RegistrationIntentService` 我们的应用程序开始运行时。编辑 `MainActivity.cs` 并修改 `OnCreate` 方法, 以便我们 `RegistrationIntentService` 后我们检查是否存在的 Google Play 服务已启动:

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);

    SetContentView(Resource.Layout.Main);
    msgText = FindViewById<TextView> (Resource.Id.msgText);

    if (IsPlayServicesAvailable ())
    {
        var intent = new Intent (this, typeof (RegistrationIntentService));
        StartService (intent);
    }
}
```

现在让我们看一看每个部分的 `RegistrationIntentService` 若要了解其工作原理。

首先, 我们添加批注我们 `RegistrationIntentService` 与要表明我们的服务时不应由系统进行实例化的以下属性:

```
[Service (Exported = false)]
```

`RegistrationIntentService` 构造函数名称的工作线程 `RegistrationIntentService` 使调试更加容易。

```
public RegistrationIntentService() : base ("RegistrationIntentService") { }
```

核心功能 `RegistrationIntentService` 驻留在 `OnHandleIntent` 方法。让我们演练一下此代码来查看它如何向 GCM 注册我们的应用程序。

`OnHandleIntent` 首先调用 Google `InstanceId.GetToken` 方法来请求从 GCM 注册令牌。我们将在该代码打包 `lock` 若要防止同时发生多个注册意向的可能性—`lock` 可确保按顺序处理这些意图。如果我们无法获取注册令牌，将引发异常和我们记录了错误。如果注册成功，`token` 设置为我们返回从 GCM 注册令牌：

```
static object locker = new object ();
...
try
{
    lock (locker)
    {
        var instanceID = InstanceID.GetInstance (this);
        var token = instanceID.GetToken (
            "YOUR_SENDER_ID", GoogleCloudMessaging.InstanceIdScope, null);
        ...
    }
}
catch (Exception e)
{
    Log.Debug ...
}
```

将注册令牌转发到应用服务器

如果我们获取注册令牌（即，未引发任何异常），我们调用 `SendRegistrationToAppServer` 关联用户的注册令牌与服务端帐户（如果有），维护通过我们的应用程序。因为此实现取决于应用程序服务器的设计，此处提供了一个空的方法：

```
void SendRegistrationToAppServer (string token)
{
    // Add custom implementation here as needed.
}
```

在某些情况下，应用程序服务器不需要用户的注册令牌；在这种情况下，可以省略此方法。在将注册令牌发送到应用服务器 `SendRegistrationToAppServer` 应保持指示令牌是否已发送到服务器的布尔值。如果此布尔值为 `false`，`SendRegistrationToAppServer` 将该令牌发送到应用服务器—否则，该令牌已发送到上一次调用中的应用程序服务器。

订阅通知主题

接下来，我们调用我们 `Subscribe` 方法，以指示到我们想要通知主题订阅的 GCM。在中 `Subscribe`，我们调用 `GcmPubSub.Subscribe` API 来订阅我们的客户端应用程序下的所有消息 `/topics/global`：

```
void Subscribe (string token)
{
    var pubSub = GcmPubSub.GetInstance(this);
    pubSub.Subscribe(token, "/topics/global", null);
}
```

应用程序服务器必须将发送通知消息到 `/topics/global` 如果我们以接收推送通知。请注意，本主题下名称 `/topics`，只要应用程序服务器和客户端应用程序都达成这些名称可以是任何所需内容。（在这里，我们选择了名称 `global` 表示我们希望支持的应用程序服务器的所有主题上接收的消息。）

在服务器端消息传送的 GCM 主题有关的信息，请参阅 Google [发送到主题消息传送](#)。

实现实例 ID 侦听器服务

注册令牌是唯一的安全；但是，客户端应用程序（或 GCM）可能需要刷新出现应用程序重新安装或安全问题时的注册令牌。出于此原因，我们必须实现 `InstanceIdListenerService` 从 GCM 令牌刷新请求的响应。

添加名为的新文件 `InstanceIdListenerService.cs` 和模板代码替换为以下：


```
using Android.App;
using Android.Content;
using Android.Gms.Gcm.Iid;

namespace ClientApp
{
    [Service(Exported = false), IntentFilter(new[] { "com.google.android.gms.iid.InstanceID" })]
    class MyInstanceIdListenerService : InstanceIDListenerService
    {
        public override void OnTokenRefresh()
        {
            var intent = new Intent (this, typeof (RegistrationIntentService));
            StartService (intent);
        }
    }
}
```

批注 `InstanceIdListenerService` 具有以下属性以指示服务处于无法进行实例化系统，它可以接收 GCM 注册令牌 (也称为 *实例 ID*) 刷新的请求：

```
[Service(Exported = false), IntentFilter(new[] { "com.google.android.gms.iid.InstanceID" })]
```

`OnTokenRefresh` 我们的服务中的方法启动 `RegistrationIntentService`，以便它可以截取新的注册令牌。

测试注册到 GCM

让我们完全重新生成并运行应用程序。如果已成功从 GCM 中接收的注册令牌，应在输出窗口中显示注册令牌。例如：

```
D/Mono (1934): Assembly Ref addrf ClientApp[0xb4ac2400] -> Xamarin.GooglePlayServices.Gcm[0xb4ac2640]: 2
I/RegistrationIntentService(1934): Calling InstanceID.GetToken
I/RegistrationIntentService(1934): GCM Registration Token: f8LdveCvXig:APA91bFIIsjUAbP-V8TPQdLR89qQbEJh1SYG38AcCbBUf34z5gSdUc50sXrgs93YFiGcRSRafPfzkz23lf3-LvYV1CwrFheMjHgwPeFSh12MywnRIhz
```

句柄的下游消息

我们到目前为止已实现的代码是仅“设置”代码；它检查以查看 Google Play 服务已安装并协商使用 GCM 和应用程序服务器准备我们的客户端应用程序接收远程通知。但是，我们尚未实现实际接收和处理下游通知消息的代码。若要执行此操作，我们必须实现 *GCM 侦听器服务*。此服务接收来自应用程序服务器的主题消息，然后将其本地广播作为通知。我们实现此服务后，我们将创建测试程序，用于将消息发送到 GCM，以便我们可以看到如果我们实现能够正常运行。

添加通知图标

让我们先添加启动我们通知时将出现在通知区域中的小图标。可以将复制[此图标](#)到你的项目或创建你自己的自定义图标。我们将命名图标文件 `ic_stat_button_click.png` 将其复制到资源/`drawable` 文件夹。请记住使用添加 > 现有项... 此图标文件包含在项目中。

实现 GCM 侦听器服务

添加名为的新文件 `GcmListenerService.cs` 和模板代码替换为以下：


```

using Android.App;
using Android.Content;
using Android.OS;
using Android.Gms.Gcm;
using Android.Util;

namespace ClientApp
{
    [Service (Exported = false), IntentFilter (new [] { "com.google.android.c2dm.intent.RECEIVE" })]
    public class MyGcmListenerService : GcmListenerService
    {
        public override void OnMessageReceived (string from, Bundle data)
        {
            var message = data.GetString ("message");
            Log.Debug ("MyGcmListenerService", "From: " + from);
            Log.Debug ("MyGcmListenerService", "Message: " + message);
            SendNotification (message);
        }

        void SendNotification (string message)
        {
            var intent = new Intent (this, typeof(MainActivity));
            intent.AddFlags (ActivityFlags.ClearTop);
            var pendingIntent = PendingIntent.GetActivity (this, 0, intent, PendingIntentFlags.OneShot);

            var notificationBuilder = new Notification.Builder(this)
                .SetSmallIcon (Resource.Drawable.ic_stat_ic_notification)
                .SetContentTitle ("GCM Message")
                .SetContentText (message)
                .SetAutoCancel (true)
                .SetContentIntent (pendingIntent);

            var notificationManager = (NotificationManager)GetSystemService(Context.NotificationService);
            notificationManager.Notify (0, notificationBuilder.Build());
        }
    }
}

```

让我们看看每个部分我们 `GcmListenerService` 若要了解其工作原理。

首先，我们添加批注 `GcmListenerService` 属性以指示，此服务并不是由系统进行实例化以及我们将添加一个意向筛选器，以指示它将接收 GCM 消息：

```
[Service (Exported = false), IntentFilter (new [] { "com.google.android.c2dm.intent.RECEIVE" })]
```

当 `GcmListenerService` 从 GCM 中接收消息 `OnMessageReceived` 调用方法。此方法从传入的提取消息内容 `Bundle`，记录消息内容（因此，我们可以在输出窗口中查看它），并调用 `SendNotification` 启动包含接收的消息内容的本地通知：

```

var message = data.GetString ("message");
Log.Debug ("MyGcmListenerService", "From: " + from);
Log.Debug ("MyGcmListenerService", "Message: " + message);
SendNotification (message);

```

`SendNotification` 方法使用 `Notification.Builder` 创建通知，然后使用 `NotificationManager` 以启动该通知。实际上，这将转换的本地通知，以便向用户显示为远程通知消息。有关使用详细信息 `Notification.Builder` 并 `NotificationManager`，请参阅[本地通知](#)。

声明在清单中的接收方

我们可以从 GCM 中接收消息之前，我们必须声明 Android 清单中的 GCM 侦听器。让我们编

辑 **AndroidManifest.xml**，并将为 `<application>` 部分使用以下 XML：

```
<application android:label="RemoteNotifications" android:icon="@drawable/Icon">
  <receiver android:name="com.google.android.gms.gcm.GcmReceiver"
    android:exported="true"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE" />
      <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
      <category android:name="YOUR_PACKAGE_NAME" />
    </intent-filter>
  </receiver>
</application>
```

在上述 XML 中，更改您的包名为您的客户端应用程序项目的包名称。在演练本示例中，包名称是 `com.xamarin.gcmexample`。

让我们看看此 XML 中的每个设置的作用：

| 设置 | 描述 |
|--|--|
| <code>com.google.android.gms.gcm.GcmReceiver</code> | 声明我们的应用程序实现 GCM 接收方，捕获和处理传入的推送通知消息。 |
| <code>com.google.android.c2dm.permission.SEND</code> | 声明仅 GCM 服务器可以直接向应用程序发送消息。 |
| <code>com.google.android.c2dm.intent.RECEIVE</code> | 广告，我们的应用程序处理广播的消息从 GCM 意向筛选器。 |
| <code>com.google.android.c2dm.intent.REGISTRATION</code> | 广告，我们的应用程序处理新注册意向的意向筛选器（即，我们已实现实例 ID 侦听器服务）。 |

或者，您可以修饰 `GcmListenerService` 使用这些属性，而无需在 XML 中指定此处我们中指定它们 **AndroidManifest.xml**，以便更轻松地遵循代码示例。

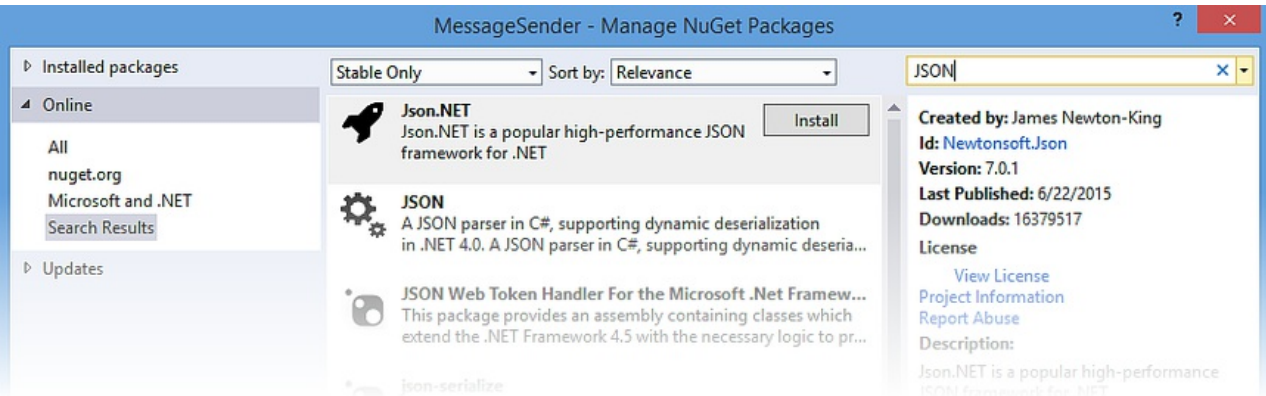
创建消息发送者，以测试应用程序

让我们将添加 C# 桌面控制台应用程序项目和解决方案并调用其 **MessageSender**。我们将使用此控制台应用程序来模拟应用程序服务器-它将通知消息发送到 **ClientApp** 通过 GCM。

添加 Json.NET 包

在此控制台应用中，我们要生成包含我们想要发送到客户端应用程序的通知消息的 JSON 有效负载。我们将使用 **Json.NET** 中打包 **MessageSender** 以便更轻松地生成所需的 GCM 的 JSON 对象。在 Visual Studio 中，右键单击引用 > 管理 NuGet 包...；在 Visual Studio for Mac 中，右键单击包 > 添加包....

我们来搜索 **Json.NET** 包并将其安装在项目中：



添加对 **System.Net.Http** 的引用

我们还需要添加对的引用 `System.Net.Http`，以便我们可以实例化 `HttpClient` 将我们的测试消息发送到 GCM。在中 **MessageSender** 项目，右键单击引用 > 添加引用 向下滚动直到您看到 **System.Net.Http**。旁边放置一个复选标记 **System.Net.Http** 然后单击 **确定**。

发送测试消息的实现代码

在中 **MessageSender**，编辑 **Program.cs** 和内容替换为以下代码：

```
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json.Linq;

namespace MessageSender
{
    class MessageSender
    {
        public const string API_KEY = "YOUR_API_KEY";
        public const string MESSAGE = "Hello, Xamarin!";

        static void Main (string[] args)
        {
            var jGcmData = new JObject();
            var jData = new JObject();

            jData.Add ("message", MESSAGE);
            jGcmData.Add ("to", "/topics/global");
            jGcmData.Add ("data", jData);

            var url = new Uri ("https://gcm-http.googleapis.com/gcm/send");
            try
            {
                using (var client = new HttpClient())
                {
                    client.DefaultRequestHeaders.Accept.Add(
                        new MediaTypeWithQualityHeaderValue("application/json"));

                    client.DefaultRequestHeaders.TryAddWithoutValidation (
                        "Authorization", "key=" + API_KEY);

                    Task.WaitAll(client.PostAsync (url,
                        new StringContent(jGcmData.ToString(), Encoding.Default, "application/json"))
                        .ContinueWith(response =>
                        {
                            Console.WriteLine(response);
                            Console.WriteLine("Message sent: check the client device notification tray.");
                        }));
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Unable to send GCM message:");
                Console.Error.WriteLine(e.StackTrace);
            }
        }
    }
}
```

在上述代码中，更改 `YOUR_API_KEY` 访问 API 密钥的客户端应用程序项目。

此测试应用程序服务器将以下 JSON 格式的消息发送到 GCM:

```
{
  "to": "/topics/global",
  "data": {
    "message": "Hello, Xamarin!"
  }
}
```

GCM, 反过来, 这将消息转发到客户端应用程序。让我们构建**MessageSender**和打开我们可以在其中运行命令行从它的控制台窗口。

尝试一下！

现在我们已准备好测试我们的客户端应用程序。如果你使用仿真程序, 或如果你的设备通过 Wi-fi 与 GCM 通信时, 必须通过获取的 GCM 消息在防火墙上打开以下 TCP 端口: 5228、5229 和 5230。

启动客户端应用, 并观察输出窗口。之后 `RegistrationIntentService` 成功接收一个注册从 GCM 令牌, 输出窗口应显示标记使用与下面类似的日志输出:

```
I/RegistrationIntentService(16103): GCM Registration Token: eX9ggabZV1Q:APA91bHjBnQXMUeBOT6JDilPrt8m2YWtY ...
```

此时客户端应用程序已准备好接收远程通知消息。从命令行中, 运行**MessageSender.exe**要将"Hello, Xamarin"的通知消息发送到客户端应用程序。如果你不生成**MessageSender**项目中, 现在执行此操作。

若要运行**MessageSender.exe**在 Visual Studio 中, 打开命令提示符下, 更改到**MessageSender/bin/Debug**目录, 并运行命令直接:

```
MessageSender.exe
```

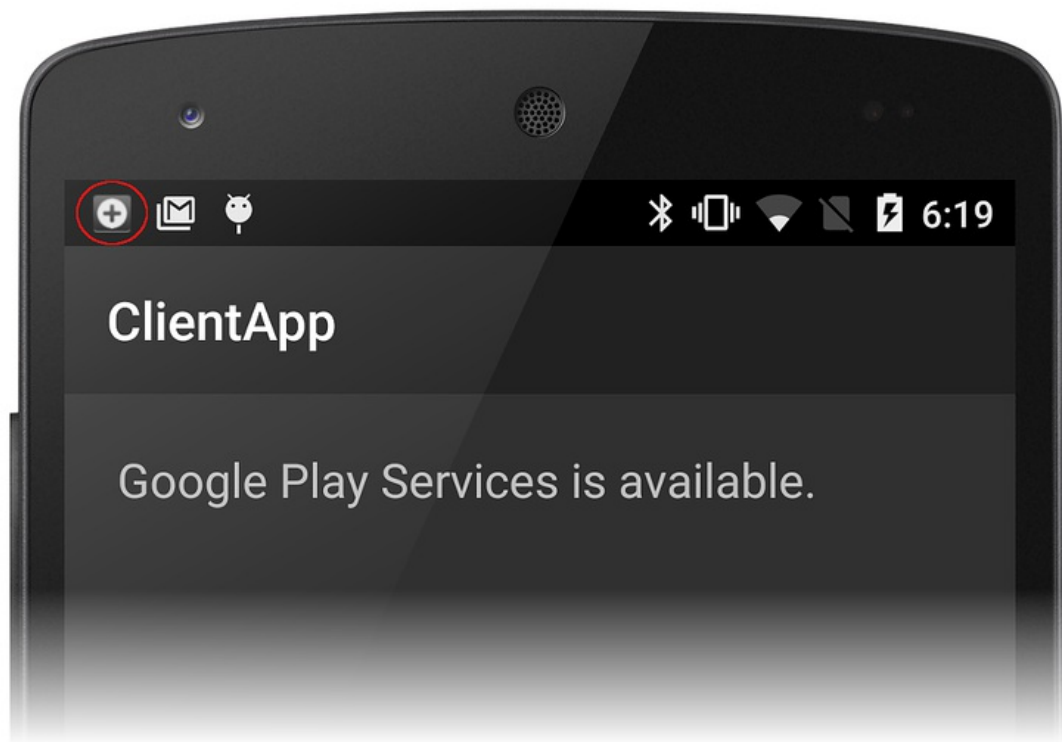
若要运行**MessageSender.exe**在 Visual Studio for Mac 中, 打开终端会话, 更改到**MessageSender/bin/Debug**目录, 并使用 `mono` 运行**MessageSender.exe**

```
mono MessageSender.exe
```

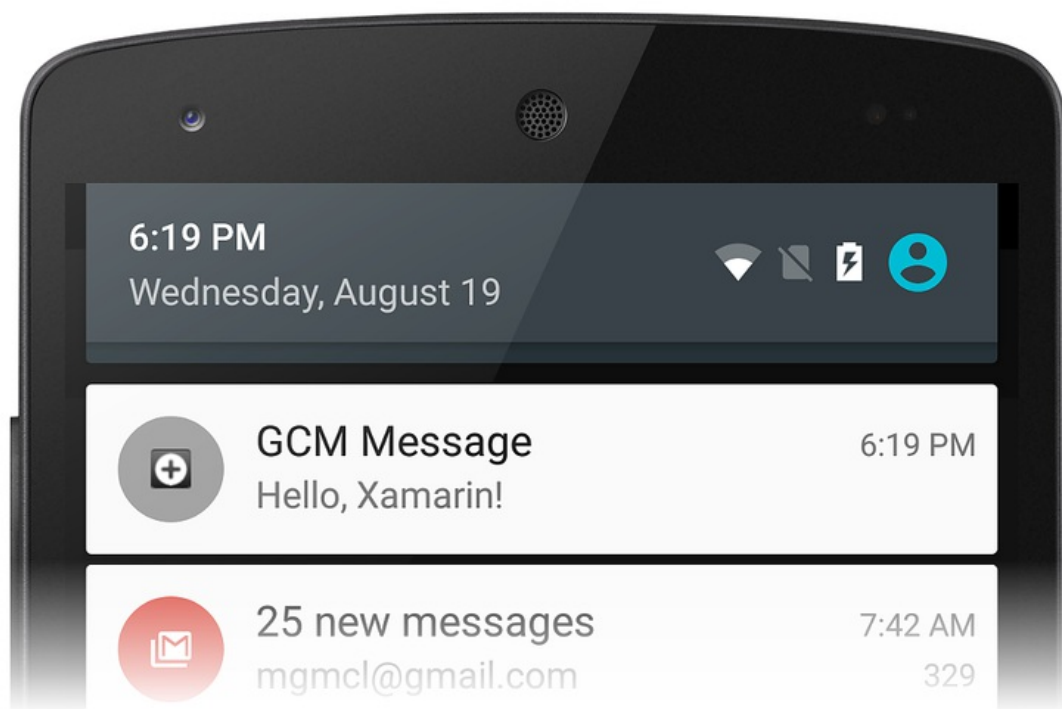
它可能需要一分钟来传播通过 GCM 和返回到客户端应用程序的消息。如果已成功收到消息时, 应该会看到类似于输出窗口中的以下输出:

```
D/MyGcmListenerService(16103): From:    /topics/global
D/MyGcmListenerService(16103): Message: Hello, Xamarin!
```

此外, 您会注意到一个新的通知图标已显示在通知栏:



当您打开的通知送纸器，若要查看通知时，应会看到我们远程通知：



恭喜，您的应用程序已收到其第一个远程通知！

请注意是否应用强制停止，将无法再接收 GCM 消息。若要强制停止后恢复通知，应用程序必须手动重新启动。有关此 Android 策略的详细信息，请参阅[启动已停止的应用程序上的控件](#)，这[堆栈溢出文章](#)。

总结

本演练详细 Xamarin.Android 应用程序中实现远程通知的步骤。介绍了如何安装所需的 GCM 通信的其他包，并介绍了如何配置应用程序对 GCM 服务器的访问权限。它提供示例代码，说明如何检查存在的 Google Play 服务、如

何实现注册意向服务和实例 ID 侦听器服务协商向 GCM 注册令牌，以及如何实现 GCM 侦听器接收并处理远程通知消息的服务。最后，我们实现了命令行测试程序，用于将测试通知发送到 GCM 通过我们的客户端应用程序。

相关链接

- [GCM 远程通知（示例）](#)
- [Google 云消息传送](#)

介绍了 Web 服务

2018/6/22 • [Edit Online](#)

本指南演示如何使用另一个 web 服务技术。涵盖的主题包括与 REST 服务、SOAP 服务和 Windows Communication Foundation 服务通信。

为了正常运行，许多移动应用程序都依赖于云中，并因此将 web 服务集成到移动应用程序是一个常用方案。Xamarin 平台支持使用另一个 web 服务技术，包括内置和第三方为使用基于 Rest、ASMX 和 Windows Communication Foundation (WCF) 服务的支持。

本文讨论以下主题：

- [REST 服务](#)
- [ASP.Net Web 服务 \(ASMX\)](#)
- [WCF 服务](#)

对于使用 Xamarin.Forms 的客户，有使用上述每种技术中的完整示例[Xamarin.Forms Web 服务文档](#)。

IMPORTANT

在 iOS 9 中，应用程序传输安全 (ATS) 强制实施安全连接之间 internet 资源（如应用程序的后端服务器）和应用程序，从而防止意外泄露的敏感信息。由于默认情况下，生成的 ios 9 应用中启用了 ATS，所有连接都将遵循 ATS 安全要求。如果连接不能满足这些要求，则会失败并出现异常。

你可以选择退出的 ATS 如果不能使用 [HTTPS](#) 协议和安全的 internet 资源的通信。这可以通过更新应用程序的实现 [Info.plist](#) 文件。有关详细信息请参阅[应用传输安全](#)。

REST

具象状态传输 (REST) 是用于生成 web 服务的架构样式。REST 请求都通过 HTTP 使用 web 浏览器使用来检索网页并将数据发送到服务器的同一 HTTP 谓词。谓词是：

- **获取**– 此操作用于从 web 服务中检索数据。
- **POST**– 此操作用于在 web 服务上创建数据的新项。
- **PUT**– 使用此操作更新某个项 web 服务上的数据。
- **修补程序**– 使用此操作通过描述有关应如何修改项的一组的说明更新 web 服务上的数据的某个项。在示例应用程序未使用此谓词。
- **删除**– 使用此操作删除某项 web 服务上的数据。

遵守 REST 的 Api 调用 RESTful Api 和使用定义的 web 服务：

- 基 URI。
- HTTP 方法，如 GET、POST、PUT、PATCH 或 DELETE。
- 数据，如 JavaScript 对象表示法 (JSON) 媒体类型。

REST 的简单性已帮助使其用于访问移动应用程序中的 web 服务的主要方法。

使用 REST 服务

有大量库和可用于使用 REST 服务的类和以下各小节讨论它们。有关使用 REST 服务的详细信息，请参阅[使用 RESTful Web 服务](#)。

HttpClient

[Microsoft HTTP 客户端库](#)提供 `HttpClient` 类, 该类用于发送和接收通过 HTTP 请求。它提供用于发送 HTTP 请求和从 URI 标识的资源的接收 HTTP 响应的功能。每个请求将作为异步操作发送。有关异步操作的详细信息, 请参阅[异步支持概述](#)。

`HttpResponseMessage` 类表示进行 HTTP 请求之后, 从 web 服务收到 HTTP 响应消息。它包含有关响应, 包括状态代码、标头和正文的信息。`HttpContent` 类表示的 HTTP 正文和内容标头, 如 `Content-Type` 和 `Content-Encoding`。可以使用任一读取内容 `ReadAs` 方法, 如 `ReadAsStringAsync` 和 `ReadAsByteArrayAsync` 根据数据的格式。

有关详细信息 `HttpClient` 类, 请参阅[创建 HttpClient 对象](#)。

HTTPWebRequest

调用 web 服务用于 `HTTPWebRequest` 涉及:

- 为特定 URI 创建请求实例。
- 在请求实例上设置各种 HTTP 属性。
- 检索 `HttpWebResponse` 请求。
- 读取响应数据。

例如, 下面的代码从美国检索数据国家/地区的故障排除库 web 服务:

```
var rxcul = "198440";
var request = HttpWebRequest.Create(string.Format(@"http://rxnav.nlm.nih.gov/REST/RxTerms/rxcui/{0}/allinfo",
rxcul));
request.ContentType = "application/json";
request.Method = "GET";

using (HttpWebResponse response = request.GetResponse() as HttpWebResponse)
{
    if (response.StatusCode != HttpStatusCode.OK)
        Console.WriteLine("Error fetching data. Server returned status code: {0}", response.StatusCode);
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        var content = reader.ReadToEnd();
        if(string.IsNullOrEmpty(content)) {
            Console.WriteLine("Response contained empty body...");
        }
        else {
            Console.WriteLine("Response Body: \r\n {0}", content);
        }

        Assert.NotNull(content);
    }
}
```

上面的示例创建 `HttpWebRequest` 这将返回 JSON 格式数据。返回的数据 `HttpWebResponse`, 从中 `StreamReader` 可以获得要读取的数据。

RestSharp

使用 REST 服务的另一种方法使用[RestSharp](#)库。RestSharp 封装 HTTP 请求, 包括用于检索结果作为原始字符串内容或反序列化的 C# 对象的支持。例如, 下面的代码在美国太平洋时区中发出了请求国家/地区的故障排除库 web 服务, 并检索结果为 JSON 格式字符串:


```

var request = new RestRequest(string.Format("{0}/allinfo", rxcul));
request.RequestFormat = DataFormat.Json;
var response = Client.Execute(request);
if(string.IsNullOrEmpty(response.Content) || response.StatusCode != System.Net.HttpStatusCode.OK) {
    return null;
}
rxTerm = DeserializeRxTerm(response.Content);

```

`DeserializeRxTerm` 是一个方法，将会从原始的 JSON 字符串 `RestSharp.RestResponse.Content` 属性并将其转换为 C# 对象。反序列化从 web 服务返回的数据将更高版本在本文中讨论。

NSURLConnection

除了单声道基中可用的类库 (BCL)，如 `HttpWebRequest`，和第三方 C# 库，如 `RestSharp`，特定于平台的类也已可用于使用 web 服务。例如，在 iOS、`NSURLConnection` 和 `NSMutableURLRequest` 可以使用类。

下面的代码示例演示如何调用美国国家/地区的故障排除库 web 服务使用 iOS 类：

```

var rxcul = "198440";
var request = new NSMutableURLRequest(new
NSURL(string.Format("http://rxnav.nlm.nih.gov/REST/RxTerms/rxcui/{0}/allinfo", rxcul)),
    NSURLRequestCachePolicy.ReloadRevalidatingCacheData, 20);
request["Accept"] = "application/json";

var connectionDelegate = new RxTermNSURLConnectionDelegate();
var connection = new NSURLConnection(request, connectionDelegate);
connection.Start();

public class RxTermNSURLConnectionDelegate : NSURLConnectionDelegate
{
    StringBuilder _ResponseBuilder;
    public bool IsFinishedLoading { get; set; }
    public string ResponseContent { get; set; }

    public RxTermNSURLConnectionDelegate()
        : base()
    {
        _ResponseBuilder = new StringBuilder();
    }

    public override void ReceivedData(NSURLConnection connection, NSData data)
    {
        if(data != null) {
            _ResponseBuilder.Append(data.ToString());
        }
    }

    public override void FinishedLoading(NSURLConnection connection)
    {
        IsFinishedLoading = true;
        ResponseContent = _ResponseBuilder.ToString();
    }
}

```

通常情况下，为使用 web 服务的特定于平台的类应限于正在将本机代码移植到 C# 的方案。如果可能，web 服务访问代码应为可移植，以便它可以是共享的跨平台。

ServiceStack

用于调用 web 服务的另一个选项是[服务堆栈](#)库。例如，下面的代码演示如何使用服务堆栈 `IServiceClient.GetAsync` 发出服务请求的方法：

```
client.GetAsync<CustomersResponse>("",
    (response) => {
        foreach(var c in response.Customers) {
            Console.WriteLine(c.CompanyName);
        }
    },
    (response, ex) => {
        Console.WriteLine(ex.Message);
    });
```

IMPORTANT

虽然工具 ServiceStack 和 RestSharp 可以轻松调用和使用 REST 服务, 但是有时重要使用 XML 或不符合标准的 JSON `_DataContract_` 序列化约定。如有必要, 调用请求并处理相应的序列化显式使用下面讨论的 ServiceStack.Text 库。

使用 rest 样式的数据

RESTful web 服务通常使用 JSON 消息来将数据返回到客户端。JSON 是基于文本的数据交换格式生成压缩的负载, 这会导致降低的带宽要求时发送数据。在此部分中, 将检查以使用 JSON 和纯旧 XML (POX) 的 RESTful 响应的机制。

System.Json

Xamarin 平台随附了现成的 json 支持。通过使用 `JsonObject`, 可以检索结果, 如下面的代码示例中所示:

```
var obj = JsonObject.Parse(json);
var properties = obj["rxtermsProperties"];
term.BrandName = properties["brandName"];
term.DisplayName = properties["displayName"];
term.Synonym = properties["synonym"];
term.FullName = properties["fullName"];
term.FullGenericName = properties["fullGenericName"];
term.Strength = properties["strength"];
```

但是, 务必请注意, `System.Json` 工具将整个数据加载到内存。

JSON.NET

[NewtonSoft JSON.NET](#) 库是用于序列化和反序列化 JSON 消息的常用的库。下面的代码示例演示如何使用 JSON.NET 反序列化到 C# 对象的 JSON 消息:

```
var term = new RxTerm();
var properties = JObject.Parse(json)["rxtermsProperties"];
term.BrandName = properties["brandName"].Value<string>();
term.DisplayName = properties["displayName"].Value<string>();
term.Synonym = properties["synonym"].Value<string>();
term.FullName = properties["fullName"].Value<string>();
term.FullGenericName = properties["fullGenericName"].Value<string>();
term.Strength = properties["strength"].Value<string>();
term.RxCUI = properties["rxcul"].Value<string>();
```

ServiceStack.Text

ServiceStack.Text 是用于处理与 ServiceStack 库的 JSON 序列化库。下面的代码示例演示如何分析 JSON 使用

```
ServiceStack.Text.JsonObject :
```

```
var result = JsonObject.Parse(json).Object("rxtermsProperties")
    .ConvertTo(x => new RxTerm {
        BrandName = x.Get("brandName"),
        DisplayName = x.Get("displayName"),
        Synonym = x.Get("synonym"),
        FullName = x.Get("fullName"),
        FullGenericName = x.Get("fullGenericName"),
        Strength = x.Get("strength"),
        RxTermDoseForm = x.Get("rxtermsDoseForm"),
        Route = x.Get("route"),
        RxCUI = x.Get("rxcul"),
        RxNormDoseForm = x.Get("rxnormDoseForm"),

    });
```

System.Xml.Linq

发生时使用基于 XML 的 REST web 服务, LINQ to XML 可分析的 XML 并填充 C# 对象内联, 如下面的代码示例中所示:

```
var doc = XDocument.Parse(xml);
var result = doc.Root.Descendants("rxtermsProperties")
    .Select(x=> new RxTerm()
    {
        BrandName = x.Element("brandName").Value,
        DisplayName = x.Element("displayName").Value,
        Synonym = x.Element("synonym").Value,
        FullName = x.Element("fullName").Value,
        FullGenericName = x.Element("fullGenericName").Value,
        //bind more here...
        RxCUI = x.Element("rxcul").Value,
    });
```

ASP.NET Web 服务 (ASMX)

ASMX 能够生成使用简单对象访问协议 (SOAP) 发送消息的 web 服务。SOAP 是一个独立于平台的独立于语言的协议用于构建和访问 web 服务。ASMX 服务的使用者不需要知道任何有关平台、对象模型或用于实现服务的编程语言。它们只需了解如何发送和接收 SOAP 消息。

SOAP 消息是 XML 文档包含以下元素:

- 名为的根元素 *信封* 标识 XML 文档作为 SOAP 消息。
- 一个可选 *标头* 包含特定于应用程序的信息, 例如身份验证数据的元素。如果 *标头* 存在元素, 则它必须是第一个子元素 *信封* 元素。
- 必需 *正文* 包含适用于接收方的 SOAP 消息的元素。
- 一个可选 *错误* 元素, 用于指示错误消息。如果 *错误* 元素存在, 它必须是子元素的 *正文* 元素。

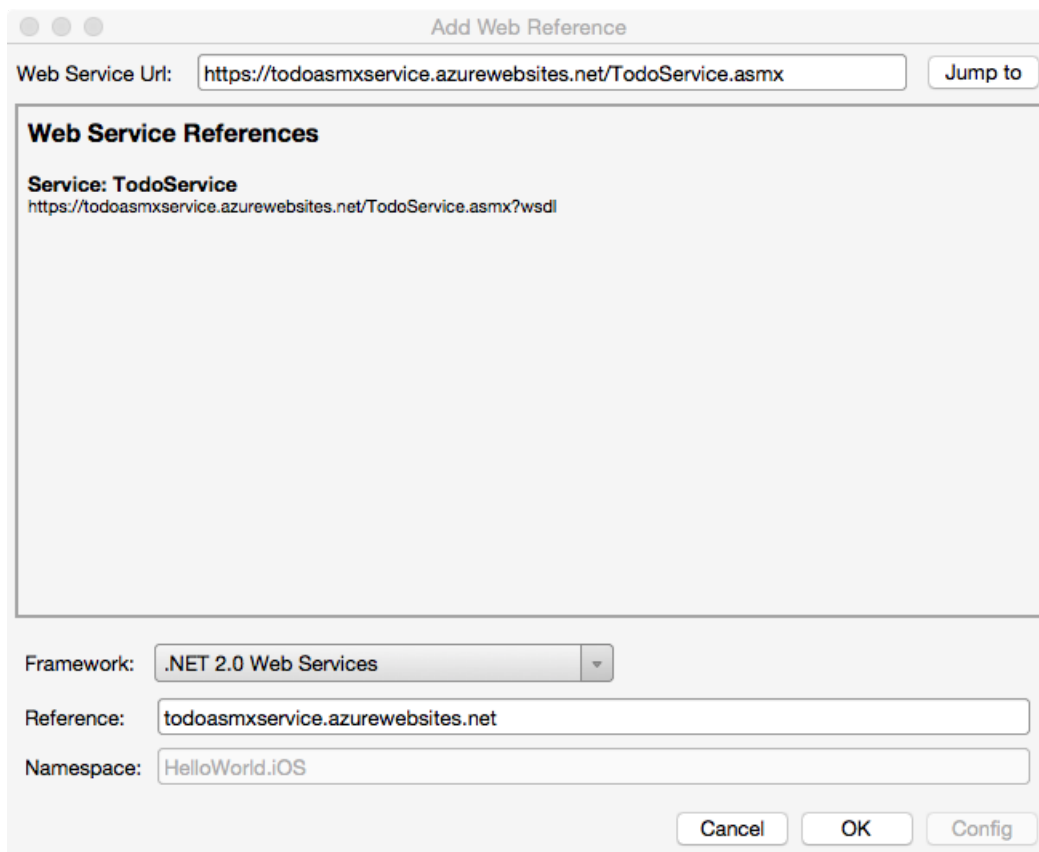
SOAP 可以对许多传输协议, 包括 HTTP、SMTP、TCP 和 UDP 进行操作。但是, ASMX 服务可以仅通过 HTTP 操作。Xamarin 平台支持标准 SOAP 1.1 实现通过 HTTP, 并且这会包含对许多标准 ASMX 服务配置的支持。

生成代理

A *代理* 必须生成要使用 ASMX 服务, 它允许应用程序连接到服务。代理是通过使用以定义的方法和关联的服务配置的服务元数据构造的。此元数据公开为生成的 web 服务的 Web 服务描述语言 (WSDL) 文档。使用适用于 Mac 的 Visual Studio 或 Visual Studio 将 web 服务的 web 引用添加到特定于平台的项目生成代理。

Web 服务 URL 可以是托管远程源或本地文件系统资源可通过访问 `file:///` 路径前缀, 例如:

```
file:///Users/myUserName/projects/MyProjectName/service.wsdl
```



这将在项目的 Web 或服务引用文件夹中生成代理。由于生成代理代码，它不应修改。

手动将代理添加到项目

如果必须使用兼容的工具生成的现有代理服务器，包括你的项目的一部分时，可以使用此输出。在适用于 Mac 的 Visual Studio，使用 **添加文件...** 将代理添加的菜单选项。此外，这需要 *System.Web.Services.dll* 使用显式引用添加引用... 对话框。

使用代理

生成的代理类提供用于使用 web 服务使用的异步编程模型 (APM) 设计模式的方法。在此模式中异步操作实现这两个方法名为 *BeginOperationName* 和 *EndOperationName*，其开始和结束异步操作。

BeginOperationName 方法开始异步操作并返回一个对象，实现 *IAsyncResult* 接口。在调用 *BeginOperationName*，应用程序可以继续调用的线程上执行指令，同时异步操作在有线程池线程上的发生。

每次调用 *BeginOperationName*，应用程序还应调用 *EndOperationName* 来获取该操作的结果。返回值 *EndOperationName* 同步 web 服务方法返回的类型相同。下面的代码示例演示了此示例：

```
public async Task<List<ToDoItem>> RefreshDataAsync ()
{
    ...
    var todoItems = await Task.Factory.FromAsync<ASMXService.ToDoItem[]> (
        todoService.BeginGetTodoItems,
        todoService.EndGetTodoItems,
        null,
        TaskCreationOptions.None);
    ...
}
```

任务并行库 (TPL) 可以简化通过封装在同一个异步操作来使用的 APM begin/end 方法对的过程 *Task* 对象。此封装提供的多个重载的 *Task.Factory.FromAsync* 方法。此方法创建 *Task* 执行 *ToDoService.EndGetTodoItems* 方法一次 *ToDoService.BeginGetTodoItems* 方法完成时，与 *null* 表示没有数据传递到参数 *BeginGetTodoItems* 委托。最后，值 *TaskCreationOptions* 枚举指定应使用的创建和执行的任务的默认行为。

APM 有关的详细信息，请参阅 [异步编程模型](#) 和 [TPL 和传统 .NET Framework 异步编程](#) MSDN 上。

有关使用 ASMX 服务的详细信息，请参阅[使用 ASP.NET Web 服务 \(ASMX\)](#)。

Windows Communication Foundation (WCF)

WCF 是 Microsoft 的统一的框架，用于构建面向服务的应用程序。它允许开发人员生成安全、可靠、事务处理，且可互操作的分布式应用程序。

WCF 描述了与各种不同的协定包括以下服务：

- **数据协定**– 定义构成一条消息中的内容的基础的数据结构。
- **消息协定搭配**– 撰写从现有数据协定的消息。
- **错误协定**– 允许指定的自定义 SOAP 错误。
- **服务协定**– 指定服务支持的操作和消息所需的每个操作与之进行交互。它们还指定可以与每个服务操作相关联的任何自定义错误行为。

ASP.NET Web 服务 (ASMX) 和 WCF，之间的差异，但是务必了解 WCF 支持相同的功能，提供的 ASMX 服务 – 通过 HTTP 的 SOAP 消息。

一般情况下，Xamarin 平台支持相同的客户端将一部分附带 Silverlight 运行时的 WCF。这包括 WCF 的最常见的编码和协议实现 — 通过 HTTP 文本编码 SOAP 消息传输协议使用 `BasicHttpBinding` 类。此外，WCF 支持需要仅在一个用于生成代理的 Windows 环境中可用的工具的用法。

有关使用 Xamarin 平台来使用 WCF 的详细信息 web 服务与 `BasicHttpBinding` 类，请参阅[演练-使用 WCF](#)。

生成代理

A 代理必须生成要使用 WCF 服务，它允许应用程序连接到服务。代理是通过使用以定义的方法和关联的服务配置的服务元数据构造的。由 web 服务生成 Web 服务描述语言 (WSDL) 文档形式公开此元数据。可以使用在 Visual Studio 2017 Microsoft WCF Web 服务引用提供程序将 web 服务的服务引用添加到 .NET 标准库生成代理。

创建在 Visual Studio 2017 中使用 Microsoft WCF Web 服务引用提供程序的代理的替代方法是使用 ServiceModel 元数据实用工具 (svcutil.exe)。有关详细信息，请参阅[ServiceModel 元数据实用工具 \(Svcutil.exe\)](#)。

配置代理服务器

配置生成的代理将通常采用两个配置参数（具体取决于 SOAP 1.1/ASMX 或 WCF）在初始化期间：`EndpointAddress` 和/或关联的绑定信息，如下面的示例中所示：

```
var binding = new BasicHttpBinding () {
    Name= "basicHttpBinding",
    MaxReceivedMessageSize = 67108864,
};

binding.ReaderQuotas = new System.Xml.XmlDictionaryReaderQuotas() {
    MaxArrayLength = 2147483646,
    MaxStringContentLength = 5242880,
};

var timeout = new TimeSpan(0,1,0);
binding.SendTimeout= timeout;
binding.OpenTimeout = timeout;
binding.ReceiveTimeout = timeout;

client = new Service1Client (binding, new EndpointAddress ("http://192.168.1.100/Service1.svc"));
```

绑定用于指定传输、编码和协议详细信息所需的应用程序和服务相互通信。`BasicHttpBinding` 指定，将通过 HTTP 传输协议发送文本编码 SOAP 消息。指定终结点地址启用应用程序连接到 WCF 服务的不同实例，前提是有多个已发布的实例。

使用代理

生成的代理类提供用于使用使用异步编程模型 (APM) 设计模式的 web 服务方法。在此模式中，异步操作实现这两个方法名为 *BeginOperationName* 和 *EndOperationName*，其开始和结束异步操作。

BeginOperationName 方法开始异步操作并返回一个对象，实现 `IAsyncResult` 接口。在调用 *BeginOperationName*，应用程序可以继续在此调用的线程上执行指令，同时异步操作在多线程池线程上发生。

每次调用 *BeginOperationName*，应用程序还应调用 *EndOperationName* 来获取该操作的结果。返回值 *EndOperationName* 同步 web 服务方法返回的类型相同。下面的代码示例演示了此示例：

```
public async Task<List<TodoItem>> RefreshDataAsync ()
{
    ...
    var todoItems = await Task.Factory.FromAsync <ObservableCollection<TodoWCFService.TodoItem>> (
        todoService.BeginGetTodoItems,
        todoService.EndGetTodoItems,
        null,
        TaskCreationOptions.None);
    ...
}
```

任务并行库 (TPL) 可以简化通过封装在同一个异步操作来使用的 APM begin/end 方法对的过程 `Task` 对象。此封装提供的多个重载的 `Task.Factory.FromAsync` 方法。此方法创建 `Task` 执行 `TodoServiceClient.EndGetTodoItems` 方法一次 `TodoServiceClient.BeginGetTodoItems` 方法完成时，与 `null` 表示没有数据传递到参数 `BeginGetTodoItems` 委托。最后，值 `TaskCreationOptions` 枚举指定应使用的创建和执行的任务的默认行为。

APM 有关的详细信息，请参阅 [异步编程模型](#) 和 [TPL 和传统 .NET Framework 异步编程](#) MSDN 上。

有关使用 WCF 服务的详细信息，请参阅 [使用 Windows Communication Foundation \(WCF\) Web 服务](#)。

使用传输安全

WCF 服务可能会使用传输级安全，以防出现截获的消息。Xamarin 平台支持采用使用 SSL 的传输级安全的绑定。但是，有时可能无法在其中堆栈可能需要验证的证书，这会导致意外行为。验证可以通过注册重写

`ServerCertificateValidationCallback` 委托之前调用服务，如下面的代码示例中所示：

```
System.Net.ServicePointManager.ServerCertificateValidationCallback +=
(se, cert, chain, sslerror) => { return true; };
```

这将保持时忽略服务器端证书验证的传输加密。但是，此方法将有效地忽略与证书关联的信任问题和可能不合适。有关详细信息，请参阅 [恭敬使用受信任的根上 mono project.com](#)。

使用客户端凭据安全

WCF 服务还可能要求使用凭据进行身份验证的服务客户端。Xamarin 平台不支持 Ws-security 协议，允许客户端在 SOAP 消息信封内发送凭据。但是，Xamarin 平台支持能够向服务器发送 HTTP 基本身份验证凭据，通过指定相应

`ClientCredentialType`：

```
basicHttpBinding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Basic;
```

然后，可以指定基本身份验证凭据：

```
client.ClientCredentials.UserName.UserName = @"foo";
client.ClientCredentials.UserName.Password = @"mrsnuggles";
```

在上面的示例中，如果你收到消息“用尽 trampolines 类型 0”则可以增加类型 0 trampolines 数加上 `-aot "trampolines={number of trampolines}"` 到生成的自变量。有关详细信息，请参阅 [故障排除](#)。

详细了解 HTTP 基本身份验证，但在 REST web 服务的上下文，请参阅 [RESTful Web 服务进行身份验证](#)。

总结

本指南演示了如何使用另一个 web 服务技术。涵盖的主题包括与 REST 服务、SOAP 服务和 Windows Communication Foundation 服务通信。

相关链接

- [WebServices 示例](#)
- [Xamarin.Forms 中的 web 服务](#)
- [ServiceModel 元数据实用工具 \(svcutil.exe\)](#)
- [BasicHttpBinding](#)

演练-使用 WCF

2018/6/22 • [Edit Online](#)

本演练介绍如何使用 Xamarin 生成的移动应用程序可以使用 WCF web 服务使用 BasicHttpBinding 类。

它是移动应用程序能够与后端系统通信的一个常见要求。有许多选项和选项用于后端框架，其中之一是 [Windows Communication Foundation \(WCF\)](#)。本演练将提供的 Xamarin 移动应用程序可以使用 WCF 服务使用示例 BasicHttpBinding 类。本演练包括以下主题：

1. **创建 WCF 服务**-在本部分中，我们将创建具有两种方法非常基本的 WCF 服务。第一种方法将需要一个字符串参数，而另一种方法将采用 C# 对象。本部分还将讨论如何配置开发人员的工作站，以允许远程访问 WCF 服务。
2. **创建一个 Xamarin.Android 应用程序**-一旦创建 WCF 服务后，我们将创建一个简单的 Xamarin.Android 应用程序将使用 WCF 服务。本部分将介绍如何创建一个 WCF 服务的代理类来促进与 WCF 服务的通信。
3. **创建一个 Xamarin.iOS 应用程序**-本教程的最后一部分涉及到创建的简单的 Xamarin.iOS 应用程序将使用 WCF 服务。

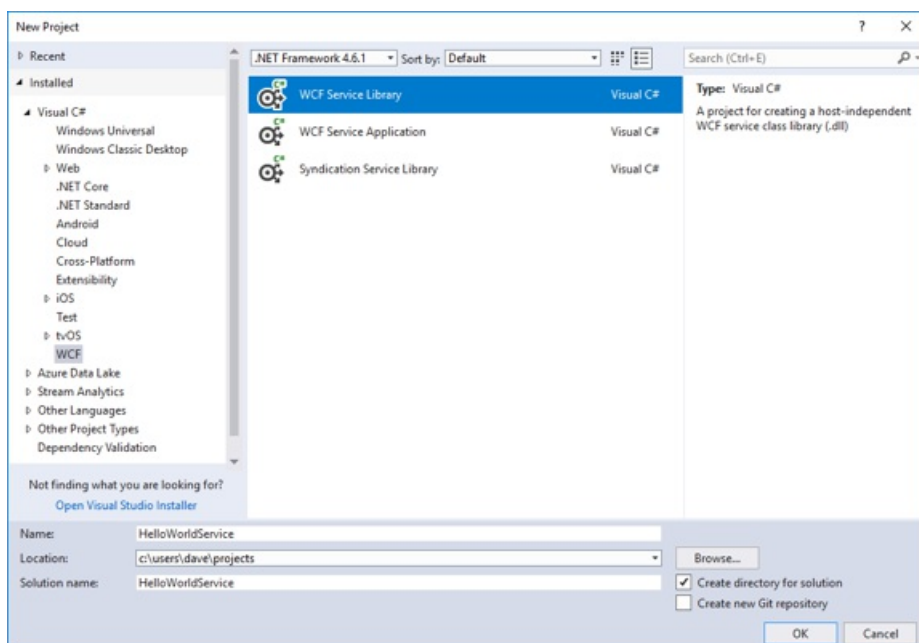
要求

本演练假定你已具备一定的创建和使用 WCF 服务。

创建 WCF 服务

前面我们的第一个任务是创建移动应用程序中，与进行通信的 WCF 服务。

1. 启动 Visual Studio 2017，并创建新项目。
2. 在新项目对话框中，选择 **WCF > WCF 服务库模板**，并将解决方案命名解决方案 `HelloWorldService`：



3. 在解决方案资源管理器，添加一个名为的新类 `HelloWorldData` 到项目：


```

using System.Runtime.Serialization;

namespace HelloWorldService
{
    [DataContract]
    public class HelloWorldData
    {
        [DataMember]
        public bool SayHello { get; set; }

        [DataMember]
        public string Name { get; set; }

        public HelloWorldData()
        {
            Name = "Hello ";
            SayHello = false;
        }
    }
}

```

4. 在解决方案资源管理器，重命名 `IService1.cs` 到 `IHelloWorldService.cs`，并将重命名 `Service1.cs` 到 `HelloWorldService.cs`。

5. 在解决方案资源管理器，打开 `IHelloWorldService.cs`，将代码替换为以下代码：

```

using System.ServiceModel;

namespace HelloWorldService
{
    [ServiceContract]
    public interface IHelloWorldService
    {
        [OperationContract]
        string SayHelloTo(string name);

        [OperationContract]
        HelloWorldData GetHelloData(HelloWorldData helloWorldData);
    }
}

```

此服务提供两种方法 – 其中一个采用字符串参数，另一个使用.NET 对象。

6. 在解决方案资源管理器，打开 `HelloWorldService.cs`，将代码替换为以下代码：

```

using System;

namespace HelloWorldService
{
    public class HelloWorldService : IHelloWorldService
    {
        public HelloWorldData GetHelloData(HelloWorldData helloWorldData)
        {
            if (helloWorldData == null)
                throw new ArgumentException("helloWorldData");

            if (helloWorldData.SayHello)
                helloWorldData.Name = "Hello World to {helloWorldData.Name}";

            return helloWorldData;
        }

        public string SayHelloTo(string name)
        {
            return "Hello World to you, {name}";
        }
    }
}

```

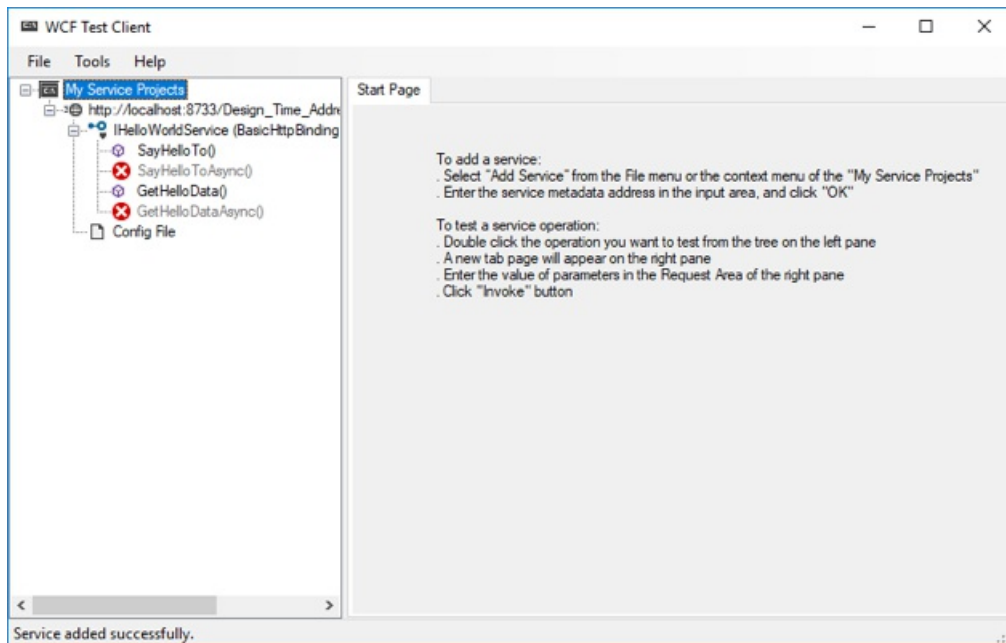
7. 在解决方案资源管理器, 打开 `App.config`, 更新 `name` 属性 `<service>` 节点, `contract` 属性 `<endpoint>` 节点, 和 `baseAddress` 属性 `<add>` 节点:

```

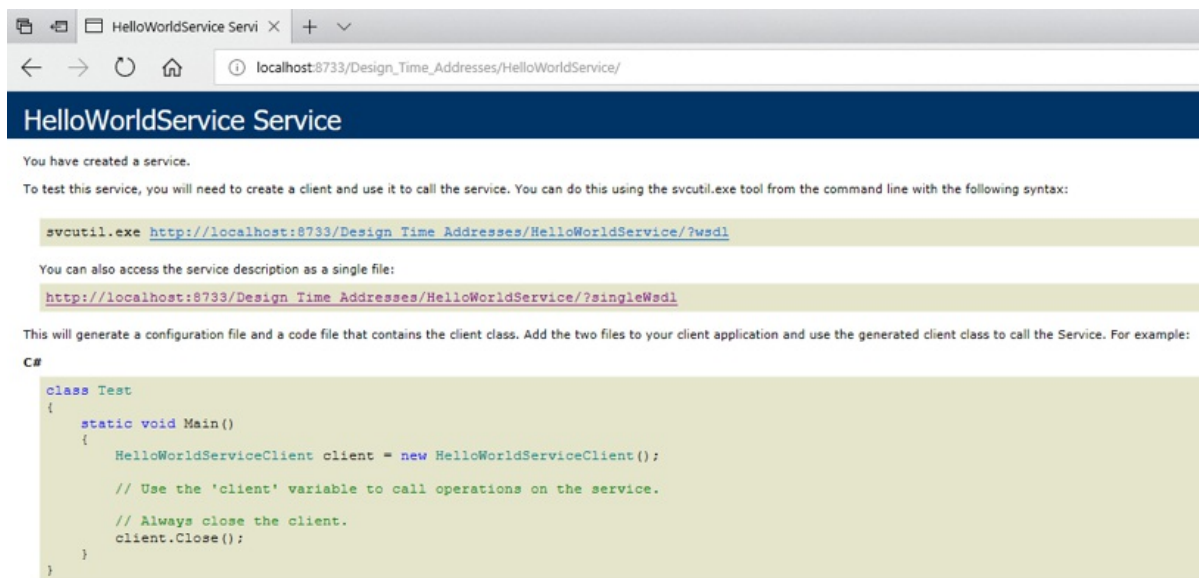
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    ...
    <services>
        <service name="HelloWorldService.HelloWorldService">
            <endpoint address="" binding="basicHttpBinding"
contract="HelloWorldService.IHelloWorldService">
                <identity>
                    <dns value="localhost" />
                </identity>
            </endpoint>
            <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
            <host>
                <baseAddresses>
                    <add baseAddress="http://localhost:8733/Design_Time_Addresses/HelloWorldService/" />
                </baseAddresses>
            </host>
        </service>
    </services>
    ...
</configuration>

```

8. 生成并运行 WCF 服务。WCF 测试客户端将承载该服务:



9. 运行 WCF 测试客户端，启动浏览器并导航到的 WCF 服务终结点：



IMPORTANT

以下部分才有必要，如果你需要接受远程连接 Windows 10 工作站上。如果你有另一种平台上部署 WCF 服务，则可以忽略该节。

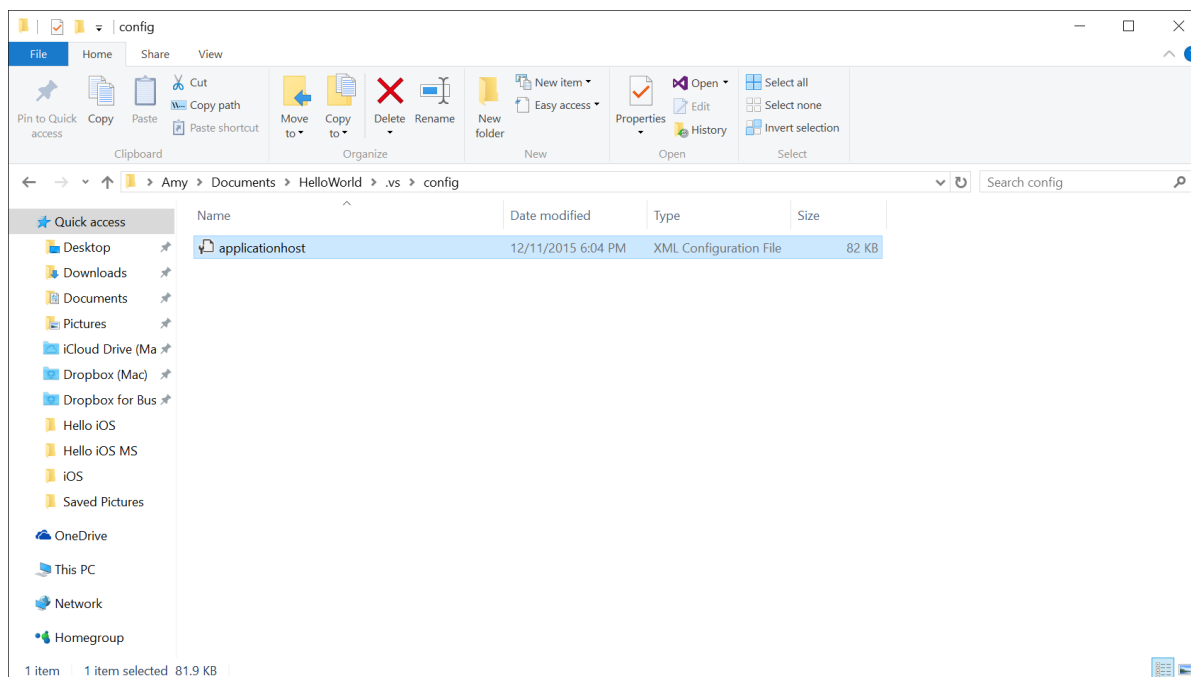
配置远程访问到 IIS Express

当连接仅来自本地计算机，则承载 WCF 本地已足够。但是，远程设备（如 Android 设备或 iPhone）不会对本地 WCF 服务的任何访问。因此，本部分介绍如何配置 Windows 10 和 IIS Express 接受远程连接：

1. 接受远程连接配置 IIS Express -此步骤包括编辑的 IIS Express，以接受特定端口上的远程连接配置文件，然后设置 IIS Express，以接受的传入流量的规则。
2. 将异常添加到 Windows 防火墙-你必须打开通过远程应用程序可以使用与 WCF 服务进行通信的 Windows 防火墙的端口。

你将需要知道你的工作站的 IP 地址。此示例的目的，我们将假定我们的工作站具有 192.168.1.143 的 IP 地址。

3. 让我们开始通过配置 IIS Express 以侦听外部请求。我们可以执行此操作通过 IIS express 在编辑配置文件 [solutiondirectory]\.vs\config\applicationhost.config, 如以下屏幕截图中所示:



找到 `site` 同名元素 `HelloWorldWcfHost`。其外观应类似于以下 XML 代码段:

```
<site name="HelloWorldWcfHost" id="2">
  <application path="/" applicationPool="Clr4IntegratedAppPool">
    <virtualDirectory path="/" physicalPath="\\vmware-host\Shared
Folders\tom\work\xamarin\code\private-samples\webservices\HelloWorld\HelloWorldWcfHost" />
  </application>
  <bindings>
    <binding protocol="http" bindingInformation="*:8733:localhost" />
  </bindings>
</site>
```

我们将需要添加另一个 `binding` 以打开端口 8734 到外部流量。添加以下 XML 到 `bindings` 元素, 将替换为你自己的 IP 地址的 IP 地址:

```
<binding protocol="http" bindingInformation="*:8734:192.168.1.143" />
```

这会将配置 IIS Express, 以接受来自任何远程 IP 地址的端口 8734 上的计算机的外部 IP 地址的 HTTP 流量。上面的代码段假定运行 IIS Express 的计算机的 IP 地址是 192.168.1.143。后所做的更改, `bindings` 元素应如下所示:

```
<site name="HelloWorldWcfHost" id="2">
  <application path="/" applicationPool="Clr4IntegratedAppPool">
    <virtualDirectory path="/" physicalPath="\\vmware-host\Shared
Folders\tom\work\xamarin\code\private-samples\webservices\HelloWorld\HelloWorldWcfHost" />
  </application>
  <bindings>
    <binding protocol="http" bindingInformation="*:8733:localhost" />
    <binding protocol="http" bindingInformation="*:8734:192.168.1.143" />
  </bindings>
</site>
```

4. 接下来, 我们需要配置 IIS Express 接受端口 8734 上的传入连接。启动管理命令提示符, 并运行此命令:

```
> netsh http add urlacl url=http://192.168.1.143:9608/ user=everyone
```

5. 最后一步是配置 Windows 防火墙以允许在端口 8734 上的外部流量。从管理命令提示符，运行以下命令：

```
> netsh advfirewall firewall add rule name="IISExpressXamarin" dir=in protocol=tcp localport=8734  
profile=private remoteip=localsubnet action=allow
```

此命令将作为 Windows 10 工作站位于同一子网允许端口 8734 从所有设备上的传入流量。

您已创建非常基本的 WCF 服务承载在 IIS Express 中，将接受来自其他设备或我们子网上的计算机的传入连接。你可以通过运行你的应用程序以及访问来测试此扩展

`http://localhost:8733/Design_Time_Addresses/HelloWorldService/` 工作站上和

`http://192.168.1.143:8734/Design_Time_Addresses/HelloWorldService/` 从子网上的另一台计算机。

若要允许 IIS Express，以保持运行并提供服务，请关闭编辑并继续选项 **项目属性** > **Web** > **调试器**。

创建 Web 服务代理

应用程序可以使用服务之前，必须为 WCF 服务，创建 web 服务代理。这可以通过以下操作实现：

1. 添加名为.NET 标准类库 `HelloWorldServiceProxy`，和删除项目中的任何类。
2. 运行 `HelloWorldService` 项目。
3. 与 `HelloWorldService` 运行项目中，添加一个新连接的服务到项目中，使用**Microsoft WCF Web 服务引用**的提供程序。
4. 在**服务终结点**选项卡**配置 WCF Web 服务引用**对话框中，单击**发现**按钮，删除 `mex` 结尾处检测到中的终结点**URI**下拉列表中，输入 `HelloWorldServiceProxy` 作为**Namespace**，然后单击**下一步**按钮。
5. 在**数据类型**选项卡**配置 WCF Web 服务引用**对话框中，通过单击**接受默认设置**下一步按钮。
6. 在**客户端**选项卡**配置 WCF Web 服务引用**对话框中，确保**公共**复选框已选中，然后单击**完成**按钮。
7. 生成 `HelloWorldServiceProxy` 项目。

NOTE

创建在 Visual Studio 2017 中使用 Microsoft WCF Web 服务引用提供程序的代理的替代方法是使用 ServiceModel 元数据实用工具 (svcutil.exe)。有关详细信息，请参阅[ServiceModel 元数据实用工具 \(Svcutil.exe\)](#)。

创建 Xamarin.Android 应用程序

WCF 服务代理可供 Xamarin.Android 应用程序，如下所示：

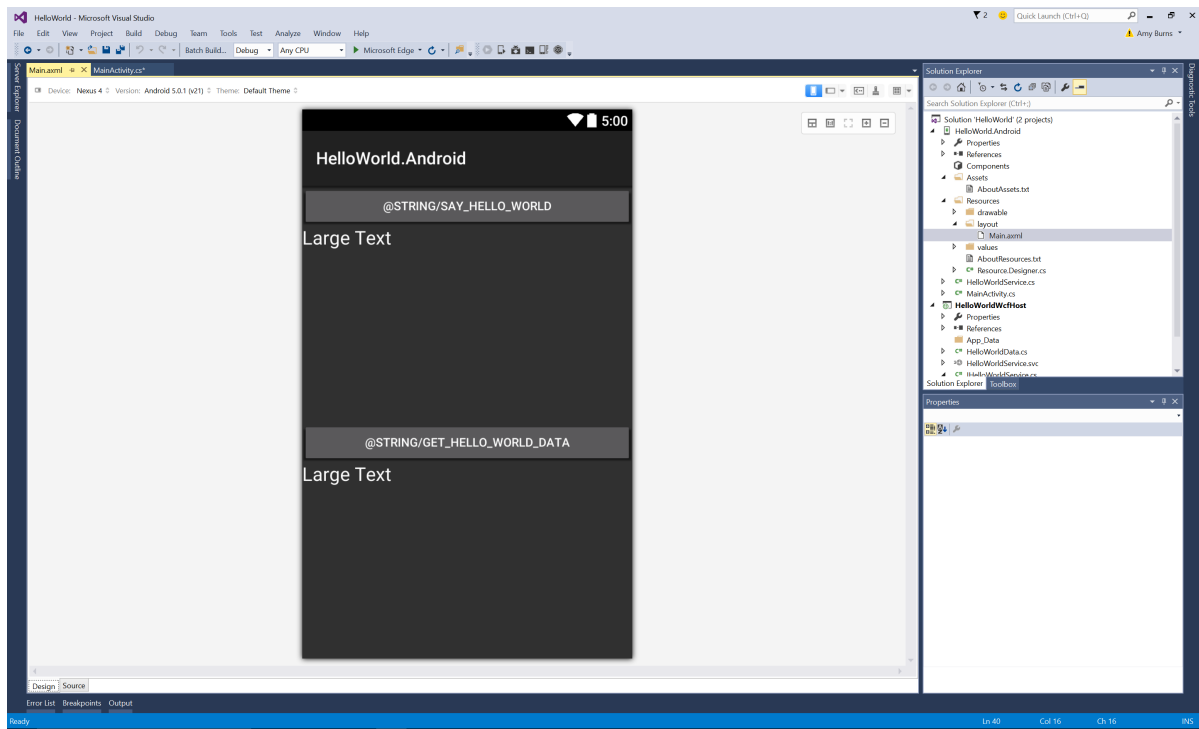
1. 在 Visual Studio 中，将新的空白 Android 项目添加到解决方案并将其命名 `HelloWorld.Android`。
2. 在 `HelloWorld.Android` 项目中，添加对引用 `HelloWorldServiceProxy` 项目中和的引用来 `System.ServiceModel` 命名空间。
3. 在**解决方案资源管理器**，打开 `Resources/layout/main.axml` 并将现有的 XML 替换为以下 XML：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="0px"
        android:layout_weight="1">
        <Button
            android:id="@+id/sayHelloWorldButton"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/say_hello_world" />
        <TextView
            android:text="Large Text"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/sayHelloWorldTextView" />
    </LinearLayout>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="0px"
        android:layout_weight="1">
        <Button
            android:id="@+id/getHelloWorldDataButton"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/get_hello_world_data" />
        <TextView
            android:text="Large Text"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/getHelloWorldDataTextView" />
    </LinearLayout>
</LinearLayout>

```

以下屏幕截图显示了在设计器的 UI:



4. 在解决方案资源管理器，打开 `Resources/values/Strings.xml` 并添加以下 XML:

```
<string name="say_hello_world">Say Hello World</string>
<string name="get_hello_world_data">Get Hello World data</string>
```

5. 在解决方案资源管理器，打开 `MainActivity.cs` 和替换为以下代码替换现有代码:

```
[Activity(Label = "HelloWorld.Android", MainLauncher = true)]
public class MainActivity : Activity
{
    static readonly EndpointAddress Endpoint = new EndpointAddress("
<insert_WCF_service_endpoint_here>");

    HelloWorldServiceClient _client;
    Button _getHelloWorldDataButton;
    TextView _getHelloWorldDataTextView;
    Button _sayHelloWorldButton;
    TextView _sayHelloWorldTextView;
    ...
}
```

替换 `<insert_WCF_service_endpoint_here>` 与 WCF 终结点的地址。

6. 在 `MainActivity.cs`，修改 `OnCreate` 方法，以便它包含以下代码:

```

protected override void OnCreate(Bundle savedInstanceState)
{
    base.OnCreate(bundle);

    SetContentView(Resource.Layout.Main);

    InitializeHelloWorldServiceClient();

    // This button will invoke the GetHelloWorldData - the method that takes a C# object as a
    parameter.
    _getHelloWorldDataButton = FindViewById<Button>(Resource.Id.getHelloWorldDataButton);
    _getHelloWorldDataButton.Click += GetHelloWorldDataButtonOnClick;
    _getHelloWorldDataTextView = FindViewById<TextView>(Resource.Id.getHelloWorldDataTextView);

    // This button will invoke SayHelloWorld - this method takes a simple string as a parameter.
    _sayHelloWorldButton = FindViewById<Button>(Resource.Id.sayHelloWorldButton);
    _sayHelloWorldButton.Click += SayHelloWorldButtonOnClick;
    _sayHelloWorldTextView = FindViewById<TextView>(Resource.Id.sayHelloWorldTextView);
}

```

上面的代码中初始化类的实例变量和电线事件处理程序。

7. 在 `MainActivity.cs` , 通过添加以下两种方法来实例化客户端代理类:

```

void InitializeHelloWorldServiceClient()
{
    BasicHttpBinding binding = CreateBasicHttpBinding();
    _client = new HelloWorldServiceClient(binding, Endpoint);
}

static BasicHttpBinding CreateBasicHttpBinding()
{
    BasicHttpBinding binding = new BasicHttpBinding
    {
        Name = "basicHttpBinding",
        MaxBufferSize = 2147483647,
        MaxReceivedMessageSize = 2147483647
    };

    TimeSpan timeout = new TimeSpan(0, 0, 30);
    binding.SendTimeout = timeout;
    binding.OpenTimeout = timeout;
    binding.ReceiveTimeout = timeout;
    return binding;
}

```

上面的代码实例化和初始化 `HelloWorldServiceClient` 对象。

8. 在 `MainActivity.cs` , 甚至为添加处理程序中的两个按钮 `Activity` :


```

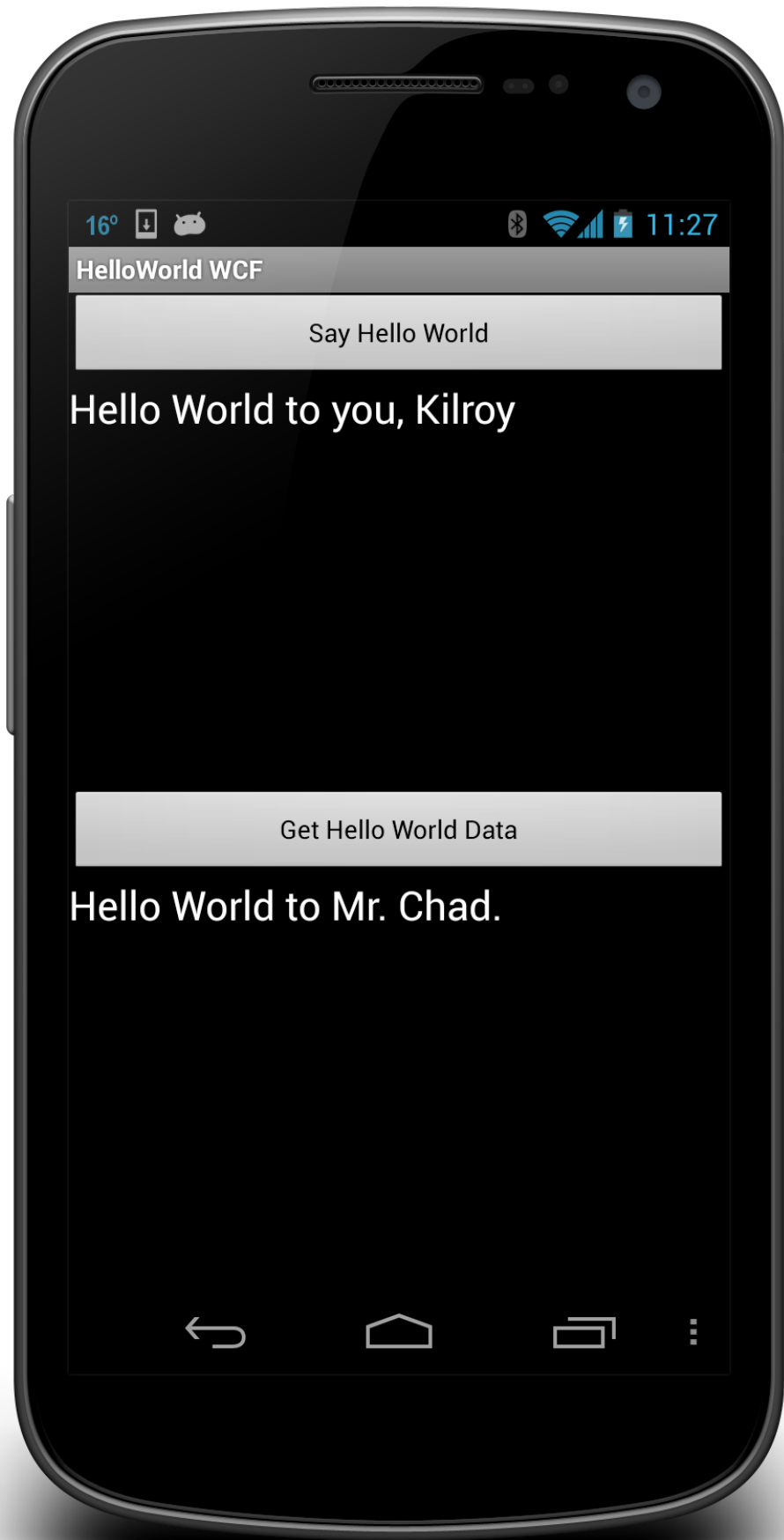
async void GetHelloWorldDataButtonOnClick(object sender, EventArgs e)
{
    var data = new HelloWorldData
    {
        Name = "Mr. Chad",
        SayHello = true
    };

    _getHelloWorldDataTextView.Text = "Waiting for WCF...";
    HelloWorldData result;
    try
    {
        result = await _client.GetHelloDataAsync(data);
        _getHelloWorldDataTextView.Text = result.Name;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

async void SayHelloWorldButtonOnClick(object sender, EventArgs e)
{
    _sayHelloWorldTextView.Text = "Waiting for WCF...";
    try
    {
        var result = await _client.SayHelloToAsync("Kilroy");
        _sayHelloWorldTextView.Text = result;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

9. 运行应用程序, 请确保 WCF 服务正在运行, 并且两个按钮上单击。应用程序将 WCF 以异步方式调用, 条件是 `Endpoint` 正确设置字段:



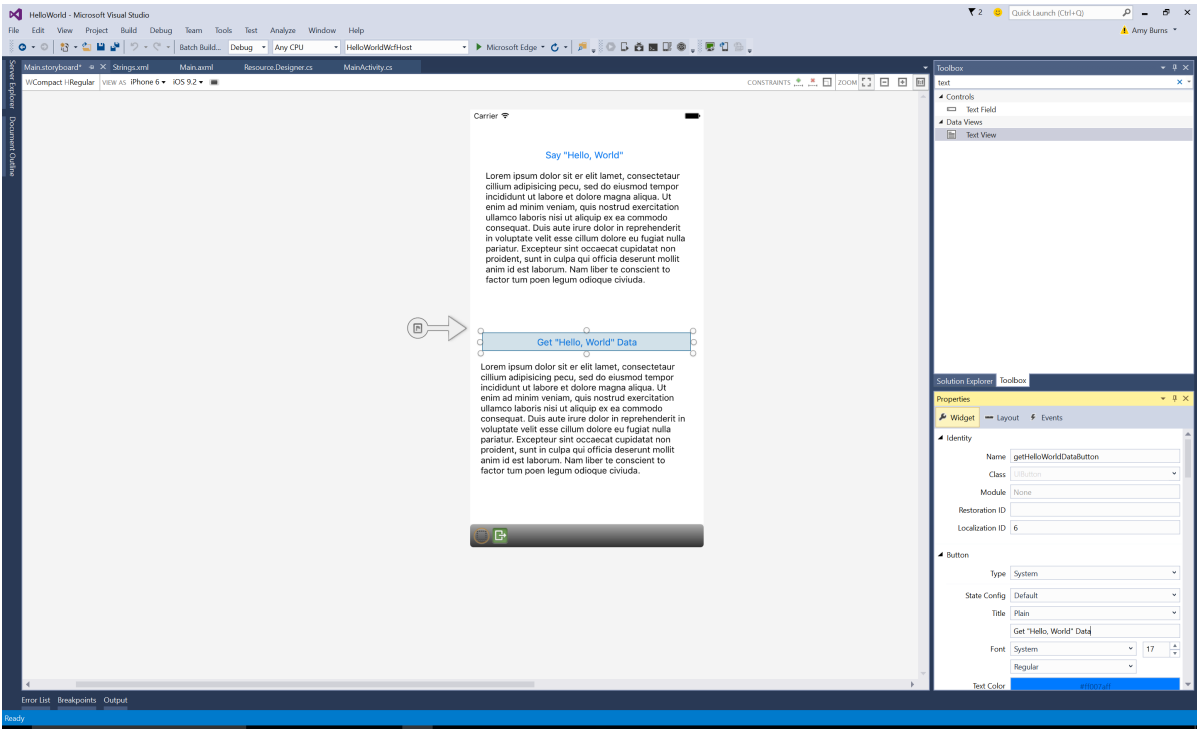
创建 Xamarin.iOS 应用程序

WCF 服务代理可供 Xamarin.iOS 应用程序，如下所示：

1. 在 Visual Studio 中，添加新的 iPhone 单视图应用程序项目合并为解决方案并将其命名 `HelloWorld.iOS`。
2. 在 `HelloWorld.iOS` 项目中，添加对引用 `HelloWorldServiceProxy` 项目中和的引用来 `System.ServiceModel` 命名空间。
3. 在解决方案资源管理器，双击 `Main.storyboard` 以在 iOS 设计器中打开该文件。然后，添加以下 `UIButton` 和 `UITextView` 控件：

| | 名称 | 标题 |
|-------------------------|--------------------------------------|--------------------|
| <code>UIButton</code> | <code>sayHelloWorldButton</code> | 说出"Hello, World" |
| <code>UITextView</code> | <code>sayHelloWorldText</code> | |
| <code>UIButton</code> | <code>getHelloWorldDataButton</code> | 获取"Hello, World"数据 |
| <code>UITextView</code> | <code>getHelloWorldDataText</code> | |

添加控件后，UI 应类似于下面的屏幕快照：



4. 在解决方案资源管理器，打开 `ViewController.cs` 并添加以下代码：

```
public partial class ViewController : UIViewController
{
    static readonly EndpointAddress Endpoint = new EndpointAddress("<insert_WCF_service_endpoint_here>");
    HelloWorldServiceClient _client;
    ...
}
```

替换 `<insert_WCF_service_endpoint_here>` 与 WCF 终结点的地址。

5. 在 `ViewController.cs` , 更新 `ViewDidLoad` 方法, 以便它如下所示:

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();
    InitializeHelloWorldServiceClient();

    getHelloWorldDataButton.TouchUpInside += GetHelloWorldDataButton_TouchUpInside;
    sayHelloWorldButton.TouchUpInside += SayHelloWorldButton_TouchUpInside;
}
```

6. 在 `ViewController.cs` , 添加 `InitializeHelloWorldServiceClient` 和 `CreateBasicHttpBinding` 方法:

```
void InitializeHelloWorldServiceClient()
{
    BasicHttpBinding binding = CreateBasicHttpBinding();
    _client = new HelloWorldServiceClient(binding, Endpoint);
}

static BasicHttpBinding CreateBasicHttpBinding()
{
    BasicHttpBinding binding = new BasicHttpBinding
    {
        Name = "basicHttpBinding",
        MaxBufferSize = 2147483647,
        MaxReceivedMessageSize = 2147483647
    };

    TimeSpan timeout = new TimeSpan(0, 0, 30);
    binding.SendTimeout = timeout;
    binding.OpenTimeout = timeout;
    binding.ReceiveTimeout = timeout;
    return binding;
}
```

7. 在 `ViewController.cs` , 添加事件处理程序 `TouchUpInside` 对两个事件 `UIButton` 实例:

```

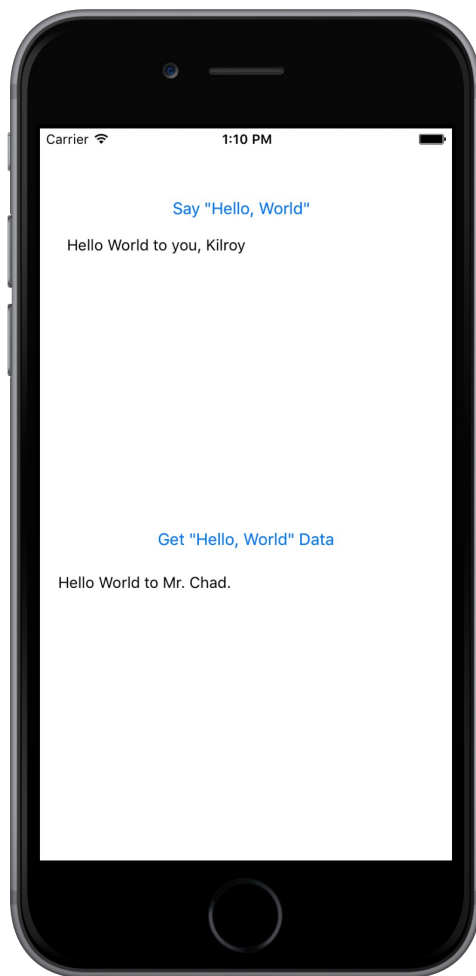
async void GetHelloWorldDataButton_TouchUpInside(object sender, EventArgs e)
{
    getHelloWorldDataText.Text = "Waiting for WCF...";
    var data = new HelloWorldData
    {
        Name = "Mr. Chad",
        SayHello = true
    };

    HelloWorldData result;
    try
    {
        result = await _client.GetHelloDataAsync(data);
        getHelloWorldDataText.Text = result.Name;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

async void SayHelloWorldButton_TouchUpInside(object sender, EventArgs e)
{
    sayHelloWorldText.Text = "Waiting for WCF...";
    try
    {
        var result = await _client.SayHelloToAsync("Kilroy");
        sayHelloWorldText.Text = result;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

8. 运行应用程序, 请确保 WCF 服务正在运行, 并且两个按钮上单击。应用程序将 WCF 以异步方式调用, 条件是 `Endpoint` 正确设置字段:



总结

本教程介绍如何使用 Xamarin.Android 和 Xamarin.iOS 的移动应用程序中的 WCF 服务使用。它介绍了如何创建 WCF 服务，并且说明了如何配置 Windows 10 和 IIS Express 以接受来自远程设备的连接。然后，介绍了如何生成 WCF 代理客户端，并演示如何在 Xamarin.Android 和 Xamarin.iOS 应用程序中使用客户端代理。

相关链接

- [HelloWorld \(示例\)](#)
- [使用 WCF 开发面向服务的应用程序](#)
- [如何：创建 Windows Communication Foundation 客户端](#)
- [ServiceModel 元数据实用工具 \(svcutil.exe\)](#)

部署和测试

2018/10/26 • [Edit Online](#)

本部分包含的指南介绍如何测试应用程序、优化应用程序的性能、做好发布准备、使用证书对应用程序进行签名以及将它发布到应用商店。

应用程序包大小

本文将讨论 Xamarin.Android 应用程序包的组成部分，以及可用于在开发的调试和发布阶段进行高效包部署的相关策略。

构建应用

本部分介绍生成过程的工作原理，并说明如何生成特定于 ABI 的 APK。

命令行仿真器

本文简要介绍如何通过命令行启动仿真器。

调试

这部分中的指南帮助你使用 Android 仿真器、实际 Android 设备和调试日志调试应用。

设置可调式属性

本文介绍如何设置可调式属性，以便 `adb` 等工具能够与 JVM 进行通信。

环境

本文介绍 Xamarin.Android 执行环境以及影响程序执行的 Android 系统属性。

GDB

本文介绍如何使用 `gdb` 调试 Xamarin.Android 应用程序。

安装系统应用

本指南介绍如何将 Xamarin.Android 应用作为 Android 设备上的系统应用程序或作为自定义 ROM 的一部分安装。

在 Android 上链接

本文讨论 Xamarin.Android 用于缩减应用程序最终大小的链接过程。其中描述了可以执行的各种级别的链接，并就缓解使用链接器可能导致的错误提供了一些指导和故障排除建议。

Xamarin.Android 性能

可以通过许多方法提高使用 Xamarin.Android 构建的应用程序的性能。总体上，这些方法可以极大地降低由 CPU 执行的工作量和应用程序占用的内存量。

分析 Android 应用

本指南介绍如何使用探查器工具来检查 Android 应用的性能和内存使用情况。

做好应用程序发布准备

应用程序经编码和测试后，必须准备一个包进行分发。准备此包的第一个任务是生成供发布的应用程序，其中主要涉及到设置应用程序的一些属性。

对 Android 应用程序包进行签名

了解如何创建 Android 签名标识、为 Android 应用程序创建新签名证书以及使用签名证书对应用程序进行签名。此外，本主题还介绍了如何将应用导出到磁盘以进行临时发布。生成的 APK 可以旁加载到 Android 设备中，而无需经过应用商店。

发布应用程序

这一系列文章介绍了公开发布使用 Xamarin.Android 创建的应用程序的步骤。可以通过电子邮件、专用 Web 服务器、Google Play 或适用于 Android 的 Amazon 应用商店等通道进行分发。

应用程序包大小

2018/10/26 • [Edit Online](#)

本文将讨论 Xamarin.Android 应用程序包的组成部分，以及可用于在开发的调试和发布阶段进行高效包部署的相关策略。

概述

Xamarin.Android 可使用各种机制来最大程度地减小包的大小，同时保持高效调试和发布部署过程。在本文中，我们将了解 Xamarin.Android 版本和调试部署工作流，并了解 Xamarin.Android 平台如何确保我们生成和发布小的应用程序包。

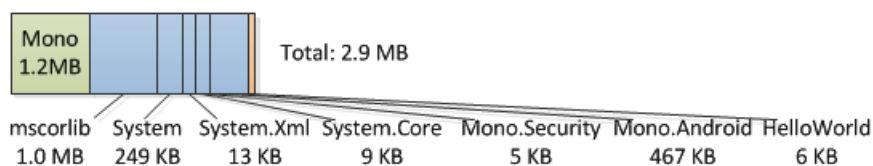
版本包

若要提供完全包含的应用程序，包必须包含应用程序、关联库、内容、Mono 运行时以及所需的基类库 (BCL) 程序集。例如，如果我们使用默认的“Hello World”模板，则完整的包生成内容将如下所示：

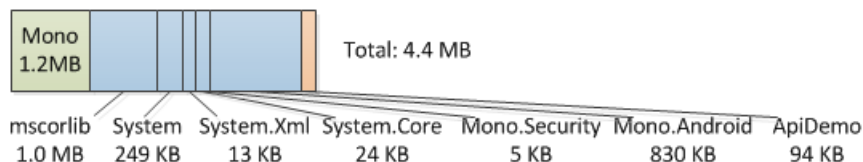


15.8 MB - 比我们所需的下载大小要大。问题在于 BCL 库，因为它们包含 mscorlib、系统和 Mono.Android，以此提供大量的必需组件来运行应用程序。但是，它们还在应用程序中提供了可能不使用的功能，因此最好将这些组件排除在外。

当我们构建用于分发的应用程序时，会执行一个称为“链接”的过程来检查应用，并移除不直接使用的任何代码。此过程类似于[垃圾回收](#)为堆分配内存提供的功能。但是，与在对象上操作不同，链接将在代码上运行。例如，System.dll 中有一个完整的命名空间，用于发送和接收电子邮件，但是，如果应用程序不使用此功能，那么该代码只会浪费空间。在 Hello World 应用程序上运行链接器之后，现在我们的包如下所示：



正如我们所看到的，这会移除大量未使用的 BCL。请注意，BCL 的最终大小要取决于实际使用的应用程序。例如，如果我们查看一个名为 ApiDemo 的更大的示例应用程序，会看到 BCL 组件的大小增加了，因为 ApiDemo 使用的 BCL 数量要大于 Hello, World：



如此处所示，应用程序包大小通常要比应用程序及其依赖项大约要大 2.9 MB。

调试包

针对调试版本的处理操作略有不同。当重新部署到设备时，应用程序需要尽可能快，因此，我们优化了调试包的部署速度，而不是大小。

Android 在复制和安装包方面相对较慢，因此，我们希望包大小尽可能地小。如上所述，最大程度降低包大小的一个可行方法就是通过链接器。但是链接速度比较慢，而且我们通常只想要部署应用程序自上次部署以来有更改的部分。为实现此目的，我们将应用程序从核心 Xamarin.Android 组件中分离了出来。

第一次在设备上调试时，我们复制了两个较大的包：共享运行时和共享平台。共享运行时包含 Mono 运行时和 BCL，而共享平台包含 Android API 级别的特定程序集：

| | | |
|---------------|--|----------------|
| Mono 1.2MB | BCL (mscorlib, System, System.Xml, System.Core, etc) 9.0 MB | Total: 10.2 MB |
|---------------|--|----------------|

复制这些核心组件只进行一次，因为这需要相当多的时间，但允许任何后续运行的应用程序在调试模式下使用它们。最后，我们将复制小而快速的实际应用程序：

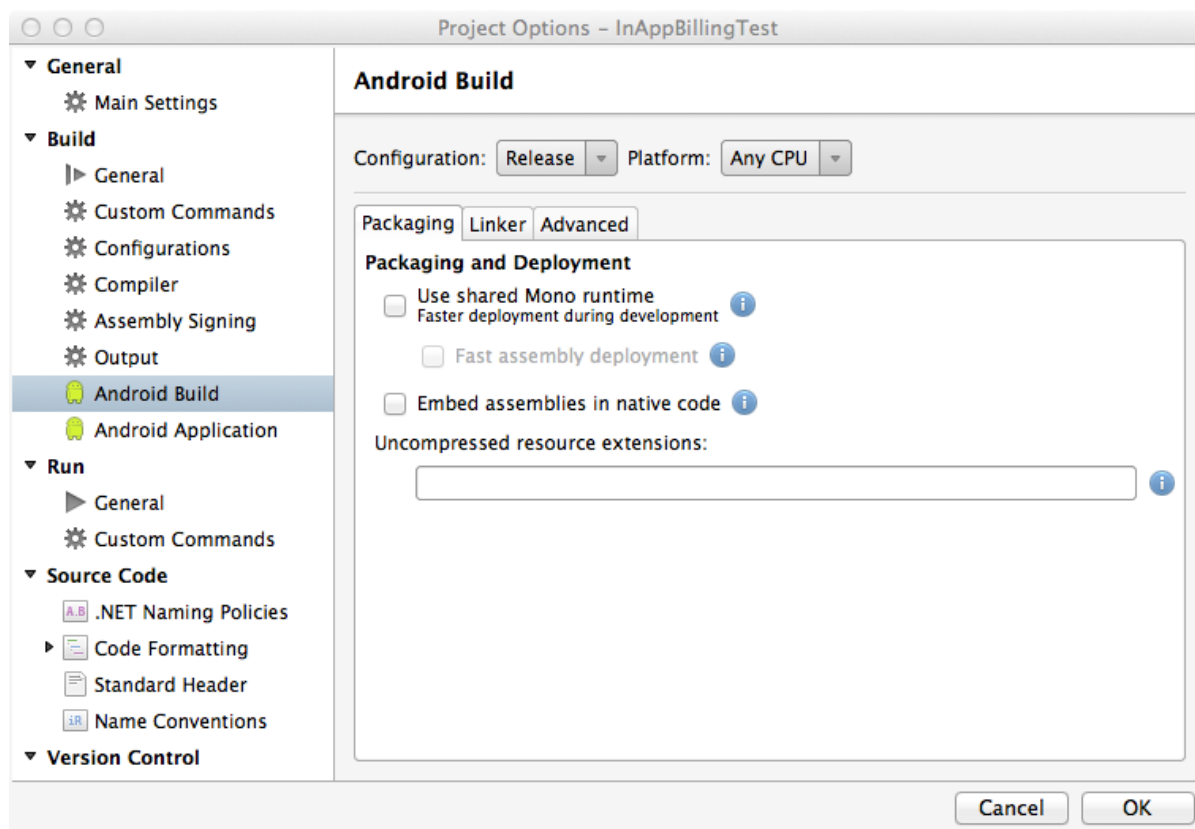
| |
|--------------------|
| HelloWorld 6 KB |
|--------------------|

快速程序集部署

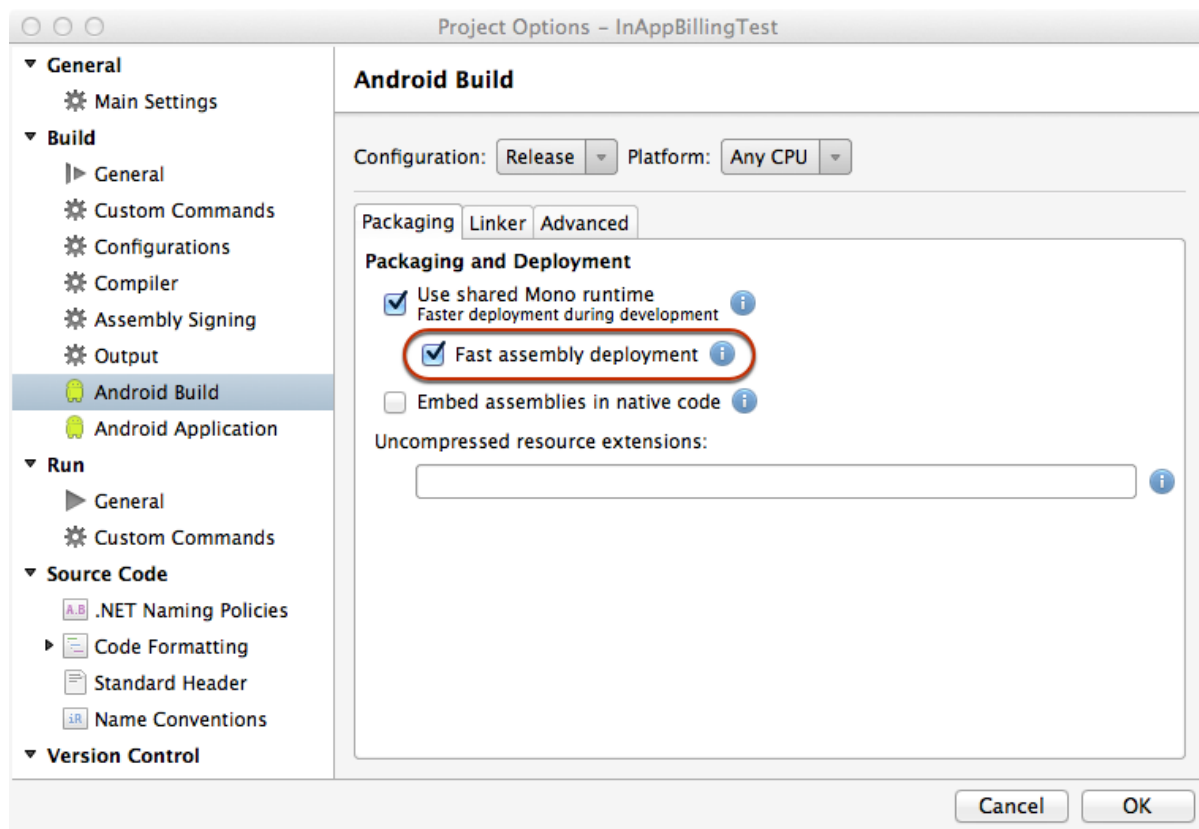
“快速程序集部署”生成选项可用于进一步减小调试安装包的大小，方法是通过在应用程序包中将程序集排除在外、直接在设备上安装程序集且只进行一次，并仅复制自上次部署以来有修改的文件。

若要启用“快速程序集部署”，请执行以下操作：

1. 右键单击“解决方案资源管理器”中的 Android 项目，并选择“选项”。
2. 在“项目选项”对话框中，选择“Android 版本”：



3. 选中“使用共享 Mono 运行时”复选框和“快速程序集部署”复选框：



4. 单击“确定”按钮，以保存所做的更改并关闭“项目选项”对话框。

下一次构建应用程序进行调试时，将直接在设备上安装程序集(如果尚未安装)，并且会在设备上安装一个较小的应用程序包(不包括程序集)。这将缩短部署应用程序更改并运行测试所需的时间。

通过持久长时间首次部署共享运行时和共享平台，每次对应用程序进行更改时，都可以快速轻松地部署新版本，以便拥有一个快速更改/部署/运行周期。

总结

本文将探讨 Xamarin.Android 版本和调试配置文件打包的各个方面。此外，我们还讨论了 Android 平台 Mono 用于在开发的调试和发布阶段进行高效包部署的相关策略。

构建应用

2018/10/26 • [Edit Online](#)

本部分介绍生成过程的工作原理, 并说明如何生成特定于 ABI 的 APK。

生成过程

本主题介绍与 Xamarin.Android 应用程序的源代码、资源和资产以及生成可在 Android 设备上安装的 APK 相关的步骤和过程。

生成特定于 ABI 的 APK

本指南介绍如何创建支持单一 CPU 体系结构和 ABI 的 Android APK。

生成过程

2018/11/2 • [Edit Online](#)

概述

Xamarin.Android 生成过程负责将所有内容集合在一起: 生成 `Resource.designer.cs`, 支持 `AndroidAsset`、`AndroidResource` 和其他生成操作, 生成 Android 可调用的包装器, 以及生成 `.apk` 以在 Android 设备上执行。

应用程序包

在广义上说, Xamarin.Android 生成系统可以生成两种类型的 Android 应用程序包 (`.apk` 文件):

- 发行版本, 完全是自包含的, 不需要额外的程序包来执行。这些是提供给应用商店的程序包。
- 调试版本则不是。

并非巧合的是, 这些版本与生成程序包的 MSBuild `Configuration` 相匹配。

共享运行时

“共享运行时”是一对额外的 Android 程序包, 它们提供基类库 (`mscorlib.dll` 等) 和 Android 绑定库 (`Mono.Android.dll` 等)。调试版本依赖共享运行时替代在 Android 应用程序包中包含基类库和绑定程序集, 从而使调试程序包更小。

通过将 `$(AndroidUseSharedRuntime)` 属性设置为 `False`, 可以在调试版本中禁用共享运行时。

快速部署

“快速部署”与共享运行时协同工作, 进一步缩小 Android 应用程序包的大小。可以通过不在程序包内捆绑应用的程序集来完成此操作。而是通过 `adb push` 将它们复制到目标上。此进程加快了生成/部署/调试周期, 因为如果只更改了程序集, 则不会重新安装该程序包。相反, 只有更新的程序集才会重新同步到目标设备。

已知快速部署在阻止 `adb` 同步到目录 `/data/data/@PACKAGE_NAME@/files/.__override__` 的设备上会失败。

快速部署在默认情况下处于启用状态, 可以通过将 `$(EmbedAssembliesIntoApk)` 属性设置为 `True` 在调试版本中禁用。

MSBuild 项目

Xamarin.Android 生成过程基于 MSBuild, 它也是 Visual Studio for Mac 和 Visual Studio 使用的项目文件格式。通常, 用户不需要手工编辑 MSBuild 文件 – IDE 将创建功能齐全的项目并使用所做的全部更改进行更新, 根据需要自动调用生成目标。

高级用户可能希望执行 IDE 的 GUI 不支持的操作, 因此, 可通过直接编辑项目文件来自定义生成过程。本页仅记录 Xamarin.Android 特定的功能和自定义 – 可以使用正常的 MSBuild 项目、属性和目标执行更多的操作。

生成目标

以下的生成目标是 Xamarin.Android 项目定义的:

- 生成 – 生成程序包。
- 清理 – 删除由生成过程生成的所有文件。
- 安装 – 将程序包安装到默认设备或虚拟设备。

- 卸载 – 从默认设备或虚拟设备中卸载程序包。
- SignAndroidPackage – 创建并对程序包进行签名 (`.apk`)。用于 `/p:Configuration=Release` , 生成自包含的“发行”包。
- UpdateAndroidResources – 更新 `Resource.designer.cs` 文件。将新的资源添加到项目中时, 这个目标通常由 IDE 调用。

生成属性

MSBuild 属性控制目标的行为。它们是在项目文件中指定的, 例如 [MSBuild PropertyGroup 元素](#) 中的 `MyApp.csproj`。

- 配置 – 指定要使用的生成配置, 例如“调试”或“发行”。配置属性用于确定其他属性(确定目标行为)的默认值。其他配置可能会在 IDE 中创建。

默认情况下, `Debug` 配置将导致 `Install` 和 `SignAndroidPackage` 目标创建更小的 Android 程序包, 这需要提供其他文件和包进行操作。

默认 `Release` 配置会导致 `Install` 和 `SignAndroidPackage` 目标创建独立的 Android 包, 无需安装其他任何包或文件, 即可使用此包。

- DebugSymbols – 确定 Android 程序包是否为 *可调试* 以及 `$(DebugType)` 属性的布尔值。可调试包包含调试符号, 将 `//application/@android:debuggable` 属性设置为 `true` , 并自动添加 `INTERNET` 权限, 以便调试器可以附加到该过程。如果 `DebugSymbols` 是 `True` , 并且 `DebugType` 是空字符串或 `Full` , 则应用程序是可调试的。
- DebugType – 指定要生成的 *调试符号的类型* 作为版本的一部分, 它还会影响应用程序是否可调试。可能的值包括:
 - Full: 生成 Full 符号。如果 `DebugSymbols` MSBuild 属性也为 `True` , 则应用程序包是可调试的。
 - PdbOnly: 生成 "PDB" 符号。应用程序包将不可调试。

如果 `DebugType` 未设置或为空字符串, 则 `DebugSymbols` 属性控制应用程序是否可调试。

安装属性

安装属性控制 `Install` 和 `Uninstall` 目标的行为。

- AdbTarget – 指定 Android 包可能要安装到或从中删除的 Android 目标设备。此属性的值与 `adb` [目标设备选项](#) 相同:

```
# Install package onto emulator via -e
# Use `./Library/Frameworks/Mono.framework/Commands/msbuild` on OS X
MSBuild /t:Install ProjectName.csproj /p:AdbTarget=-e
```

打包属性

打包属性控制如何创建 Android 包, 由 `Install` 和 `SignAndroidPackage` 目标使用。打包发布应用程序时, *签名属性* 也是相关的。

- AndroidApkSigningAlgorithm – 字符串值, 用于指定对 `jarsigner -sigalg` 使用的签名算法。

默认值为 `md5withRSA` 。

在 Xamarin.Android 8.2 中新增。

- AndroidApplication – 一个布尔值, 指示项目是用于 Android 应用程序 (`True`) 还是用于 Android 库项目 (`False` 或不存在)。

在 Android 包中, 可能只存在一个具有 `<AndroidApplication>True</AndroidApplication>` 的项目。(遗憾的是, 这点尚未得到验证, 这可能会导致与 Android 资源有关的微妙和奇怪的错误。)

- **AndroidBuildApplicationPackage** – 一个布尔值, 指示是否创建包 (.apk) 并为其签名。将此值设置为 `True` 相当于使用 [SignAndroidPackage](#) 生成目标。

在 Xamarin.Android 7.1 之后添加了对该属性的支持。

该属性默认为 `False`。

- **AndroidEnableMultiDex** – 一个布尔属性, 用于确定是否将在最终的 `.apk` 中使用 multi-dex 支持。

Xamarin.Android 5.1 中增加了对该属性的支持。

该属性默认为 `False`。

- **AndroidEnableSGenConcurrent** – 一个布尔属性, 用于确定是否使用 Mono 的[并发垃圾收集器](#)。

在 Xamarin.Android 7.2 中增加了对该属性的支持。

该属性默认为 `False`。

- **AndroidErrorOnCustomJavaObject** – 布尔属性, 用于确定类型能否实现 `Android.Runtime.IJavaObject`, 而无需同时继承自 `Java.Lang.Object` 或 `Java.Lang.Throwable` :

```
class BadType : IJavaObject {
    public IntPtr Handle {
        get {return IntPtr.Zero;}
    }

    public void Dispose()
    {
    }
}
```

若为 `True`, 这些类型生成 XA4212 错误; 否则, 生成 XA4212 警告。

Xamarin.Android 8.1 现已开始支持此属性。

该属性默认为 `True`。

- **AndroidFastDeploymentType** – `:` (冒号) 分隔的值列表, `$(EmbedAssembliesIntoApk)` MSBuild 属性为 `False` 时可用于控制部署到目标设备上的[快速部署目录](#)的类型。如果资源是快速部署的, 则不会嵌入到生成的 `.apk` 中, 这样做可以加快部署时间。(部署的速度越快, `.apk` 需要重建的频率越低, 安装过程可能会更快。)有效值包括:

- **Assemblies** : 部署应用程序程序集。
- **Dexes** : 部署 `.dex` 文件、Android 资源和 Android 资产。此值仅可以在运行 **Android 4.4 或更高版本 (API-19)** 的设备上使用。

默认值为 `Assemblies`。

“实验”。已在 Xamarin.Android 6.1 中添加。

- **AndroidApplicationJavaClass** – 类继承自 [Android.App.Application](#) 时, 用于替代 `android.app.Application` 的完整 Java 类名称。

此属性通常由其他属性(如 `$(AndroidEnableMultiDex)` MSBuild 属性)设置。

已在 Xamarin.Android 6.1 中添加。

- **AndroidHttpClientHandlerType** – 控制 `System.Net.Http.HttpClient` 默认构造函数使用的默认 `System.Net.Http.HttpMessageHandler` 实现。值是 `HttpMessageHandler` 子类的程序集限定类型名称，适用于 `System.Type.GetType(string)`。

默认值为 `System.Net.Http.HttpClientHandler, System.Net.Http`。

这可能会被重写为包含 `Xamarin.Android.Net.AndroidClientHandler`，后者使用 Android Java API 执行网络请求。这样，如果基础 Android 版本支持 TLS 1.2，就可以访问 TLS 1.2 URL。

只有 Android 5.0 及更高版本通过 Java 可靠提供 TLS 1.2 支持。

注意：如果低于 5.0 的 Android 版本必须提供 TLS 1.2 支持，或必须对 `System.Net.WebClient` 和相关 API 提供 TLS 1.2 支持，应使用 `$(AndroidTlsProvider)`。

注意：通过设置 `XA_HTTP_CLIENT_HANDLER_TYPE` 环境变量，可以支持此属性。在生成操作为 `@(AndroidEnvironment)` 的文件中，发现的 `$XA_HTTP_CLIENT_HANDLER_TYPE` 值的优先级更高。

已在 Xamarin.Android 6.1 中添加。

- **AndroidTlsProvider** – 一个字符串值，指定应用程序中应使用哪个 TLS 提供程序。可能的值有：
 - `btls`：使用 [Boring SSL](#) 与 `HttpWebRequest` 进行 TLS 通信。这样，可以对所有 Android 版本使用 TLS 1.2。
 - `legacy`：使用历史托管 SSL 实施进行网络交互。这不支持 TLS 1.2。
 - `default`：允许 Mono 选择默认 TLS 提供程序。这相当于 `legacy`，即使在 Xamarin.Android 7.3 中，也不例外。
注意：此值不太可能会出现在 `.csproj` 值中，因为 IDE“Default”值会导致 `$(AndroidTlsProvider)` 属性遭删除。
 - 取消设置/空字符串：在 Xamarin.Android 7.1 中，这相当于 `legacy`。
在 Xamarin.Android 7.3 中，这相当于 `btls`。

默认值为空字符串。

已在 Xamarin.Android 7.1 中添加。

- **AndroidLinkMode** – 指定应在 Android 包中包含的程序集上执行[链接](#)的类型。仅在 Android 应用程序项目中使用。默认值是 `SdkOnly`。有效值为：
 - `None`：不会尝试链接。
 - `SdkOnly`：仅在基类库上执行链接，而不是用户程序集。
 - `Full`：将在基类库和用户程序集上执行链接。注意：使用 `Full` 的 `AndroidLinkMode` 值通常会导致应用程序损坏，尤其是在使用 `Reflection` 时。除非你真正知道在做什么，否则请避免。

```
<AndroidLinkMode>SdkOnly</AndroidLinkMode>
```

- **AndroidLinkSkip** – 指定不应链接的程序集名称的分号分隔(;)列表，没有文件扩展名。仅在 Android 应用程序项目中使用。

```
<AndroidLinkSkip>Assembly1;Assembly2</AndroidLinkSkip>
```

- **AndroidManagedSymbols** – 一个布尔属性，用于控制是否生成序列点，以便可以从 `Release` 堆栈跟踪中提取文件名和行号信息。

已在 Xamarin.Android 6.1 中添加。

- AndroidManifest – 指定用于应用 `AndroidManifest.xml` 的模板的文件名。在生成期间，将合并任何其他必要的值以生成实际的 `AndroidManifest.xml`。`$(AndroidManifest)` 必须在 `/manifest/@package` 属性中包含程序包名称。
- AndroidSdkBuildToolsVersion – Android SDK 生成工具包提供 aapt 和 zipalign 工具等。可以同时安装多个不同版本的生成工具包。若要选择用于打包的生成工具包，请检查是否有“首选”生成工具版本。如果有，请使用它；如果没有“首选”版本，请使用版本最高的已安装生成工具包。

`$(AndroidSdkBuildToolsVersion)` MSBuild 属性包含首选的生成工具版本。如果(例如)已知上一 aapt 版本可用，而此时最新的 aapt 发生崩溃，则 Xamarin.Android 生成系统会在 `Xamarin.Android.Common.targets` 中提供默认值，并且可在项目文件中替代该默认值，选择备用的生成工具版本。

- AndroidSupportedAbis – 包含分号 (;) 分隔的 ABI 列表的字符串属性，应包含到 `.apk` 中。

支持的值包括：

- `armeabi`
- `armeabi-v7a`
- `x86`
- `arm64-v8a` : 需要 Xamarin.Android 5.1 及更高版本。
- `x86_64` : 需要 Xamarin.Android 5.1 及更高版本。

- AndroidUseSharedRuntime – 一个布尔属性，用于确定是否需要“共享运行时包”才能在目标设备上运行应用程序。依靠共享运行时包以允许应用程序包更小，加快包创建和部署过程，从而加快生成/部署/调试周转周期。

对于调试版本，该属性应为 `True`，对于发行项目应为 `False`。

- AotAssemblies – 一个布尔属性，用于确定程序集是否会被预编译为本机代码并包含在 `.apk` 中。

Xamarin.Android 5.1 中增加了对该属性的支持。

该属性默认为 `False`。

- EmbedAssembliesIntoApk – 一个布尔属性，用于确定应用程序的程序集是否应嵌入到应用程序包中。

对于发行版本，该属性应为 `True`，对于调试版本应为 `False`。如果“快速部署”不支持目标设备，则调试版本中可能必须为 `True`。

该属性为 `False` 时，`$(AndroidFastDeploymentType)` MSBuild 属性还会控制嵌入到 `.apk` 中的内容，这会影晌部署和重新生成时间。

- EnableLLVM – 一个布尔属性，用于确定在将程序集先编译为本机代码时是否使用 LLVM。

Xamarin.Android 5.1 中增加了对该属性的支持。

该属性默认为 `False`。

除非 `$(AotAssemblies)` MSBuild 属性为 `True`，否则该属性将被忽略。

- EnableProguard – 一个布尔属性，用于确定是否将 `proguard` 作为打包过程的一部分运行以链接 Java 代码。

Xamarin.Android 5.1 中增加了对该属性的支持。

该属性默认为 `False`。

如果 `True`，`ProguardConfiguration` 文件将用于控制 `proguard` 的执行。

- JavaMaximumHeapSize – 指定构建 `.dex` 文件作为打包过程一部分时使用的 java `-Xmx` 参数值的值。如果未指定，则不会为 java 提供 `-Xmx` 选项。

如果 `_CompileDex` 目标引发 `java.lang.OutOfMemoryError`，则指定该属性是必需的。

```
<JavaMaximumHeapSize>1G</JavaMaximumHeapSize>
```

- `JavaOptions` – 指定在生成 `.dex` 文件时传递给 `java` 的附加命令行选项。
- `Mandroidl18n` – 指定应用程序附带的国际化支持，例如排序规则和排序表。该值是以下一个或多个不区分大小写值的以逗号或分号分隔的列表：
 - 无: 不包含其他编码。
 - 全部: 包含所有可用的编码。
 - CJK: 包括中文、日语和朝鲜语编码，例如 *日语(EUC)* [enc-jp, CP51932]、*日语(Shift-JIS)* [iso-2022-jp, shift_jis, CP932]、*日语(JIS)* [CP50220]、*简体中文(GB2312)* [gb2312, CP936]、*朝鲜语(UHC)* [ks_c_5601-1987, CP949]、*朝鲜语(EUC)* [euc-kr, CP51949]、*繁体中文(Big5)* [big5, CP950] 以及 *简体中文(GB18030)* [GB18030, CP54936]。
 - 中东: 包括中东编码，例如 *土耳其语(Windows)* [iso-8859-9, CP1254]、*希伯来语(Windows)* [windows-1255, CP1255]、*阿拉伯语(Windows)* [windows-1256, CP1256]、*阿拉伯语(ISO)* [iso-8859-6, CP28596]、*希伯来语(ISO)* [iso-8859-8, CP28598]、*拉丁语 5 (ISO)* [iso-8859-9, CP28599] 以及 *希伯来语(Iso 备用)* [iso-8859-8, CP38598]。
 - 其他: 包括其他编码，例如 *西里尔文(Windows)* [CP1251]、*波罗的语(Windows)* [iso-8859-4, CP1257]、*越南语(Windows)* [CP1258]、*西里尔文(KOI8-R)* [koi8-r, CP1251]、*乌克兰语(KOI8-U)* [koi8-u, CP1251]、*波罗的语(ISO)* [iso-8859-4, CP1257]、*西里尔文(ISO)* [iso-8859-5, CP1251]、*ISCII Davenagari* [x-iscii-de, CP57002]、*ISCII 孟加拉语* [x-iscii-be, CP57003]、*ISCII 泰米尔语* [x-iscii-ta, CP57004]、*ISCII 泰卢固语* [x-iscii-te, CP57005]、*ISCII 阿萨姆语* [x-iscii-as, CP57006]、*ISCII 奥里亚语* [x-iscii-or, CP57007]、*ISCII 卡纳达语* [x-iscii-ka, CP57008]、*ISCII 马拉雅拉姆语* [x-iscii-ma, CP57009]、*ISCII 古吉拉特语* [x-iscii-gu, CP57010]、*ISCII 旁遮普文* [x-iscii-pa, CP57011] 以及 *泰语(Windows)* [CP874]。
 - 少数: 包括少数编码，例如 *IBM EBCDIC(土耳其语)* [CP1026]、*IBM EBCDIC(开放系统拉丁语 1)* [CP1047]、*IBM EBCDIC(美国-加拿大与欧洲)* [CP1140]、*IBM EBCDIC(德国与欧洲)* [CP1141]、*IBM EBCDIC(丹麦/挪威与欧洲)* [CP1142]、*IBM EBCDIC(芬兰/瑞典与欧洲)* [CP1143]、*IBM EBCDIC(意大利与欧洲)* [CP1144]、*IBM EBCDIC(拉丁美洲/西班牙与欧洲)* [CP1145]、*IBM EBCDIC(英国与欧洲)* [CP1146]、*IBM EBCDIC(法国与欧洲)* [CP1147]、*IBM EBCDIC(国际与欧洲)* [CP1148]、*IBM EBCDIC(冰岛与欧洲)* [CP1149]、*IBM EBCDIC(德国)* [CP20273]、*IBM EBCDIC(丹麦/挪威)* [CP20277]、*IBM EBCDIC(芬兰/瑞典)* [CP20278]、*IBM EBCDIC(意大利)* [CP20280]、*IBM EBCDIC(拉丁美洲/西班牙)* [CP20284]、*IBM EBCDIC(英国)* [CP20285]、*IBM EBCDIC(扩展式日语片假名)* [CP20290]、*IBM EBCDIC(法国)* [CP20297]、*IBM EBCDIC(阿拉伯语)* [CP20420]、*IBM EBCDIC(希伯来语)* [CP20424]、*IBM EBCDIC(冰岛语)* [CP20871]、*IBM EBCDIC(西里尔文-塞尔维亚文, 保加利亚语)* [CP21025]、*IBM EBCDIC(美国-加拿大)* [CP37]、*IBM EBCDIC(国际)* [CP500]、*阿拉伯语(ASMO 708)* [CP708]、*中欧语言(DOS)* [CP852]、*西里尔文(DOS)* [CP855]、*土耳其语(DOS)* [CP857]、*西欧(DOS 与欧洲)* [CP858]、*希伯来语(DOS)* [CP862]、*阿拉伯语(DOS)* [CP864]、*俄语(DOS)* [CP866]、*希腊语(DOS)* [CP869]、*IBM EBCDIC(拉丁语 2)* [CP870] 以及 *IBM EBCDIC(希腊语)* [CP875]。
 - 西部: 包括西部编码，例如 *西欧(Mac)* [macintosh, CP10000]、*冰岛语(Mac)* [x-mac-icelandic, CP10079]、*中欧(Windows)* [iso-8859-2, CP1250]、*西欧(Windows)* [iso-8859-1, CP1252]、*希腊语(Windows)* [iso-8859-7, CP1253]、*中欧(ISO)* [iso-8859-2, CP28592]、*拉丁语 3 (ISO)* [iso-8859-3, CP28593]、*希腊语(ISO)* [iso-8859-7, CP28597]、*拉丁语 9 (ISO)* [iso-8859-15, CP28605]、*OEM 美国* [CP437]、*西欧(DOS)* [CP850]、*葡萄牙语(DOS)* [CP860]、*冰岛语(DOS)* [CP861]、*加拿大法语(DOS)* [CP863] 以及 *北欧文(DOS)* [CP865]。

```
<MandroidI18n>West</MandroidI18n>
```

- MonoSymbolArchive – 一个布尔属性，用于控制是否创建 `.msym` 项目供以后与 `mono-symbolicate` 一起使用，从版本堆栈跟踪中提取真实文件名和行号信息。

对于已启用调试符号的“发行”应用，默认情况下为 `True`：`$(EmbedAssembliesIntoApk)` 为 `True`，`$(DebugSymbols)` 为 `True` 且 `$(Optimize)` 为 `True`。

已在 Xamarin.Android 7.1 中添加。

- AndroidVersionCodePattern – 允许开发人员在清单中自定义 `versionCode` 的字符串属性。有关决定 `versionCode` 的信息，请参阅[APK 创建版本代码](#)。

例如，如果 `abi` 是 `armeabi`，清单中的 `versionCode` 为 `123`，则当 `$(AndroidCreatePackagePerAbi)` 为 `True` 时，`{abi}{versionCode}` 将生成 `1123` 的 `versionCode`，否则将生成值 `123`。如果 `abi` 是 `x86_64`，则清单中的 `versionCode` 是 `44`。当 `$(AndroidCreatePackagePerAbi)` 为 `True` 时，这将生成 `544`，否则会生成值 `44`。

如果我们包含左填充格式字符串 `{abi}{versionCode:0000}`，则会生成 `50044`，因为我们用 `0` 在左边填充 `versionCode`。此外，也可以使用十进制填充（例如 `{abi}{versionCode:D4}`），该操作与前一个示例的效果相同。

由于值必须是整数，因此只支持 `0` 和 `Dx` 填充格式字符串。

预定义的键项

- **abi** – 插入应用的目标 abi
 - 1 – `armeabi`
 - 2 – `armeabi-v7a`
 - 3 – `x86`
 - 4 – `arm64-v8a`
 - 5 – `x86_64`
- **minSDK** – 如果没有定义，则插入 `AndroidManifest.xml` 或 `11` 中支持的最小 SDK 值。
- **versionCode** – 直接使用 `Properties\AndroidManifest.xml` 中的版本代码。

你可以使用（下文中定义的）`$(AndroidVersionCodeProperties)` 属性定义自定义项。

默认情况下，值设置为 `{abi}{versionCode:D6}`。如果开发人员要保留旧行为，可将 `$(AndroidUseLegacyVersionCode)` 属性设置为 `true`，从而替代默认值

已在 Xamarin.Android 7.2 中添加。

- AndroidVersionCodeProperties – 一个字符串属性，它允许开发人员定义要与 `AndroidVersionCodePattern` 一起使用的自定义项。它们采用 `key=value` 对的形式。`value` 中的所有项都应是整数值。例如：`screen=23;target=$(_SupportedApiLevel)`。正如你所看到的，你可以使用字符串中现有或自定义的 MSBuild 属性。

已在 Xamarin.Android 7.2 中添加。

- **AndroidUseLegacyVersionCode** – 一个布尔属性，允许开发人员将 `versionCode` 计算还原到先前的 Xamarin.Android 8.2 旧行为。这只能适用于在 Google Play 商店中已发布应用程序的开发人员。强烈建议使用新 `$(AndroidVersionCodePattern)` 属性。

在 Xamarin.Android 8.2 中新增。

- **AndroidUseManagedDesignTimeResourceGenerator** – 布尔属性，用于将设计时生成切换为使用受管理资源分析程序，而不是 `aapt`。

在 Xamarin.Android 8.1 中新增。

- **AndroidUseApkSigner** – 布尔属性，允许开发人员使用 `apksigner` 工具，而不是 `jarsigner`。

在 Xamarin.Android 8.2 中新增。

- **AndroidApkSignerAdditionalArguments** – 字符串属性，允许开发人员向 `apksigner` 工具提供其他参数。

在 Xamarin.Android 8.2 中新增。

绑定项目生成属性

以下 MSBuild 属性与[绑定项目](#)一起使用：

- **AndroidClassParser** – 一个字符串属性，用于控制如何分析 `.jar` 文件。可能的值包括：
 - `class-parse`: 使用 `class-parse.exe` 直接解析 Java 字节码，无需 JVM 的帮助。此值处于试验阶段。
 - `jar2xml`: 使用 `jar2xml.jar` 以使用 Java 反射从 `.jar` 文件中提取类型和成员。

`class-parse` 优于 `jar2xml` 的优势是：

- `class-parse` 能够从包含调试符号的 Java 字节码(例如，用 `javac -g` 编译的字节码)提取参数名称。
- `class-parse` 不会“跳过”从无法解析的类型继承或者包含无法解析类型成员的类。

“实验”。已在 Xamarin.Android 6.0 中添加。

默认值为 `jar2xml`。

默认值将会在未来版本中更改。

- **AndroidCodegenTarget** – 控制代码生成目标 ABI 的字符串属性。可能的值包括：
 - `XamarinAndroid`: 使用自 Mono for Android 1.0 以来的 JNI 绑定 API。使用 Xamarin.Android 5.0 或更高版本生成的绑定程序集只能在 Xamarin.Android 5.0 或更高版本(API/ABI 附加程序)上运行，但源与先前的产品版本兼容。
 - `XAJavaInterop1`: 使用 `Java.Interop` 进行 JNI 调用。只能通过 Xamarin.Android 6.1 或更高版本构建和执行使用 `XAJavaInterop1` 的绑定程序集。Xamarin.Android 6.1 和更高版本会将 `Mono.Android.dll` 与此值绑定。

`XAJavaInterop1` 的好处包括：

- 程序集较小。
- 只要继承层次结构中的所有其他绑定类型均使用 `XAJavaInterop1` 或更高版本构建，即可使用 `base` 方法调用的 `jmethodID` 缓存。
- 用于托管子类的 `Java Callable Wrapper` 构造函数的 `jmethodID` 缓存。

默认值为 `XamarinAndroid`。

默认值将会在未来版本中更改。

资源属性

资源属性控制 `Resource.designer.cs` 文件的生成，该文件提供对 Android 资源的访问。

- `AndroidResgenExtraArgs` – 指定处理 Android 资产和资源时传递给 `aapt` 命令的附加命令行选项。
- `AndroidResgenFile` – 指定要生成的资源文件的名称。默认模板将其设置为 `Resource.designer.cs`。
- `MonoAndroidResourcePrefix` – 通过 `AndroidResource` 生成操作指定从文件名开头删除的“路径前缀”。这是为了允许更改资源所在的位置。

默认值为 `Resources`。将此项更改为 `res` 以获得 Java 项目结构。

- `AndroidExplicitCrunch` – 如果你正在生成具有大量本地绘图的应用，则需要花费数分钟才能完成初始生成（或重新生成）。要加快生成过程，请尝试包含该属性并将其设置为 `True`。设置该属性时，生成过程会预处理 `.png` 文件。

“实验”。已在 Xamarin.Android 7.0 中添加。

签名属性

签名属性控制应用程序包的签名方式，以便将其安装到 Android 设备上。为了更快地生成迭代，Xamarin.Android 任务不会在生成过程中为包签名，因为签名很慢。相反，它们在安装之前或导出过程中由 IDE 或安装生成目标进行签名（如有必要）。调用 `SignAndroidPackage` 目标将生成一个在输出目录中带有 `-Signed.apk` 后缀的包。

默认情况下，如果需要，签名目标会生成一个新的调试签名密钥。如果你希望使用特定的密钥，例如在生成服务器上，则可以使用以下 MSBuild 属性：

- `AndroidKeyStore` – 一个布尔值，指示是否应使用自定义签名信息。默认值是 `False`，这意味着将使用默认的调试签名密钥来对包进行签名。
- `AndroidSigningKeyAlias` – 指定密钥存储中密钥的别名。这是创建密钥存储时使用的 `keytool -alias` 值。
- `AndroidSigningKeyPass` – 指定密钥存储文件中密钥的密码。这是在 `keytool` 要求“输入 `$(AndroidSigningKeyAlias)` 的密匙密码”时输入的值。
- `AndroidSigningKeyStore` – 指定由 `keytool` 创建的密钥存储文件的文件名。这对应于提供给 `keytool -keystore` 选项的值。
- `AndroidSigningStorePass` – 指定 `$(AndroidSigningKeyStore)` 的密码。这是在创建密钥存储文件并要求“输入密钥存储密码:”时为 `keytool` 提供的值。

例如，请考虑以下 `keytool` 调用：

```
$ keytool -genkey -v -keystore filename.keystore -alias keystore.alias -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password: keystore.filename password
Re-enter new password: keystore.filename password
...
Is CN=... correct?
[no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA1withRSA) with a validity of 10,000 days
for: ...
Enter key password for keystore.alias
(RETURN if same as keystore password): keystore.alias password
[Storing filename.keystore]
```

要使用上面生成的密钥存储，请使用属性组：

```
<PropertyGroup>
  <AndroidKeyStore>True</AndroidKeyStore>
  <AndroidSigningKeyStore>filename.keystore</AndroidSigningKeyStore>
  <AndroidSigningStorePass>keystore.filename password</AndroidSigningStorePass>
  <AndroidSigningKeyAlias>keystore.alias</AndroidSigningKeyAlias>
  <AndroidSigningKeyPass>keystore.alias password</AndroidSigningKeyPass>
</PropertyGroup>
```

- **AndroidDebugKeyAlgorithm** – 指定要用于 `debug.keystore` 的默认算法。默认为 `RSA`。
- **AndroidDebugKeyValidity** – 指定要用于 `debug.keystore` 的默认有效期。默认值为 `10950`、`30 * 365` 或 `30 years`。

生成操作

生成操作将[应用于项目中的文件](#)，并控制文件的处理方式。

AndroidEnvironment

生成操作为 `AndroidEnvironment` 的文件用于[在过程启动期间初始化环境变量和系统属性](#)。`AndroidEnvironment` 生成操作可能会应用于多个文件，并且它们将以特定顺序进行评估（因此，不要在多个文件中指定相同的环境变量或系统属性）。

AndroidJavaSource

生成操作为 `AndroidJavaSource` 的文件是 Java 源代码，将包含在最终的 Android 程序包中。

AndroidJavaLibrary

生成操作为 `AndroidJavaLibrary` 的文件是 Java 归档（`.jar` 文件），它将包含在最终的 Android 程序包中。

AndroidResource

生成操作为 `AndroidResource` 的所有文件都将在生成过程中编译为 Android 资源，并可通过 `$(AndroidResgenFile)` 访问。

```
<ItemGroup>
  <AndroidResource Include="Resources\values\strings.xml" />
</ItemGroup>
```

更高级的用户可能希望在不同的配置中使用不同的资源，但使用相同的有效路径。为此，可以使用多个资源目录并使具有相同相对路径的文件放在这些不同的目录中，以及使用 MSBuild 条件来有条件地在不同配置中包括不同文件。例如：

```
<ItemGroup Condition="'$(Configuration)'!='Debug'">
  <AndroidResource Include="Resources\values\strings.xml" />
</ItemGroup>
<ItemGroup Condition="'$(Configuration)'=='Debug'">
  <AndroidResource Include="Resources-Debug\values\strings.xml"/>
</ItemGroup>
<PropertyGroup>
  <MonoAndroidResourcePrefix>Resources;Resources-Debug<MonoAndroidResourcePrefix>
</PropertyGroup>
```

LogicalName – 显式指定资源路径。允许使用“aliasing”文件，以便它们可用作多个不同的资源名称。


```
<ItemGroup Condition="'$(Configuration)'!='Debug'">
  <AndroidResource Include="Resources/values/strings.xml"/>
</ItemGroup>
<ItemGroup Condition="'$(Configuration)'=='Debug'">
  <AndroidResource Include="Resources-Debug/values/strings.xml">
    <LogicalName>values/strings.xml</LogicalName>
  </AndroidResource>
</ItemGroup>
```

AndroidNativeLibrary

通过将生成操作设置为 `AndroidNativeLibrary`，将本机库添加到版本中。

请注意，由于 Android 支持多个应用程序二进制接口 (ABI)，因此生成系统必须知道本机库是为哪个 ABI 生成的。可以通过两种方法完成：

1. 路径“探查”。
2. 使用 `Abi` 项目属性。

通过路径探查，本机库的父目录名称用于指定库的目标 ABI。因此，如果将 `lib/armeabi/libfoo.so` 添加到版本中，则 ABI 将被“探查”为 `armeabi`。

项属性名称

Abi – 指定本机库的 ABI。

```
<ItemGroup>
  <AndroidNativeLibrary Include="path/to/libfoo.so">
    <Abi>armeabi</Abi>
  </AndroidNativeLibrary>
</ItemGroup>
```

AndroidAarLibrary

生成操作 `AndroidAarLibrary` 应用于直接引用 .aar 文件。Xamarin 组件最常使用此生成操作。也就是说，要添加对 .aar 文件的引用，它们是 Google Play 和其他服务正常运行所必需。

包含此生成操作的文件的处理方式类似于库项目中嵌入的资源。.aar 会被提取到中间目录。然后，任何资产、资源和 .jar 文件都会被添加到相应项组中。

内容

不支持正常的 `Content` 生成操作（因为我们还未想出如何在没有成本可能昂贵的首次运行步骤的情况下支持它）。

从 Xamarin.Android 5.1 开始，尝试使用 `@(Content)` 生成操作将导致 `XA0101` 警告。

LinkDescription

生成操作为 LinkDescription 的文件用于控制链接器行为。

ProguardConfiguration

生成操作为 ProguardConfiguration 的文件包含用于控制 `proguard` 行为的选项。有关此生成操作的更多信息，请参阅 [ProGuard](#)。

除非 `$(EnableProguard)` MSBuild 属性为 `True`，否则这些文件将被忽略。

目标定义

生成过程的 Xamarin.Android 特定部分在

`$(MSBuildExtensionsPath)\Xamarin\Android\Xamarin.Android.CSharp.targets` 中定义，但生成该程序集时也需要使

用正常的语言特定目标, 如 Microsoft.CSharp.targets。

导入任何语言目标之前, 必须设置以下生成属性:

```
<PropertyGroup>
  <TargetFrameworkIdentifier>MonoDroid</TargetFrameworkIdentifier>
  <MonoDroidVersion>v1.0</MonoDroidVersion>
  <TargetFrameworkVersion>v2.2</TargetFrameworkVersion>
</PropertyGroup>
```

通过导入 Xamarin.Android.CSharp.targets, 可在 C# 中包含所有这些目标和属性:

```
<Import Project="$(MSBuildExtensionsPath)\Xamarin\Android\Xamarin.Android.CSharp.targets" />
```

该文件可轻松适应于其他语言。

构建特定于 ABI 的 APK

2018/10/26 • [Edit Online](#)

本文讨论如何构建一个使用 *Xamarin.Android* 以单个 ABI 为目标的 APK。

概述

在某些情况下，应用程序拥有多个 APK 可能更有利 - 每个 APK 都使用相同的密钥存储进行签名，并共享相同的软件包名称，但它是针对特定设备或 Android 配置进行编译的。这不是推荐的方法 - 拥有一个可以支持多种设备和配置的 APK 是非常简单的。在某些情况下，创建多个 APK 可能很有用，例如：

- 减小 APK 的大小 - Google Play 对 APK 文件强加了 100MB 大小的限制。创建特定于设备的 APK 可以减小 APK 的大小，因为你只需为应用程序提供一部分资产和资源。
- 支持不同的 CPU 体系结构 - 如果你的应用程序具有特定 CPU 的共享库，则只能分发该 CPU 的共享库。

多个 APK 可能会使分发变得复杂 - 这是 Google Play 解决的问题。Google Play 将确保根据应用程序的版本代码和 AndroidManifest.XML 中包含的其他元数据将正确的 APK 传递到设备。有关 Google Play 如何支持应用程序多个 APK 的具体详细信息和限制，请查阅[关于多个 APK 支持的 Google 文档](#)。

本指南介绍如何为 *Xamarin.Android* 应用程序构建多个 APK 的脚本，每个 APK 都针对一个特定的 ABI。它包含以下主题：

1. 为 APK 创建一个唯一的版本代码。
2. 创建一个将用于此 APK 的 AndroidManifest.XML 的临时版本。
3. 使用上一步中的 AndroidManifest.XML 构建应用程序。
4. 通过对 APK 进行签名并使用 zipalign 为其优化来准备发布。

本指南最后将演示如何使用 [Rake](#) 编写这些步骤的脚本。

为 APK 创建版本代码

Google 为使用七位数版本代码的版本代码推荐了一种特定算法(请参阅[多 APK 支持文档](#)中的“使用版本代码方案”一节)。通过将此版本代码方案扩展为八位数字，可以将一些 ABI 信息包含在版本代码中，以确保 Google Play 将正确的 APK 分发到设备。以下列表解释了这个八位数版本代码格式(从左到右编制索引)：

- **索引 0**(下图中的红色)– ABI 的整数：
 - 1 – armeabi
 - 2 – armeabi-v7a
 - 6 – x86
- **索引 1-2**(下图中的橙色)– 应用程序支持的最低 API 级别。
- **索引 3-4**(下图中的蓝色)– 支持的屏幕大小：
 - 1 – 小
 - 2 – 常规
 - 3 – 大
 - 4 – 加大
- **索引 5-7**(下图中的绿色)– 版本代码的唯一编号。这由开发人员设置。它应为应用程序的每个公开发布版本增加。

下图说明了上面列表中描述的每个代码的位置：



Google Play 确保根据 `versionCode` 和 APK 配置将正确的 APK 发送到设备。将具有最高版本代码的 APK 发送到设备。例如, 应用程序可能有三个 APK, 其版本代码如下:

- 11413456 - ABI 是 `armeabi`; 针对 API 级别 14; 小屏幕到大屏幕; 版本号为 456。
- 21423456 - ABI 是 `armeabi-v7a`; 针对 API 级别 14; 常规 & 大屏幕; 版本号为 456。
- 61423456 - ABI 是 `x86`; 针对 API 级别 14; 常规 & 大屏幕; 版本号为 456。

要继续使用此示例, 假设已修复一个特定于 `armeabi-v7a` 的错误。应用版本增加到 457, 并且将 `android:versionCode` 设置为 21423457 来构建新的 APK。 `armeabi` 和 `x86` 版本的版本代码将保持不变。

现在假设 x86 版本接收一些针对更新 API (API 级别 19) 的更新或错误修复, 使该应用版本成为 500。当 `armeabi/armeabi-v7a` 保持不变时, 新的 `versionCode` 将更改为 61923500。此时, 版本代码将是:

- 11413456 - ABI 是 `armeabi`; 针对 API 级别 14; 小屏幕到大屏幕; 版本号为 456。
- 21423457 - ABI 是 `armeabi-v7a`; 针对 API 级别 14; 常规 & 大屏幕; 版本号为 457。
- 61923500 - ABI 是 `x86`; 针对 API 级别 19; 常规 & 大屏幕; 版本号为 500。

手动维护这些版本代码可能会对开发人员带来沉重的负担。计算正确的 `android:versionCode` 然后构建 APK 的过程应该是自动执行的。本文末尾的演练将举例介绍如何执行此操作。

创建临时的 **AndroidManifest.XML**

虽然不是绝对必要, 但为每个 ABI 创建临时的 **AndroidManifest.XML** 可以帮助防止由于在 APK 之间泄露信息可能出现的问题。例如, `android:versionCode` 属性对于每个 APK 都是唯一的, 这一点至关重要。

完成此任务的方式取决于所涉及的脚本系统, 但通常涉及获取开发过程中使用的 Android 清单副本, 修改副本, 然后在构建过程中使用修改后的清单。

编译 **APK**

如下示例命令行中所示, 通过使用 `xbuild` 或 `msbuild` 能最好地完成每个 ABI 构建 APK 的任务:

```
/Library/Frameworks/Mono.framework/Commands/xbuild /t:Package /p:AndroidSupportedAbis=<TARGET_ABI>
/p:IntermediateOutputPath=obj.<TARGET_ABI>/ /p:AndroidManifest=<PATH_TO_ANDROIDMANIFEST.XML>
/p:OutputPath=bin.<TARGET_ABI> /p:Configuration=Release <CSProj FILE>
```

下表解释了每个命令行参数:

- `/t:Package` - 创建一个使用调试密钥存储签名的 Android APK
- `/p:AndroidSupportedAbis=<TARGET_ABI>` - 这是要面向的 ABI。必须为 `armeabi`、`armeabi-v7a` 或 `x86` 之一。
- `/p:IntermediateOutputPath=obj.<TARGET_ABI>/` - 这是保存作为版本一部分创建的中间文件的目录。如果需要, Xamarin.Android 将创建一个以 ABI 命名的目录, 例如 `obj.armeabi-v7a`。建议为每个 ABI 使用一个文件夹, 因为这样可以防止在版本之间“泄露”文件导致的问题。请注意, 此值以目录分隔符 (在 OS X 的情况下为 `/`) 结束。
- `/p:AndroidManifest` - 此属性指定将在构建期间使用的 **AndroidManifest.XML** 文件的路径。
- `/p:OutputPath=bin.<TARGET_ABI>` - 这是容纳最终 APK 的目录。Xamarin.Android 将创建一个以 ABI 命名的目录, 例如 `bin.armeabi-v7a`。
- `/p:Configuration=Release` - 执行 APK 的发布版本。调试版本可能无法上传到 Google Play。
- `<CS_PROJ FILE>` - 这是 Xamarin.Android 项目的 `.csproj` 文件路径。

为 APK 签名并使用 Zipalign 为其优化

在通过 Google Play 发布 APK 之前，必须先为其签名。这可以通过使用属于 Java 开发人员工具包的 `jarsigner` 应用程序来执行。下面的命令行演示如何在命令中使用 `jarsigner`：

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore <PATH/TO/KEYSTORE> -storepass <PASSWORD> -signedjar <PATH/FOR/SIGNED_JAR> <PATH/FOR/JAR/TO/SIGN> <NAME_OF_KEY_IN_KEYSTORE>
```

必须先使用 Zipalign 为所有的 Xamarin.Android 应用程序优化，然后才能在设备上运行。这是要使用的命令行格式：

```
zipalign -f -v 4 <SIGNED_APK_TO_ZIPALIGN> <PATH/TO/ZIP_ALIGNED.APK>
```

通过 Rake 自动创建 APK

示例项目 [OneABIPerAPK](#) 是一个简单的 Android 项目，演示如何计算 ABI 特定的版本号并为以下所有的 ABI 构建三个独立的 APK：

- armeabi
- armeabi-v7a
- x86

示例项目中的 [rakefile](#) 执行前几节中描述的所有步骤：

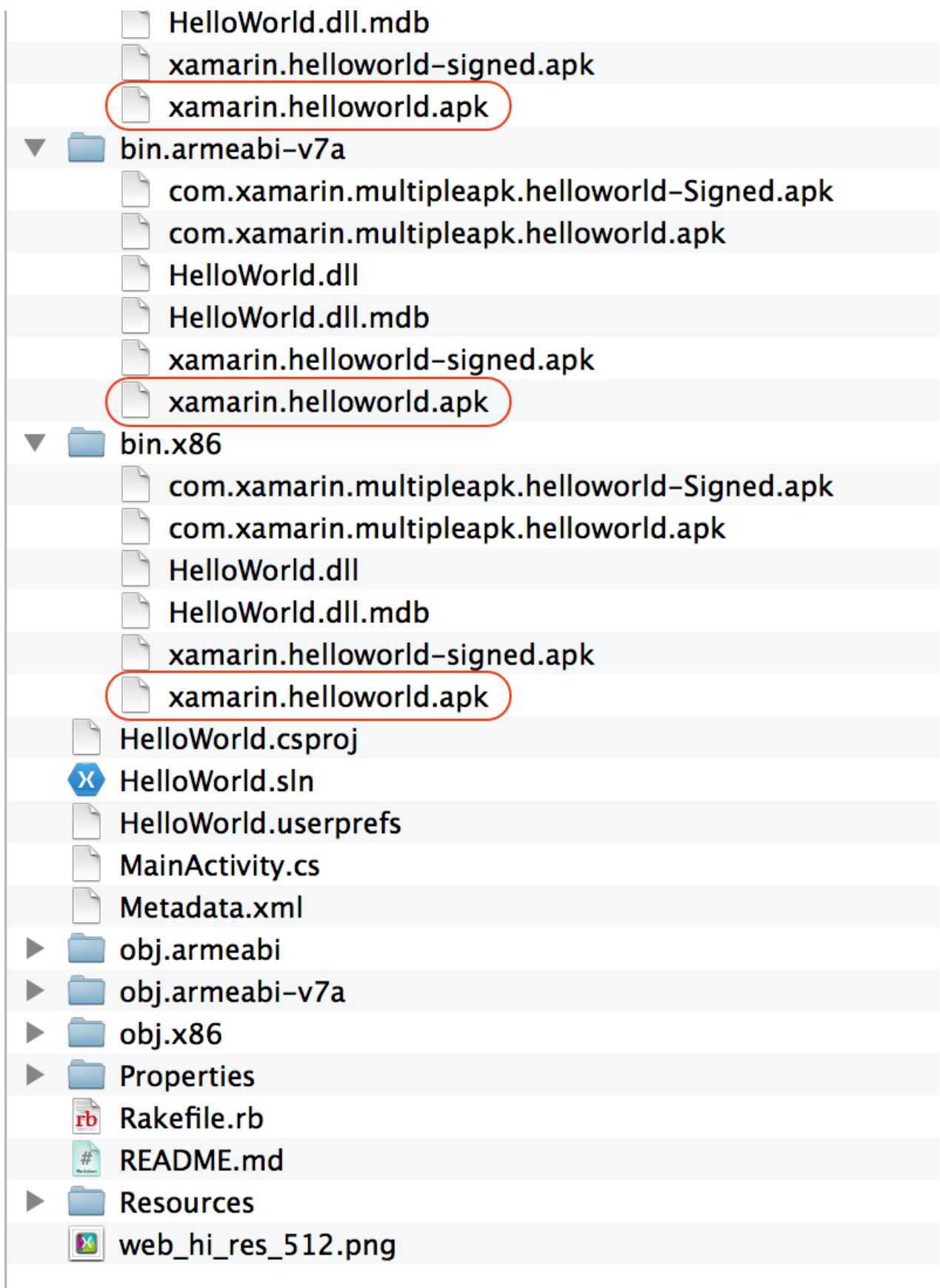
1. 为 APK [创建 android:versionCode](#)。
2. 将 [android:versionCode](#) 写入该 APK 的自定义 **AndroidManifest.XML**。
3. 为 Xamarin.Android 项目 [编译发布版本](#)，该项目特别面向 ABI 并使用在上一步中创建的 **AndroidManifest.XML**。
4. 使用生产密钥存储为 [APK 签名](#)。
5. 使用 [Zipalign](#) 优化 APK。

要构建应用程序的所有 APK，请从命令行运行 `build` Rake 任务：

```
$ rake build
==> Building an APK for ABI armeabi with ./Properties/AndroidManifest.xml.armeabi, android:versionCode = 10814120.
==> Building an APK for ABI x86 with ./Properties/AndroidManifest.xml.x86, android:versionCode = 60814120.
==> Building an APK for ABI armeabi-v7a with ./Properties/AndroidManifest.xml.armeabi-v7a, android:versionCode = 20814120.
```

rake 任务完成后，将有三个包含文件 `xamarin.helloworld.apk` 的 `bin` 文件夹。下一个屏幕截图显示了每个文件夹及其内容：

| Name | |
|------|---|
| ▶ | Assets |
| ▼ | bin.armeabi |
| | com.xamarin.multipleapk.helloworld-Signed.apk |
| | com.xamarin.multipleapk.helloworld.apk |
| | HelloWorld.dll |



NOTE

本指南中概述的构建过程可以在许多不同的构建系统之一中实现。尽管我们没有预先编写的示例，但 [Powershell](#) / [psake](#) 或 [Fake](#) 应该也适用。

总结

本指南提供了一些关于如何创建针对指定 ABI 的 Android APK 的建议。此外，还讨论了创建 `android:versionCodes` 的一种可能的方案，该方案将识别该 APK 所针对的 CPU 体系结构。演练包括一个使用 Rake 编写脚本的示例项目。

相关链接

- [OneABIPerAPK \(示例\)](#)
- [发布应用程序](#)
- [面向 Google Play 的多 APK 支持](#)

命令行仿真器

2018/11/10 • [Edit Online](#)

从命令行运行 Android 仿真器

要从命令行运行 Android 仿真器，可以使用 Android SDK 提供的“仿真器”工具。此工具可用于从 OS X 上的终端或 Windows 计算机上的命令提示符运行仿真器。

要启动特定的 Android 仿真器，请从 Android SDK 位置的 tools 目录(例如 C:\android-sdk-windows\tools)运行以下命令：

在 Windows 上

```
emulator.exe -avd NameOfYourEmulator -partition-size 512
```

在 macOS 上

```
./emulator -avd NameOfYourEmulator -partition-size 512
```

需要分区大小的原因是为了让仿真器具有足够的空间，以在仿真器上安装 Xamarin.Android 平台，因为默认情况下仿真器的大小很小。

有关额外参数的详细信息，可访问此处的 Android 站点 - <https://developer.android.com/studio/run/emulator-commandline>

调试

2018/10/26 • [Edit Online](#)

本节讨论如何在设备或仿真器上调试 Xamarin.Android 应用。

调试概述

开发 Android 应用程序需要在物理硬件上或使用仿真器运行应用程序。使用硬件是最好的方法，但并不总是最实用的方法。在许多情况下，使用如下所述的仿真器之一模拟/仿真 Android 硬件，这可简化操作且更具成本效益。

在 Android Emulator 上调试

本文介绍如何从 Visual Studio 启动 Android Emulator 并在虚拟设备中运行应用。

在设备上进行调试

本文介绍如何配置物理 Android 设备，以便可以直接从 Visual Studio 或 Visual Studio for Mac 将 Xamarin.Android 应用程序部署到此设备。

Android 调试日志

开发人员用于调试应用程序的一个非常常见的技巧是使用 `Console.WriteLine`。但是，在移动平台（如 Android）上没有控制台。Android 设备会提供日志，你在编写应用时可能需要利用它。这有时称为 **logcat**，原因在于为检索它而输入的命令。本文介绍如何使用 **logcat**。

WARNING

请注意，**Xamarin Android Player** 已弃用。有关详细信息，请参阅[此博客文章中的公告](#)。此外，自 Visual Studio 2017 起，已弃用 **Visual Studio Android Emulator**。

在 Android Emulator 上调试

2018/10/26 • [Edit Online](#)

本指南介绍如何在 Android Emulator 中启动虚拟设备以调试和测试应用。

概述

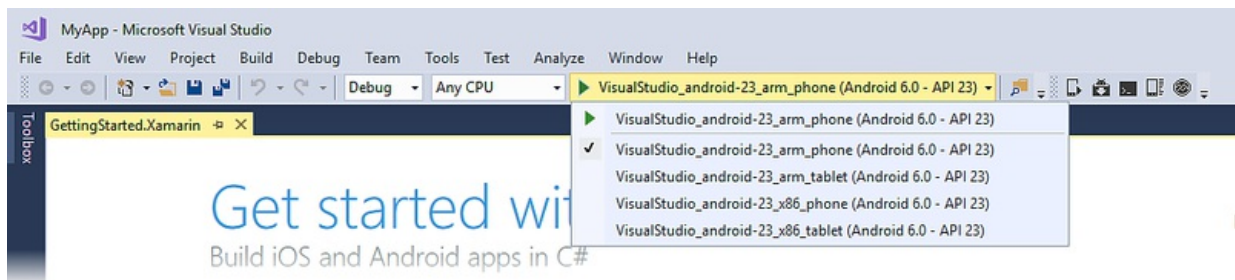
可使用各种配置运行 Android Emulator (作为“使用 .NET 进行移动开发”工作负荷的一部分进行安装) 来模拟各种 Android 设备。其中的每个配置都创建为虚拟设备。本指南介绍如何从 Visual Studio 启动模拟器以及如何虚拟设备中运行应用。有关配置 Android Emulator 和创建新的虚拟设备的信息, 请参阅 [Android Emulator 设置](#)。

使用预配置的虚拟设备

- [Visual Studio](#)
- [Visual Studio for Mac](#)

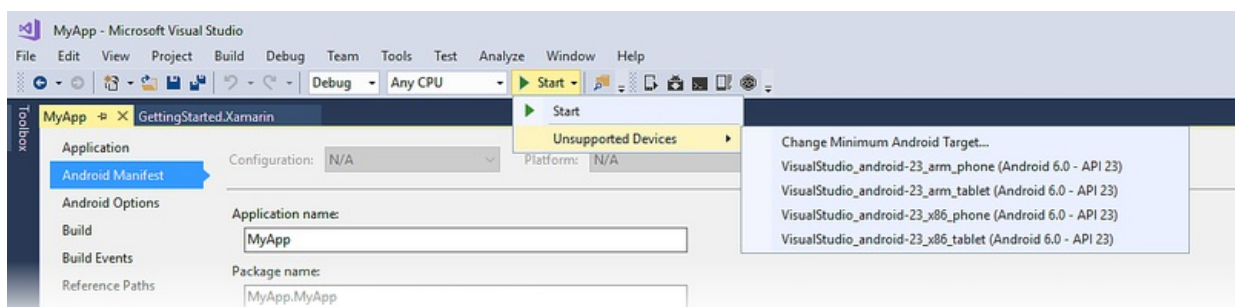
Visual Studio 包含预配置虚拟设备, 可以在“设备”下拉菜单中查看。例如, 在下面的 Visual Studio 2017 屏幕截图中, 多个预配置虚拟设备都可用:

- **VisualStudio_android-23_arm_phone**
- **VisualStudio_android-23_arm_tablet**
- **VisualStudio_android-23_x86_phone**
- **VisualStudio_android-23_x86_tablet**



通常都会选择使用“VisualStudio_android-23_x86_phone”虚拟设备来测试和调试手机应用。如果这些预配置虚拟设备中有一个能够满足需求(即与应用的目标 API 级别一致), 请跳转到[启动仿真器](#), 开始在仿真器中运行应用。(如果尚不熟悉 Android API 级别, 请参阅[了解 Android API 级别](#)。)

如果 Xamarin.Android 项目使用的目标框架级别与可用虚拟设备都不兼容, 不可用的虚拟设备会在下拉菜单中的“不支持的设备”下列出。例如, 以下项目的目标框架设为 Android 7.1 Nougat (API 25), 这与此示例中所列的 Android 6.0 虚拟设备不兼容:



可以单击“更改最低 Android 目标”，更改项目的最低 Android 版本，使其与可用虚拟设备的 API 级别一致。也可以使用 [Android Device Manager](#) 新建支持目标 API 级别的虚拟设备。必须先安装新 API 级别对应的系统映像(请参阅[设置用于 Xamarin.Android 的 Android SDK](#))，然后才能为虚拟设备配置此 API 级别。

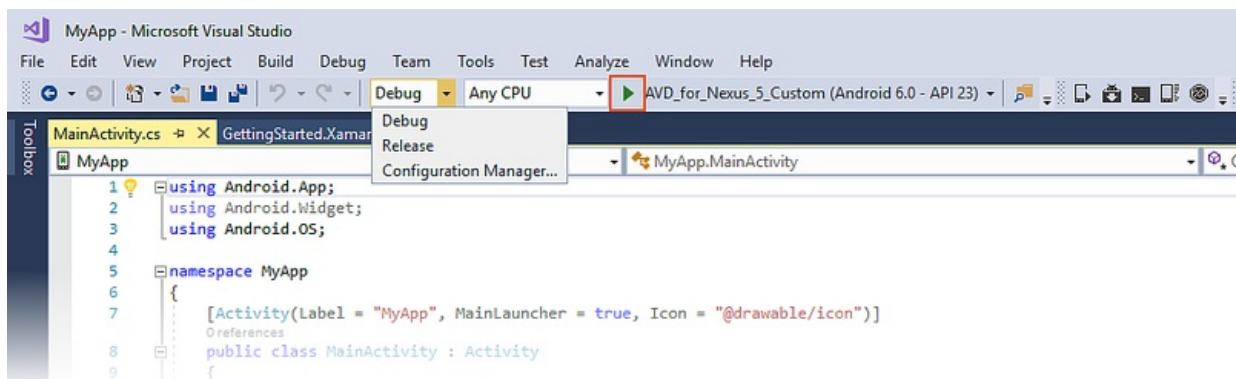
编辑虚拟设备

若要修改虚拟设备(或创建新的虚拟设备)，必须使用 [Android Device Manager](#)。

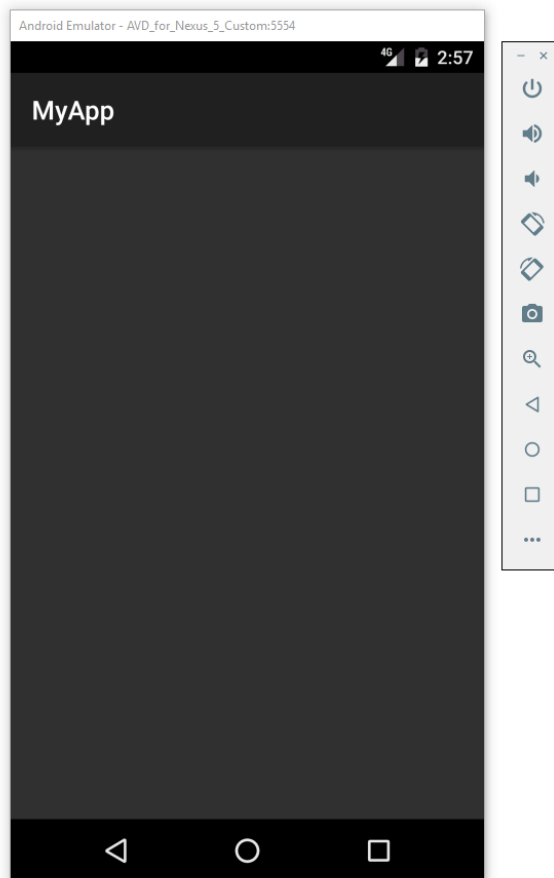
启动仿真器

在 Visual Studio 的顶部附近，有一个可用于选择“调试”或“发布”模式的下拉菜单。选择“调试”可在应用启动后将调试器附加到仿真器内运行的应用程序进程。选择“发布”模式可禁用调试器(但是，仍可运行应用并使用 LOG 语句进行调试)。在设备下拉菜单中选择虚拟设备后，选择“调试”或“发布”模式，然后单击“播放”按钮运行应用程序：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



仿真器启动后，Xamarin.Android 将应用部署到仿真器。仿真器会使用配置的虚拟设备映像运行此应用。Android Emulator 的示例屏幕截图如下所示。在此示例中，仿真器在运行名为 MyApp 的空白应用：



可以一直运行仿真器:无需关闭仿真器并在每次启动应用时等待重启。当 Xamarin.Android 应用首次在仿真器中运行时,将会先安装面向目标 API 级别的 Xamarin.Android 共享运行时,再安装应用。运行时安装过程可能需要一段时间,请耐心等待。仅当首次向仿真器部署 Xamarin.Android 应用时,才会安装运行时 – 后续部署速度更快,因为仅将应用复制到仿真器。

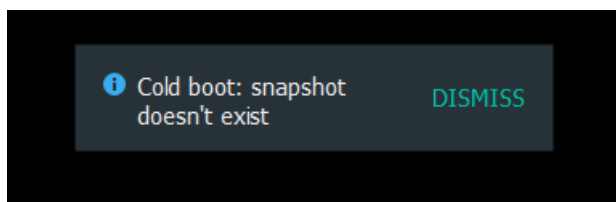
快速启动

较新版本的 Android Emulator 包含一个名为“快速启动”的功能,只需几秒钟即可启动模拟器。关闭仿真器时,它会拍摄虚拟设备状态的快照,以便在重启时从该状态快速还原。要使用此功能,需要以下工具:

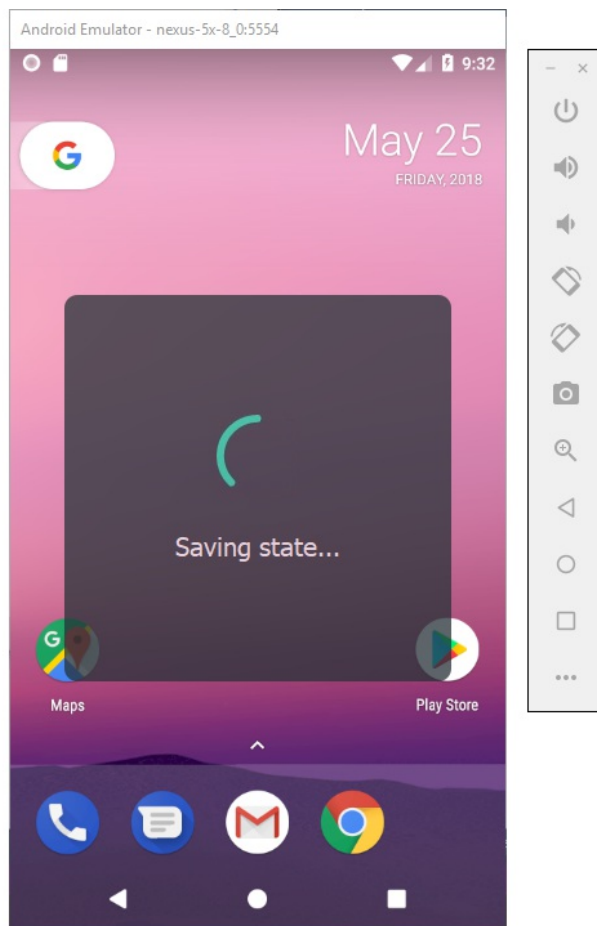
- Android Emulator 版本 27.0.2 或更高版本
- Android SDK Tools 版本 26.1.1 或更高版本

安装了上述版本的仿真器和 SDK 工具后,默认情况下会启用“快速启动”功能。

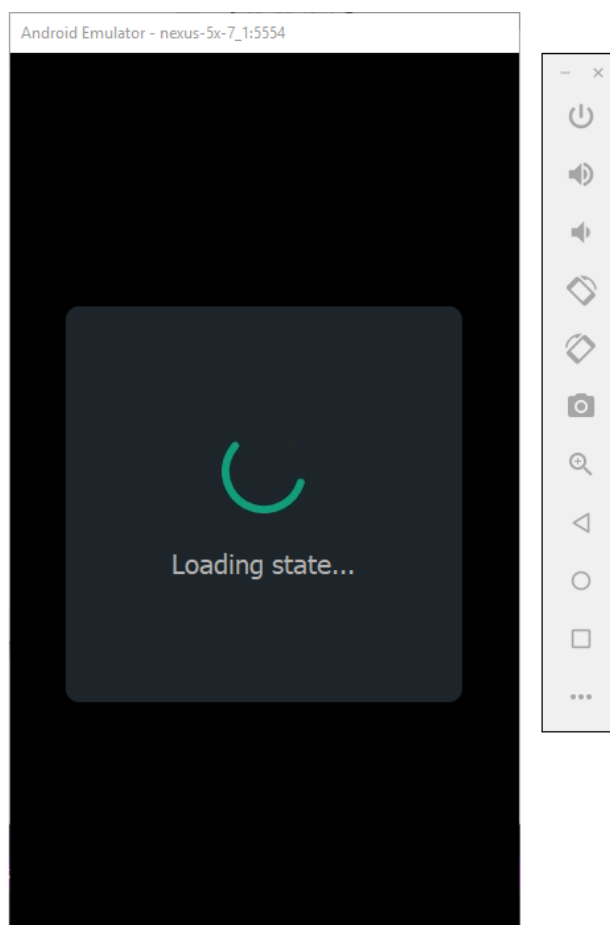
由于尚未创建快照,虚拟设备的首次冷启动没有加快速度:



退出仿真器时,“快速启动”在快照中保存仿真器的状态:



之后，启动虚拟设备的速度大幅加快，因为仿真器只需还原关闭仿真器时的状态。



疑难解答

有关常见模拟器问题的技巧和解决方法, 请参阅 [Android Emulator 疑难解答](#)。

总结

本指南介绍如何配置 Android Emulator, 以便能够运行和测试 Xamarin.Android 应用。其中介绍了使用预配置的虚拟设备启动仿真器的步骤, 并提供了将应用程序从 Visual Studio 部署到仿真器的步骤。

若要深入了解如何使用 Android Developer, 请参阅以下 Android 开发者主题:

- [屏幕导航](#)
- [在仿真器中执行基本任务](#)
- [使用扩展控件、设置和帮助](#)
- [使用快速启动运行仿真器](#)

在设备上调试

2018/10/26 • [Edit Online](#)

本文介绍如何在物理 Android 设备上调试 Xamarin.Android 应用程序。

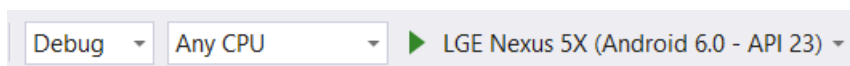
在设备上调试概述

可在 Android 设备上使用 Visual Studio for Mac 或 Visual Studio 调试 Xamarin.Android 应用。在设备上执行调试之前，必须[设置设备进行开发](#)并将其连接到电脑或 Mac。

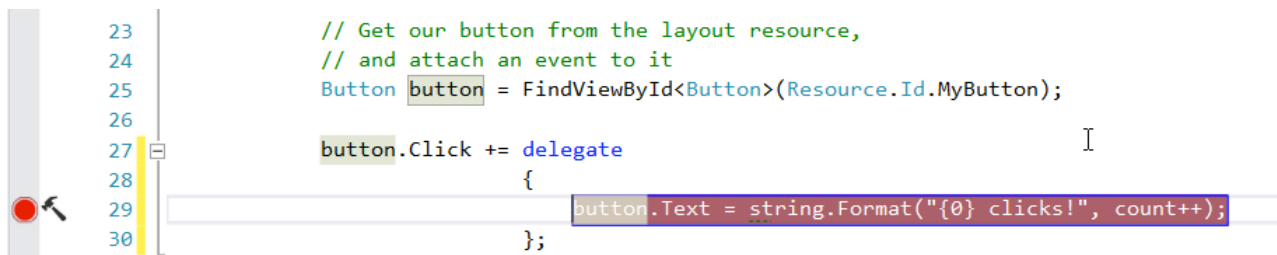
调试应用程序

设备连接到计算机后，即可采用与其他任何 Xamarin 产品或 .NET 应用程序相同的方式调试 Xamarin.Android 应用程序。请务必在 IDE 中选择“调试”配置和外部设备，确保必要的调试符号可用且 IDE 可连接到运行的应用程序：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



随后，在代码中设置断点：



选择设备后，Xamarin.Android 会连接到设备、部署应用程序，然后运行该程序。达到断点时，调试程序会停止应用程序，允许以类似于任何其他 C# 应用程序的方式调试该程序：



总结

本文档介绍了如何通过设置断点和选择目标设备来调试 Xamarin.Android 应用程序。

相关链接

- [设置设备进行开发](#)
- [设置可调试属性](#)

Android 调试日志

2018/11/2 • [Edit Online](#)

开发人员用于调试应用程序的一个非常常见的技巧是调用 `Console.WriteLine`。但是，在移动平台（如 Android）上没有控制台。Android 设备会提供日志，可以在编写应用时使用。这有时称为“logcat”，原因在于为检索它而输入的命令。使用“调试日志”工具，可查看记录的数据。

Android 调试日志概述

“调试日志”工具使你能够在通过 Visual Studio 调试应用时查看日志输出。调试日志支持以下设备：

- 物理 Android 手机、平板电脑和可穿戴设备。
- 在 Android Emulator 上运行的 Android 虚拟设备。

NOTE

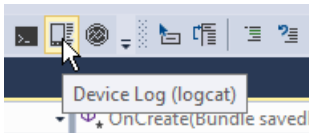
“调试日志”工具并不适用于 Xamarin Live Player。

“调试日志”不会显示在设备上独立运行应用时生成的日志消息（即，在从 Visual Studio 断开连接时）。

从 Visual Studio 访问调试日志

- [Visual Studio](#)
- [Visual Studio for Mac](#)

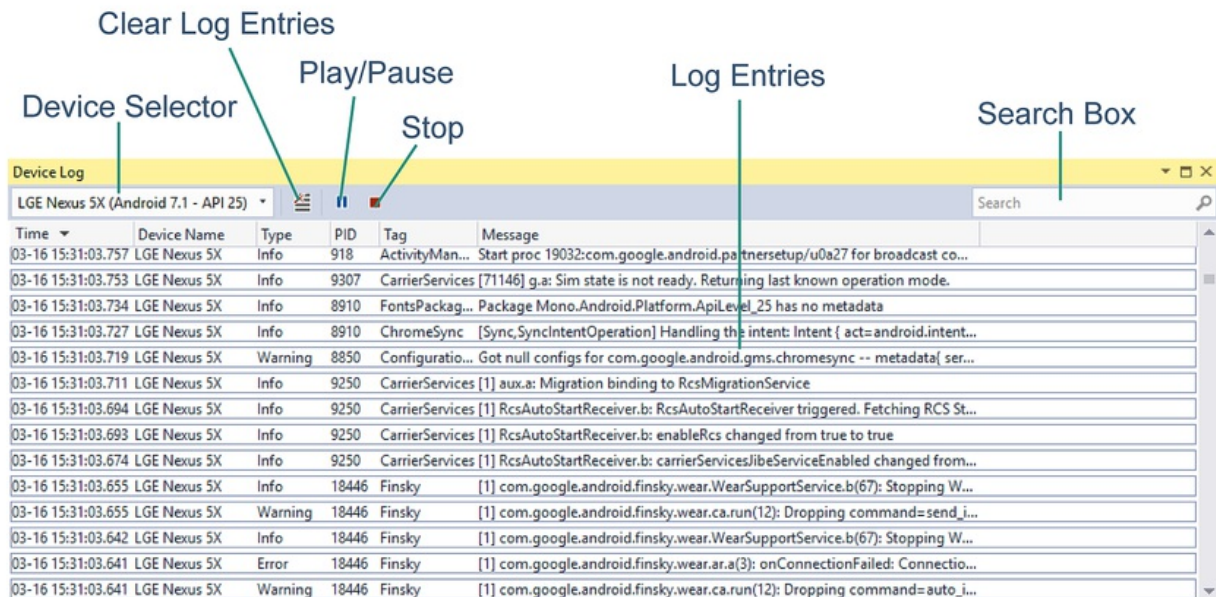
若要打开“设备日志”工具，单击工具栏上的“设备日志 (logcat)”图标：



或者，也可以从以下菜单选项之一启动“设备日志”工具：

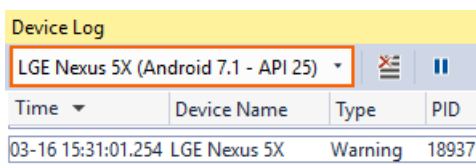
- “视图”>“其他窗口”>“设备日志”
- “工具”>“Android”>“设备日志”

下面的屏幕截图演示了“调试工具”窗口的各个部分：



- 设备选择器 – 选择要监视哪些物理设备或运行的仿真器。
- 日志条目 – logcat 中的日志消息表。
- 清除日志条目 – 清除表中所有当前日志条目。
- 播放/暂停 – 在更新或暂停显示新日志条目之间切换。
- 停止 – 暂停显示新日志条目。
- 搜索框 – 在此框中输入搜索字符串以筛选日志条目子集。

当显示“调试日志”工具窗口时，使用设备下拉菜单来选择要监视的 Android 设备：



在选择设备后，“设备日志”工具会从正在运行的应用中自动添加日志条目 – 这些日志条目显示在日志条目表中。在设备之间切换将停止和启动设备日志记录。请注意，在任何设备显示在设备选择器中之前，必须加载 Android 项目。如果设备未显示在设备选择器中，请确认它在“启动”按钮旁边的 Visual Studio 设备下拉列表菜单中可用。

从命令行访问

- [Visual Studio](#)
- [Visual Studio for Mac](#)

另一种方法是通过命令行查看调试日志。打开命令提示符窗口，并导航到 Android SDK 平台工具文件夹（通常情况下，SDK 平台工具文件夹位于 C:\Program Files (x86)\Android\android-sdk\platform-tools 中）。

如果仅连接了单个设备（物理设备或仿真器），则可以通过输入以下命令查看日志：

```
$ adb logcat
```

如果连接了多个设备，则必须对设备进行显式标识。例如，adb -d logcat 显示已连接的唯一物理设备的日志，而 adb -e logcat 显示运行中的唯一仿真器的日志。

通过输入 adb 和读取帮助消息，可以找到更多命令。

写入调试日志

使用 `Android.Util.Log` 类方法可将消息写入“调试日志”。例如:

```
string tag = "myapp";

Log.Info (tag, "this is an info message");
Log.Warn (tag, "this is a warning message");
Log.Error (tag, "this is an error message");
```

这将生成类似下面的输出:

```
I/myapp (11103): this is an info message
W/myapp (11103): this is a warning message
E/myapp (11103): this is an error message
```

也可使用 `Console.WriteLine` 写入调试日志 – 这些消息将在 logcat 中以略有不同的输出格式显示(调试 Android 上的 Xamarin.Forms 应用时, 此方法非常有用):

```
System.Console.WriteLine ("DEBUG - Button Clicked!");
```

这将在 logcat 中生成类似下面的输出:

```
Info (19543) / mono-stdout: DEBUG - Button Clicked!
```

兴趣消息

在读取日志时(尤其是在向其他用户提供日志代码片段时), 浏览完整的日志文件通常非常麻烦。若要更加轻松地浏览日志消息, 首先查找类似于下面的日志条目:

```
I/ActivityManager(12944): Starting: Intent { act=android.intent.action.MAIN cat=
[android.intent.category.LAUNCHER] flg=0x10200000 cmp=GcTest.GcTest/gctest.Activity1 } from pid 24175
```

具体而言, 查找与也包含应用程序包名称的正则表达式匹配的行:

```
^I.*ActivityManager.*Starting: Intent
```

这是对应于活动开始的行, 大多数(并非全部)以下消息均应与应用程序相关。

请注意, 每条消息均包含生成该消息的进程的进程标识符(pid)。在上一 `ActivityManager` 消息中, 进程 `12944` 生成了消息。若要确定哪个进程是正在调试的应用程序进程, 请查找 `mono.MonoRuntimeProvider` 消息:

```
I/ActivityThread( 602): Pub TouchTest.TouchTest.__mono_init__: mono.MonoRuntimeProvider
```

此消息来自已启动的进程。包含此 pid 的所有后续消息均来自同一进程。

可调试属性

2018/10/26 • [Edit Online](#)

若要进行调试，Android 需支持 Java 调试线协议 (JDWP)。这是一种允许 ADB 等工具与 JVM 通信的技术。尽管在开发期间 JDWP 非常重要，仍应在发布应用程序之前将其禁用。

JDWP 可以是 Android 应用程序中 `android:debuggable` 属性的值。Xamarin.Android 提供了以下方式来设置此属性：

1. 创建 `AndroidManifest.xml` 文件，并在其中设置 `android:debuggable` 属性。
2. 将 `ApplicationAttribute` 包含在 `.cs` 文件中，例如：`[assembly: Application(Debuggable=false)]`。

如果 `AndroidManifest.xml` 和 `ApplicationAttribute` 均存在，则 `AndroidManifest.xml` 的内容将优先于 `ApplicationAttribute` 所指定的内容。

如果既没有 `AndroidManifest.xml` 也没有 `ApplicationAttribute`，则 `android:debuggable` 属性的默认值将取决于是否生成了调试符号。如果调试符号存在，则 Xamarin.Android 会将 `android:debuggable` 属性设为 `true`。

请注意，`android:debuggable` 属性的值不一定依赖于生成配置。发行版本可能会将 `android:debuggable` 属性设为 `true`。

相关链接

- [Android 市场中的可调试应用](#)

Xamarin.Android 环境

2018/11/2 • [Edit Online](#)

执行环境

执行环境是一系列影响程序执行的环境变量和 Android 系统属性。Android 系统属性可通过 `adb shell setprop` 命令设置, 而环境变量可通过设置 `debug.mono.env` 系统属性进行设置:

```
## Enable GREF logging
adb shell setprop debug.mono.log gref

## Set the MONO_LOG_LEVEL and MONO_LOG_MASK environment variables
## so that additional Mono messages will be written to `adb logcat`.
adb shell setprop debug.mono.env "'MONO_LOG_LEVEL=info|MONO_LOG_MASK=asm'"
```

Android 系统属性是针对目标设备上的所有进程而设置的。

从 Xamarin.Android 4.6 开始, 可能会在每个应用的基础上设置或重写系统属性和环境变量, 方法是将环境文件添加到项目。环境文件是 Unix 格式的纯文本文件, 附带 [AndroidEnvironment](#) 的生成操作。环境文件包含键=值格式的行。注释是以 `#` 开头的行。将忽略空行。

如果密钥以大写字母开头, 密钥将被视为一个环境变量, 且 `setenv(3)` 用于在进程启动过程中将环境变量设置为指定值。

如果密钥以小写字母开头, 则密钥会被视为 Android 系统属性, 且值为默认值: 首先从 Android 系统属性存储查找控制 Xamarin.Android 执行行为的 Android 系统属性, 如果未指定任何值, 则使用环境文件中指定的值。这是为了允许 `adb shell setprop` 用于重写来自环境文件的值, 以进行诊断。

Xamarin.Android 环境变量

Xamarin.Android 支持 `XA_HTTP_CLIENT_HANDLER_TYPE` 变量, 可通过 `adb shell setprop debug.mono.env` 或 `$(AndroidEnvironment)` 生成操作进行设置。

```
XA_HTTP_CLIENT_HANDLER_TYPE
```

程序集限定类型必须继承自 [HttpMessageHandler](#), 并从 [HttpClient\(\)](#) 默认构造函数进行构造。

在 Xamarin.Android 6.1 中, 默认情况下不会设置此环境变量, 且会使用 [HttpClientHandler](#)。

或者, 可以指定值 `Xamarin.Android.Net.AndroidClientHandler` 使用 [java.net.URLConnection](#) 进行网络访问, 该操作可允许使用 TLS 1.2(在获得 Android 支持的情况下)。

已在 Xamarin.Android 6.1 中添加。

Xamarin.Android 系统属性

Xamarin.Android 支持以下系统属性, 可通过 `adb shell setprop` 或 `$(AndroidEnvironment)` 生成操作进行设置。

- `debug.mono.debug`
- `debug.mono.env`
- `debug.mono.gc`
- `debug.mono.log`
- `debug.mono.max_grefc`

- `debug.mono.profile`
- `debug.mono.runtime_args`
- `debug.mono.trace`
- `debug.mono.wref`
- `XA_HTTP_CLIENT_HANDLER_TYPE`

`debug.mono.debug`

`debug.mono.debug` 系统属性的值是一个整数。若为 `1`，则表现为就好像进程是以 `mono --debug` 开始的。这通常会在堆栈跟踪中显示文件和行信息等内容，而无需应用通过调试器启动。

`debug.mono.env`

包含环境变量以 `|` 分隔的列表。

`debug.mono.gc`

`debug.mono.debug` 系统属性的值是一个整数。若为 `1`，则应记录 GC 信息。

这相当于使 `debug.mono.log` 系统属性包含 `gc`。

`debug.mono.log`

控制 Xamarin.Android 将记录到 `adb logcat` 的其他信息。它是一个以逗号分隔的字符串 (`,`)，包含以下值之一：

- `all` : 打印所有消息。这不是一个好主意，因为它包含 `lref` 消息。
- `assembly` : 打印 `.apk` 和程序集分析消息。
- `gc` : 打印与 GC 相关的消息。
- `gref` : 打印 JNI 全局引用消息。
- `lref` : 打印 JNI 本地引用消息。

注意：这将是真正的垃圾邮件 `adb logcat`。

在 Xamarin.Android 5.1 中，它还会创建 `.__override__/lrefs.txt` 文件，从中获取 gigantic。

请避免。

- `timing` : 打印某些方法计时信息。这还将创建文件 `.__override__/methods.txt` 和 `.__override__/counters.txt`。

`debug.mono.max_grefc`

`debug.mono.max_grefc` 系统属性的值是一个整数。其值会重写默认检测到的目标设备的最大 GREF 计数。

注意：这只能与 `adb shell setprop debug.mono.max_grefc` 一起使用，因为当此值在 **environment.txt** 文件中时将不可用。

`debug.mono.profile`

`debug.mono.profile` 系统属性将启用探查器。它等效于 `mono --profile` 选项，并使用与之相同的值。（请参阅 [mono\(1\)](#) 手册页了解详细信息。）

`debug.mono.runtime_args`

`debug.mono.runtime_args` 系统属性包含应通过 mono 分析的其他选项。

`debug.mono.trace`

`debug.mono.trace` 系统属性将启用跟踪。它等效于 `mono --trace` 选项，并使用与之相同的值。（请参阅 [mono\(1\)](#) 手册页了解详细信息。）

一般情况下，**不使用**。使用跟踪将发送垃圾邮件 `adb logcat` 输出，严重减慢程序行为，并更改程序行为（直至并包括添加其他错误情况）。

但是有些时候，它允许执行某些进一步研究...

debug.mono.wref

debug.mono.wref 系统属性可重写默认检测到的 JNI 弱引用机制。有两个支持的值：

- `jni` : 使用 JNI 弱引用, 由 `JNIEnv::NewWeakGlobalRef()` 创建并由 `JNIEnv::DeleteWeakGlobalRef()` 销毁。
- `java` : 使用引用 `java.lang.WeakReference` 实例的 JNI 全局引用。

默认情况下使用 `java`, 通过 API 7 和 API-19 (Kit Katt) 启用, 同时启用 ART。(API 8 添加了 `jni` 引用, 而 ART 中断了 `jni` 引用。)

此系统属性可用于测试和进行某些形式的调查。一般情况下, 不应更改此属性。

XA_HTTP_CLIENT_HANDLER_TYPE

在 Xamarin.Android 6.1 中首次引入, 此环境变量声明将由 `HttpClient` 使用的默认 `HttpMessageHandler` 实现。默认情况下不设置此变量, Xamarin.Android 将使用 `HttpClientHandler`。

```
XA_HTTP_CLIENT_HANDLER_TYPE=Xamarin.Android.Net.AndroidClientHandler
```

NOTE

基础 Android 设备必须支持 TLS 1.2。Android 5.0 及更高版本支持 TLS 1.2

示例

```
## Comments are lines which start with '#'
## Blank lines are ignored.

## Enable GREF messages to `adb logcat`
debug.mono.log=gref

## Clear out a Mono environment variable to decrease logging
MONO_LOG_LEVEL=
```

相关链接

- [传输层安全性](#)

概述

Xamarin.Android 4.10 通过使用 `_Gdb` MSBuild 目标引入了对使用 `gdb` 的部分支持。

NOTE

`gdb` 支持要求安装 Android NDK。

有三种使用 `gdb` 的方法：

1. [启用快速部署的调试版本](#)。
2. [禁用快速部署的调试版本](#)。
3. [发行版本](#)。

出现错误时，请参阅[故障排除](#)部分。

使用快速部署的调试版本

构建和部署启用快速部署的调试版本时，可以使用 `_Gdb` MSBuild 目标附加 `gdb`。

首先，安装应用。可以通过 IDE 或命令行完成此操作：

```
$ /Library/Frameworks/Mono.framework/Commands/xbuild /t:Install *.csproj
```

其次，运行 `_Gdb` 目标。执行结束时，会打印 `gdb` 命令行：

```
$ /Library/Frameworks/Mono.framework/Commands/xbuild /t:_Gdb *.csproj
...
Target _Gdb:
    "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb" -x "/Users/jon/Development/Projects/Scratch.HelloXamarin20//gdb-symbols/gdb.env"
...
```

`_Gdb` 目标将启动在你的 `AndroidManifest.xml` 文件中声明的任意启动器活动。若要显式指定要运行的活动，请使用 `RunActivity` MSBuild 属性。此时不支持启动服务和其他 Android 构造。

`_Gdb` 目标将创建一个 `gdb-symbols` 目录，并将目标 `/system/lib` 和 `$APPDIR/lib` 目录的内容复制到那里。

NOTE

`gdb-symbols` 目录的内容绑定到你部署到的 Android 目标，如果你更改目标，则不会自动替换。（请注意这个 bug。）如果你更改 Android 目标设备，则需要手动删除此目录。

最后，复制生成的 `gdb` 命令并在你的 shell 中执行它：

```
$ "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb"
-x "/Users/jon/Development/Projects/Scratch.HelloXamarin20/gdb-symbols/gdb.env"
GNU gdb (GDB) 7.3.1-gg2
...
(gdb) bt
#0  0x40082e84 in nanosleep () from /Users/jon/Development/Projects/Scratch.HelloXamarin20/gdb-symbols/libc.so
#1  0x4008ffe6 in sleep () from /Users/jon/Development/Projects/Scratch.HelloXamarin20/gdb-symbols/libc.so
#2  0x74e46240 in ?? ()
#3  0x74e46240 in ?? ()
(gdb) c
```

不使用快速部署的调试版本

使用快速部署的调试版本, 方法是将 Android NDK 的 `gdbserver` 程序复制到快速部署 `__override__` 目录。快速部署禁用时, 此目录可能不存在。

有两种解决方法:

- 设置 `debug.mono.log` 系统属性, 以便创建 `__override__` 目录。
- 将 `gdbserver` 包含在你的 `.apk` 中。

设置 `debug.mono.log` 系统属性

要设置 `debug.mono.log` 系统属性, 请使用 `adb` 命令:

```
$ adb shell setprop debug.mono.log gc
```

系统属性设置后, 请执行 `_Gdb` 目标和打印的 `gdb` 命令, 就像“使用快速部署的调试版本”配置一样:

```
$ /Library/Frameworks/Mono.framework/Commands/xbuild /t:_Gdb *.csproj
...
Target _Gdb:
  "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb" -x "/Users/jon/Development/Projects/Scratch.HelloXamarin20/gdb-symbols/gdb.env"
...
$ "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb"
-x "/Users/jon/Development/Projects/Scratch.HelloXamarin20/gdb-symbols/gdb.env"
GNU gdb (GDB) 7.3.1-gg2
...
(gdb) c
```

将 `gdbserver` 包含在你的应用中

将 `gdbserver` 包含在你的应用中:

1. 在你的 Android NDK 中找到 `gdbserver` (它应位于 `$ANDROID_NDK_PATH/prebuilt/android-arm/gdbserver/gdbserver` 中), 并将其复制到你的 Project 目录中。
2. 将 `gdbserver` 重命名为 `libs/armeabi-v7a/libgdbserver.so`。
3. 使用 `AndroidNativeLibrary` 的“生成操作”将 `libs/armeabi-v7a/libgdbserver.so` 添加到你的项目。
4. 重新生成并重新安装你的应用程序。

重新安装应用后, 请执行 `_Gdb` 目标和打印的 `gdb` 命令, 就像“使用快速部署的调试版本”配置一样:

```
$ /Library/Frameworks/Mono.framework/Commands/xbuild /t:_Gdb *.csproj
...
Target _Gdb:
    "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb" -x "/Users/jon/Development/Projects/Scratch.HelloXamarin20//gdb-symbols/gdb.env"
...
$ "/opt/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/darwin-x86/bin/arm-linux-androideabi-gdb"
-x "/Users/jon/Development/Projects/Scratch.HelloXamarin20//gdb-symbols/gdb.env"
GNU gdb (GDB) 7.3.1-gg2
...
(gdb) c
```

发行版本

`gdb` 支持需要三项内容：

1. `INTERNET` 权限。
2. 应用调试已启用。
3. 可访问的 `gdbserver`。

默认在调试应用中启用 `INTERNET` 权限。如果它在你的应用程序中尚未出现，可以通过编辑 **Properties/AndroidManifest.xml** 或“项目属性”来添加它。

应用调试可通过下面的方法启用：将 `ApplicationAttribute.Debugging` 自定义属性设置为 `true`，或编辑 **Properties/AndroidManifest.xml**，并将 `//application/@android:debuggable` 属性设置为 `true`：

```
<application android:label="Example.Name.Here" android:debuggable="true">
```

通过遵守[不使用快速部署的调试版本](#)部分的内容，可以提供可访问的 `gdbserver`。

一个问题：`_Gdb` MSBuild 目标将终止以前运行的任何应用实例。这不适用于 Android v4.0 之前的目标。

疑难解答

`mono_pmip` 不起作用

从 `libmonosgen-2.0.so` 导出 `mono_pmip` 函数（用于[获取托管堆栈帧](#)），`_Gdb` 目标目前不会下拉。（未来版本会修复此问题。）

要启用位于 `libmonosgen-2.0.so` 中的调用功能，请将其从目标设备复制到 `gdb-symbols` 目录中：

```
$ adb pull /data/data/Mono.Android.DebugRuntime/lib/libmonosgen-2.0.so Project/gdb-symbols
```

然后重新启动调试会话。

总线错误：运行 `gdb` 命令时为 10

当 `gdb` 命令出错，显示 "Bus error: 10" 时，重新启动 Android 设备。

```
$ "/path/to/arm-linux-androideabi-gdb" -x "Project/gdb-symbols/gdb.env"
GNU gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 Free Software Foundation, Inc.
...
Bus error: 10
$
```


连接后没有堆栈跟踪

```
$ "/path/to/arm-linux-androideabi-gdb" -x "Project/gdb-symbols/gdb.env"
GNU gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 Free Software Foundation, Inc.
...
(gdb) bt
No stack.
```

这通常表示 `gdb-symbols` 目录的内容与你的 Android 目标不同步。(是否更改 Android 目标?)

请删除 `gdb-symbols` 目录并重试。

在 Android 上链接

2018/10/26 • [Edit Online](#)

Xamarin.Android 应用程序使用链接器缩减应用程序大小。链接器使用应用程序的静态分析来确定实际使用的程序集、类型以及成员。然后，链接器像垃圾回收器一样运行，不断寻找被引用的程序集、类型和成员，直到找到引用的程序集、类型和成员的完整闭包。然后，放弃此闭包之外的所有内容。

例如，[Hello, Android](#) 示例：

| 配置 | 1.2.0 大小 | 4.0.1 大小 |
|---------------|----------|----------|
| 在不链接的情况下进行发布： | 14.0 MB | 16.0 MB |
| 在链接的情况下进行发布： | 4.2 MB | 2.9 MB |

链接导致程序包的大小是 1.2.0 中原始(未链接)程序包大小的 30%，是 4.0.1 中未链接程序包大小的 18%。

控件

链接基于静态分析。因此，不会检测到任何依赖于运行时环境的内容：

```
// To play along at home, Example must be in a different assembly from MyActivity.
public class Example {
    // Compiler provides default constructor...
}

[Activity (Label="Linker Example", MainLauncher=true)]
public class MyActivity {
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Will this work?
        var o = Activator.CreateInstance (typeof (ExampleLibrary.Example));
    }
}
```

链接器行为

用于控制链接器的主要机制为“项目选项”对话框中的“链接器行为”(在 Visual Studio 中为“链接”)下拉列表。有三个选项：

1. 不链接(在 Visual Studio 中为“无”)
2. 链接 **SDK 程序集**(仅 SDK 程序集)
3. 链接所有程序集(SDK 和用户程序集)

“不链接”选项会关闭链接器；上述“在不链接的情况下进行发布”应用程序大小示例使用了此行为。这对排除运行时故障很有用，可了解链接器是否负责。通常不建议将此设置用于生产版本。

“链接 SDK 程序集”选项仅链接 [Xamarin.Android 附带的程序集](#)。不会链接所有其他程序集(如你的代码)。

“链接所有程序集”选项将链接所有程序集，这意味着如果没有静态引用，你的代码也可能会被删除。

上述示例将使用“不链接”和“链接 SDK 程序集”选项，并且使用“链接所有程序集”行为将会失败，并生成以下错误：

```

E/mono (17755): [0xafd4d440:] EXCEPTION handling: System.MissingMethodException: Default constructor not
found for type ExampleLibrary.Example.
I/MonoDroid(17755): UNHANDLED EXCEPTION: System.MissingMethodException: Default constructor not found for
type ExampleLibrary.Example.
I/MonoDroid(17755): at System.Activator.CreateInstance (System.Type,bool) <0x00180>
I/MonoDroid(17755): at System.Activator.CreateInstance (System.Type) <0x00017>
I/MonoDroid(17755): at LinkerScratch2.Activity1.OnCreate (Android.OS.Bundle) <0x00027>
I/MonoDroid(17755): at Android.App.Activity.n_OnCreate_Landroid_os_Bundle_ (IntPtr,intptr,intptr) <0x00057>
I/MonoDroid(17755): at (wrapper dynamic-method) object:95bb4fbe-bef8-4e5b-8e99-ca83a5d7a124
(IntPtr,intptr,intptr) <0x00033>
E/mono (17755): [0xafd4d440:] EXCEPTION handling: System.MissingMethodException: Default constructor not
found for type ExampleLibrary.Example.
E/mono (17755):
E/mono (17755): Unhandled Exception: System.MissingMethodException: Default constructor not found for type
ExampleLibrary.Example.
E/mono (17755): at System.Activator.CreateInstance (System.Type type, Boolean nonPublic) [0x00000] in
<filename unknown>:0
E/mono (17755): at System.Activator.CreateInstance (System.Type type) [0x00000] in <filename unknown>:0
E/mono (17755): at LinkerScratch2.Activity1.OnCreate (Android.OS.Bundle bundle) [0x00000] in <filename
unknown>:0
E/mono (17755): at Android.App.Activity.n_OnCreate_Landroid_os_Bundle_ (IntPtr JNIEnv, IntPtr
native_this, IntPtr native_savedInstanceState) [0x00000] in <filename unknown>:0
E/mono (17755): at (wrapper dynamic-method) object:95bb4fbe-bef8-4e5b-8e99-ca83a5d7a124
(IntPtr,intptr,intptr)

```

保留代码

链接器有时会删除你想要保留的代码。例如:

- 你可能拥有通过 `System.Reflection.MemberInfo.Invoke` 动态调用的代码。
- 如果你动态实例化类型,则需要保留类型的默认构造函数。
- 如果你使用 XML 序列化,则需要保留类型的属性。

在这些情况下,你可以使用 [Android.Runtime.Preserve](#) 属性。应用程序未静态链接的每个成员都可能被删除,因此可以使用此属性标记未被静态引用但应用程序仍然需要的成员。你可以将此属性应用于某种类型的每个成员或类型本身。

在以下示例中,此属性用于保留 `Example` 类的构造函数:

```

public class Example
{
    [Android.Runtime.Preserve]
    public Example ()
    {
    }
}

```

如果要保留整个类型,可以使用以下属性语法:

```
[Android.Runtime.Preserve (AllMembers = true)]
```

例如,在以下代码段中,整个 `Example` 类保留用于 XML 序列化:

```

[Android.Runtime.Preserve (AllMembers = true)]
class Example
{
    // Compiler provides default constructor...
}

```

有时你需要保留某些成员，但只有在保留了包含类型的情况下才可以操作。在这些种况下，请使用以下属性语法：

```
[Android.Runtime.Preserve (Conditional = true)]
```

如果不想采用 Xamarin 库上的依赖项(例如，生成一个跨平台可移植类库(PCL))，你仍然可以使用

`Android.Runtime.Preserve` 属性。为此，请在 `Android.Runtime` 命名空间内声明一个 `PreserveAttribute` 类，如下所示：

```
namespace Android.Runtime
{
    public sealed class PreserveAttribute : System.Attribute
    {
        public bool AllMembers;
        public bool Conditional;
    }
}
```

在以上示例中，`Preserve` 属性在 `Android.Runtime` 命名空间进行了声明；但是，你可以在任何命名空间使用 `Preserve` 属性，因为链接器会按类型名称查找此属性。

falseflag

如果不能使用 `[Preserve]` 属性，提供一段代码以便链接器相信该类型被使用通常很有用，但同时需防止代码块在运行时被执行。若要利用此技术，我们可以执行以下操作：

```
[Activity (Label="Linker Example", MainLauncher=true)]
class MyActivity {

    #pragma warning disable 0219, 0649
    static bool falseflag = false;
    static MyActivity ()
    {
        if (falseflag) {
            var ignore = new Example ();
        }
    }
    #pragma warning restore 0219, 0649

    // ...
}
```

linkskip

可以指定根本不应链接一组用户提供的程序集，同时允许使用 `AndroidLinkSkip` MSBuild 属性通过“链接 SDK 程序集”行为跳过其他用户程序集：

```
<PropertyGroup>
    <AndroidLinkSkip>Assembly1;Assembly2</AndroidLinkSkip>
</PropertyGroup>
```

LinkDescription

可以在包含 [自定义链接器配置文件](#) 的文件上使用 `@(LinkDescription)` 生成操作。要保留需要保留的 `internal` 或 `private` 成员，可能需要自定义链接器配置文件。

自定义特性

链接程序集时，将从所有成员中删除以下自定义属性类型：

- `System.ObsoleteAttribute`

- `System.MonoDocumentationNoteAttribute`
- `System.MonoExtensionAttribute`
- `System.MonoInternalNoteAttribute`
- `System.MonoLimitationAttribute`
- `System.MonoNotSupportedAttribute`
- `System.MonoTODOAttribute`
- `System.Xml.MonoFIXAttribute`

链接程序集时，将从发行版本的所有成员中删除以下自定义属性类型：

- `System.Diagnostics.DebuggableAttribute`
- `System.Diagnostics.DebuggerBrowsableAttribute`
- `System.Diagnostics.DebuggerDisplayAttribute`
- `System.Diagnostics.DebuggerHiddenAttribute`
- `System.Diagnostics.DebuggerNonUserCodeAttribute`
- `System.Diagnostics.DebuggerStepperBoundaryAttribute`
- `System.Diagnostics.DebuggerStepThroughAttribute`
- `System.Diagnostics.DebuggerTypeProxyAttribute`
- `System.Diagnostics.DebuggerVisualizerAttribute`

相关链接

- [自定义链接器配置](#)
- [在 iOS 上链接](#)

多核设备和 Xamarin.Android

2018/11/5 • [Edit Online](#)

Android 可以在几种不同的计算机体系结构上运行。本文档讨论可为一个 Xamarin.Android 应用程序部署的不同 CPU 体系结构。本文还将介绍如何打包 Android 应用程序以支持不同的 CPU 体系结构。将介绍应用程序二进制接口 (ABI)，并且将提供有关在 Xamarin.Android 应用程序中使用哪些 ABI 的指导。

概述

Android 允许创建“胖二进制文件”，这是一个 `.apk` 文件，其中包含将支持多种不同的 CPU 体系结构的计算机代码。这是通过将每段计算机代码与应用程序二进制接口相关联来实现的。ABI 用于控制哪些计算机代码将在给定的硬件设备上运行。例如，对于在 x86 设备上运行的 Android 应用程序，编译应用程序时需要包含 x86 ABI 支持。

具体而言，每个 Android 应用程序将至少支持一个嵌入式应用程序二进制接口 (EABI)。EABI 是特定于嵌入式软件程序的约定。典型的 EABI 将描述如下内容：

- CPU 指令集。
- 内存的字节排序方式在运行时存储和加载。
- 对象文件和程序库的二进制格式，以及这些文件和库中允许或支持的内容类型。
- 用于在应用程序代码和系统之间传递数据的各种约定（例如：调用函数时如何使用寄存器和/或堆栈、对齐约束等）
- 枚举类型、结构、字段和数组的对齐和大小约束。
- 运行时可用于计算机代码的函数符号列表，通常来自一组专门选择的库。

armeabi 和线程安全性

应用程序二进制接口将在下面详细讨论，但务必要记住 Xamarin.Android 使用的 `armeabi` 运行时不具线程安全性。如果将具有 `armeabi` 支持的应用程序部署到 `armeabi-v7a` 设备，则将会发生许多奇怪且无法解释的异常情况。

由于 Android 4.0.0、4.0.1、4.0.2 和 4.0.3 中存在 bug，即使存在 `armeabi-v7a` 目录且设备为 `armeabi-v7a` 设备，也会从 `armeabi` 目录中选取本机库。

NOTE

Xamarin.Android 将确保 `.so` 以正确的顺序添加到 APK。此 bug 对 Xamarin.Android 的用户来说应该不是问题。

ABI 说明

Android 支持的每个 ABI 均由唯一名称标识。

armeabi

这是基于 ARM 的至少支持 ARMv5TE 指令集的 CPU 的 EABI 名称。Android 遵循小字节序 ARM GNU/Linux ABI。此 ABI 不支持硬件辅助的浮点计算。所有 FP 操作都来自编译器的 `libgcc.a` 静态库的软件帮助程序函数执行。`armeabi` 不支持 SMP 设备。

注意：Xamarin.Android 的 `armeabi` 代码不具线程安全性，不应在多 CPU `armeabi-v7a` 设备上使用（如下所述）。在单核 `armeabi-v7a` 设备上使用 `arembi` 代码是安全的。

armeabi-v7a

这是另一种基于 ARM 的 CPU 指令集, 可扩展上述 `armeabi` EABI。`armeabi-v7a` EABI 支持硬件浮点操作和多个 CPU (SMP) 设备。使用 `armeabi-v7a` EABI 的应用程序与使用 `armeabi` 的应用程序相比, 可以获得极大的性能改进。

注意: `armeabi-v7a` 计算机代码将不会在 ARMv5 设备上运行。

arm64-v8a

这是基于 ARMv8 CPU 体系结构的 64 位指令集。此体系结构用于 *Nexus 9*。Xamarin.Android 5.1 为此体系结构提供实验性支持(有关详细信息, 请参阅[实验性功能](#))。

x86

这是支持通常名为 x86 或 IA-32 的指令集的 CPU 的 ABI 名称。此 ABI 对应于 Pentium Pro 指令集的指令, 包括 MMX、SSE、SSE2 和 SSE3 指令集。不包括任何其他可选的 IA-32 指令集扩展, 例如:

- MOVBE 指令。
- 补充 SSE3 扩展 (SSSE3)。
- SSE4 的任何变体。

注意: 虽然 Google TV 在 x86 上运行, 但不受 Android 的 NDK 支持。

x86_64

这是支持 64 位 x86 指令集(也称为 x64 或 AMD64)的 CPU 的 ABI 名称。Xamarin.Android 5.1 为此体系结构提供实验性支持(有关详细信息, 请参阅[实验性功能](#))。

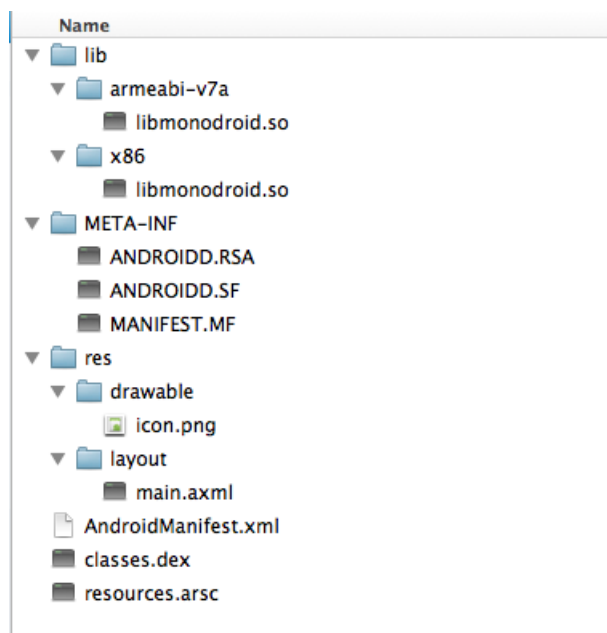
mips

这是基于 MIPS 的至少支持 `MIPS32r1` 指令集的 CPU 的 ABI 名称。Android 不支持 MIPS 16 和 `micromips`。

注意: Xamarin.Android 目前不支持 MIPS 设备, 但在未来的版本中将支持。

APK 文件格式

Android 应用程序包是包含 Android 应用程序所需的所有代码、资产、资源和证书的文件格式。它是一个 `.zip` 文件, 但使用 `.apk` 文件扩展名。展开后, Xamarin.Android 创建的 `.apk` 的内容可以在下面的屏幕截图中看到:



`.apk` 文件内容的快速说明:

- AndroidManifest.xml – 这是 `AndroidManifest.xml` 文件, 采用二进制 XML 格式。
- classes.dex – 这包含应用程序代码, 编译为 Android 运行时 VM 使用的 `dex` 文件格式。
- resources.arsc – 此文件包含应用程序的所有预编译资源。

- **lib** – 此目录包含每个 ABI 的编译代码。它将包含上一节中所述的每个 ABI 的一个子文件夹。在上面的屏幕截图中, 所涉及的 `.apk` 都具有 `armeabi-v7a` 和 `x86` 的本机库。
- **META-INF** – 此目录(如果存在)用于存储签名信息、包和扩展配置数据。
- **res** – 此目录包含未编译为 `resources.arsc` 的资源。

NOTE

文件 `libmonodroid.so` 是所有 Xamarin.Android 应用程序所需的本机库。

Android 设备 ABI 支持

每个 Android 设备支持在最多两个 ABI 中执行本机代码:

- “主”ABI – 这对应于系统映像中使用的计算机代码。
- “辅助”ABI – 这是也受系统映像支持的可选 ABI。

例如, 典型的 ARMv5TE 设备将仅具有主 ABI `armeabi`, 而 ARMv7 设备将指定主 ABI `armeabi-v7a` 和辅助 ABI `armeabi`。典型的 x86 设备将仅指定主 ABI `x86`。

Android 本机库安装

在包安装时, `.apk` 中的本机库被提取到应用的本机库目录中, 通常为 `/data/data/<package-name>/lib`, 此后称为 `$APP/lib`。

Android 的本机库安装行为在 Android 版本之间会发生显著变化。

安装本机库: Android 4.0 之前

4.0 Ice Cream Sandwich 之前的 Android 将仅从 `.apk` 内的单个 ABI 中提取本机库。此年份的 Android 应用将首先尝试为主 ABI 提取所有本机库, 如果不存在此类库, 则 Android 将为辅助 ABI 提取所有本机库。没有执行任何“合并”。

例如, 请考虑在 `armeabi-v7a` 设备上安装应用程序的情况。同时支持 `armeabi` 和 `armeabi-v7a` 的 `.apk`, 具有以下 ABI `lib` 目录以及其中的文件:

```
lib/armeabi/libone.so
lib/armeabi/libtwo.so
lib/armeabi-v7a/libtwo.so
```

安装完成后, 本机库目录将包含:

```
$APP/lib/libtwo.so # from the armeabi-v7a directory in the apk
```

换言之, 没有安装 `libone.so`。这将导致出现问题, 因为对于在运行时要加载的应用程序 `libone.so` 不存在。此行为是意外发生, 已被记录为 bug 并重新归类为“[按预期方式工作](#)”。

因此, 针对 4.0 之前的 Android 版本, 有必要为应用程序将支持的每个 ABI 提供所有本机库, 即 `.apk` 应包含:

```
lib/armeabi/libone.so
lib/armeabi/libtwo.so
lib/armeabi-v7a/libone.so
lib/armeabi-v7a/libtwo.so
```

安装本机库: Android 4.0 – Android 4.0.3

Android 4.0 Ice Cream Sandwich 更改了提取逻辑。它将枚举所有本机库, 查看是否已经提取文件的基本名称, 并且如果满足以下两个条件, 则将提取该库:

- 它还没有被提取。
- 本机库的 ABI 匹配目标的主或辅助 ABI。

满足这些条件即允许“合并”行为;也就是说,如果我们有一个具有以下内容的 `.apk`:

```
lib/armeabi/libone.so
lib/armeabi/libtwo.so
lib/armeabi-v7a/libtwo.so
```

安装完成后,本机库目录将包含:

```
$APP/lib/libone.so
$APP/lib/libtwo.so
```

遗憾的是,此行为依赖于顺序,如以下文档中所述 - [问题 24321:当 armeabi 和 armeabi-v7a 同时包含在 apk 中时, Galaxy Nexus 4.0.2 使用 armeabi 本机代码。](#)

本机库“按顺序”(例如,通过解压缩列出的顺序)进行处理,并提取第一个匹配项。由于 `.apk` 包含 `libtwo.so` 的 `armeabi` 和 `armeabi-v7a` 版本,并且首先列出 `armeabi`,它是提取的 `armeabi` 版本,不是 `armeabi-v7a` 版本:

```
$APP/lib/libone.so # armeabi
$APP/lib/libtwo.so # armeabi, NOT armeabi-v7a!
```

此外,即使同时指定了 `armeabi` 和 `armeabi-v7a` ABI(如下面的声明支持的 ABI 部分所述),Xamarin.Android 将在 `.csproj`:

```
<AndroidSupportedAbis>armeabi,armeabi-v7a</AndroidSupportedAbis>
```

因此,首先将找到 `.apk` 中的 `armeabi` `libmonodroid.so`,并且将会提取 `armeabi` `libmonodroid.so`,即使 `armeabi-v7a` `libmonodroid.so` 存在并针对目标进行了优化。这也会导致隐蔽的运行时错误,因为 `armeabi` 不具 SMP 安全性。

安装本机库:Android 4.0.4 及更高版本

Android 4.0.4 更改了提取逻辑:它将枚举所有本机库,读取文件的基本名称,然后提取主 ABI 版本(如果存在)或辅助 ABI(如果存在)。这允许“合并”行为;也就是说,如果我们有一个具有以下内容的 `.apk`:

```
lib/armeabi/libone.so
lib/armeabi/libtwo.so
lib/armeabi-v7a/libtwo.so
```

安装完成后,本机库目录将包含:

```
$APP/lib/libone.so # from armeabi
$APP/lib/libtwo.so # from armeabi-v7a
```

Xamarin.Android 和 ABI

Xamarin.Android 支持以下体系结构:

- `armeabi`
- `armeabi-v7a`
- `x86`

Xamarin.Android 为以下体系结构提供实验性支持：

- arm64-v8a
- x86_64

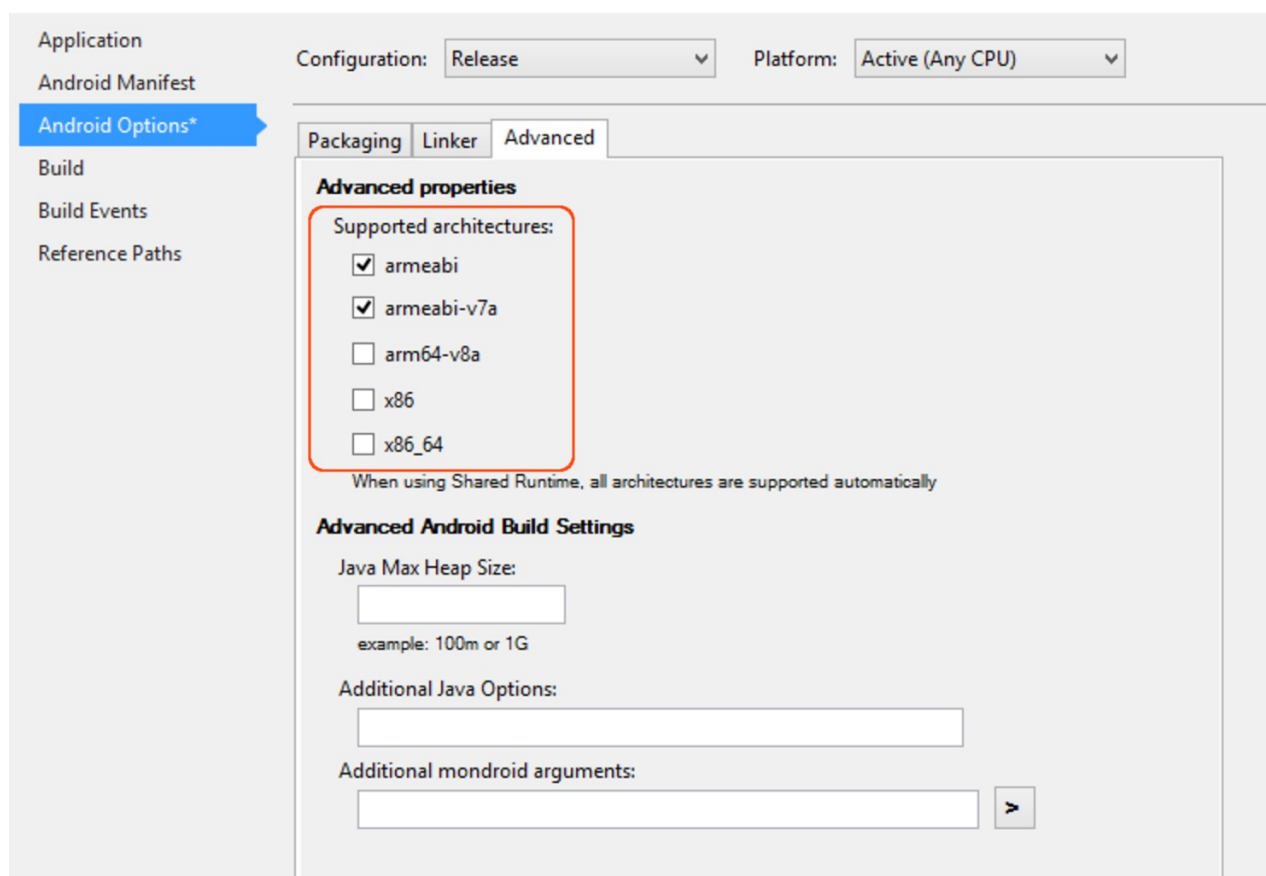
NOTE

自 2018 年 8 月起新应用需要面向 API 级别 26, 自 2019 年 8 月起, 除 32 位版本之外, 应用还[需要提供 64 位版本](#)。

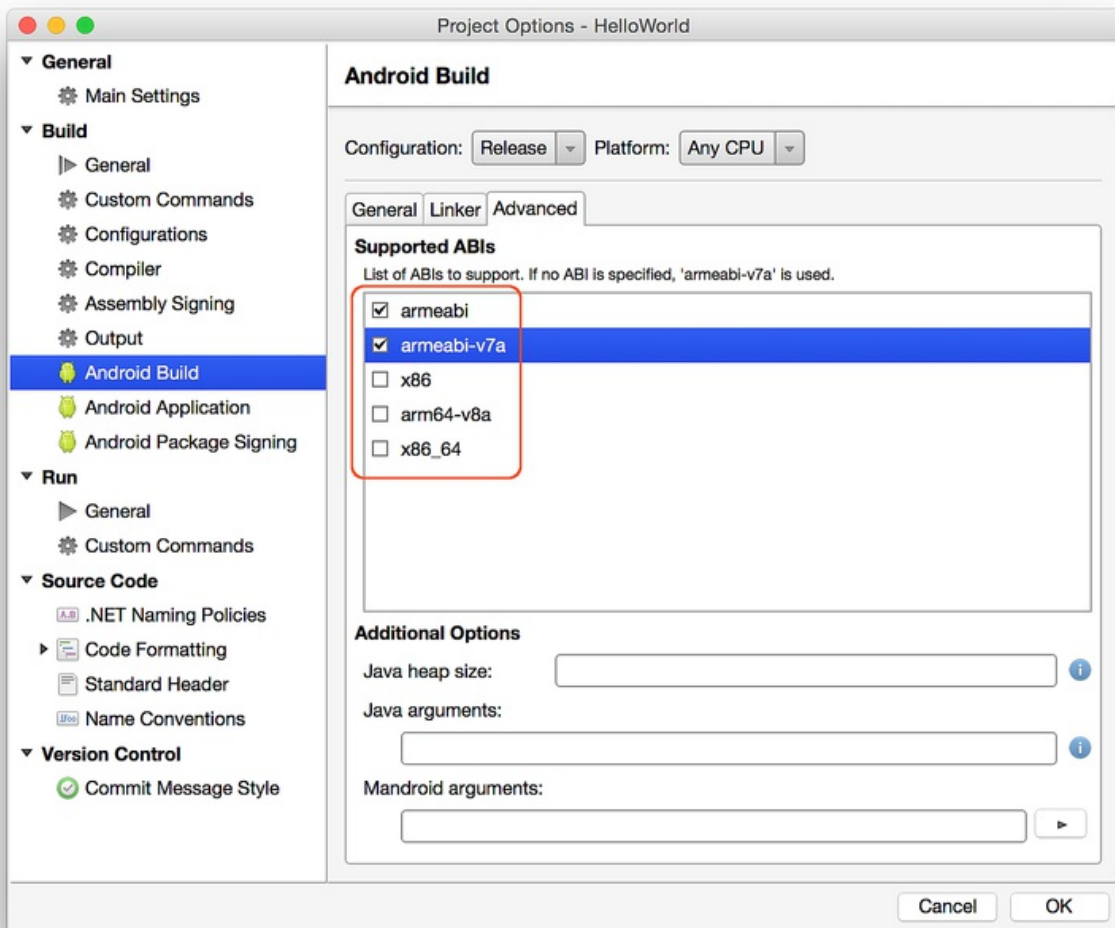
Xamarin.Android 当前不支持 mips。

声明受支持的 ABI

默认情况下, 对于发行版本, Xamarin.Android 默认为 armeabi-v7a, 对于调试版本, 则为 armeabi-v7a 和 x86。通过 Xamarin.Android 项目的项目选项可以设置对不同 ABI 的支持。在 Visual Studio 中, 可在“高级”选项卡下的项目“属性”的“Android 选项”页中进行此设置, 如以下屏幕截图所示：



在 Visual Studio for Mac 中, 可在“高级”选项卡下的“项目选项”的“Android 版本”页上选择支持的体系结构, 如以下屏幕截图所示：



在某些可能需要声明额外的 ABI 支持的情况，例如以下情况：

- 将应用程序部署到 `x86` 设备。
- 将应用程序部署到 `armeabi-v7a` 设备以确保线程安全性。

总结

本文档讨论可运行 Android 应用程序的不同 CPU 体系结构。它介绍了应用程序二进制接口以及 Android 如何使用它来支持不同的 CPU 体系结构。然后接着讨论如何在 Xamarin.Android 应用程序中指定 ABI 支持，并强调了在仅适用于 `armeabi` 的 `armeabi-v7a` 设备上使用 Xamarin.Android 应用程序时出现的问题。

相关链接

- [MIPS 体系结构](#)
- [ARM 体系结构的 ABI \(PDF\)](#)
- [Android NDK](#)
- [问题 9089:Nexus One - 如果 armeabi-v7a 中至少有一个库，则不会加载来自 armeabi 的任何本机库](#)
- [问题 24321:armeabi 和 armeabi-v7a 同时包含在 apk 中时，Galaxy Nexus 4.0.2 使用 armeabi 本机代码](#)

Xamarin.Android 性能

2018/10/26 • [Edit Online](#)

可以通过很多方法来提高使用 Xamarin.Android 构建的应用程序的性能。这些方法共同可以极大地降低由 CPU 执行的工作量和应用程序占用的内存量。本文将介绍并讨论这些方法。

性能概述

应用程序性能差表现在许多方面。这会造成应用程序看起来无响应，导致滚动缓慢，还可降低电池寿命。但是，优化性能不止需要实现高效的代码。还必须考虑用户对应用程序性能的体验。例如，确保操作执行不会妨碍用户执行其他活动，这有助于改进用户的体验。

可以通过许多方法提高使用 Xamarin.Android 构建的应用程序的性能和感知性能。它们包括：

- [优化布局层次结构](#)
- [优化列表视图](#)
- [在活动中删除事件处理程序](#)
- [限制服务的有效期](#)
- [收到通知时释放资源](#)
- [隐藏用户界面时释放资源](#)
- [优化图像资源](#)
- [释放未使用的图像资源](#)
- [避免使用浮点运算](#)
- [关闭对话框](#)

NOTE

阅读本文之前，首先应阅读[跨平台性能](#)，其中讨论了非平台特定方法，可用于改善使用 Xamarin 平台生成的应用程序的内存使用情况和性能。

优化布局层次结构

添加到应用程序的每个布局都需要执行初始化、布局和绘制。嵌套使用 `weight` 参数的 `LinearLayout` 实例时，布局过程可能很昂贵，因为每个子级都将测量两次。使用 `LinearLayout` 的嵌套实例可能产生深层视图层次结构，这可能导致将布局的较差性能加倍放大，例如在 `ListView` 中。因此，对这种布局进行优化很重要，因为之后的性能优势会成倍增加。

例如，考虑对具有图标、标题和说明的列表视图行使用 `LinearLayout` 的情况。`LinearLayout` 将包含 `ImageView` 以及包含两个 `TextView` 实例的垂直 `LinearLayout`：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="5dip">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="5dip"
        android:src="@drawable/icon" />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="0dip"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Mei tempor iuvaret ad." />
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:singleLine="true"
            android:ellipsize="marquee"
            android:text="Lorem ipsum dolor sit amet." />
    </LinearLayout>
</LinearLayout>

```

此布局的层级为 3 级，当放大 `ListView` 行时则是一种浪费。但是，可以通过平展布局改善这种情况，如以下代码示例所示：

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="5dip">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentBottom="true"
        android:layout_marginRight="5dip"
        android:src="@drawable/icon" />
    <TextView
        android:id="@+id/secondLine"
        android:layout_width="fill_parent"
        android:layout_height="25dip"
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:text="Lorem ipsum dolor sit amet." />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_above="@id/secondLine"
        android:layout_alignWithParentIfMissing="true"
        android:gravity="center_vertical"
        android:text="Mei tempor iuvaret ad." />
</RelativeLayout>

```

以前的 3 级层次结构已缩减为 2 级层次结构，并且一个 `RelativeLayout` 取代了两个 `LinearLayout` 实例。放大布局的每个 `ListView` 行时，性能将显著提高。

优化列表视图

用户期望平滑滚动并快速加载 `ListView` 实例。但是，当每个列表视图行包含深度嵌套的视图层次结构或列表视图行包含复杂布局时，滚动性能会降低。但是，可以使用一些方法避免出现不佳的 `ListView` 性能：

- 重复使用行视图 有关详细信息，请参阅[重复使用行视图](#)。
- 尽量平展布局。
- 缓存从 Web 服务检索的行内容。
- 避免缩放图像。

结合使用这些方法有助于保持 `ListView` 实例的平滑滚动。

重复使用行视图

在 `ListView` 中显示数百个行时，若一次仅在屏幕上显示其中一小部分，创建数百个 `View` 对象则是在浪费内存。相反，应仅将屏幕上行中显示的 `View` 对象加载到内存中，同时内容将加载到这些重复使用的对象中。这可以防止实例化数百个其他对象，从而节省时间和内存。

因此，当某一行从屏幕上消失后，可以将其视图放到队列中以供重复使用，如下面的代码示例所示：

```
public override View GetView(int position, View convertView, ViewGroup parent)
{
    View view = convertView; // re-use an existing view, if one is supplied
    if (view == null) // otherwise create a new one
        view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);
    // set view properties to reflect data for the given row
    view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = items[position];
    // return the view, populated with data, for display
    return view;
}
```

当用户滚动时，`ListView` 调用 `GetView` 重写以请求显示新视图，（如果有）它会在 `convertView` 参数中传递一个未使用的视图。如果此值为 `null`，则代码将创建一个新的 `View` 实例，否则可以重置和重复使用 `convertView` 属性。

有关详细信息，请参阅[用数据填充 ListView](#) 中的[重复使用行视图](#)。

在活动中删除事件处理程序

当在 Android 运行时中销毁活动时，该活动仍可能在 Mono 运行时处于活动状态。因此，请在 `Activity.OnPause` 中删除外部对象的事件处理程序，以防止运行时继续引用已销毁的活动。

在活动中，在类级别处声明事件处理程序：

```
EventHandler<UpdatingEventArgs> service1UpdateHandler;
```

然后在活动中实现该处理程序，如在 `OnResume` 中：

```
service1UpdateHandler = (object s, UpdatingEventArgs args) => {
    this.RunOnUiThread (() => {
        this.updateStatusText1.Text = args.Message;
    });
};
App.Current.Service1.Updated += service1UpdateHandler;
```

当活动退出运行状态时，会调用 `OnPause`。在 `OnPause` 实现中，删除处理程序，如下所示：

```
App.Current.Service1.Updated -= service1UpdateHandler;
```

限制服务的有效期

服务启动时，Android 会持续运行该服务进程。这使得进程很昂贵，因为其内存不能分页或在其他地方使用。如果持续运行已不需要的服务，会增加应用程序由于内存限制而出现性能降低的风险。还可导致应用程序切换效率降低，因为它减少了 Android 可以缓存的进程数。

使用 `IntentService` 可以限制服务的有效期，处理完最初启动的目的时服务便可自动终止。

收到通知时释放资源

在应用程序生命周期内，`OnTrimMemory` 回调会在设备内存较低时发出通知。应实现此回调以侦听以下内存级别的通知：

- `TrimMemoryRunningModerate` - 应用程序可能需要释放一些不需要的资源。
- `TrimMemoryRunningLow` - 应用程序应释放不需要的资源。

- `TrimMemoryRunningCritical` - 应用程序应尽可能多地释放非关键进程。

此外，缓存应用程序进程时，`OnTrimMemory` 回调可能会收到以下内存级别的通知：

- `TrimMemoryBackground` - 释放如果用户返回到应用，则可以快速高效地重新生成的资源。
- `TrimMemoryModerate` - 释放资源有助于系统继续缓存其他进程以提高整体性能。
- `TrimMemoryComplete` - 如果未能尽快恢复更多内存，将很快终止应用程序进程。

应根据接收到的级别释放资源来响应通知。

隐藏用户界面时释放资源

在用户导航到另一应用时，释放该应用的用户界面使用的所有资源，因为这样可以显著提高 Android 的缓存进程容量，这反过来会影响用户体验质量。

若要在用户退出用户界面时接收通知，需在 `Activity` 类中实现 `OnTrimMemory` 回调，并侦听 `TrimMemoryUiHidden` 级别，该值指示用户界面已从视图中隐藏。仅当应用程序的所有用户界面组件均对用户隐藏时，才会收到此通知。收到此通知时释放用户界面资源，可确保当用户从应用中的另一个活动导航回来后，用户界面资源仍可用于快速恢复活动。

优化图像资源

图像是应用程序使用的一些最昂贵的资源，通常以高分辨率捕获。因此，显示图像时，请采用设备屏幕所必需的分辨率显示它。如果图像的分辨率比屏幕高，则应降低。

有关详细信息，请参阅[跨平台性能](#)指南中的[优化图像资源](#)。

释放未使用的图像资源

为了节省内存使用，最好释放不再需要的大型图像资源。但是，请务必确保图像被正确地释放。可以利用 `using` 语句以确保正确使用 `.Dispose()` 对话，而不利用显式 `IDisposable` 调用。

例如，[位图](#)类可实现 `IDisposable`。在 `using` 块中包装 `Bitmap` 对象的实例化可确保从块中退出时它能被正确地释放：

```
using (Bitmap smallPic = BitmapFactory.DecodeByteArray(smallImageByte, 0, smallImageByte.Length))
{
    // Use the smallPic bit map here
}
```

有关释放可释放资源的详细信息，请参阅[释放 IDisposable 资源](#)。

避免使用浮点运算

在 Android 设备上，浮点运算比整数运算的速度大约慢两倍。因此，应尽可能用整数运算取代浮点运算。但是，`float` 和 `double` 运算在最新硬件上不存在执行时间差异。

NOTE

甚至对于整数运算，一些 CPU 也缺少硬件划分功能。因此，通常在软件中执行整数除法和取模运算。

关闭对话框

对话框的操作目的完成后，当使用 `ProgressDialog` 类(或任何对话框或警报)而不是调用 `Hide` 方法时，请调用

[Dismiss](#) 方法。否则，该对话框将仍处于活动状态，并由于继续引用活动而导致泄露活动。

总结

本文介绍和讨论了提高使用 Xamarin.Android 构建的应用程序的性能的方法。这些方法共同可以极大地降低由 CPU 执行的工作量和应用程序占用的内存量。

相关链接

- [跨平台性能](#)

分析 Android 应用

2018/10/26 • [Edit Online](#)

在将应用部署到应用商店之前，必须先识别并修复任何性能瓶颈、过度占用内存或网络资源利用效率低下的问题。可以使用两个探查器工具来实现此目的：

- Xamarin Profiler
- Android Studio 中的 Android Profiler

本指南介绍了 Xamarin Profiler，并提供了有关如何开始使用 Android Profiler 的详细信息。

Xamarin Profiler

Xamarin Profiler 是独立的应用程序，与 Visual Studio 和 Visual Studio for Mac 集成，用于从 IDE 中分析 Xamarin 应用。有关使用 Xamarin Profiler 的详细信息，请参阅 [Xamarin Profiler](#)。

NOTE

必须订阅 [Visual Studio Enterprise](#) 才能解锁 Windows 版 Visual Studio Enterprise 或 Visual Studio for Mac 中的 Xamarin Profiler 功能。

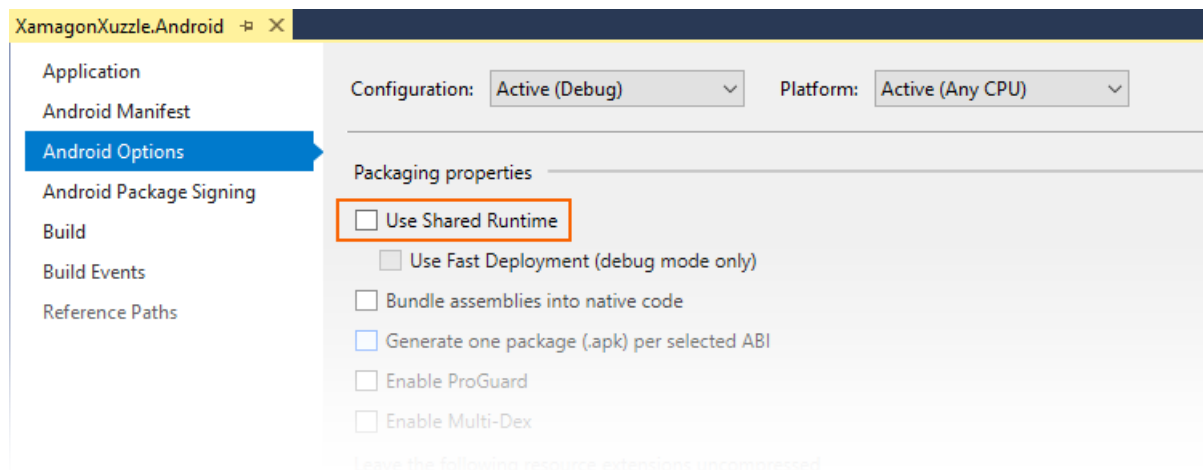
Android Studio Profiler

Android Studio 3.0 及更高版本含有 Android Profiler 工具。可以使用 Android Profiler 来衡量使用 Visual Studio 生成的 Xamarin Android 应用的性能 – 而无需 Visual Studio Enterprise 许可证。但是，与 Xamarin Profiler 不同，Android Profiler 没有与 Visual Studio 集成，并且只能用于分析已提前生成并导入 Android Profiler 的 Android 应用程序包 (APK)。

在 Android Profiler 中启动 Xamarin Android 应用

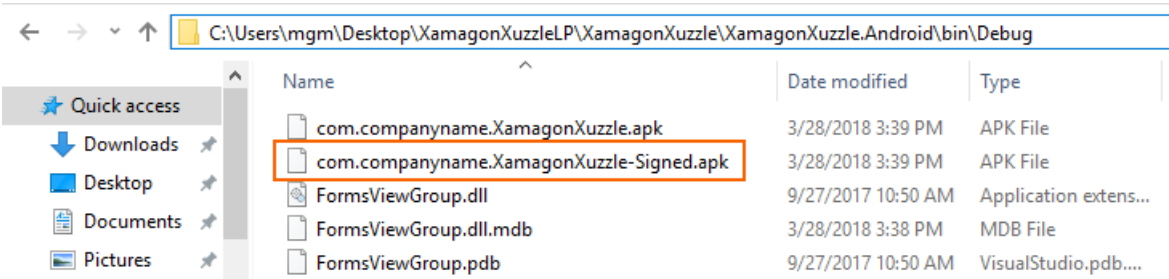
下列步骤介绍如何在 Android Studio 的 Android Profiler 工具中启动 Xamarin Android 应用程序。在下面的示例屏幕截图中，Xamarin 窗体 [XamagonXuzzle](#) 应用是使用 Android Profiler 生成和分析的：

1. 在 Android 项目生成选项中，禁用“使用共享运行时”。这可确保 Android 应用程序包 (APK) 的生成不依赖于共享开发时间 Mono 运行时。

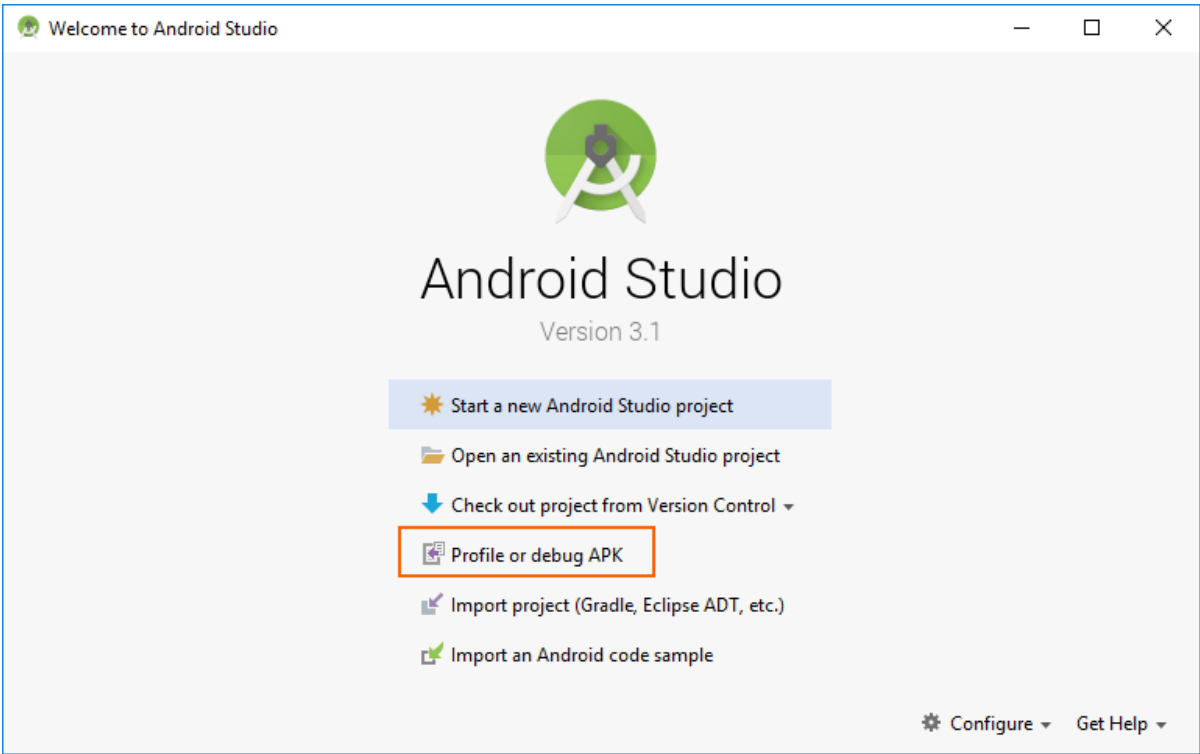


2. 生成用于“调试”的应用，并将其部署到物理设备或仿真器中。这可生成 APK 的已签名“调试”版本。对于 XamagonXuzzle 示例，生成的 APK 名为 com.companyname.XamagonXuzzle Signed.apk。

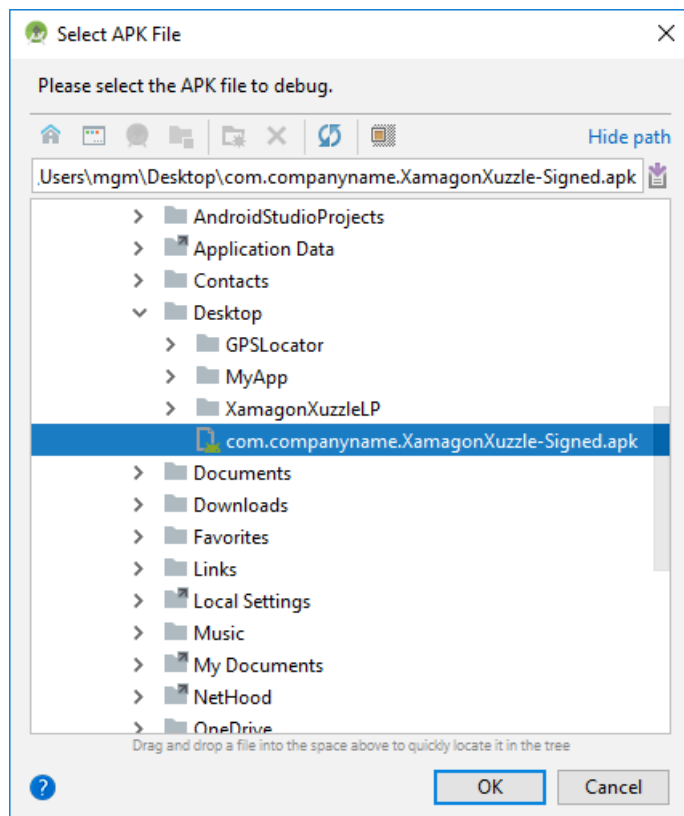
3. 打开项目文件夹，然后导航到 bin/Debug。在此文件夹中，找到 Signed.apk 版本的应用并将其复制到易于访问的位置(例如桌面)。在下面的屏幕截图中，已找到 APK com.companyname.XamagonXuzzle Signed.apk，并已将其复制到桌面：



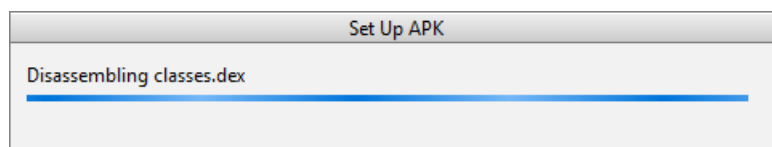
4. 启动 Android Studio，然后选择“分析或调试 APK”：



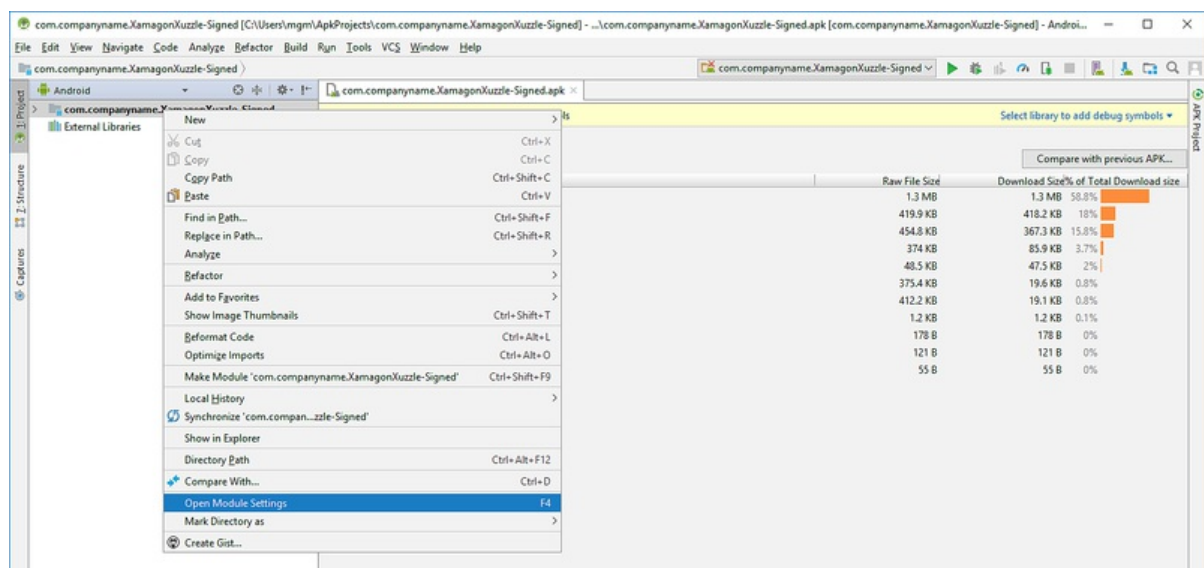
5. 在“选择 APK 文件”对话框中，导航到先前生成和复制的 APK。选择 APK，然后单击“确定”：



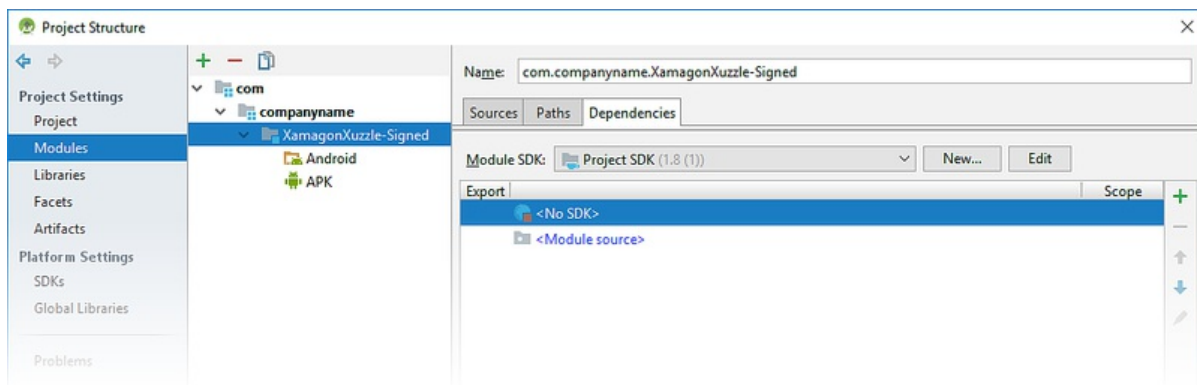
6. Android Studio 将加载 APK 并反汇编 classes.dex:



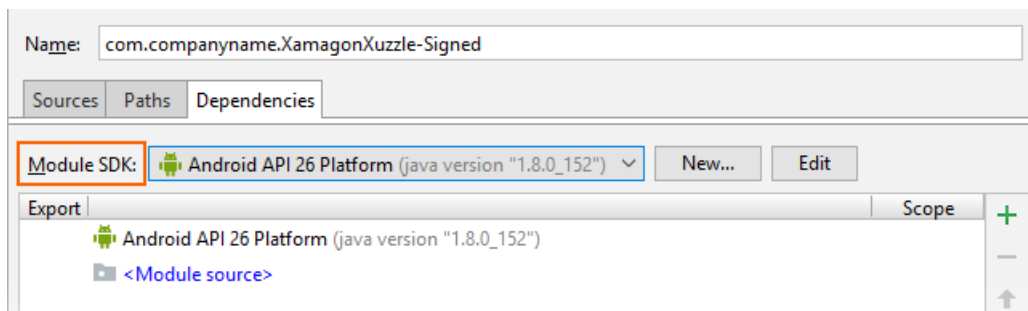
7. 加载 APK 后, Android Studio 显示以下 APK 项目屏幕。右键单击左侧树视图中的应用名称, 然后选择“打开模块设置”:



8. 导航到“项目设置”>“模块”, 选择应用的 -Signed 节点, 然后单击“<无 SDK>”:

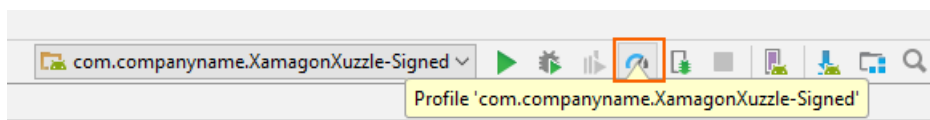


9. 在“模块 SDK”下拉菜单中，选择用于生成应用的 Android SDK 级别(在此示例中，使用 API 级别 26 来生成 XamagonXuzzle)：

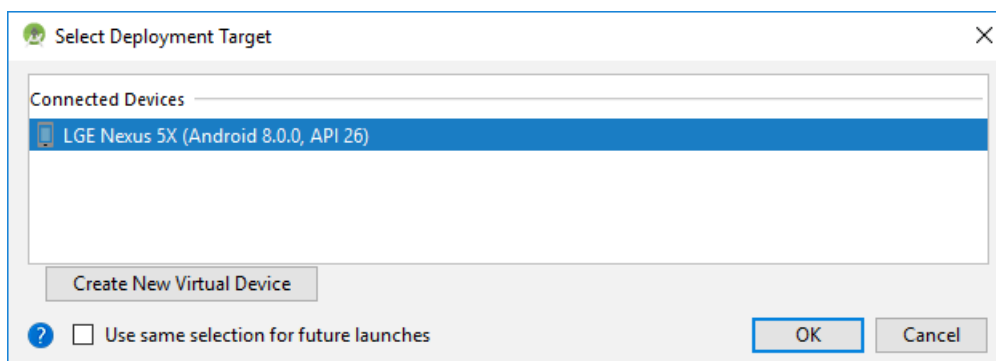


单击“应用”和“确定”保存此设置。

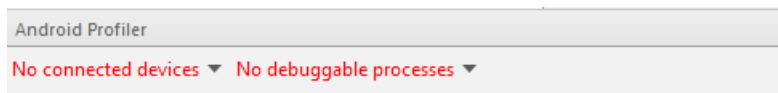
10. 从工具栏图标启动探查器：



11. 选择运行/分析应用的部署目标，然后单击“确定”。部署目标可以是物理设备，也可以是在仿真器中运行的虚拟设备。此示例使用的是 Nexus 5X 设备：

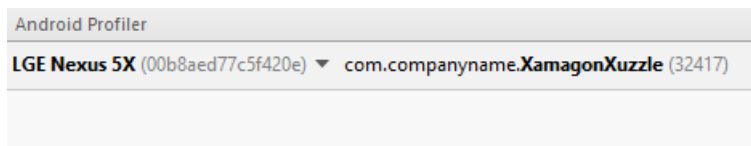


12. 探查器启动后，需要几秒钟时间才能连接到部署设备和应用进程。安装 APK 时，Android Profiler 将报告“无已连接设备”和“无可调试进程”。



No device detected. Please plug in a device,
or launch the emulator. [Learn More](#)

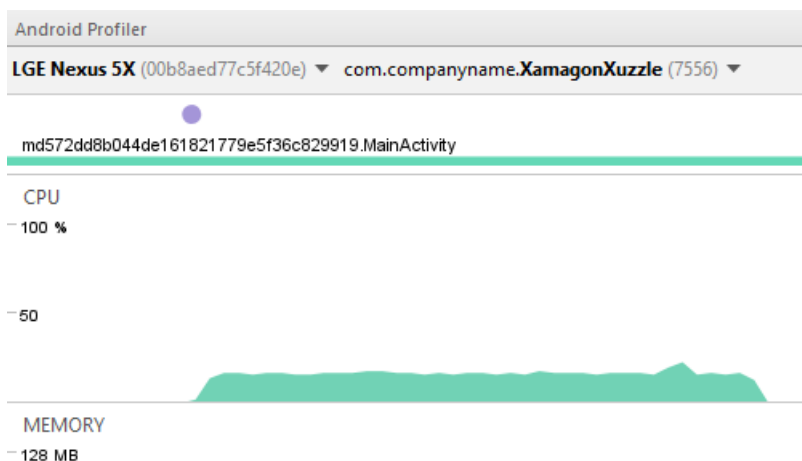
13. 几秒钟后, Android Profiler 完成 APK 安装并启动 APK, 报告正在分析的设备名称和应用进程名称(在此示例中, 分别是 LGE Nexus 5X 和 com.companyname.XamagonXuzzle):



14. 识别设备和可调试进程后, Android Profiler 开始分析应用:



15. 如果点击 XamagonXuzzle 上的“随机化”按钮(这会导致其移动和随机化磁贴), 可看到 CPU 使用率在应用的随机化间隔期间增加:



使用 Android Profiler

[Android Studio 文档](#)中记录了有关使用 Android Profiler 的详细信息。Xamarin Android 开发人员可能对以下主题感兴趣:

- [CPU Profiler](#) – 说明如何实时检查应用的 CPU 使用情况和线程活动。
- [内存 Profiler](#) – 显示应用内存使用情况的实时图, 并包括记录内存分配以进行分析的按钮。
- [网络 Profiler](#) – 显示应用发送和接收的数据的实时网络活动。

做好应用程序发布准备

2018/10/26 • [Edit Online](#)

应用程序经编码和测试后，必须准备一个包进行分发。准备此包的第一个任务是生成供发布的应用程序，其中主要涉及到设置应用程序的一些属性。

使用以下步骤生成供发布的应用：

- **指定应用程序图标** – 每个 Xamarin.Android 应用程序应指定一个应用程序图标。虽然在技术层面并不需要这么做；但是，某些应用商店（例如 Google Play）对此提出了要求。
- **应用程序版本控制** – 此步骤涉及初始化或更新版本信息。这对应用程序将来的更新以及确保用户知道安装的应用程序版本非常重要。
- **压缩 APK** – 通过托管代码上的 Xamarin.Android 链接器和 Java 字节码上的 ProGuard，可大幅压缩最终 APK。
- **保护应用程序** – 通过禁用调试、模糊处理托管代码、添加防调试和防篡改，并使用本机编译来阻止用户或攻击者对应用程序进行调试、篡改或反向工程。
- **设置打包属性** – 打包属性控制 Android 应用程序包 (APK) 的创建。此步骤会优化 APK，保护其资产并根据需要模块化打包。
- **编译** – 此步骤编译代码和资产，以确认按发布模式生成。
- **存档以供发布** – 此步骤生成应用，并将其放置在存档中以供签名和发布。

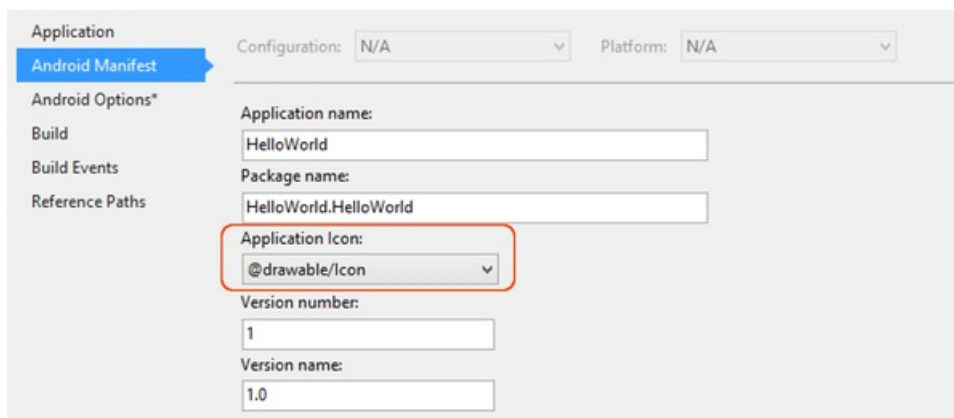
下面详细说明了上述各步骤。

指定应用程序图标

强烈建议每个 Xamarin.Android 应用程序都指定一个应用程序图标。某些应用程序市场将不允许发布没有图标的 Android 应用程序。`Application` 特性的 `Icon` 属性用于指定 Xamarin.Android 项目的应用程序图标。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

在 Visual Studio 2015 及更高版本中，可通过项目“属性”的“Android 清单”部分指定应用程序图标，如以下屏幕截图所示：



在这些示例中，`@drawable/icon` 指位于 `Resources/drawable/icon.png`（请注意，.png 扩展名不包含在资源名称中）中的一个图标文件。另外，此属性也可在文件 `Properties\AssemblyInfo.cs` 中声明，如以下示例代码片段所示：

```
[assembly: Application(Icon = "@drawable/icon")]
```

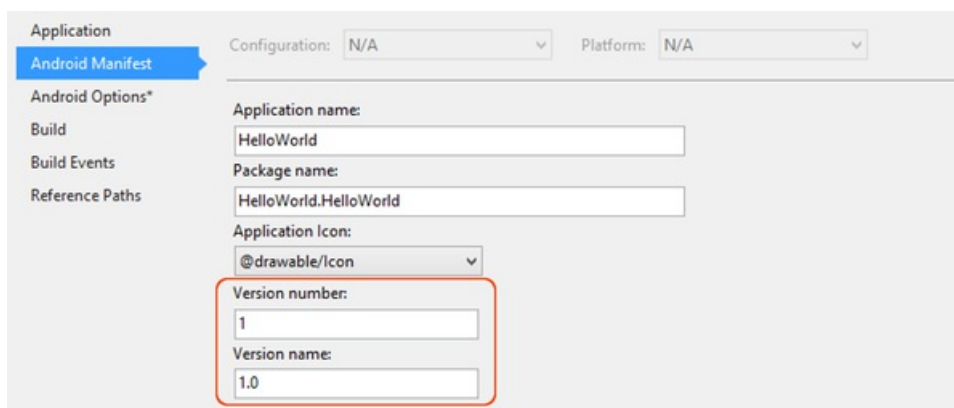
通常情况下，`using Android.App` 在 **AssemblyInfo.cs** (`Application` 属性的命名空间为 `Android.App`) 顶部声明；不过，可能需要添加此 `using` 语句（如果尚不存在）。

应用程序版本控制

对于 Android 应用程序维护和分发而言，版本控制很重要。如果没有版本控制，则很难确定应用程序是否应更新或如何更新。为了辅助版本控制，Android 可识别两种不同类型的信息：

- **版本号** – 表示应用程序版本的整数值，供 Android 和应用程序内部使用。对大多数应用程序而言，此值的初始设置为 1，之后随每个内部版本递增。此值与版本名称属性（见下文）没有关系或关联。应用程序和发布服务不应向用户显示此值。此值在 **AndroidManifest.xml** 文件中存储为 `android:versionCode`。
 - **版本名称** – 仅用于向用户传递应用程序（如安装在特定设备上）的版本相关信息的字符串。版本名称将向用户显示，或在 Google Play 中显示。此字符串不供 Android 内部使用。版本名称可以是任何字符串值，它能帮助用户了解其设备上安装的版本。此值在 **AndroidManifest.xml** 文件中存储为 `android:versionName`。
- [Visual Studio](#)
 - [Visual Studio for Mac](#)

在 Visual Studio 中，可在项目“属性”的“Android 清单”部分设置这些值，如以下屏幕截图所示：



缩小 APK

可通过结合使用 Xamarin.Android 链接器（删除不必要的托管代码）和 Android SDK 中的 ProGuard 工具（删除未使用的 Java 字节码）缩小 Xamarin.Android APK。生成过程首先使用 Xamarin.Android 链接器以托管代码（C#）级别优化应用，然后使用 ProGuard（如已启用）以 Java 字节码级别优化 APK。

配置链接器

发布模式会关闭共享运行时并打开链接，使应用程序只提供运行时需要的 Xamarin.Android 部分。Xamarin.Android 中的链接器使用静态分析来确定 Xamarin.Android 应用程序所使用或引用的程序集、类型和类型成员。然后，链接器将放弃所有未使用（或引用）的程序集、类型和成员。这可显著减小包的大小。例如，[HelloWorld](#) 示例，其 APK 的最终大小减少了 83%：

- 配置：无 – Xamarin.Android 4.2.5 大小 = 17.4 MB。
 - 配置：仅 SDK 程序集 – Xamarin.Android 4.2.5 大小 = 3.0 MB。
- [Visual Studio](#)
 - [Visual Studio for Mac](#)

通过项目“属性”的“Android 选项”部分设置链接器选项：



“链接”下拉菜单提供以下选项，用于控制链接器：

- 无 – 这将关闭链接器；不会执行任何链接。
- 仅 **SDK 程序集** – 这会仅链接 [Xamarin.Android](#) 所需的程序集。不会链接其他程序集。
- **SDK 和用户程序集** – 这会链接应用程序所需的所有程序集，而不是仅链接 [Xamarin.Android](#) 所需的程序集。

链接可能产生一些意外的副作用，因此必须在物理设备上的发布模式下重新测试应用程序。

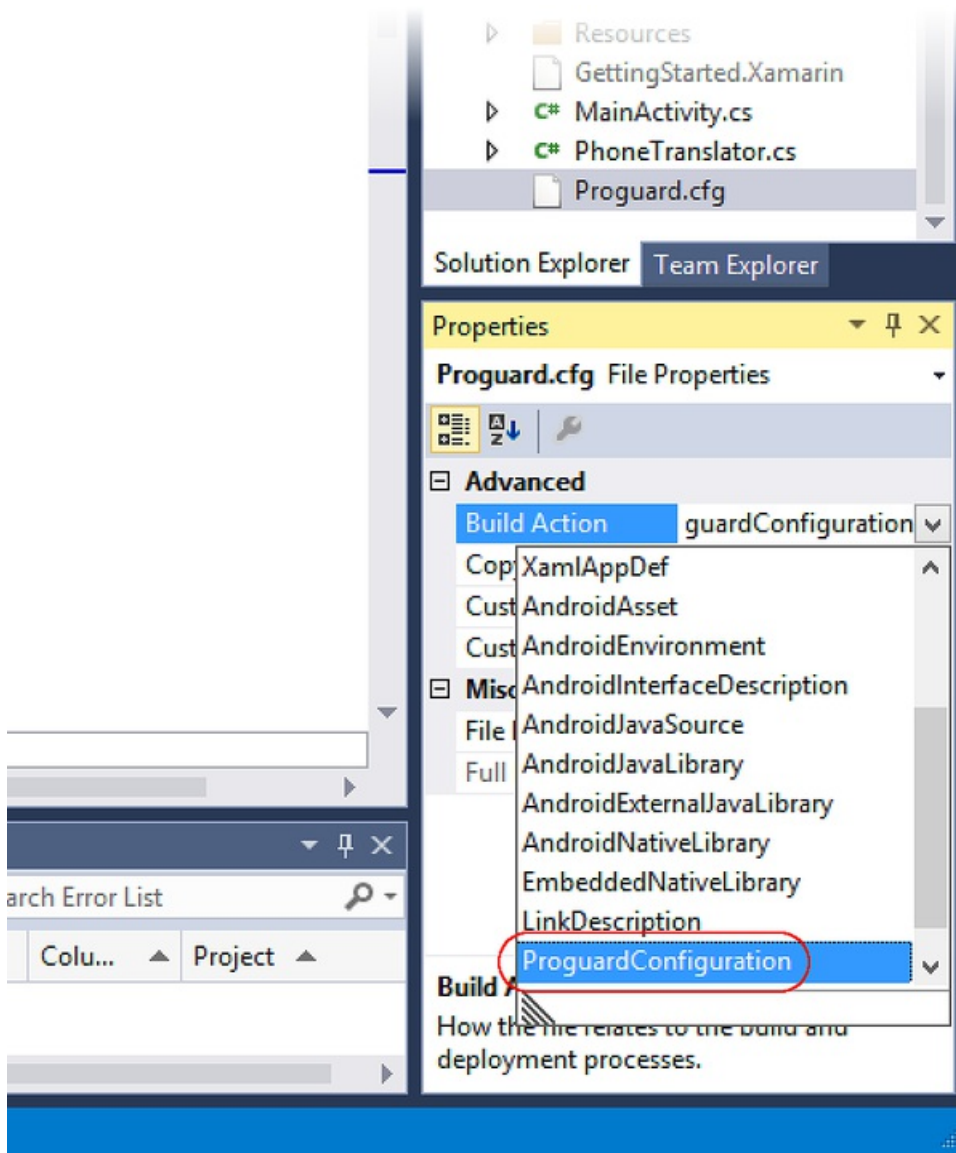
ProGuard

ProGuard 是一种链接和模糊处理 Java 代码的 Android SDK 工具。ProGuard 通常用于创建小型应用程序，工作原理是减少 APK 中包含的大型库的内存占用。ProGuard 将删除未使用的 Java 字节码，使生成的应用变得更小。例如，在小型 [Xamarin.Android](#) 应用中使用 ProGuard 通常可减少约 24% 大小– 在具有多个库依赖关系的大型应用中使用 ProGuard 通常可实现更大幅度的大小缩减。

ProGuard 不是 [Xamarin.Android](#) 链接器的替代工具。[Xamarin.Android](#) 链接器链接托管代码，而 ProGuard 链接 Java 字节码。生成过程首先在应用中使用 [Xamarin.Android](#) 链接器优化托管的 (C#) 代码，然后在 Java 字节码级别使用 ProGuard(若已启用)优化 APK。

选择“启用 ProGuard”时，[Xamarin.Android](#) 将在生成的 APK 中运行 ProGuard 工具。ProGuard 配置文件由 ProGuard 在生成时生成和使用。[Xamarin.Android](#) 还支持自定义 ProguardConfiguration 生成操作。可以将自定义 ProGuard 配置文件添加到项目中，右键单击并选中该文件作为生成操作，如此示例中所示：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



默认情况下，禁用 ProGuard。仅在项目设置为“发布”模式时才能使用“启用 ProGuard”选项。除非选中“启用 ProGuard”，否则会忽略所有 ProGuard 生成操作。Xamarin.Android ProGuard 配置不会模糊处理 APK，且不能启用模糊处理，即使处理自定义配置文件也不例外。如果想要模糊处理，请参阅[使用 Dotfuscator 保护应用程序](#)。

有关 ProGuard 工具用法的详细信息，请参阅[ProGuard](#)。

保护应用程序

禁用调试

在 Android 应用程序开发期间，将使用 Java 调试线路协议 (JDWP) 执行调试。这是一种技术，它允许 **adb** 等工具出于调试目的与 JVM 通信。默认对 Xamarin.Android 应用程序的调试版本启用 JDWP。虽然 JDWP 在开发过程中很重要，但它会对已发布的应用程序造成安全问题。

IMPORTANT

请始终禁用已发布应用程序中的调试状态，因为如果不禁用此状态，则可能(通过 JDWP)获得 Java 进程的完全访问权限并在应用程序的上下文中执行任意代码。

Android 清单包含 `android:debuggable` 属性，该属性控制是否可以调试应用程序。将 `android:debuggable` 属性设置为 `false` 被视为一种很好的做法。执行此操作最简单的方法是在 **AssemblyInfo.cs** 中添加条件编译语句：

```
#if DEBUG
[assembly: Application(Debuggable=true)]
#else
[assembly: Application(Debuggable=false)]
#endif
```

注意，调试版本会自动设置某些权限以简化调试(如 **Internet** 和 **ReadExternalStorage**)。但是，发布版本只使用显式配置的权限。若发现切换到发布版本会导致应用失去可在调试版本中使用的权限，请验证是否已在“所需权限”列表中显式启用了此权限，如[权限](#)中所述。

使用 Dotfuscator 保护应用程序

- [Visual Studio](#)
- [Visual Studio for Mac](#)

即使已禁用调试，攻击者仍可能重新打包应用程序，从而添加或删除配置选项或权限。这可使他们对应用程序进行反向工程、调试或篡改。[Dotfuscator Community Edition \(CE\)](#) 可用于混淆托管代码，并在生成时向 Xamarin.Android 应用插入运行时安全状态检测代码，对应用是否在根设备上运行进行检测和响应。

Dotfuscator CE 随附在 Visual Studio 中，但是仅 Visual Studio 2015 Update 3 (及更高版本)具有用于 Xamarin.Android 的正确版本。若要使用 Dotfuscator，请单击“工具”>“PreEmptive Protection - Dotfuscator”。

若要配置 Dotfuscator CE，请参阅 [Using Dotfuscator Community Edition with Xamarin](#) (结合使用 Dotfuscator Community Edition 和 Xamarin)。完成配置后，Dotfuscator CE 将自动保护创建的每个生成。

将程序集捆绑到本机代码

此选项启用时，程序集会捆绑到本机共享库中。此选项使代码保持安全；它通过在本机二进制文件中嵌入这些托管程序集来保护它们。

此选项需要 Enterprise 许可证，仅当“使用快速部署”禁用时才可用。“将程序集捆绑到本机代码”在默认情况下处于禁用状态。

请注意，“捆绑到本机代码”选项执行并不意味着程序集会编译到本机代码中。无法使用 [AOT 编译](#) 将程序集编译到本机代码中(当前只是试验性功能，不用于生产用途)。

AOT 编译

[打包属性](#)页上的AOT 编译选项支持预先编译程序集。启用此选项后，通过在运行时之前预编译程序集可将实时(JIT)启动开销降到最低。生成的本机代码包括在 APK 以及未编译程序集中。这可缩短应用程序启动时间，但代价是 APK 大小会变得稍大。

“AOT 编译”选项要求使用 Enterprise 或更高版本的许可证。仅在项目配置为发布模式时，才可使用“AOT 编译”，并且该选项默认处于禁用状态。有关 AOT 编译的详细信息，请参阅 [AOT](#)。

LLVM 优化编译器

LLVM 优化编译器会创建更小更快速的编译代码，并将 AOT 编译的程序集转换为本机代码，但生成时间会变缓慢。默认情况下，LLVM 编译器处于禁用状态。要使用 LLVM 编译器，必须首先启用“AOT 编译”选项(在[打包属性](#)页面上)。

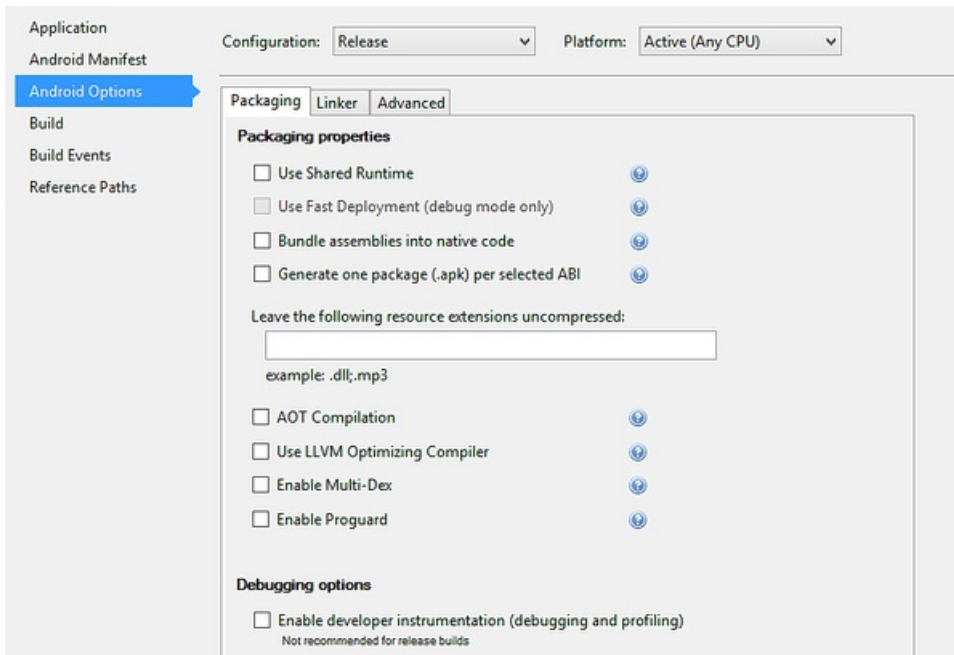
NOTE

“LLVM 优化编译器”选项需要业务许可证。

设置打包属性

- [Visual Studio](#)
- [Visual Studio for Mac](#)

可在项目“属性”的“Android 选项”部分设置打包属性，如以下屏幕截图所示：



其中许多属性(例如“使用共享运行时”和“使用快速部署”)专用于调试模式。但是，在发布模式下配置应用程序时，还需要进行其他设置，这些设置用于确定如何[针对大小和执行速度优化应用](#)、如何[防止篡改应用](#)，以及如何打包应用以支持不同的体系结构和大小限制。

指定支持的体系结构

准备 Xamarin.Android 应用进行发布时，必须指定支持的 CPU 体系结构。单个 APK 可包含计算机代码，以支持多个不同的体系结构。请参阅 [CPU 体系结构](#)，深入了解如何支持多个 CPU 体系结构。

每个选定 ABI 生成一个包 (.APK)

启用此选项后，会为每个支持的 ABI(在“高级”选项卡上进行选择，如 [CPU 体系结构](#)中所述)分别创建一个 APK，而不是为所有支持的 ABI 创建单个大型 APK。仅在项目配置为用于发布模式时，才可使用此选项，并且其默认处于禁用状态。

Multi-Dex

如果启用“启用 Multi-Dex”选项，Android SDK 工具将用于绕过 **.dex** 文件格式的 65K 方法限制。65K 方法限制基于应用_引用_的 Java 方法数(包括应用依赖的任何库中的方法数)– 不基于_源代码中写入_的方法数。如果应用程序只定义了几个方法，却使用了多个方法或大型库，则可能超出 65K 限制。

应用可能未使用每个引用库中的每个方法；因此，ProGuard(见上文)等工具可能会将未使用的方法从代码中删除。最佳做法是仅在必要时启用“启用 Multi-Dex”，也就是说，即使使用 ProGuard，应用引用的 Java 方法仍然超过 65K。

若要深入了解 Multi-Dex，请参阅[配置超出 64K 方法的应用](#)。

Compile

- [Visual Studio](#)
- [Visual Studio for Mac](#)

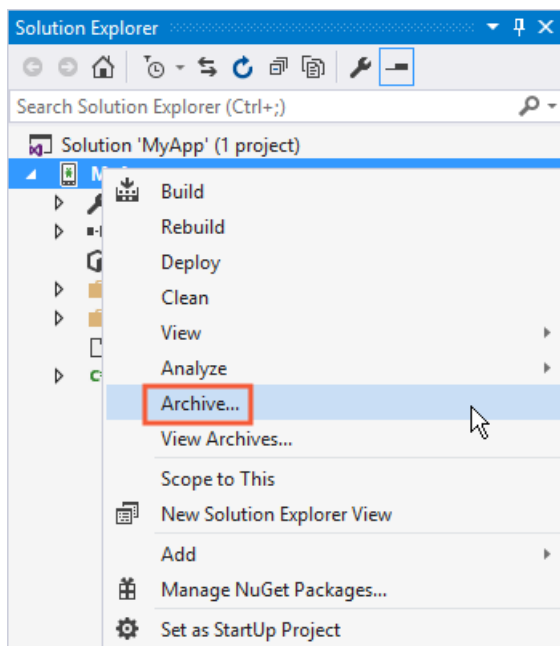
完成上述所有步骤后，应用即可用于编译。选择“生成”>“重新生成解决方案”以验证其是否在发布模式下成功生成。请注意，此步骤尚不会产生 APK。

[对应用包进行签名](#)中更详细地讨论了打包和签名。

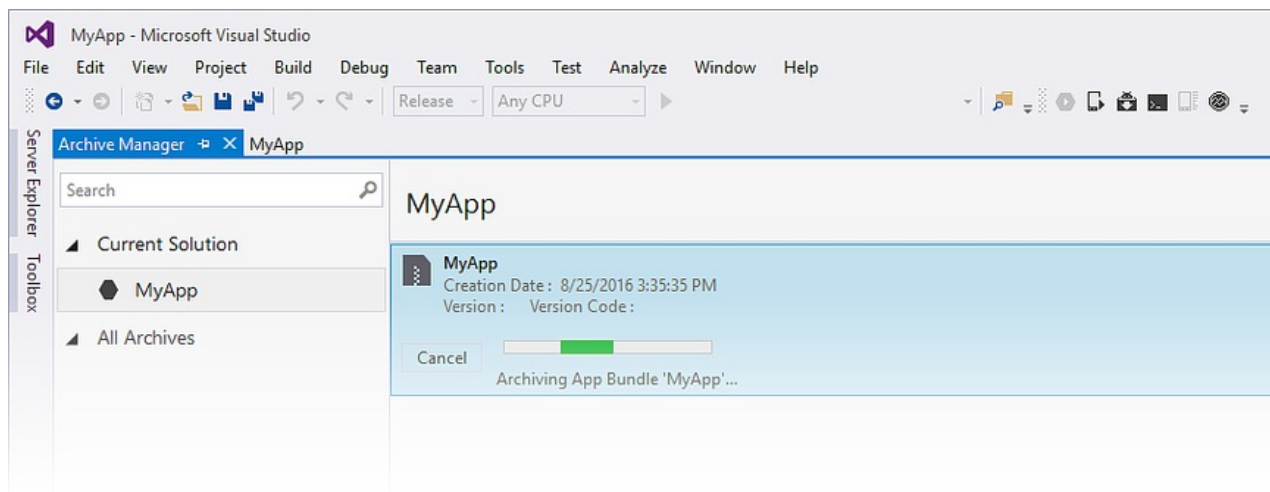
存档以供发布

- [Visual Studio](#)
- [Visual Studio for Mac](#)

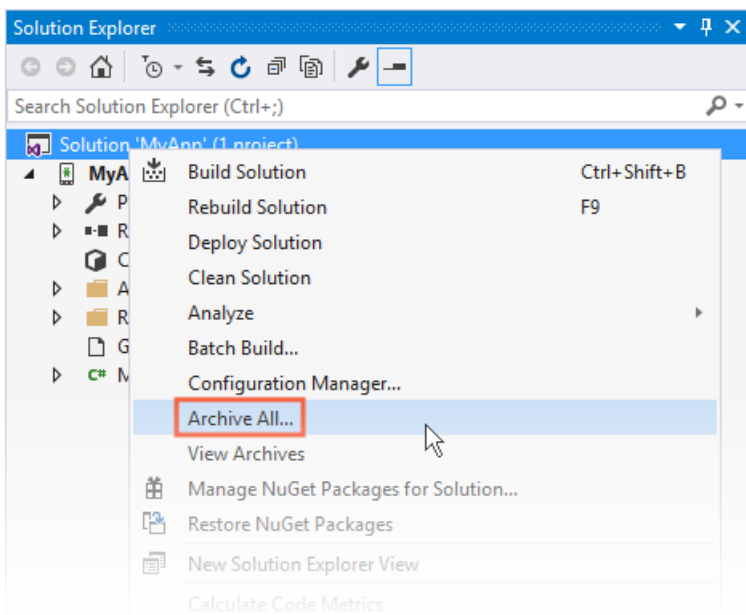
若要开始发布过程，请在**解决方案资源管理器**中右键单击项目，然后选择“存档...”上下文菜单项：



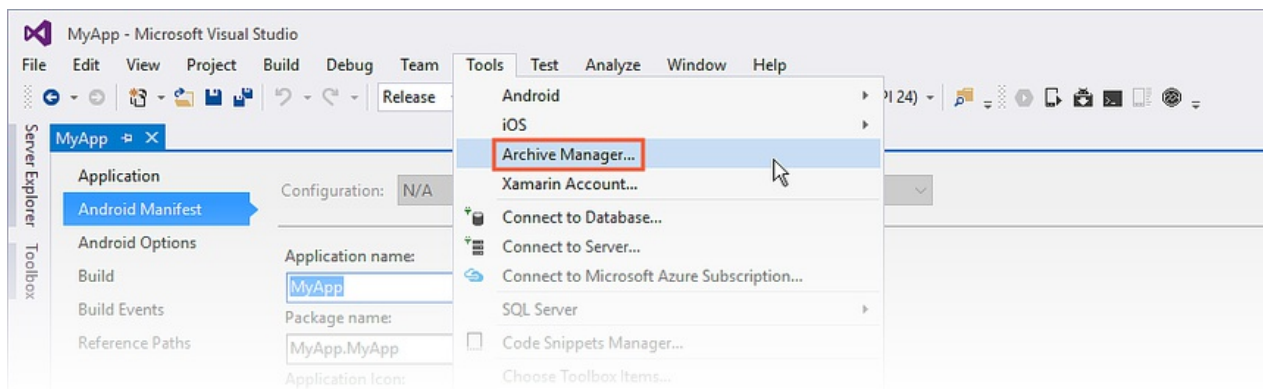
选择“存档...”选项将启动**存档管理器**并开始应用程序包的存档过程，如以下屏幕截图所示：



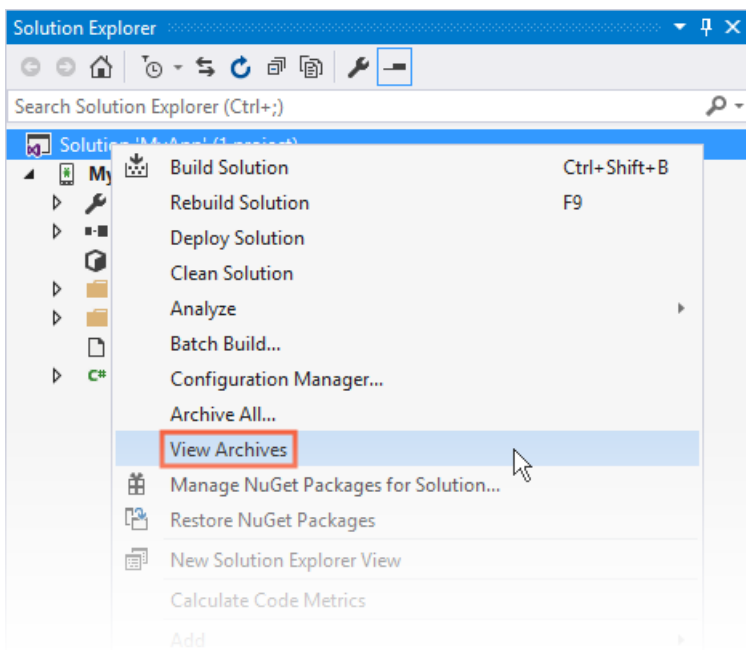
另一种创建存档的方法是：在**解决方案资源管理器**中，右键单击“解决方案”，然后选择“全部存档...”，这会生成解决方案并存档可生成存档的所有 Xamarin 项目：



“存档”和“全部存档”都会自动启动存档管理器。若要直接启动存档管理器，请单击“工具”>“存档管理器...”菜单项：

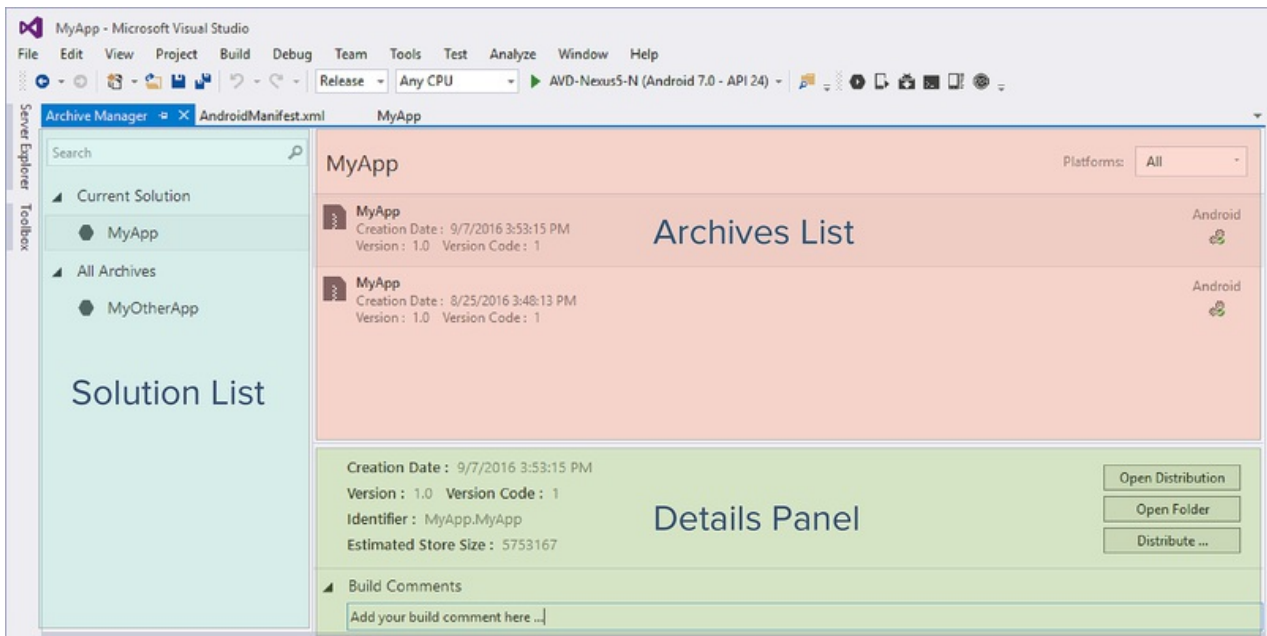


右键单击“解决方案”节点并选择“查看存档”可随时查看该解决方案的存档：



存档管理器

存档管理器由“解决方案列表”窗格、“存档列表”和“详细信息面板”组成：



“解决方案列表”将显示所有解决方案，其中至少有一个项目已存档。“解决方案列表”包括以下各部分：

- **当前解决方案** – 显示当前的解决方案。请注意，如果当前解决方案不含现有存档，此区域可能为空。
- **全部存档** – 显示包含存档的所有解决方案。
- **搜索文本框** (顶部) – 根据文本框中输入的搜索字符串筛选“全部存档”列表中列出的解决方案。

“存档列表”显示有关所选解决方案的所有存档的列表。“存档列表”包括以下各部分：

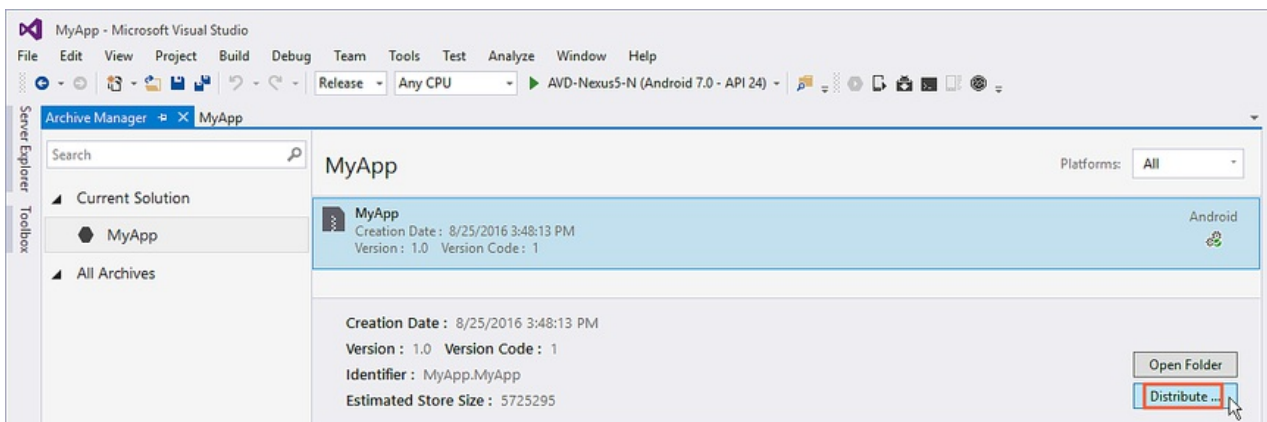
- **所选解决方案名称** – 显示“解决方案列表”中所选解决方案的名称。“存档列表”中显示的所有信息均与此选定的解决方案有关。
- **平台筛选器** – 此字段可按平台类型 (如 iOS 或 Android) 筛选存档。
- **存档项目** – 选定解决方案的存档列表。此列表中的每个项均包括项目名称、创建日期和平台。还可以显示其他信息，例如存档或发布项目时的进度。

“详细信息面板”显示有关每个存档的其他信息。用户还可以从此面板启动分发 workflow 或打开创建分发的文件夹。

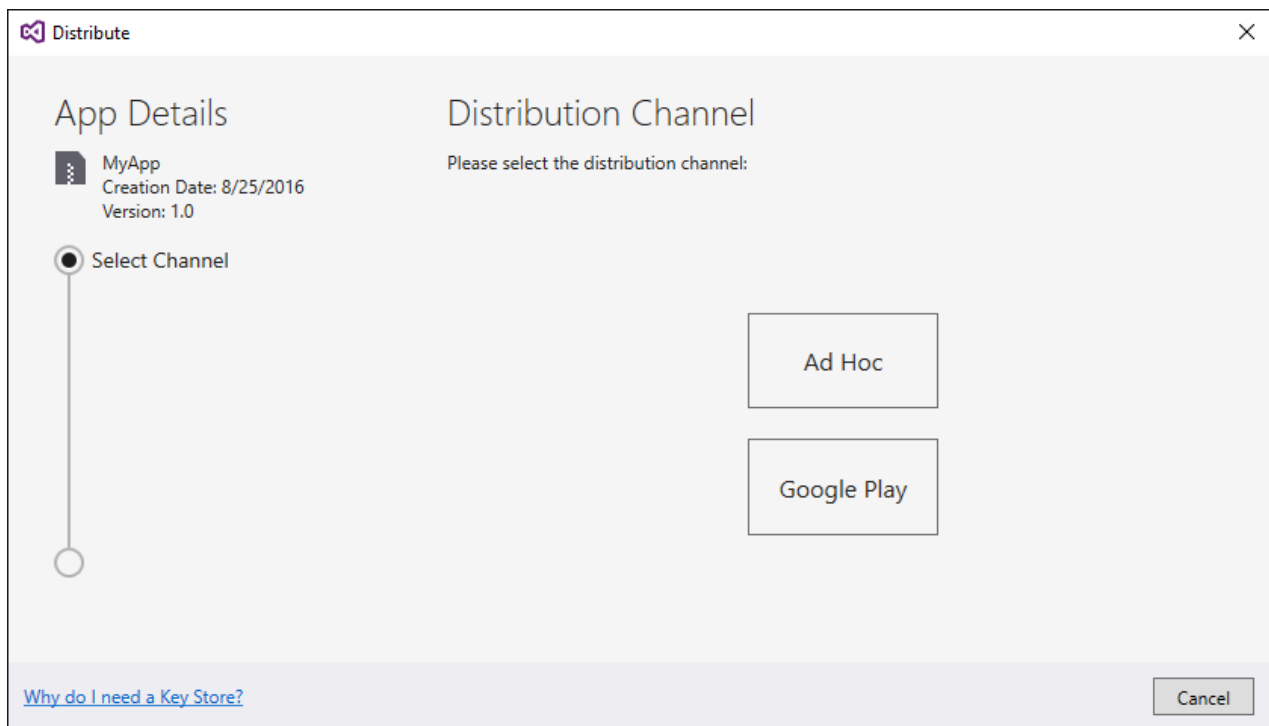
“生成注释”部分可将生成注释包括在存档中。

分布

准备好发布存档版应用程序后，请在“存档管理器”中选择该存档，然后单击“分发...”按钮：



“分发通道”对话框包括以下方面的信息：应用、分发 workflow 进度指示以及分发渠道选项。首次运行时，提供两个选项：



可选择以下分发通道之一：

- **Ad-Hoc** – 将已签名的 APK 保存到磁盘，以将其旁加载到 Android 设备。继续查看[对应用包进行签名](#)，了解如何创建 Android 签名标识、为 Android 应用程序创建新的签名证书以及将“临时”版本的应用发布到磁盘。这是为测试创建 APK 的好方法。
- **Google Play** – 将已签名的 APK 发布到 Google Play。继续查看[发布到 Google Play](#)，了解如何对 APK 进行签名并将其发布到 Google Play 商店。

相关链接

- [多核设备和 Xamarin.Android](#)
- [CPU 体系结构](#)
- [AOT](#)
- [收缩代码和资源](#)
- [配置方法数超过 64K 的应用](#)

ProGuard

2018/10/26 • [Edit Online](#)

Xamarin.Android ProGuard 是一个 Java 类文件压缩器、优化器和预验证器。它会检测和删除未使用的代码，分析和优化字节码。本指南阐释了 ProGuard 的工作原理、如何在项目中启用它，以及如何配置。同时提供了几个 ProGuard 配置示例。_

概述

ProGuard 从打包的应用程序中检测并删除未使用的类、字段、方法和属性。它甚至可对引用的库执行相同操作（这有助于避免 64k 引用限制）。Android SDK 中的 ProGuard 工具还将优化字节码和删除未使用的代码说明。

ProGuard 读取输入文件，然后将其压缩、优化和预验证，并将结果写入一个或多个输出文件。

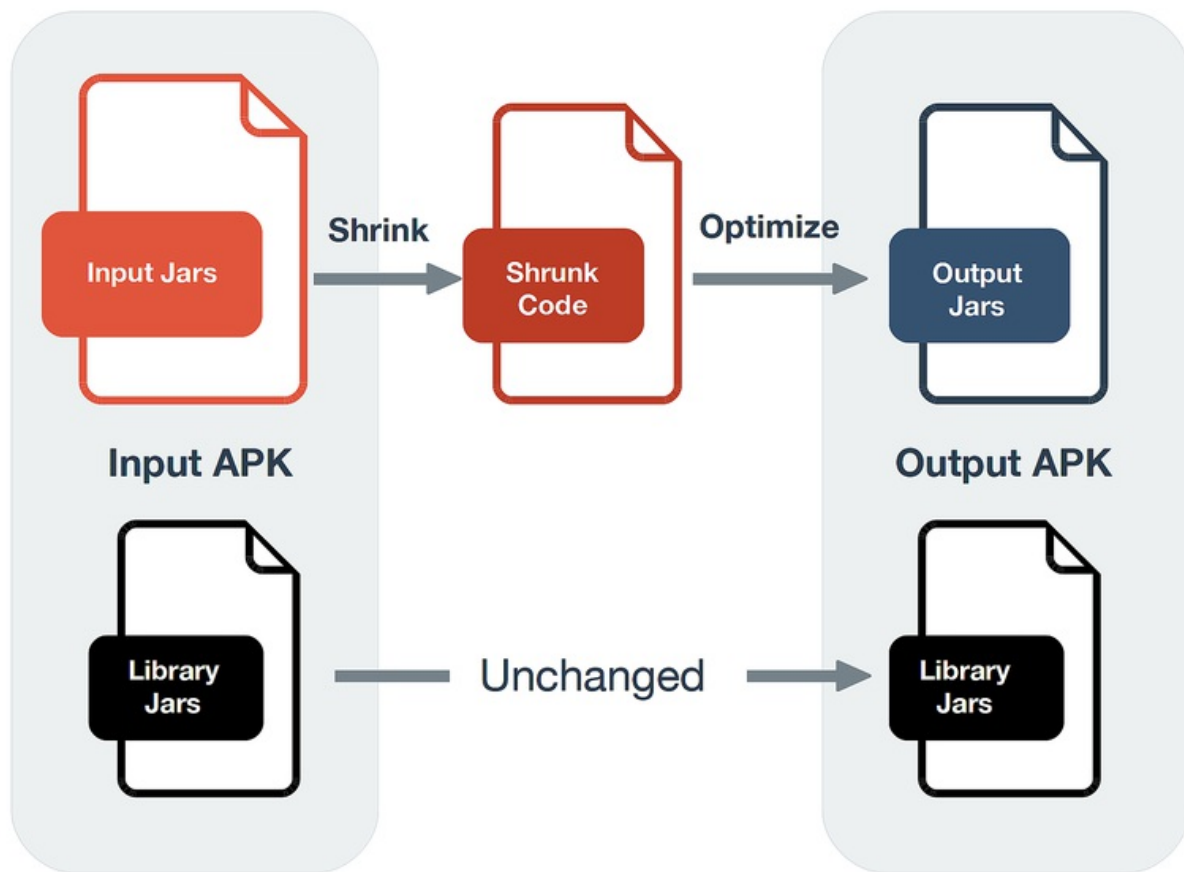
ProGuard 使用以下步骤处理输入 APK：

1. **压缩步骤** – ProGuard 以递归方式确定使用的类和类成员。将丢弃所有其他类和类成员。
2. **优化步骤** – ProGuard 进一步优化代码。在其他优化中，可将非入口点的类和方法设置为私有、静态或最终，可删除未使用的参数，并且可内联一些方法。
3. **模糊处理步骤** – 在本机 Android 开发中，ProGuard 重命名非入口点的类和类成员。保留入口点，确保它们仍可通过其原始名称访问。但是，Xamarin.Android 并不支持此步骤，因为该应用是使用中间语言 (IL) 编译的。
4. **预验证步骤** – 在运行时前检查 Java 字节码，并对 Java VM 权益的类文件进行批注。只有此步骤无需知道入口点。

上述每个步骤均可选。Xamarin.Android ProGuard 要使用其中的部分步骤，详见下一节。

Xamarin.Android 中的 ProGuard

Xamarin.Android ProGuard 配置不会模糊处理 APK。事实上，无法通过 ProGuard 进行模糊处理（即使使用自定义配置文件）。因此，Xamarin.Android 的 ProGuard 只执行压缩和优化步骤：



使用 ProGuard 前, 必须知道其在 `Xamarin.Android` 生成过程中的工作原理。此过程使用两个独立的步骤:

1. Xamarin Android 链接器
2. ProGuard

接下来将说明上述各步骤。

链接器步骤

Xamarin.Android 链接器使用应用程序的静态分析来确定以下内容:

- 实际使用的程序集。
- 实际使用的类型。
- 实际使用的成员。

将始终在 ProGuard 步骤前运行链接器。因此, 链接器可剥离因此, 链接器可剥离想要 ProGuard 在其上运行的程序集/类型/成员。(若要详细了解 Xamarin.Android 中的链接, 请参阅[在 Android 上链接](#)。)

ProGuard 步骤

链接器步骤成功完成后, 运行 ProGuard 删除未使用的 Java 字节码。此步骤用于优化 APK。

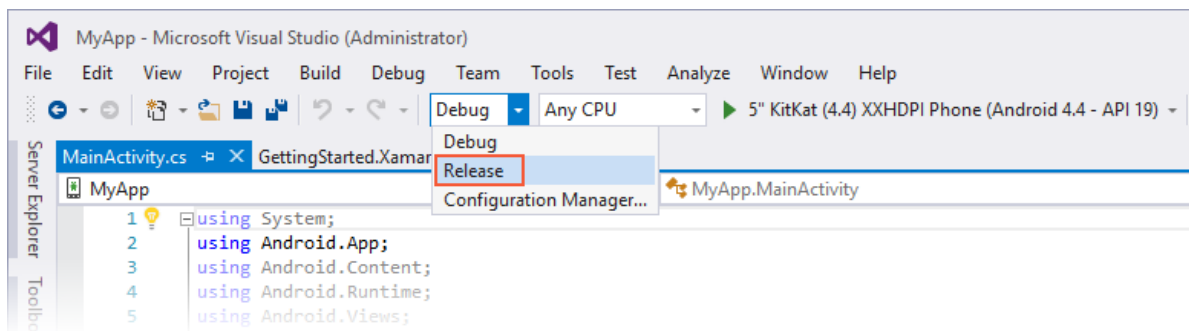
使用 ProGuard

必须先启用 ProGuard, 才可在应用项目中使用它。接下来, 可让 Xamarin.Android 生成过程使用默认的 ProGuard 配置文件, 也可自行创建自定义配置文件供 ProGuard 使用。

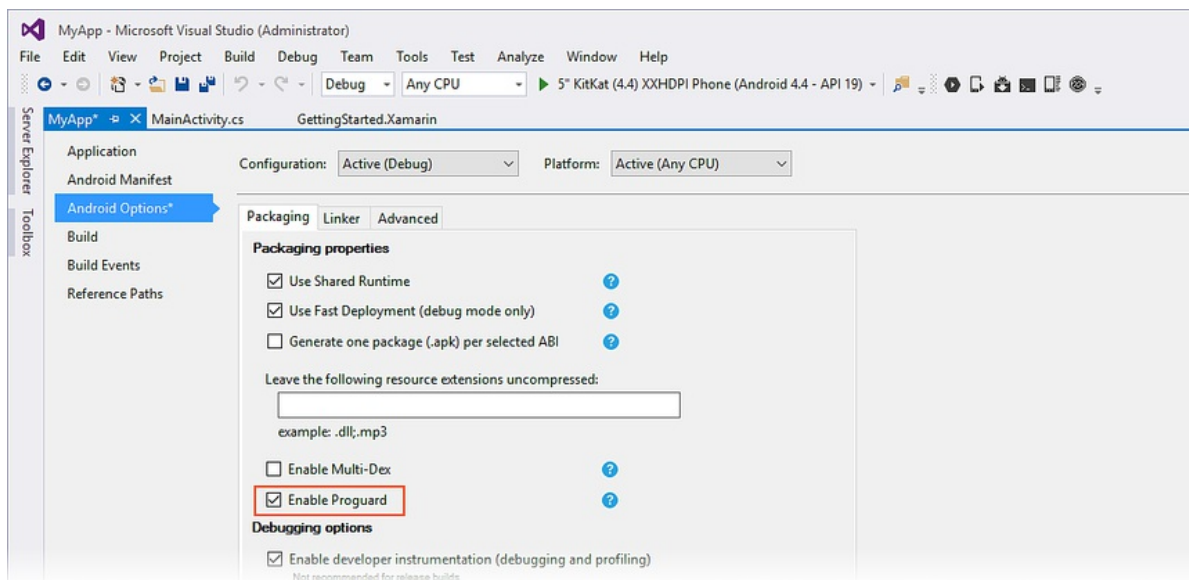
启用 ProGuard

在应用项目中使用以下步骤启用 ProGuard:

1. 确保项目设置为“发布”配置(这很重要, 因为必须先运行链接器才能运行 ProGuard):



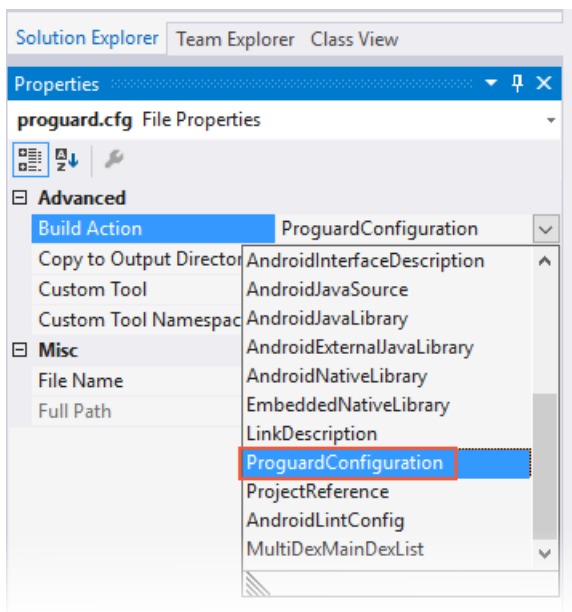
2. 在“属性”>“Android 选项”的“包装”选项卡下，选中“启用 ProGuard”选项来启用 ProGuard:



对于大多数 Xamarin.Android 应用, Xamarin.Android 提供的默认 ProGuard 配置文件足以删除所有(仅)未使用的代码。若要查看默认 ProGuard 配置, 请打开 **obj\Release\proguard\proguard_xamarin.cfg** 处的文件。下一节将介绍如何创建自定义 ProGuard 配置文件。

自定义 ProGuard

或者, 可添加自定义 ProGuard 配置文件, 实现对 ProGuard 工具的更多掌控。例如, 你可能想就要保留的类显式通知 ProGuard。为此, 请新建 **.cfg** 文件, 并在解决方案资源管理器的“属性”窗格中应用 **ProGuardConfiguration** 生成操作:



请记住, 该配置文件不会替换 Xamarin.Android proguard_xamarin.cfg 文件, 因为 ProGuard 将使用这两者。

以下示例说明了典型的 ProGuard 配置文件：

```
# This is Xamarin-specific (and enhanced) configuration.

-dontobfuscate

-keep class mono.MonoRuntimeProvider { *; <init>(...); }
-keep class mono.MonoPackageManager { *; <init>(...); }
-keep class mono.MonoPackageManager_Resources { *; <init>(...); }
-keep class mono.android.** { *; <init>(...); }
-keep class mono.java.** { *; <init>(...); }
-keep class mono.javax.** { *; <init>(...); }
-keep class opentk.platform.android.AndroidGameView { *; <init>(...); }
-keep class opentk.GameViewBase { *; <init>(...); }
-keep class opentk_1_0.platform.android.AndroidGameView { *; <init>(...); }
-keep class opentk_1_0.GameViewBase { *; <init>(...); }

-keep class android.runtime.** { <init>(**); }
-keep class assembly_mono_android.android.runtime.** { <init>(**); }
# hash for android.runtime and assembly_mono_android.android.runtime.
-keep class md52ce486a14f4bcd95899665e9d932190b.** { *; <init>(...); }
-keepclassmembers class md52ce486a14f4bcd95899665e9d932190b.** { *; <init>(...); }

# Android's template misses fluent setters...
-keepclassmembers class * extends android.view.View {
    *** set*(**);
}

# also misses those inflated custom layout stuff from xml...
-keepclassmembers class * extends android.view.View {
    <init>(android.content.Context,android.util.AttributeSet);
    <init>(android.content.Context,android.util.AttributeSet,int);
}
```

有些情况下，ProGuard 可能无法正确分析应用程序，它可能会删除应用程序实际需要的代码。若发生此情况，可将 `-keep` 行添加到自定义 ProGuard 配置文件：

```
-keep public class MyClass
```

本例中，`MyClass` 设置为想让 ProGuard 跳过的类的实际名称。

还可使用 `[Register]` 注释来注册自己的名称，并使用这些名称来自定义 ProGuard 规则。可为 Adapter、View、BroadcastReceiver、Service、ContentProvider、Activity 和 Fragment 注册名称。有关使用 `[Register]` 自定义属性的详细信息，请参阅[使用 JNI](#)。

ProGuard 选项

ProGuard 提供了许多选项，可配置实现更精细的操作控制。[ProGuard 手册](#)提供了 ProGuard 用法的完整参考文档。

Xamarin.Android 支持以下 ProGuard 选项：

- [输入/输出选项](#)
- [保留选项](#)
- [压缩选项](#)
- [常规选项](#)
- [类路径](#)
- [文件名](#)

- [文件筛选器](#)
- [筛选器](#)
- [Keep](#) [选项概述](#)
- [保留选项修饰符](#)
- [类规范](#)

Xamarin.Android 忽略以下选项：

- [优化选项](#)
- [模糊处理选项](#)
- [预验证选项](#)

ProGuard 和 Android Nougat

如果尝试在 Android 7.0 或更高版本上使用 ProGuard，必须下载较新版本的 ProGuard，因为 Android SDK 未提供与 JDK 1.8 兼容的新版本。

可使用此 [NuGet 包](#) 安装 `proguard.jar` 的较新版本。有关更新默认 Android SDK `proguard.jar` 的详细信息，请参阅此[堆栈溢出](#)讨论。

可在 [SourceForge 页面](#)找到所有版本的 ProGuard。

ProGuard 配置示例

下面列出了两个 ProGuard 配置文件示例。请注意，在这些情况下，Xamarin.Android 生成过程将提供输入、输出和库 jar。因此，可专注于其他选项，如 `-keep`。

一个简单的 Android 活动

下例演示了一个简单 Android 活动的配置：

```
-injars bin/classes
-outjars bin/classes-processed.jar
-libraryjars /usr/local/java/android-sdk/platforms/android-9/android.jar

-dontpreverify
-repackageclasses ''
-allowaccessmodification
-optimizations !code/simplification/arithmetic

-keep public class mypackage.MyActivity
```

一个完整的 Android 应用程序

下例演示了一个完整 Android 应用的配置：

```

-injars bin/classes
-injars libs
-outjars bin/classes-processed.jar
-libraryjars /usr/local/java/android-sdk/platforms/android-9/android.jar

-dontpreverify
-repackageclasses ''
-allowaccessmodification
-optimizations !code/simplification/arithmetic
-keepattributes *Annotation*

-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider

-keep public class * extends android.view.View {
public <init>(android.content.Context);
public <init>(android.content.Context, android.util.AttributeSet);
public <init>(android.content.Context, android.util.AttributeSet, int);
public void set*(...);
}

-keepclasseswithmembers class * {
public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembers class * {
public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers class * implements android.os.Parcelable {
static android.os.Parcelable$Creator CREATOR;
}

-keepclassmembers class **.R$* {
public static <fields>;
}

```

ProGuard 和 Xamarin.Android 生成过程

以下部分介绍了 ProGuard 在 Xamarin.Android 发布生成期间的运行方式。

ProGuard 正在运行什么命令？

ProGuard 只是随附 Android SDK 提供的 `.jar`。因此，它会在命令中被调用：

```
java -jar proguard.jar options ...
```

ProGuard 任务

ProGuard 任务位于 **Xamarin.Android.Build.Tasks.dll** 程序集中。它是 `_CompileToDalvikWithDx` 目标的一部分，后者则是 `_CompileDex` 目标的一部分。

以下列表提供了使用“文件”>“新建项目”创建新项目后生成的默认参数示例：

```
ProGuardJarPath = C:\Android\android-sdk\tools\proguard\lib\proguard.jar
AndroidSdkDirectory = C:\Android\android-sdk\
JavaToolPath = C:\Program Files (x86)\Java\jdk1.8.0_92\bin
ProGuardToolPath = C:\Android\android-sdk\tools\proguard\
JavaPlatformJarPath = C:\Android\android-sdk\platforms\android-25\android.jar
ClassesOutputDirectory = obj\Release\android\bin\classes
AcwMapFile = obj\Release\acw-map.txt
ProGuardCommonXamarinConfiguration = obj\Release\proguard\proguard_xamarin.cfg
ProGuardGeneratedReferenceConfiguration = obj\Release\proguard\proguard_project_references.cfg
ProGuardGeneratedApplicationConfiguration = obj\Release\proguard\proguard_project_primary.cfg
ProGuardConfigurationFiles

    {sdk.dir}\tools\proguard\proguard-android.txt;
    {intermediate.common.xamarin};
    {intermediate.references};
    {intermediate.application};
    ;

JavaLibrariesToEmbed = C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\MonoAndroid\v7.0\mono.android.jar
ProGuardJarInput = obj\Release\proguard\__proguard_input__.jar
ProGuardJarOutput = obj\Release\proguard\__proguard_output__.jar
DumpOutput = obj\Release\proguard\dump.txt
PrintSeedsOutput = obj\Release\proguard\seeds.txt
PrintUsageOutput = obj\Release\proguard\usage.txt
PrintMappingOutput = obj\Release\proguard\mapping.txt
```

下一示例展示了从 IDE 运行的典型 ProGuard 命令：

```
C:\Program Files (x86)\Java\jdk1.8.0_92\bin\java.exe -jar C:\Android\android-
sdk\tools\proguard\lib\proguard.jar -include obj\Release\proguard\proguard_xamarin.cfg -include
obj\Release\proguard\proguard_project_references.cfg -include
obj\Release\proguard\proguard_project_primary.cfg "-injars
'obj\Release\proguard\__proguard_input__.jar'; 'C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\MonoAndroid\v7.0\mono.android.jar'" "-libraryjars 'C:\Android\android-
sdk\platforms\android-25\android.jar'" -outjars "obj\Release\proguard\__proguard_output__.jar" -optimizations
!code/allocation/variable
```

疑难解答

文件问题

当 ProGuard 读取其配置文件时，可能会显示以下错误消息：

```
Unknown option '-keep' in line 1 of file 'proguard.cfg'
```

此问题通常发生在 Windows 上，因为 `.cfg` 文件编码错误。ProGuard 不能处理字节顺序标记 (BOM)，它可能出现在文本文件中。如果存在 BOM，ProGuard 将退出并显示上述错误。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

为避免此问题，请从允许不使用 BOM 保存文件的文本编辑器中编辑自定义配置文件。若要解决此问题，请确保文本编辑器的编码设置为 `UTF-8`。例如，保存文件时，文本编辑器 [Notepad++](#) 可通过选择“编码”>“无 BOM 时采用 UTF-8 编码”在没有 BOM 的情况下保存文件。

其他问题

ProGuard [疑难解答](#) 页面讨论了使用 ProGuard 时可能遇到的常见问题（及解决方案）。

总结

本指南阐释了 ProGuard 在 Xamarin.Android 中的工作原理、如何在应用项目中启用它，以及如何配置。提供了示例 ProGuard 配置，并描述了常见问题的解决方案。有关 ProGuard 工具和 Android 的详细信息，请参阅[压缩代码和资源](#)。

相关链接

- [做好应用程序发布准备](#)

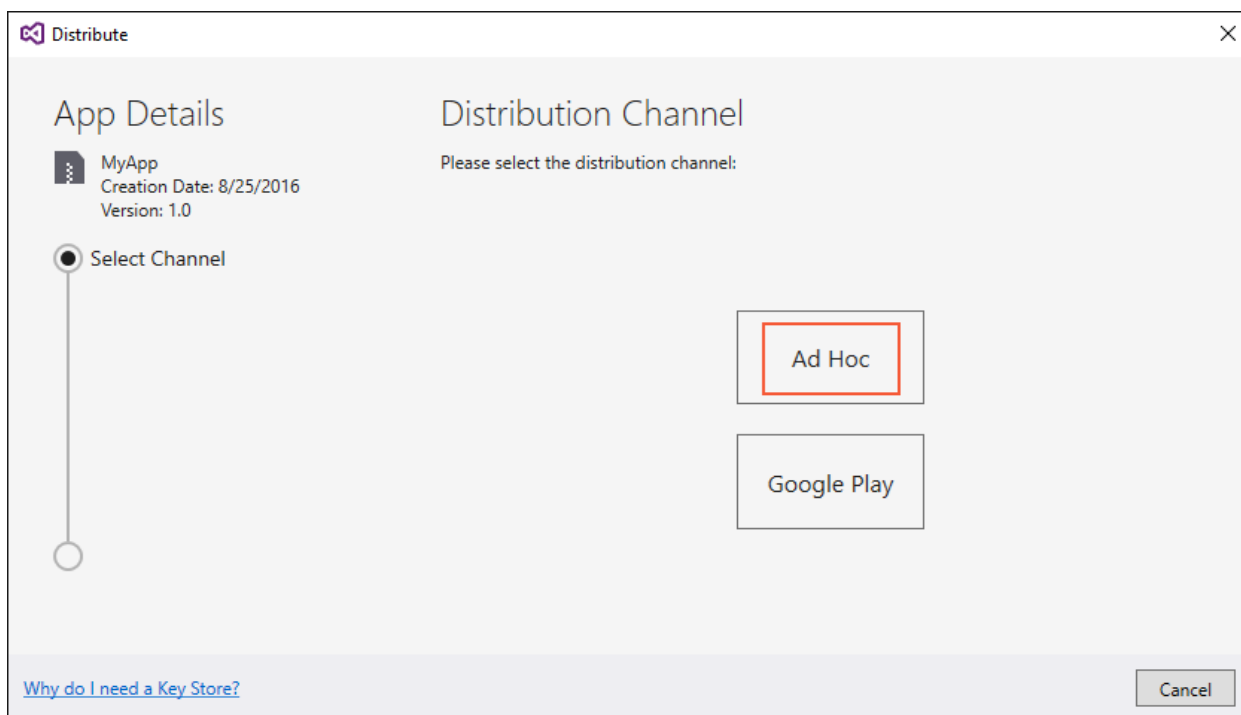
对 Android 应用程序包进行签名

2018/10/26 • [Edit Online](#)

在[做好应用程序发布准备](#)中，使用了“存档管理器”以生成应用并将它放置在存档中以进行签名和发布。此部分说明如何创建 Android 签名标识、为 Android 应用程序创建新签名证书以及将存档应用即席发布到磁盘。生成的 APK 可以旁加载到 Android 设备中，而无需经过应用商店。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

在[存档以便进行发布](#)中，“分发渠道”对话框提供了两种分发选择。选择“即席”：

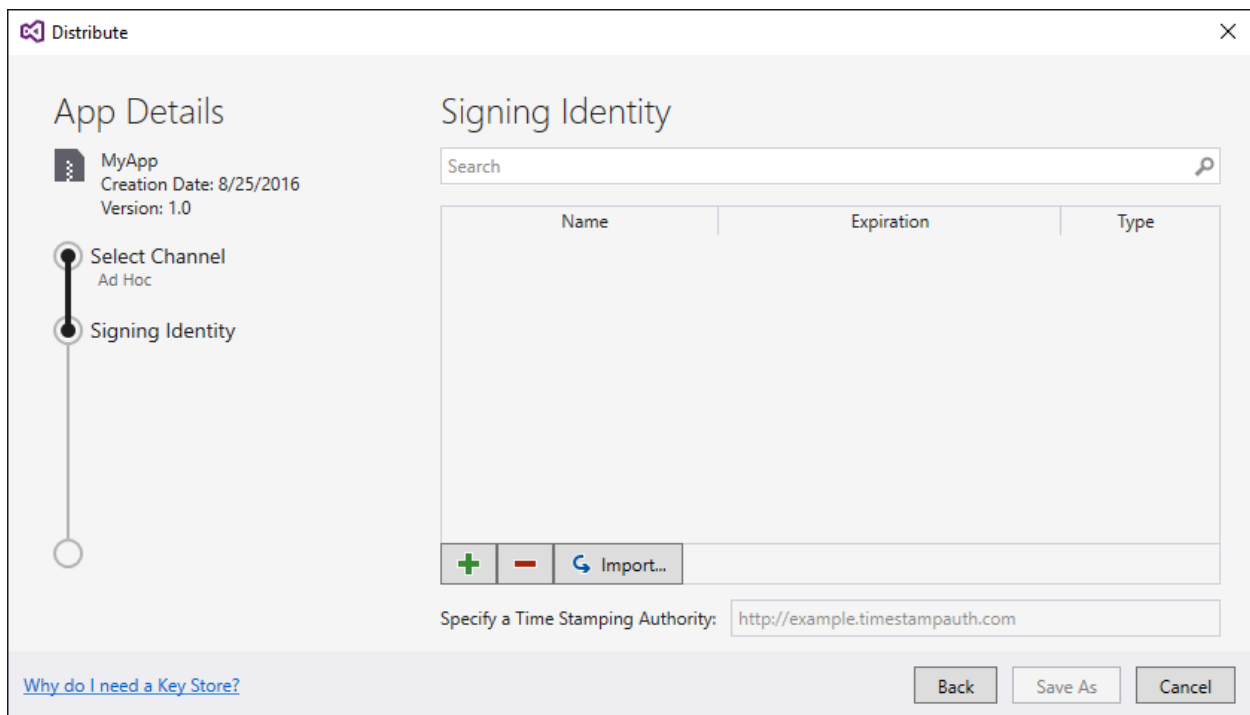


创建新证书

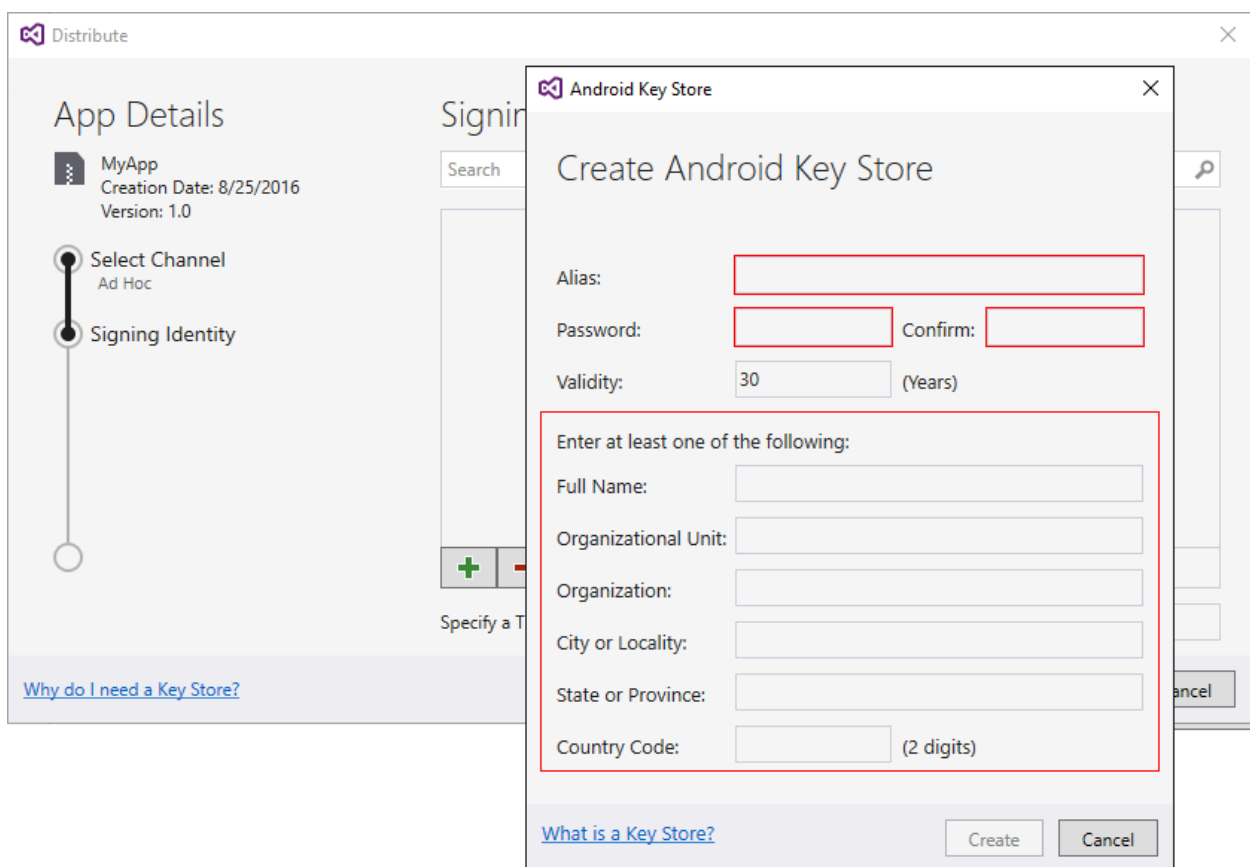
- [Visual Studio](#)
- [Visual Studio for Mac](#)

选择“即席”之后，Visual Studio 会打开对话框的“签名标识”页，如下一个屏幕截图所示。若要发布 .APK，必须首先使用签名密钥(也称为证书)对它进行签名。

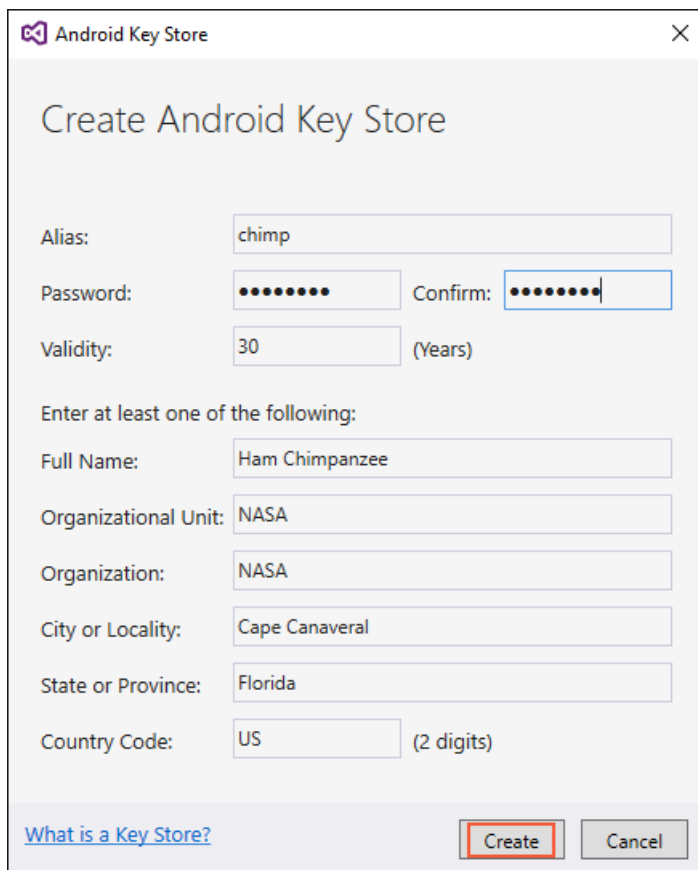
可以通过单击“导入”按钮，然后前进到[对 APK 进行签名](#)，来使用现有证书。否则，单击“+”按钮创建新证书：



会显示“创建 Android 密钥存储”对话框;使用此对话框可创建可以用于对 Android 应用程序进行签名的新签名证书。输入所需信息(具有红色边框), 如此对话框中所示:



下面的示例说明必须提供的信息的种类。单击“创建”以创建新证书:



生成的密钥存储位于以下位置：

C:\Users\USERNAME\AppData\Local\Xamarin\Mono for Android\Keystore\ALIAS\ALIAS.keystore

例如，使用“chimp”作为别名，以上步骤会在以下位置创建新签名密钥：

C:\Users\USERNAME\AppData\Local\Xamarin\Mono for Android\Keystore\chimp\chimp.keystore

NOTE

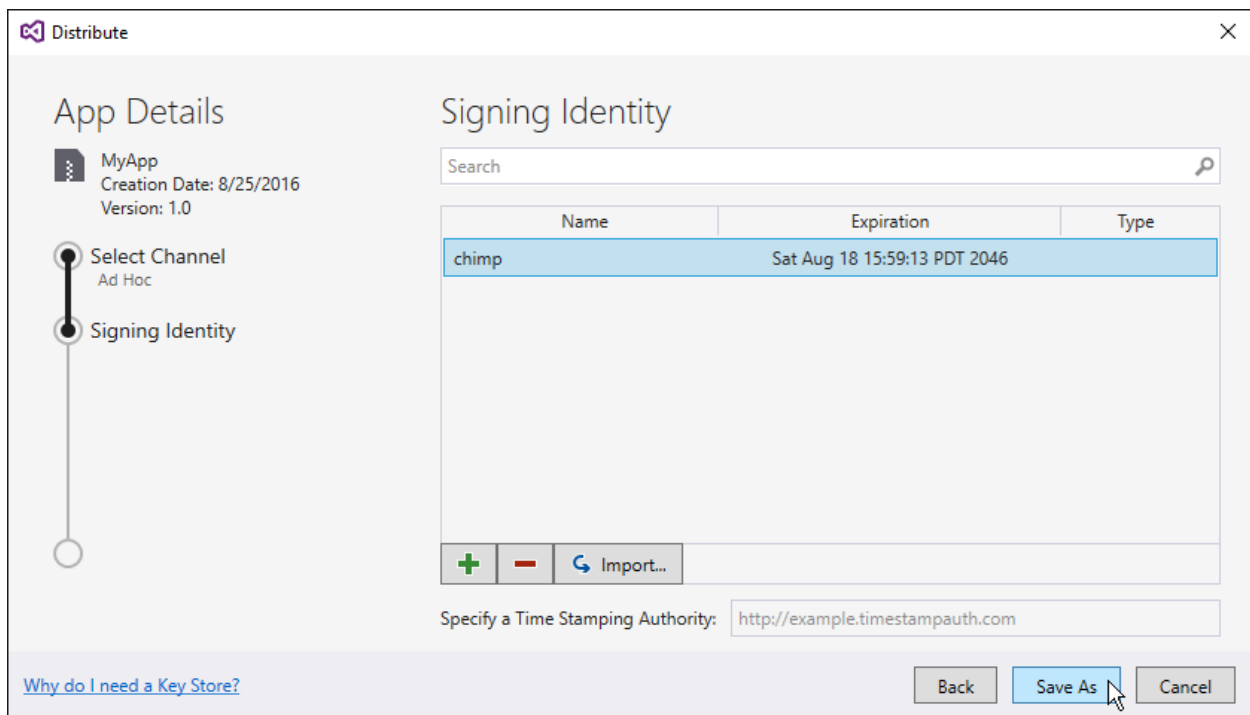
请确保将生成的密钥存储文件和密码备份在安全的位置 – 它不包含在解决方案中。如果密钥存储文件(例如，因为移动到另一台计算机或重新安装了 Windows)丢失，将无法使用与以前版本相同的证书对应用签名。

有关密钥存储的详细信息，请参阅[查找密钥存储的 MD5 或 SHA1 签名](#)。

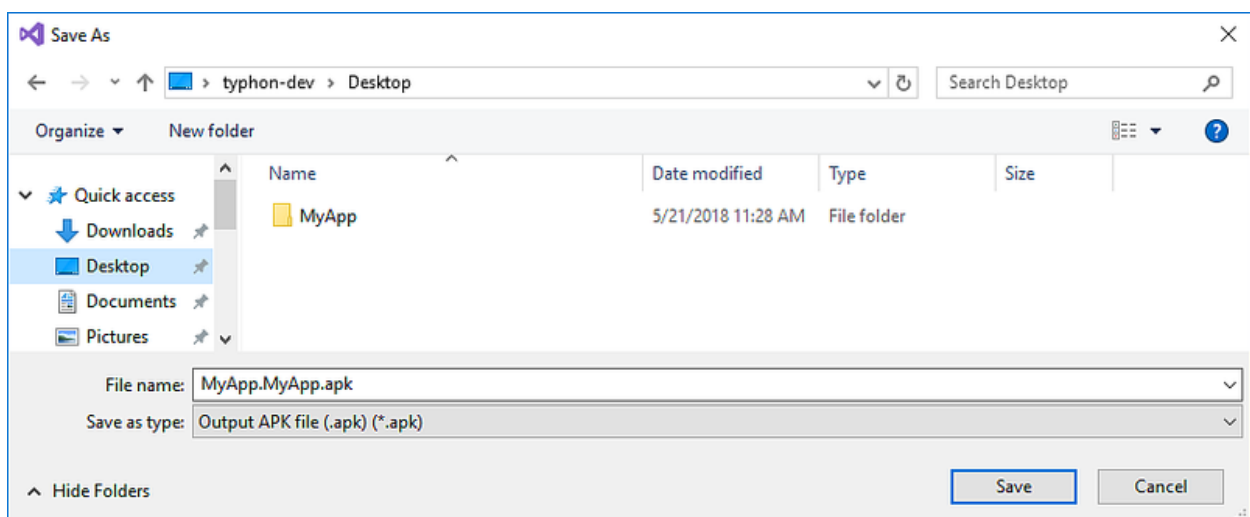
对 APK 进行签名

- [Visual Studio](#)
- [Visual Studio for Mac](#)

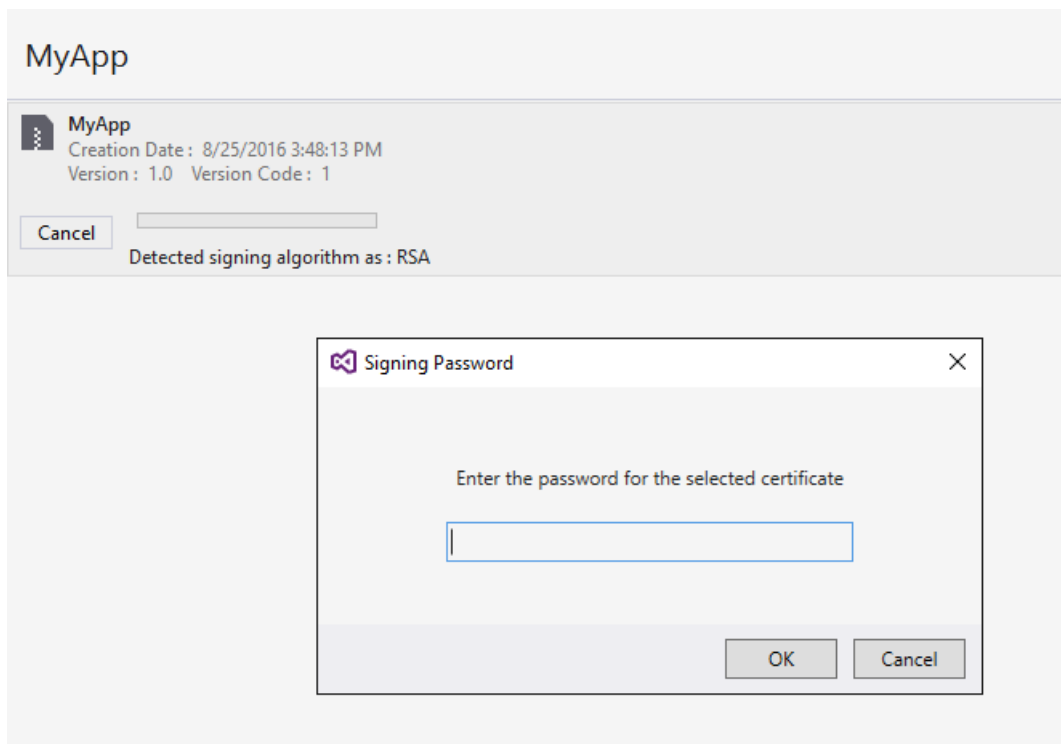
单击“创建”时，新密钥存储(包含新证书)会进行保存并在“签名标识”下列出，如下一个屏幕截图所示。若要在 Google Play 上发布应用，请单击“取消”并转至[发布到 Google Play](#)。若要即席发布，请选择要用于签名的签名标识并单击“另存为”以发布应用以用于独立分发。例如，在此屏幕截图中选择了 **chimp** 签名标识(在前面创建)：



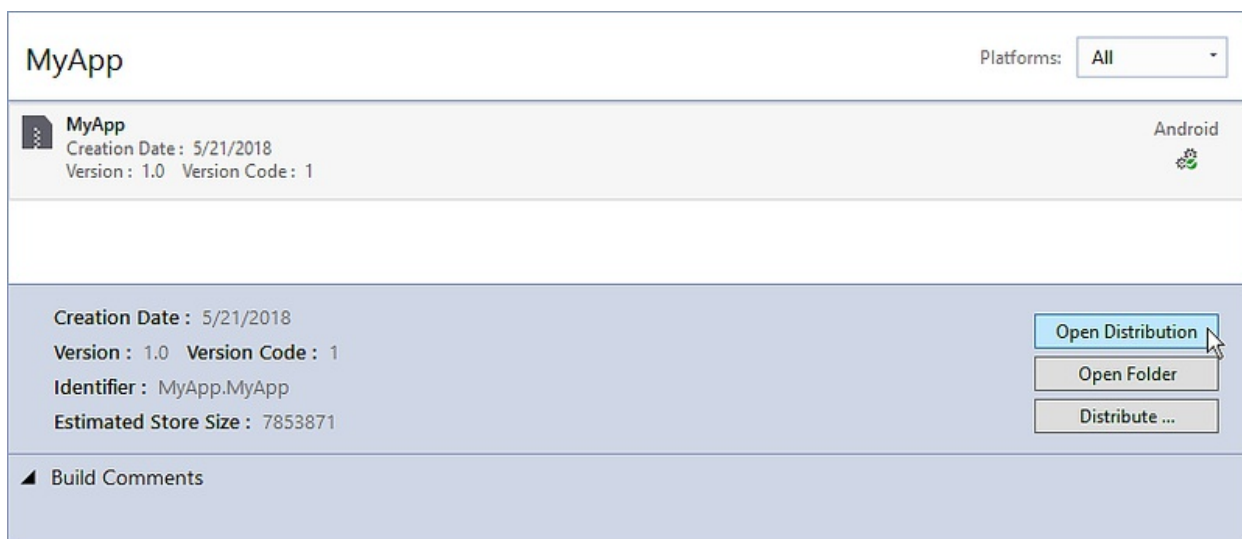
接下来，存档管理器会显示发布进度。发布过程完成时，“另存为”对话框会打开，要求提供要在其中存储生成的 .APK 文件的位置：



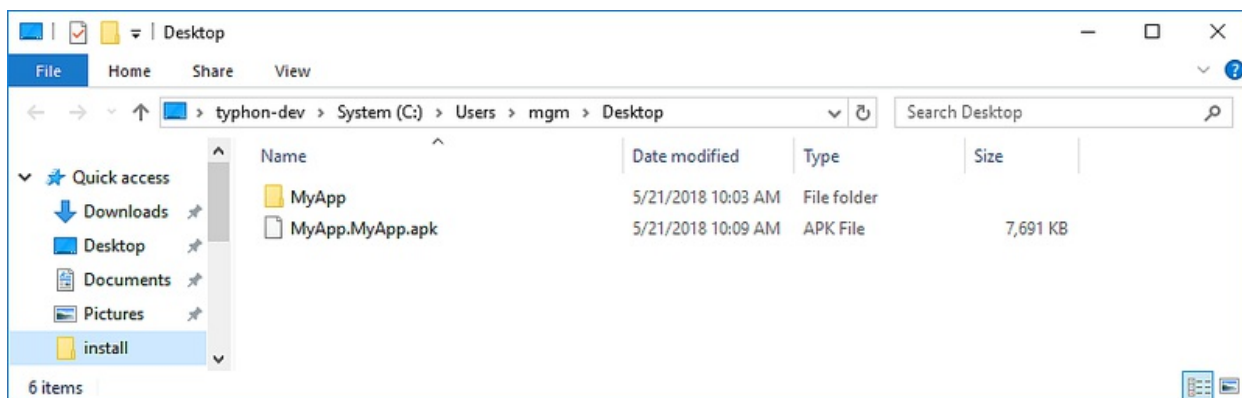
导航到所需位置并单击“保存”。如果密钥密码未知，则“签名密码”对话框会出现，提示输入所选证书的密码：



签名过程完成之后，单击“打开分发”：



这会使 Windows 资源管理器打开包含生成的 APK 文件的文件夹。此时，Visual Studio 已将 Xamarin.Android 应用程序编译为准备好进行分发的 APK。下面的屏幕截图显示准备好进行发布的应用 (**MyApp.MyApp.apk**) 的示例：



后续步骤

对要发行的应用程序包进行签名之后，必须发布它。以下各部分介绍用于发布应用程序的几种方法。

对 APK 进行手动签名

2018/11/5 • [Edit Online](#)

生成用于发布的应用程序后，应先对 APK 进行签名，然后再进行分发，以便它能够在 Android 设备上运行。此过程通常在 IDE 中处理，但某些情况下，需要在命令行中手动对 APK 进行签名。对 APK 进行签名的步骤如下：

1. **创建私钥** – 此步骤仅需执行一次。对 APK 进行数字签名需要私钥。准备好私钥后，可对将来的发布版本跳过此步骤。
2. **使用 Zipalign 优化 APK** – *Zipalign* 是对应用程序执行的优化过程。它可使 Android 和 APK 在运行时的交互更高效。Xamarin.Android 会在运行时执行检查，如果 APK 尚未使用 zipalign 进行优化，则不允许应用程序运行。
3. **对 APK 进行签名** – 此步骤涉及使用来自 Android SDK 的 *apksigner* 实用工具，并通过上一步创建的私钥对 APK 进行签名。对于使用 v24.0.3 之前的 Android SDK 生成工具版本进行开发的应用程序，它们将使用来自 JDK 的 *jarsigner* 应用。下面更详细地讨论这两种工具。

步骤的顺序至关重要，取决于使用何种工具对 APK 进行签名。当使用 *apksigner* 时，务必先使用 *zipalign* 优化应用程序，然后再通过 *apksigner* 对其签名。如需使用 *jarsigner* 对 APK 进行签名，则需先对 APK 进行签名，然后再运行 *zipalign*。

系统必备

本指南主要介绍使用来自 Android SDK 生成工具 v24.0.3 或更高版本的 *apksigner*。假设已生成 APK。

通过早期 Android SDK 生成工具版本生成的应用程序必须使用 *jarsigner*，如下面的[使用 jarsigner 对 APK 进行签名](#)中所述。

创建专用密钥存储

密钥存储是使用 Java SDK 中的 *keytool* 程序创建的安全证书数据库。密钥存储对发布 Xamarin.Android 应用程序至关重要，因为 Android 仅运行经数字签名的应用程序。

在开发期间，Xamarin.Android 使用调试密钥存储对应用程序进行签名，从而使应用程序可直接部署到仿真程序或配置为使用可调试应用程序的设备中。但若要分发应用程序，此密钥存储不会被视为有效的密钥存储。

因此，必须创建专用密钥存储，并用其对应用程序签名。该步骤应该仅执行一次，因为同一密钥可用于发布更新，然后用于对其他应用程序进行签名。

请务必保护此密钥存储。如果丢失，则无法通过 Google Play 发布应用程序更新。若要解决密钥存储丢失造成的问题，只能创建新的密钥存储、使用新密钥对 APK 重新签名，然后提交新版应用程序。然后，必须从 Google Play 中删除旧版应用程序。同样，如果新的密钥存储被泄露或公开分发，则可能会分发非官方版本或恶意版本的应用程序。

创建新的密钥存储

需要 Java SDK 中的命令行工具 *keytool* 才可新建密钥存储。下面的示例代码段演示了如何使用 *keytool* (将

`<my-filename>` 替换为密钥存储的文件名，将 `<key-name>` 替换为密钥存储内密钥的名称)：

```
$ keytool -genkeypair -v -keystore <filename>.keystore -alias <key-name> -keyalg RSA \
    -keysize 2048 -validity 10000
```

keytool 首先要求提供密钥存储的密码。然后要求提供一些有助于创建密钥的信息。下面的示例代码段演示了如何

创建将存储在文件 `xample.keystore` 中的名为 `publishingdoc` 的新密钥：

```
$ keytool -genkeypair -v -keystore xample.keystore -alias publishingdoc -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Ham Chimpanze
What is the name of your organizational unit?
  [Unknown]:  NASA
What is the name of your organization?
  [Unknown]:  NASA
What is the name of your City or Locality?
  [Unknown]:  Cape Canaveral
What is the name of your State or Province?
  [Unknown]:  Florida
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Ham Chimpanze, OU=NASA, O=NASA, L=Cape Canaveral, ST=Florida, C=US correct?
[no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA1withRSA) with a validity of 10,000 days
    for: CN=Ham Chimpanze, OU=NASA, O=NASA, L=Cape Canaveral, ST=Florida, C=US
Enter key password for <publishingdoc>
    (RETURN if same as keystore password):
Re-enter new password:
[Storing xample.keystore]
```

若要列出密钥存储中存储的密钥，请通过 **keytool** 中使用 `-list` 选项：

```
$ keytool -list -keystore xample.keystore
```

使用 Zipalign 优化 APK

在使用 `apksigner` 对 APK 进行签名之前，首先要使用来自 Android SDK 的 `zipalign` 工具对文件进行优化。`zipalign` 将沿着 4 字节边界重组 APK 中的资源。这种调整使 Android 可快速加载 APK 中的资源，提高应用程序的性能，并有可能降低内存使用。`Xamarin.Android` 将执行运行时检查，确定是否已使用 `zipalign` 优化 APK。对 APK 进行 `zipalign` 优化后，应用程序才会运行。

以下命令将使用已签名的 APK 并生成一个名为 **helloworld.apk** 的 APK，后者已经过签名和 `zipalign` 优化且可分发。

```
$ zipalign -f -v 4 mono.samples.helloworld-unsigned.apk helloworld.apk
```

对 APK 进行签名

对 APK 进行 `zipalign` 优化后，务必使用密钥存储对其进行签名。此操作需使用 `apksigner` 工具来完成，该工具可在 SDK 生成工具版本的 `build-tools` 目录中找到。例如，如果安装了 Android SDK 生成工具 v25.0.3，则可在目录中找到 `apksigner`：

```
$ ls $ANDROID_HOME/build-tools/25.0.3/apksigner
/Users/tom/android-sdk-macosx/build-tools/25.0.3/apksigner*
```

以下代码片段假设 `apksigner` 可由 `PATH` 环境变量访问。它使用包含在文件 `xample.keystore` 中的密钥别名 `publishingdoc` 来签名 APK：


```
$ apksigner sign --ks xample.keystore --ks-key-alias publishingdoc mono.samples.helloworld.apk
```

运行此命令时，apksigner 将要求提供密钥存储的密码(如有必要)。

有关如何使用 apksigner 的详细信息，请参阅 [Google 文档](#)。

NOTE

根据 [Google 问题 62696222](#)，Android SDK 中“缺少”apksigner。此问题的解决方案是安装 Android SDK 生成工具 v25.0.3，并使用该版本的 apksigner。

使用 jarsigner 对 APK 进行签名

WARNING

本节仅适用于需要使用 jarsigner 实用工具对 APK 进行签名的情况。推荐开发者使用 apksigner 对 APK 进行签名。

此技术包括使用来自 Java SDK 的 [jarsigner](#) 命令对 APK 文件进行签名。jarsigner 工具由 Java SDK 提供。

以下示例演示如何使用 **jarsigner** 和 **xample.keystore** 密钥存储文件中包含的密钥 `publishingdoc` 对 APK 进行签名：

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore xample.keystore mono.samples.helloworld.apk publishingdoc
```

NOTE

使用 jarsigner 时，务必首先对 APK 进行签名，然后再使用 zipalign。

相关链接

- [应用程序签名](#)
- [Java JAR 签名](#)
- [jarsigner](#)
- [keytool](#)
- [zipalign](#)
- [生成工具 26.0.0 - apksigner 去哪里了？](#)

查找密钥存储的签名

2018/11/2 • [Edit Online](#)

Xamarin.Android 应用的 MD5 或 SHA1 签名取决于用于对 APK 进行签名的 **.keystore** 文件。通常，调试版本会使用不同于发布版本的 **.keystore** 文件。

对于调试/非自定义签名版本

Xamarin.Android 使用相同的 **debug.keystore** 文件对所有调试版本进行签名。初次安装 Xamarin.Android 时会生成此文件。以下步骤详细介绍了默认 Xamarin.Android **debug.keystore** 文件的 MD5 或 SHA1 签名查找过程。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

找到用于对应用进行签名的 Xamarin **debug.keystore** 文件。默认情况下，用于对 Xamarin.Android 应用程序调试版本进行签名的密钥存储位于以下位置：

C:\Users\USERNAME\AppData\Local\Xamarin\Mono for Android\debug.keystore

可通过从 JDK 运行 `keytool.exe` 命令来获取有关密钥存储的信息。此工具通常位于以下位置：

C:\Program Files (x86)\Java\jdkVERSION\bin\keytool.exe

将包含 **keytool.exe** 的目录添加到 `PATH` 环境变量。打开命令提示符，使用下面的命令运行 `keytool.exe`：

```
keytool.exe -list -v -keystore "%LocalAppData%\Xamarin\Mono for Android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

运行时，**keytool.exe** 应输出以下文本。**MD5:** 和 **SHA1:** 标签标识相应的签名：

```
Alias name: androiddebugkey
Creation date: Aug 19, 2014
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 53f3b126
Valid from: Tue Aug 19 13:18:46 PDT 2014 until: Sun Nov 15 12:18:46 PST 2043
Certificate fingerprints:
    MD5: 27:78:7C:31:64:C2:79:C6:ED:E5:80:51:33:9C:03:57
    SHA1: 00:E5:8B:DA:29:49:9D:FC:1D:DA:E7:EE:EE:1A:8A:C7:85:E7:31:23
    SHA256: 21:0D:73:90:1D:D6:3D:AB:4C:80:4E:C4:A9:CB:97:FF:34:DD:B4:42:FC:
08:13:E0:49:51:65:A6:7C:7C:90:45
Signature algorithm name: SHA1withRSA
Version: 3
```

对于发布/自定义签名版本

对于使用自定义 **.keystore** 文件签名的发布版本，过程与上述相同，将 Xamarin.Android 使用的 **debug.keystore** 文件替换为发布 **.keystore** 文件。发布密钥存储文件创建后，替换为自己的密钥存储密码值和别名。

- [Visual Studio](#)

- [Visual Studio for Mac](#)

使用 Visual Studio“分发”向导对 Xamarin.Android 应用进行签名时，产生的密钥存储位于以下位置：

C:\Users\USERNAME\AppData\Local\Xamarin\Mono for Android\Keystore\alias\alias.keystore

例如，如果按照[创建新证书](#)中的步骤创建新的签名密钥，则产生的示例密钥存储位于以下位置：

C:\Users\USERNAME\AppData\Local\Xamarin\Mono for Android\Keystore\chimp\chimp.keystore

有关对 Xamarin.Android 应用进行签名的详细信息，请参阅[对 Android 应用程序包进行签名](#)。

发布应用程序

2018/10/26 • [Edit Online](#)

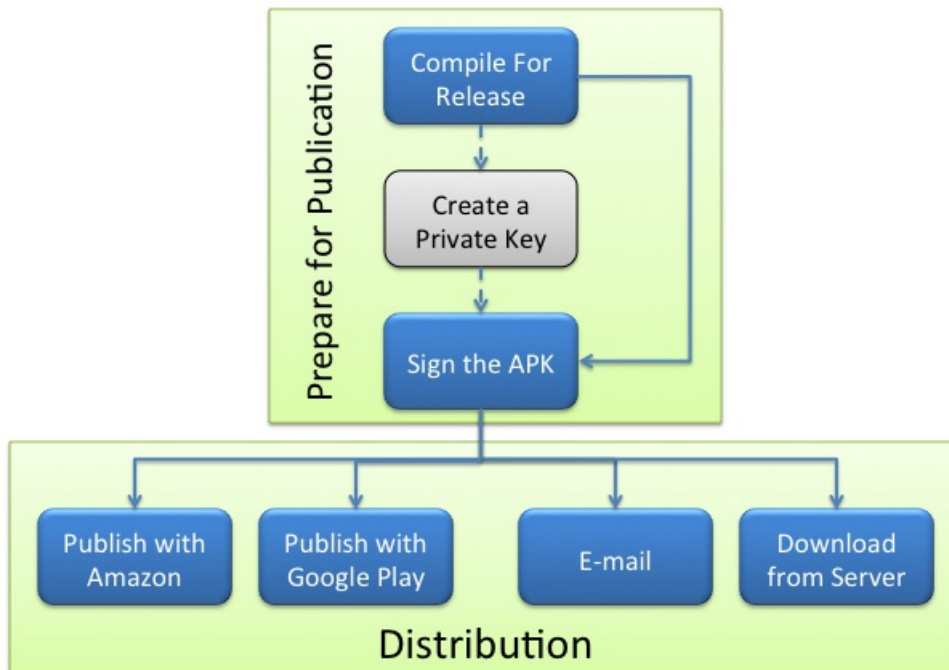
创建优秀的应用程序后，用户自然希望进行使用。本部分介绍公开发分使用 Xamarin.Android 创建的应用程序所涉及的步骤，其中销售渠道包括电子邮件、专用 Web 服务器、Google Play 或 Android 版 Amazon 应用商店。

概述

Xamarin.Android 应用程序开发的最终步骤是发布应用程序。发布是编译 Xamarin.Android 应用程序，使其可供用户安装在设备上的过程，其中包括 2 个基本任务：

- 做好发布准备 – 创建可部署到支持 Android 的设备的程序的一个发布版本(有关发布准备的详细信息，请参阅[做好应用程序发布准备](#))。
- 分发 – 通过一个或多个不同的分发渠道提供应用程序的发行版本。

下图说明了发布 Xamarin.Android 应用程序涉及的步骤：



如上图所示，所有分发方式的准备过程都是相同的。可通过以下几种方式将 Android 应用程序发布给用户：

- 通过网站 – Xamarin.Android 应用程序可通过在网站下载获得，用户可在网站中单击链接安装应用程序。
- 通过电子邮件 – 用户可以从电子邮件安装 Xamarin.Android 应用程序。在 Android 设备上打开附件时会安装应用程序。
- 通过应用商店 – 有多个应用市场可用于分发，例如 [Google Play](#) 或 [Android 版 Amazon 应用商店](#)。

发布应用程序最常见的方法是通过知名的应用商店，因为它可以提供最广的市场范围和达到最佳的分发控制。但是，通过应用市场发布应用程序需要进行其他工作。

Xamarin.Android 应用程序可同时通过多个渠道分发。例如，应用程序可在 Google Play 和 Android 版 Amazon 应用商店上发布，也可从 Web 服务器下载。

其他两种分发方法(下载或电子邮件)非常适用于受控数量的用户，例如仅面向少数或特定用户的企业环境或应用程序。服务器和电子邮件分发也是比较简单的发布模型，发布应用程序所需的准备较少。

通过 Amazon 移动应用分发程序, 移动应用开发人员可在 Amazon 上分发和销售应用程序。用户可使用 Amazon 应用商店应用程序在 Android 设备上发现和购买应用。Android 设备上运行的 Amazon 应用商店的屏幕截图如下所示:

Google Play 无疑是最全面和最受欢迎的 Android 应用程序应用市场。在 Google Play 中, 用户可以仅通过单击设备或计算机上的图标便可发现、下载、评价和购买应用程序。Google Play 还提供工具以帮助分析销售和市场趋势和控制可能下载应用程序的设备和用户。Android 设备上运行的 Google Play 的屏幕截图如下所示:

23°



17:58



Apps



CATEGORIES

FEATURED

TOP PAID



pocket



Staff Picks



Games



Editors' Choice



TED

Ideas worth spreading



hipmunk



flights



这部分演示如何将应用程序与相应的宣传材料上传到 Google Play 等商店。对 APK 扩展文件进行了解释, 从概念上概述了其内容和工作原理。还对 Google 授权服务进行了说明。最后, 介绍了其他的分发方式, 包括使用 HTTP Web 服务器、简单的电子邮件分发以及适用于 Android 的 Amazon 应用商店。

相关链接

- [HelloWorldPublishing\(示例\)](#)
- [生成过程](#)
- [链接](#)
- [获取 Google Maps API 密钥](#)
- [应用程序签名](#)
- [发布到 Google Play](#)
- [Google 应用程序授权](#)
- [Android.Play.ExpansionLibrary](#)
- [移动应用分发门户](#)
- [Amazon 移动应用分发常见问题解答](#)

发布到 Google Play

2018/10/26 • [Edit Online](#)

尽管有许多应用市场可以分发应用程序，但 Google Play 无疑是世界上最大和访问次数最多的 Android 应用商店。Google Play 提供单一平台，可用于 Android 应用程序的分发、宣传推广、销售和营销分析。

本部分将介绍一些特定于 Google Play 的主题，例如注册成为发布者、收集资产以便帮助 Google Play 推广和宣传应用程序、Google Play 上的应用程序评分指南以及使用筛选器限定应用程序只部署到某些特定设备。

要求

若要通过 Google Play 分发应用程序，必须创建开发者帐户。只需执行一次此操作，并且支付一次性费用 25 美元。

所有应用程序都需使用加密密钥进行签名，该密钥将于 2033 年 10 月 22 日后过期。

在 Google Play 上发布的 APK 的最大大小为 100MB。如果应用程序超出该大小，Google Play 允许通过 APK 扩展文件交付额外资产。Android 扩展文件允许 APK 具有 2 个额外文件，每个文件大小最大为 2GB。Google Play 会免费托管和分发这些文件。将在另一部分中对扩展文件进行讨论。

Google Play 并非全球可用。一些区域可能不支持应用程序分发。

成为发布者

若要在 Google Play 上发布应用程序，需要具有发布者帐户。若要注册发布者帐户，请按照以下步骤操作：

1. 访问 [Google Play 开发者控制台](#)。
2. 输入开发者基本身份信息。
3. 阅读并接受适用于所在区域的开发者分发协议。
4. 支付 25 美元注册费用。
5. 通过电子邮件确认验证。
6. 创建帐户后，便可使用 Google Play 发布应用程序。

并非所有的国家/地区都支持 Google Play。可通过以下链接获取最新的国家/地区列表：

1. [开发者 & 商户注册支持区域](#) – 该列表包括支持开发者注册成为商户并销售付费应用程序的所有国家/地区。
2. [支持向 Google Play 用户分发的区域](#) – 该列表包括支持分发应用程序的所有国家/地区。

准备促销资产

为在 Google Play 上有效宣传和推广应用程序，Google 允许开发人员提交屏幕截图、图形和视频等促销资产。Google Play 随后会使用这些资产宣传和推广应用程序。

启动器图标

启动器图标是表示应用程序的图形。每个启动器图标应为具有透明度 alpha 通道的 32 位 PNG。应用程序应具有适合所有通用屏幕密度的图标，如下面的列表中所述：

- **ldpi** (120dpi) – 36 x 36 px
- **mdpi** (160dpi) – 48 x 48 px
- **hdpi** (240dpi) – 72 x 72 px
- **xhdpi** (320dpi) – 96 x 96 px

用户在 Google Play 上首先看到的是应用程序的启动器图标，因此请务必使启动器图标具有视觉吸引力和一定意义。

启动器图标提示：

1. **简单整洁** – 启动器图标应保持简单整洁。这意味着图标中不应包含应用程序名称。越简单的图标越容易记住，并且在尺寸较小的情况下也更易于识别。
2. **图标不应细窄** – 过窄的图标在所有背景上不易突出。
3. **使用 alpha 通道** – 图标应使用 alpha 通道，并且不应为全帧图像。

高分辨率应用程序图标

Google Play 上的应用程序需要使用高保真版本的应用程序图标。其仅供 Google Play 使用，并不会替换应用程序启动器图标。高分辨率图标的规格为：

1. 具有 alpha 通道的 32 位 PNG
2. 512 x 512 像素
3. 最大大小为 1024KB

[Android Asset Studio](#) 是非常有用的工具，可用于创建合适的启动器图标以及高分辨率应用程序图标。

屏幕截图

对于每个应用程序，Google Play 需要至少 2 张，最多 8 张屏幕截图。它们会显示在 Google Play 中的应用程序详细信息页面。

屏幕截图规格如下：

1. 无 alpha 通道的 24 位 PNG 或 JPG
2. 320w x 480h、480w x 800h 或 480w x 854h。会对横向的图像进行裁剪。

促销图

这是 Google Play 使用的可选图像：

1. 规格为 180w x 120h 无 alpha 通道 24 位 PNG 或 JPG。
2. 图像无边框。

特征图形

由 Google Play 特征部分使用。此图形可能会单独显示，不会附带应用程序图标。

1. 无 alpha 通道和透明度的 1024w x 500h 大小的 PNG 或 JPG。
2. 所有重要内容应在 924x500 大小范围内。为满足样式要求，可能会裁剪超过此范围的像素。
3. 此图形可能会按比例缩小：使用较大的文本并简化图形。

视频链接

这是一个展示应用程序的 YouTube 视频 URL。视频长度应在 30 秒到 2 分钟之间，并能展示应用程序的最佳部分。

发布到 Google Play

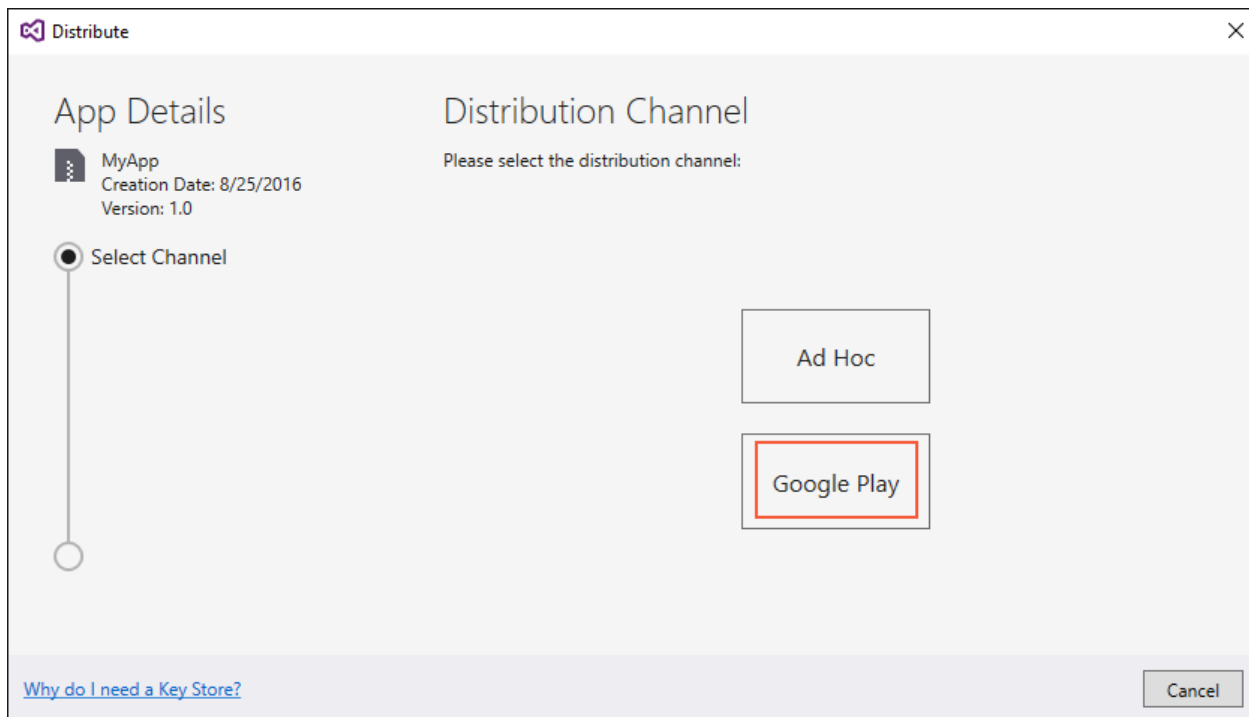
- [Visual Studio](#)
- [Visual Studio for Mac](#)

Xamarin Android 7.0 采用集成工作流，将应用从 Visual Studio 发布到 Google Play。如果使用早于 7.0 版本的 Xamarin Android，则必须通过 Google Play 开发者控制台手动上传 APK。此外，必须已至少上传 1 个 APK，才能使用此集成工作流。如果尚未上传第一个 APK，则必须手动进行上传。有关详细信息，请参阅[手动上传 APK](#)。

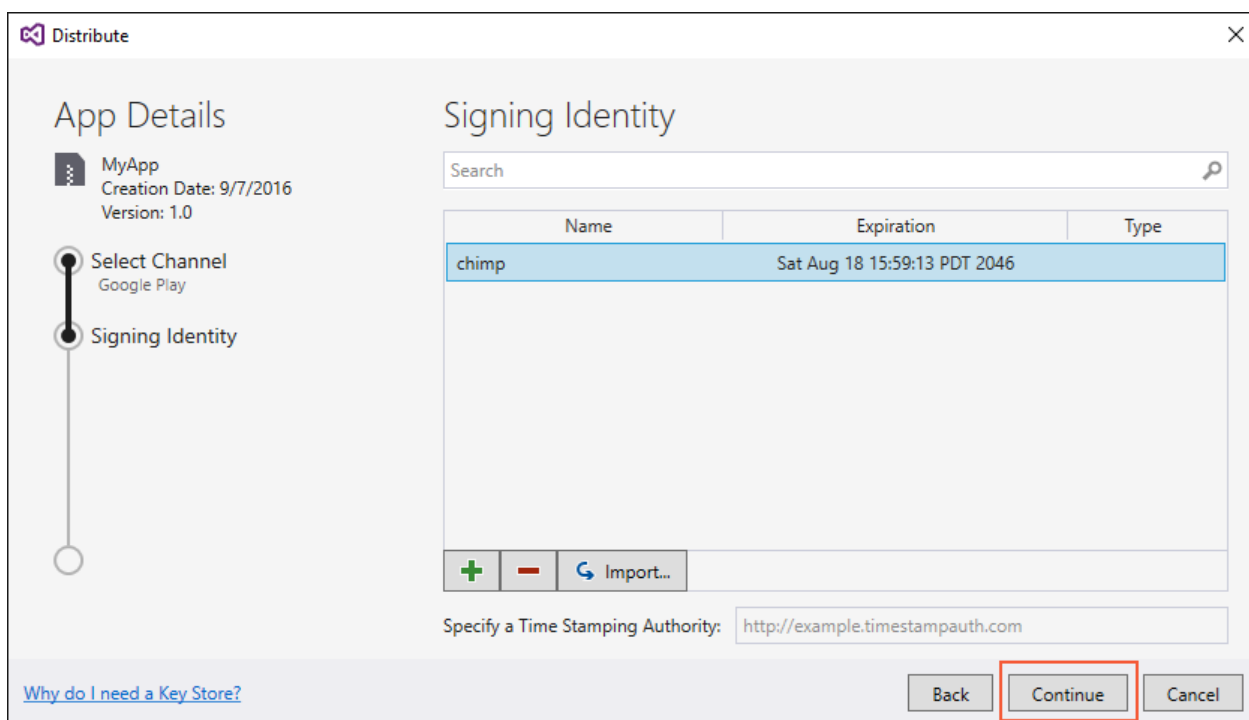
[创建新的证书](#)说明了如何创建新的证书以对 Android 应用进行签名。下一步，将已签名的应用发布到 Google Play：

1. 登录 Google Play 开发者帐户，创建链接到 Google Play 开发者帐户的新项目。
2. 创建对应用进行身份验证的 **OAuth 客户端**。
3. 在 Visual Studio 中输入生成的客户端 ID 和客户端密码。
4. 通过 Visual Studio 注册帐户。
5. 使用证书对应用进行签名。
6. 将已签名应用发布到 Google Play。

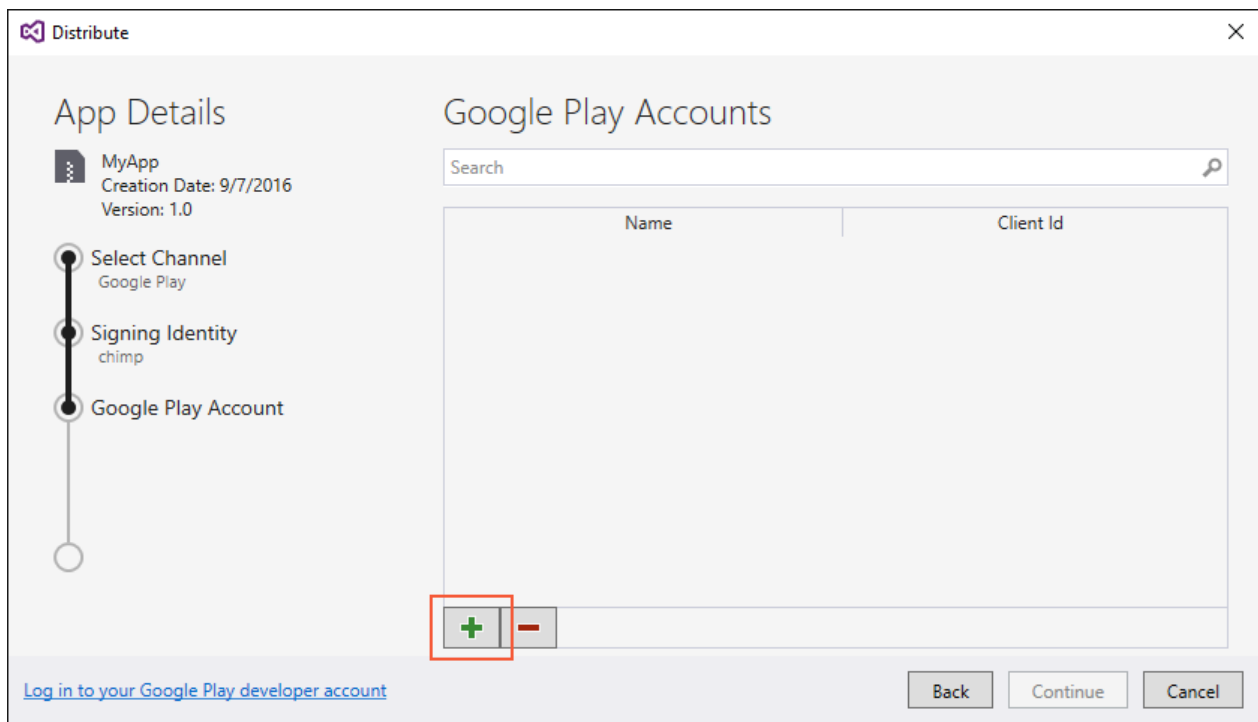
在[用于发布的存档](#)中，“分发渠道”对话框提供了两种发布选择：**Ad Hoc** 和 **Google Play**。如果显示的是“签名标识”对话框，请单击“返回”，返回到“分发渠道”对话框。选择“Google Play”，然后单击“下一步”：



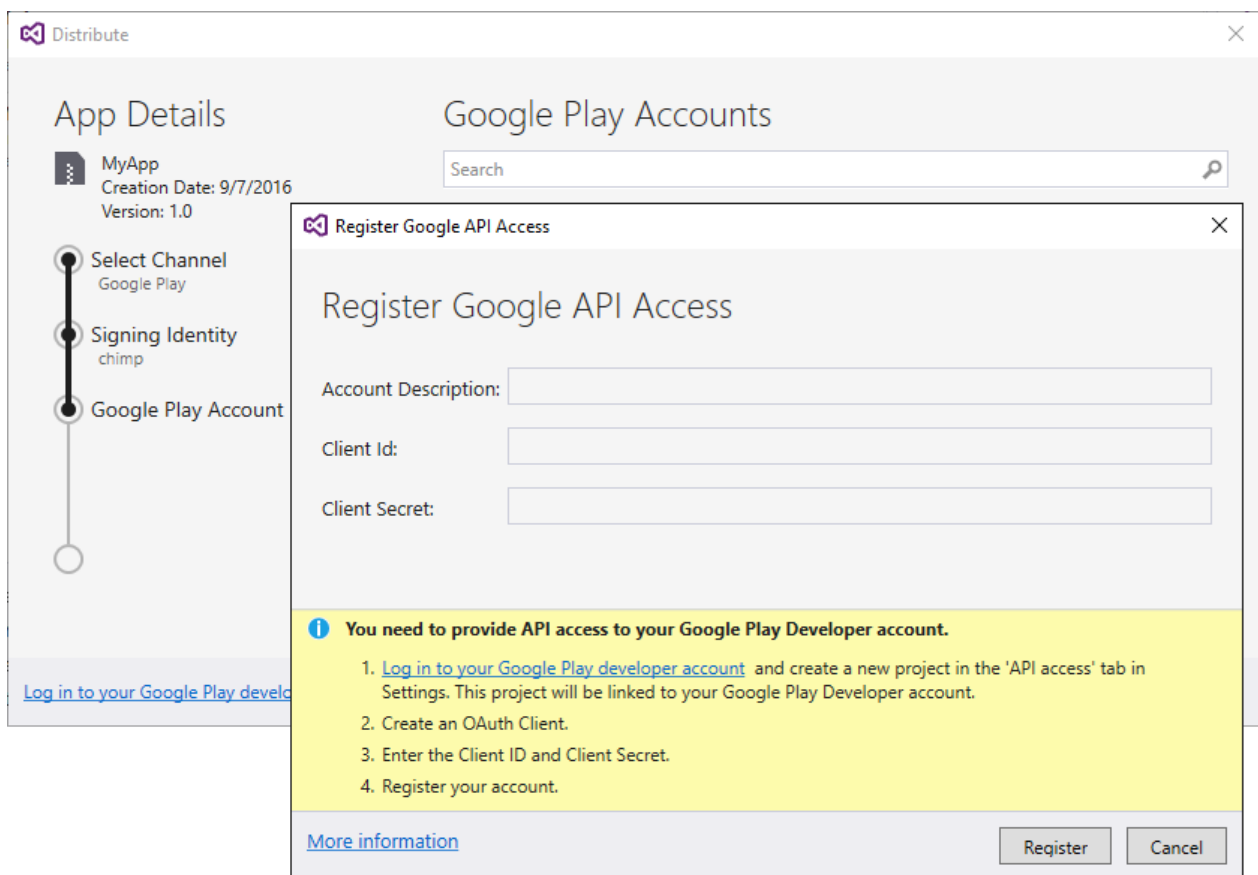
在“签名标识”对话框中，选择在[创建新的证书](#)中创建的标识，然后单击“继续”：



在“Google Play 帐户”对话框中，单击“+”按钮，添加新的 Google Play 帐户：



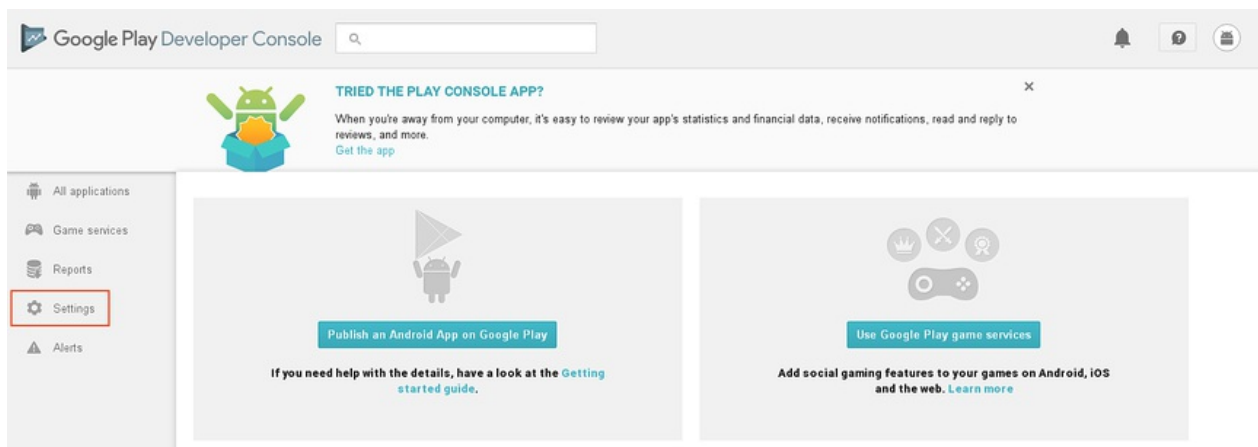
在“注册 Google Play API 访问权限”对话框中，必须提供客户端 ID 和客户端密码，从而向 Google Play 开发者帐户提供 API 访问权限：



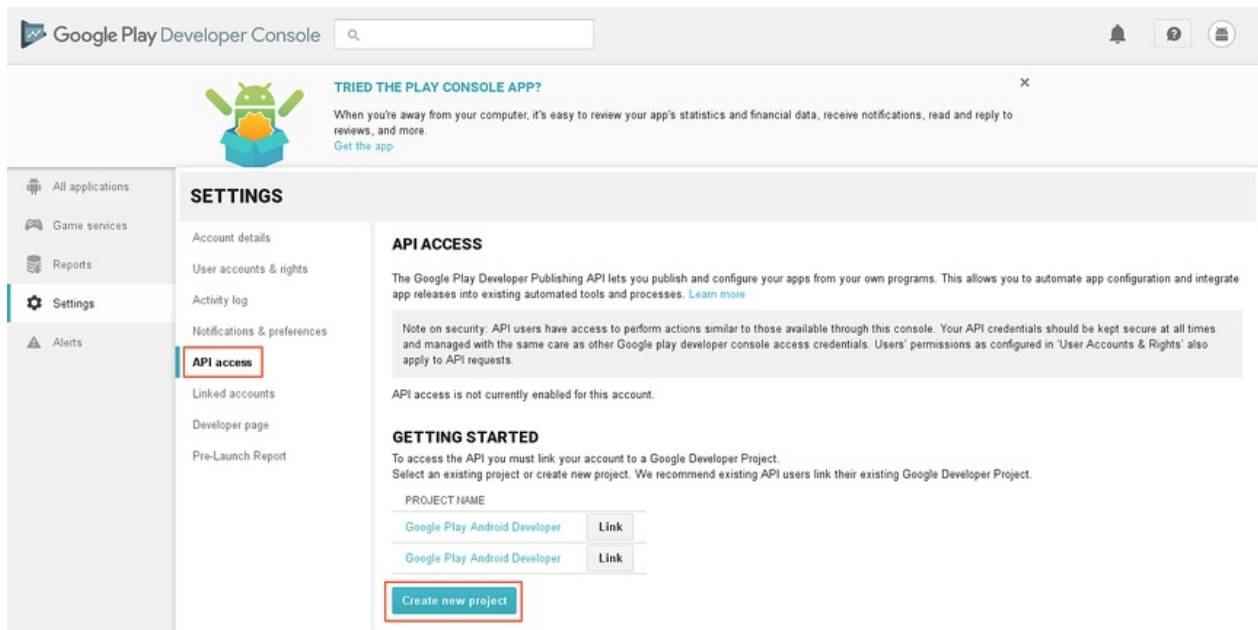
下一部分介绍如何新建 Google API 项目以及如何生成所需的_客户端 ID_ 和_客户端密码_。

创建 Google API 项目

首先，登录 [Google Play 开发者帐户](#)。如果尚没有 Google Play 开发者帐户，请参阅[发布入门](#)。此外，Google Play 开发者 API [入门](#)还介绍了如何使用 Google Play 开发者 API。登录 Google Play 开发者控制台后，请单击“设置”：

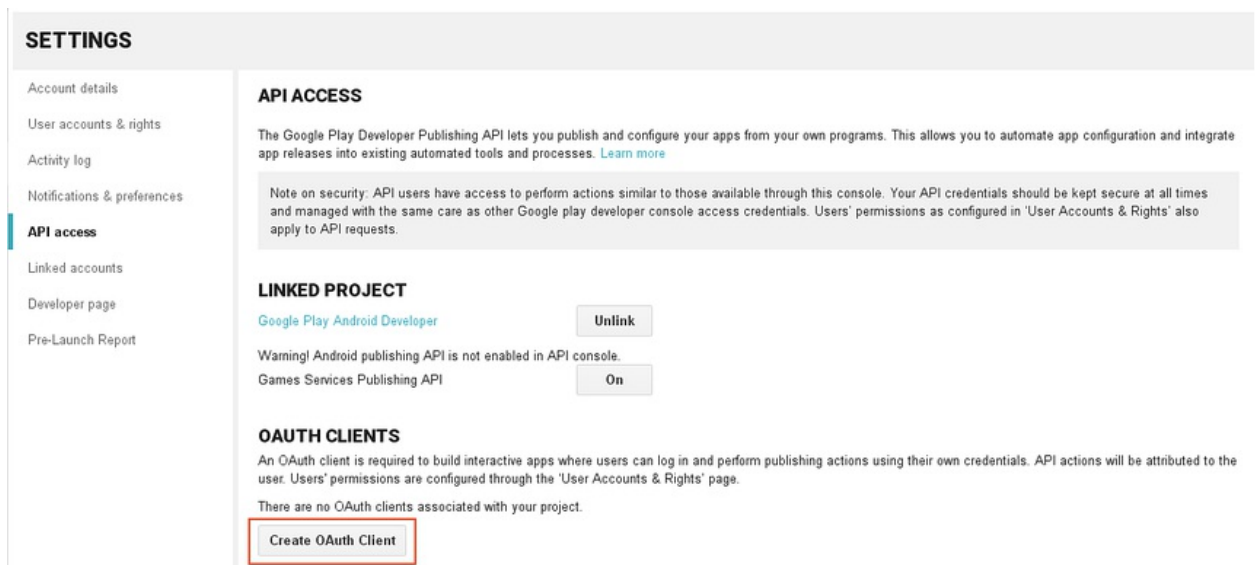


在“设置”页上，选择“API 访问”，然后单击“新建项目”按钮：



约一分钟后，新 API 项目会自动生成，并链接到 Google Play 开发者控制台帐户。

下一步，为应用创建 OAuth 客户端 (如果尚未创建)。用户使用应用请求访问个人数据时，使用 OAuth 客户端 ID 对应用进行身份验证。单击“创建 OAuth 客户端”，创建新的 OAuth 客户端：



几秒钟后将生成新的客户端 ID。单击“在 Google 开发者控制台中查看”，查看 Google 开发者控制台中的新客户 ID：

LINKED PROJECT

Google Play Android Developer

Unlink

Games Services Publishing API

On

OAUTH CLIENTS

An OAuth client is required to build interactive apps where users can log in and perform publishing actions using their own credentials. API actions will be attributed to the user. Users' permissions are configured through the 'User Accounts & Rights' page.

CLIENT ID

65179385121-t15sahafgl0c6roajdu7s.apps.googleusercontent.com

CLIENT SECRET

[View in Google Developers Console](#)

Create OAuth Client

显示客户端 ID 及其名称和创建日期。单击“编辑 OAuth 客户端”图标，查看应用的客户端密码：

| Name | Creation date | Type | Client ID |
|-------------------------------|---------------|-------|--|
| Google Play Android Developer | Sep 12, 2016 | Other | 65179385121-t15sahafgl0c6roajdu7s.apps.googleusercontent.com |

OAuth 客户端的默认名称是 Google Play Android 开发者。可将其更改为 Xamarin.Android 应用的名称或其他任何合适的名称。本示例中将 OAuth 客户端名称更改为此应用的名称，即 **MyApp**：

Client ID for Other

Client ID: 65179385121-t15sahafgl0c6roajdu7s.apps.googleusercontent.com

Client secret: BWwsMNl0MbJtIBUjuN6Q

Creation date: Sep 12, 2016, 2:06:19 PM

Name:

Save Cancel

单击“保存”以保存更改。此操作会返回“凭据”页面，在此处可通过单击“下载 JSON”图标下载凭据：

Client ID for Other

Client ID: 65179385121-t15sahafgl0c6roajdu7s.apps.googleusercontent.com

Client secret: BWwsMNl0MbJtIBUjuN6Q

Creation date: Sep 12, 2016, 2:06:19 PM

Name:

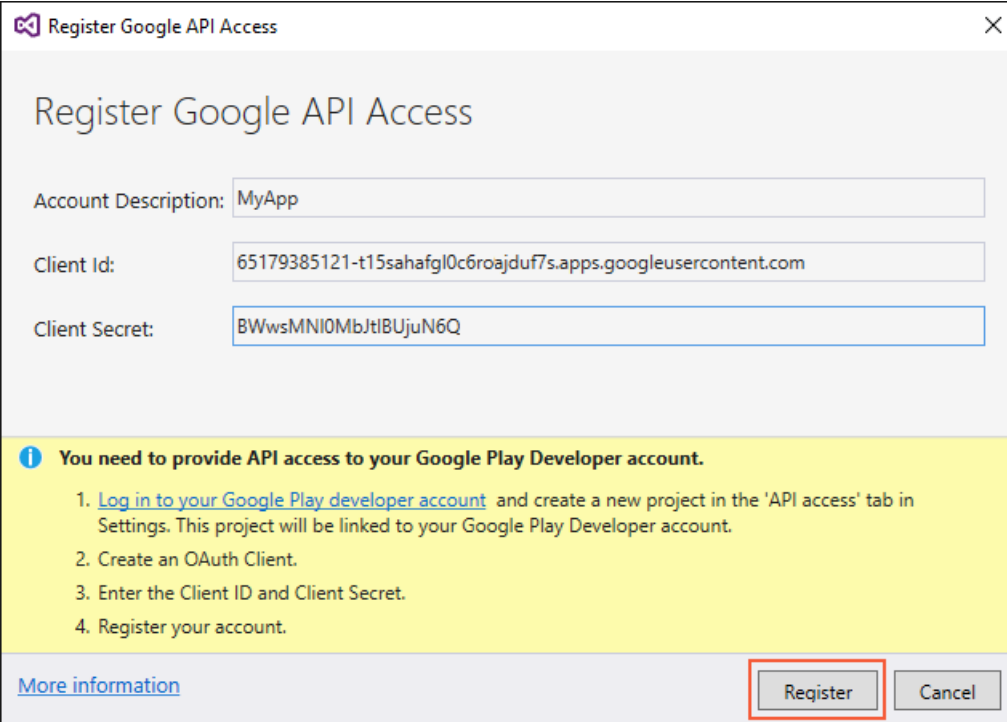
Save Cancel

此 JSON 文件包含客户端 ID 和客户端密码，可将其剪切并粘贴到下一步中的“签名和分发”对话框。

注册 Google API 访问权限

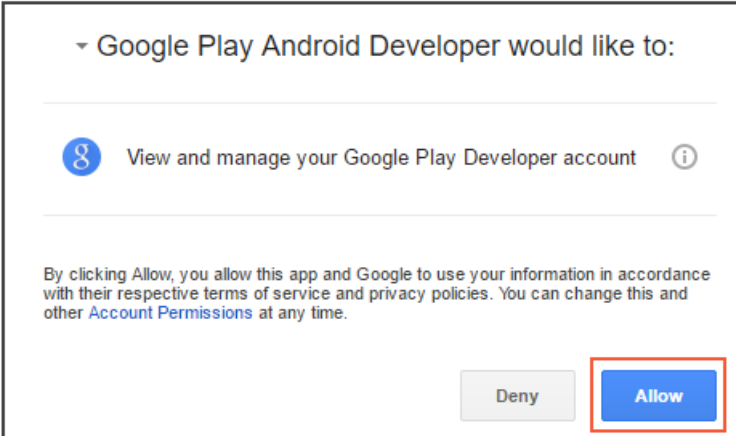
- [Visual Studio](#)
- [Visual Studio for Mac](#)

使用客户端 ID 和客户端密码填写 Visual Studio for Mac 中的“Google Play API 帐户”对话框。可以为帐户提供说明 – 这样可以注册多个 Google Play 帐户并可将以后的 APK 上传到其他 Google Play 帐户。将客户端 ID 和客户端密码复制到此对话框，然后单击“注册”：



The image shows a 'Register Google API Access' dialog box. It has a title bar with a close button. The main area contains three input fields: 'Account Description' with the value 'MyApp', 'Client Id' with the value '65179385121-t15sahafgl0c6roajduf7s.apps.googleusercontent.com', and 'Client Secret' with the value 'BWwsMNIOMbJtIBUjuN6Q'. Below these fields is a yellow information box with a blue 'i' icon and the text 'You need to provide API access to your Google Play Developer account.' followed by a numbered list of four steps: 1. Log in to your Google Play developer account and create a new project in the 'API access' tab in Settings. This project will be linked to your Google Play Developer account. 2. Create an OAuth Client. 3. Enter the Client ID and Client Secret. 4. Register your account. At the bottom left is a blue link 'More information'. At the bottom right are two buttons: 'Register' (highlighted with a red rectangle) and 'Cancel'.

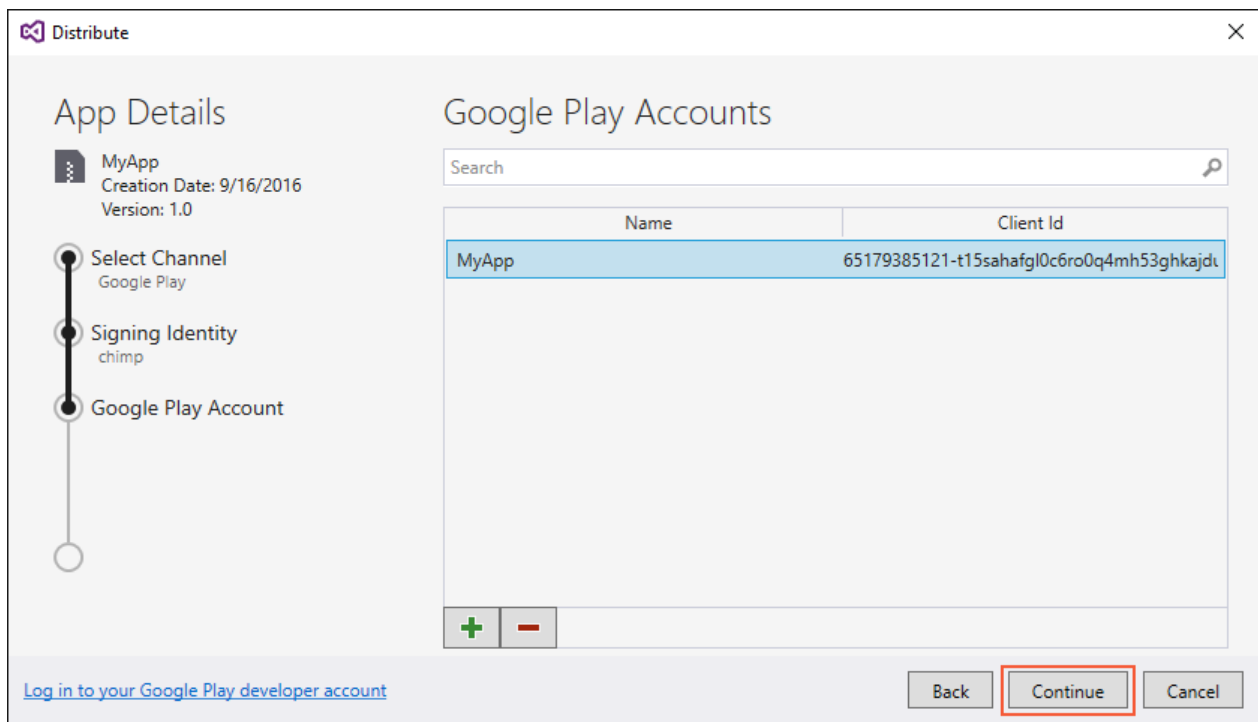
将打开 Web 浏览器，并提示登录 Google Play Android 开发者帐户 (如果尚未登录)。登录后，Web 浏览器中会显示以下提示。单击“允许”对应用授权：



The image shows a permission prompt from Google Play Android Developer. It has a title bar with a close button. The main area has a heading 'Google Play Android Developer would like to:' followed by a horizontal line. Below the line is a blue circle with a white 'g' icon, followed by the text 'View and manage your Google Play Developer account' and a small 'i' icon. Below this is another horizontal line. At the bottom, there is a paragraph of text: 'By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.' At the bottom right are two buttons: 'Deny' and 'Allow' (highlighted with a red rectangle).

发布

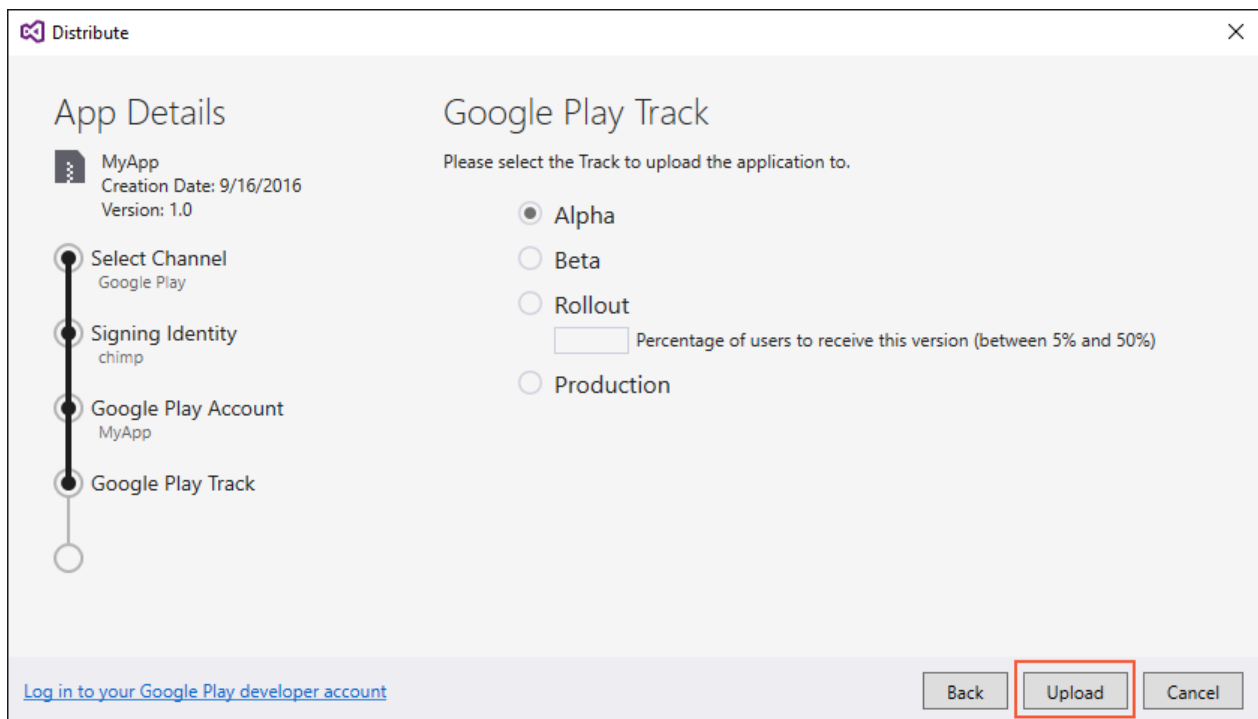
单击“允许”后，浏览器会报告已接收验证码。即将关闭...，且应用将被添加到 Visual Studio 中的 Google Play 帐户列表中。在“Google Play 帐户”对话框中，单击“继续”：



接下来，会显示“Google Play 轨道”对话框。Google Play 提供 4 个可用于上传应用的轨道：

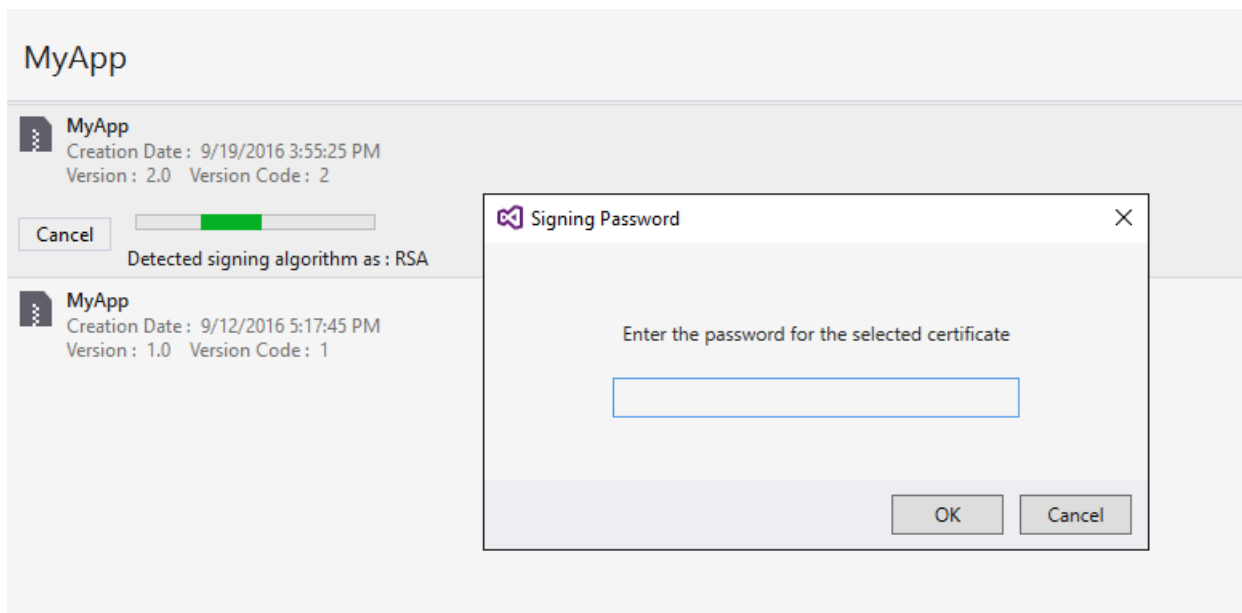
- **Alpha** – 用于将早期版本的应用上传到较小的测试员列表。
- **Beta** – 用于将早期版本的应用上传到较大的测试员列表。
- **Rollout** – 允许一定比例的用户接收应用的更新版本；这样可逐渐增加比例，例如，以 10% 的用户开始，消除 bug 后增加到 100% 用户。
- **Production** – 准备好从 Google Play 应用商店全面分发应用时，请选择此轨道。

选择用于上传应用的 Google Play 轨道，然后单击“上传”。如果选择 **Rollout**，请确保输入百分比值：

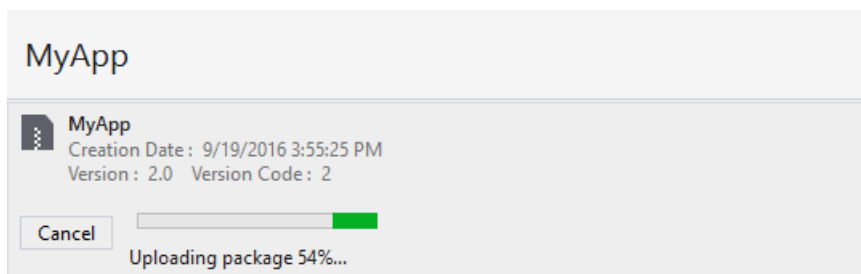


有关 Google Play 测试和分步推出的详细信息，请参阅[设置 alpha/beta 测试](#)。

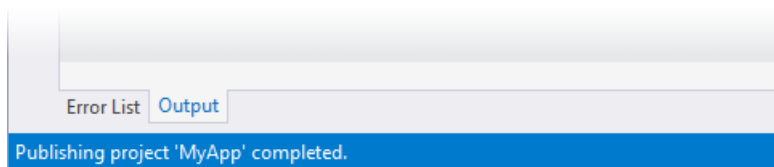
接下来，在出现的对话框中输入签名证书密码。输入密码，然后单击“确定”：



存档管理器会显示上传进度：

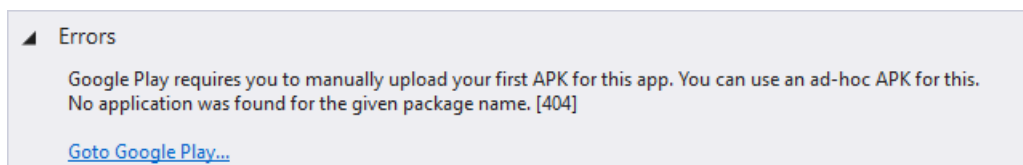


上传完成时，Visual Studio 左下角会显示完成状态：

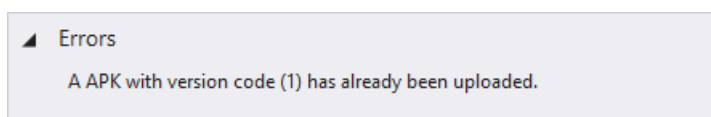


疑难解答

请注意，使用发布到 **Google Play** 前，必须已经向 Google Play 应用商店提交了 1 个 APK。如果尚未上传 APK，则“发布向导”会在“错误”窗格中显示如下错误：



出现此错误时，请通过 Google Play 开发者控制台手动上传 APK (例如 Ad-Hoc 构建版本)，并使用“分发渠道”对话框进行后续 APK 更新。有关详细信息，请参阅[手动上传 APK](#)。每次上传时必须更改 APK 版本代码，否则会出现如下错误：



若要解决此错误，请使用不同的版本号重新生成应用，然后通过“分发渠道”对话框重新提交到 Google Play。

Google 授权服务

2018/10/26 • [Edit Online](#)

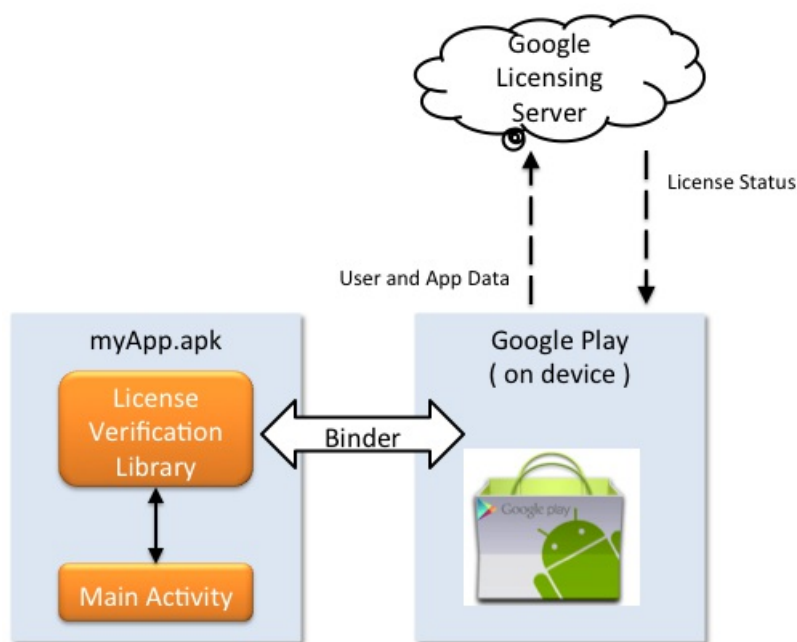
在 Google Play 推出之前, Android 应用程序依靠 Google Market 提供的旧版复制保护功能, 确保只有经过授权的用户才可在其设备上运行应用程序。复制保护机制具有局限, 是不太理想的应用程序保护解决方案。

Google 授权取代了这种旧的复制保护机制。Google 授权是一款灵活安全且基于网络的服务, Android 应用程序可查询该服务, 确定某应用程序是否可在给定设备上运行。

Google 授权非常灵活, Android 应用程序可完全掌控何时检查许可证、多久检查许可证一次以及如何处理来自授权服务器的响应。

Google 授权安全可靠, 每个响应均使用 RSA 键对进行签名, 且该键对在 Google Play 服务器和应用程序之间专属共享。Google Play 为开发人员提供了公钥, 该公钥嵌入到 Android 应用程序中且可用于验证响应的身份。Google Play 服务器将内部保存私钥。

采用了 Google 授权的应用程序将请求使用设备上 Google Play 托管的服务。然后, Google Play 将此请求发送到 Google 授权服务器, 后者使用许可状态进行响应:



上图描述了此 workflow:

- 应用程序会提供包名称、用于验证服务器响应的 *nonce* (加密验证器), 并提供可异步处理响应的回调。
- Google Play 会提供 Google 帐户等信息, 并会提供设备本身的信息 (如 IMSI 编号)。

Google 授权服务还是 APK 扩展文件的关键组件 (将在本文档后面讨论)。APK 扩展文件利用 Google 授权服务获取要下载的扩展文件的 URL。

要求

只有通过 Google Play 购买的应用程序才可享受 Google 授权服务的权益。即使设备上未安装 Google Play, 使用授权服务的应用程序仍将在该设备上正常运行。

Google Play 需连接 Internet 才可正常使用。应用程序可缓存许可证, 应对设备无权访问 Google Play 授权服务器的情况。

仅当使用 APK 扩展文件时，免费应用程序才需要 Google 授权。

APK 扩展文件

2018/10/26 • [Edit Online](#)

某些应用程序(例如一些游戏)需要的资源和资产超出了 Google Play 规定的最大 Android 应用大小限制。此限制取决于 APK 所适用的 Android 版本:

- 适用于 Android 4.0 或更高版本(API 级别 14 或更高)的 APK 的限制为 100MB。
- 适用于 Android 3.2 或更低版本(API 级别 13 或更高)的 APK 的限制为 50MB。

若要克服此限制, Google Play 将承载和分发 APK 随附的两个扩展文件, 使应用程序可直接超过此限制。

大多数设备上, 下载应用程序后, 扩展文件会随 APK 一并下载, 并保存到设备上的共享存储位置(SD 卡或可安装 USB 的分区)。在少数旧版设备上, 扩展文件可能不会随 APK 一并自动安装。这些情况下, 应用程序有必要包含用户首次运行应用程序时要下载扩展文件的代码。

扩展文件会被视为不透明二进制 blob (obb), 其大小最大为 2GB。这些文件下载后, Android 不会对其执行任何特殊处理 – 这些文件可以采用适合相应应用程序的任何格式。从概念上讲, 推荐的扩展文件方式应如下所示:

- 主扩展 – 此文件是不适合 APK 大小限制的资源和资产的主扩展文件。主扩展文件应包含应用程序所需的主资产, 并且应很少进行更新。
- 修补扩展 – 这适用于对主扩展文件进行少量更新。此文件可更新。应用程序负责从此文件执行任何必要的修补或更新。

上传 APK 的同时必须上传相应扩展文件。Google Play 不允许向现有 APK 上传扩展文件或者不允许上传现有 APK。如果有必要更新扩展文件, 则必须上传新的 APK, 同时更新 `versionCode`。

扩展文件存储

文件下载到设备后, 会存储在 `shared-store/Android/obb/package-name` 中:

- `shared-store` – 这是 `Android.OS.Environment.getExternalStorageDirectory` 指定的目录。
- `package-name` – 这是应用程序 Java 样式的包名称。

下载完成后, 不应移动、更改、重命名扩展文件或从设备上的位置中删除扩展文件。否则会导致再次下载扩展文件, 这样会删除旧文件。此外, 扩展文件目录应仅包含扩展包文件。

扩展文件未对内容提供任何安全保护 – 其他应用程序或用户可访问存储在共享存储上的任何文件。

如果需要解压缩扩展文件, 则解压缩文件应存储在单独的目录中, 例如

`Android.OS.Environment.getExternalStorageDirectory` 中的一个目录。

从扩展文件提取文件的另一方法是直接从扩展文件读取资产或资源。扩展文件只是可通过合适的 `ContentProvider` 进行使用的 zip 文件。[Android.Play.ExpansionLibrary](#) 包含的程序集 `System.IO.Compression.Zip` 中包括了一个允许对某些媒体文件直接进行文件访问的 `ContentProvider`。如果将媒体文件打包为 zip 文件, 媒体播放调用可能直接使用 zip 中的文件而无需解压缩 zip 文件。添加到 zip 文件时不应压缩媒体文件。

文件名格式

下载扩展文件后, Google Play 会使用下面的方案来对扩展进行命名:

```
[main|patch].<expansion-version>.<package-name>.obb
```

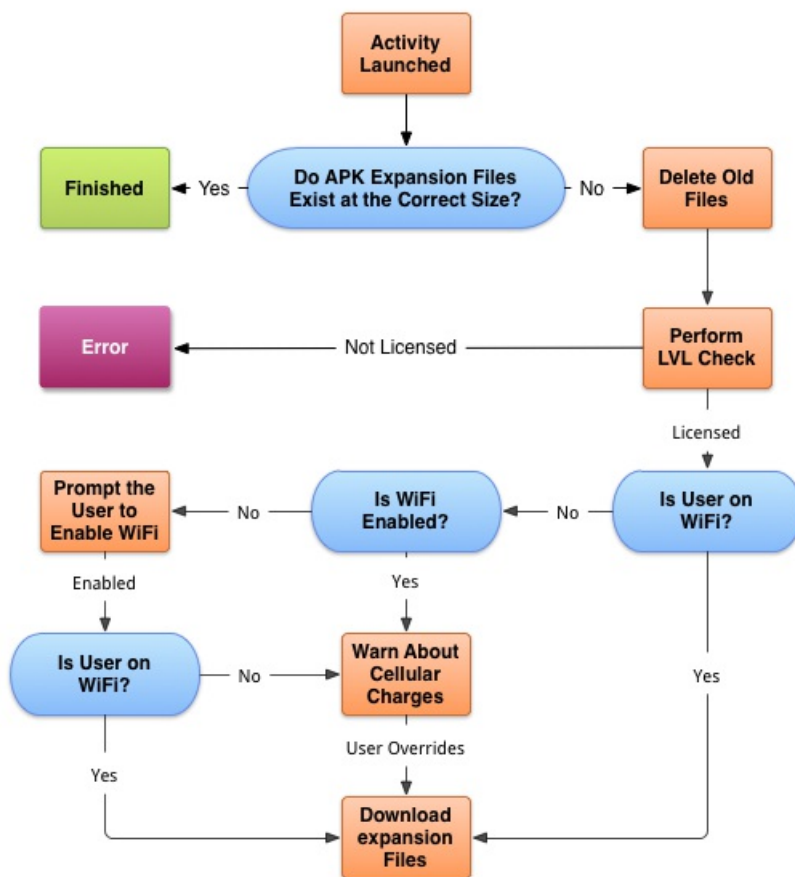
此方案的三个组件包括:

- `main` 或 `patch` – 这会指定是主扩展文件还是修补扩展文件。二者只能选其一。
- `<expansion-version>` – 这是一个整数，该整数与文件首次关联的 APK 的 `versionCode` 匹配。
- `<package-name>` – 这是应用程序的 Java 样式包名称。

例如，如果 APK 版本是 21，包名称是 `mono.samples.helloworld`，则主扩展文件应命名为 `main.21.mono.samples.helloworld`。

下载过程

从 Google Play 安装应用程序时，扩展文件会随 APK 一并下载和保存。在某些情况下可能有所例外，或者扩展文件可能会被删除。为处理这种情况，应用需检查扩展文件是否存在，然后根据需要进行下载。下方流程图显示了此过程的推荐流程：



应用程序启动时，应检查当前设备上是否存在合适的扩展文件。如果不存在，则应用程序必须在 Google Play 的[应用程序授权](#)中作出请求。使用许可验证库 (LVL) 执行该检查，并且免费或许可应用程序都必须执行该检查。LVL 主要由付费应用程序用于实施许可证限制。但是，Google 已扩展 LVL，使其也可用于扩展库。免费应用程序必须执行 LVL 检查，但可以忽略许可限制。LVL 请求负责提供以下有关应用程序所需扩展文件的信息：

- 文件大小 – 扩展文件的文件大小作为检查的一部分，可用于确定是否已经下载正确的扩展文件。
- 文件名 – 这是扩展包必须保存到其下的文件名（当前设备上）。
- 下载 URL – 用于下载扩展包的 URL。每次下载所用的 URL 都是唯一的，提供之后很快会过期。

执行 LVL 检查后，应用程序应会下载扩展文件，对于下载过程，请考虑到以下几点：

- 设备可能没有足够的空间来存储扩展文件。
- 如果 Wi-Fi 不可用，应允许用户暂停或取消下载，以避免产生不需要的数据费用。
- 在后台下载扩展文件以避免阻止用户交互。
- 在后台进行下载时，应显示进度指示。
- 下载期间出现的错误可轻松进行处理和恢复。

体系结构概述

主活动启动时，会检查是否已下载了扩展文件。如果已下载文件，则必须检查其有效性。

如果未下载扩展文件，或者当前文件无效，则必须下载新的扩展文件。创建绑定服务作为应用程序的一部分。应用程序主活动启动时，会使用绑定服务根据 Google 授权服务执行检查，以找出要下载文件的文件名和 URL。绑定服务然后会在后台线程下载文件。

为减轻将扩展文件集成到应用程序所需的工作量，Google 在 Java 中创建了几个库。这些库包括：

- 下载程序库 – 该库可减少将扩展文件集成到应用程序所需的工作量。该库会在后台服务下载扩展文件、显示用户通知、处理网络连接问题、恢复下载以及执行其他任务。
- 许可验证库 (LVL) – 该库用于执行和处理对应用程序授权服务的调用。也可用于执行授权检查，以检查应用程序是否经授权在设备上使用。
- APK 扩展 Zip 库(可选)– 如果扩展文件在 zip 文件中，则该库会作为内容提供程序，允许应用程序直接从 zip 文件读取资源和资产，而无需展开 zip 文件。

这些库已经移植到 C#，可用于 Apache 2.0 许可证。若要将扩展文件快速集成到现有应用程序，可将这些库添加到现有 Xamarin.Android 应用程序。GitHub 上的 [Android.Play.ExpansionLibrary](#) 中提供有相应代码。

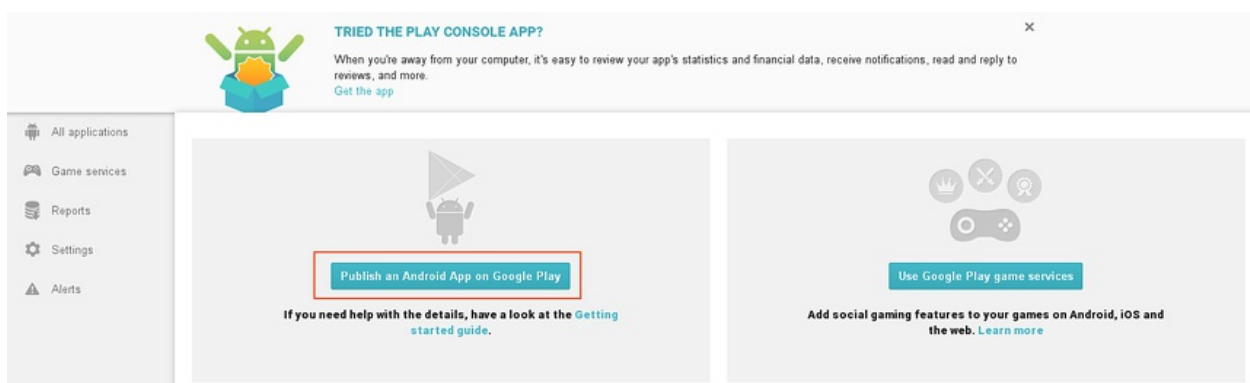
手动上传 APK

2018/10/26 • [Edit Online](#)

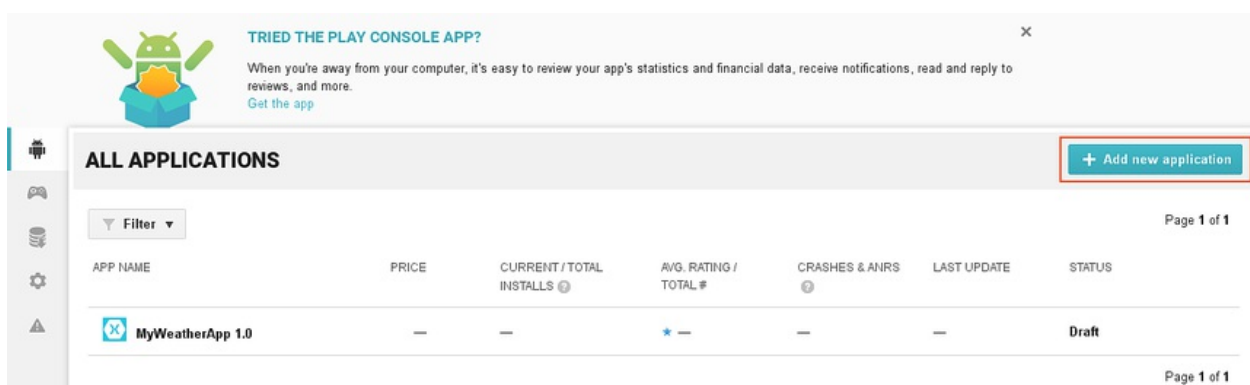
第一次将 APK 提交到 Google Play (或如果使用 Xamarin.Android 的早期版本) 时, 必须通过 [Google Play 开发者控制台](#) 手动上传 APK。本指南介绍此过程所需的步骤。

Google Play 开发者控制台

已编译 APK 并准备好促销资产后, 必须将该应用程序上传到 Google Play。通过登录到 [Google Play 开发者控制台](#) 完成此操作, 如图所示。单击“在 Google Play 上发布 Android 应用”按钮, 启动分发应用程序的进程。



如果某个现有应用已注册了 Google Play, 请单击“添加新应用程序”按钮:



当显示“添加新应用程序”对话框时, 输入应用名称并单击“上传 APK”:

ADD NEW APPLICATION

Default language *

English (United States) – en-US ▼

Title *

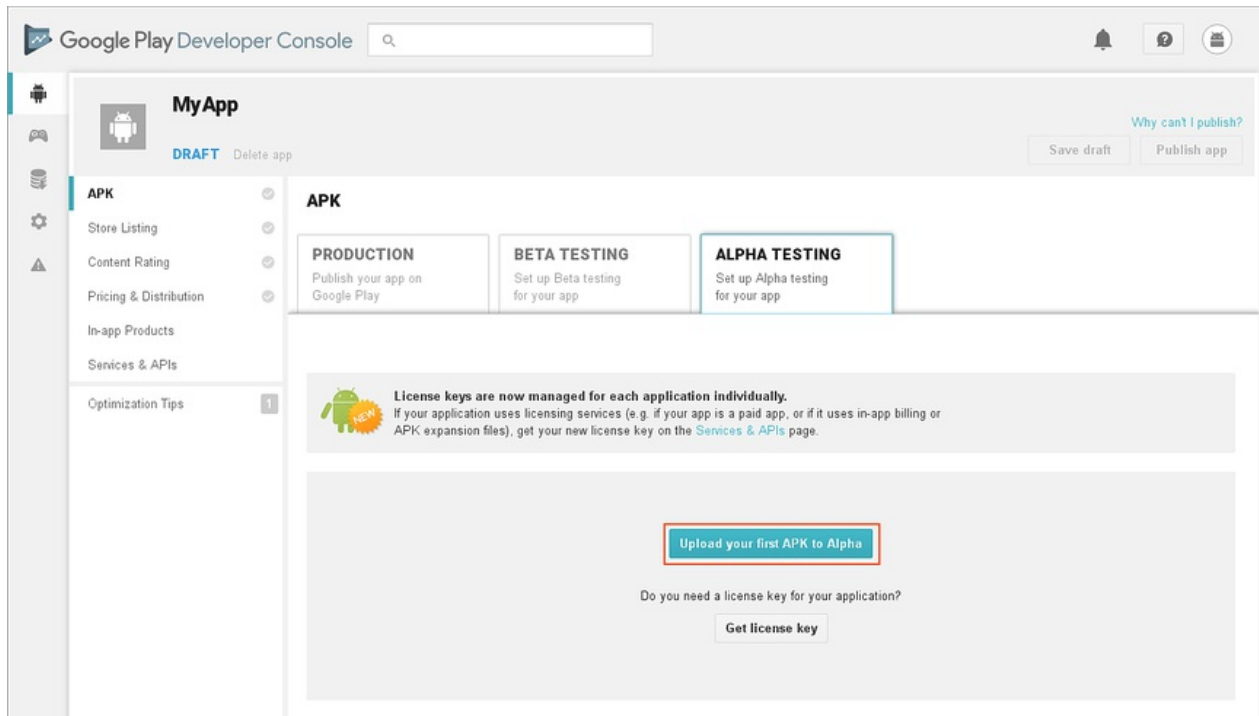
MyApp

5 of 30 characters

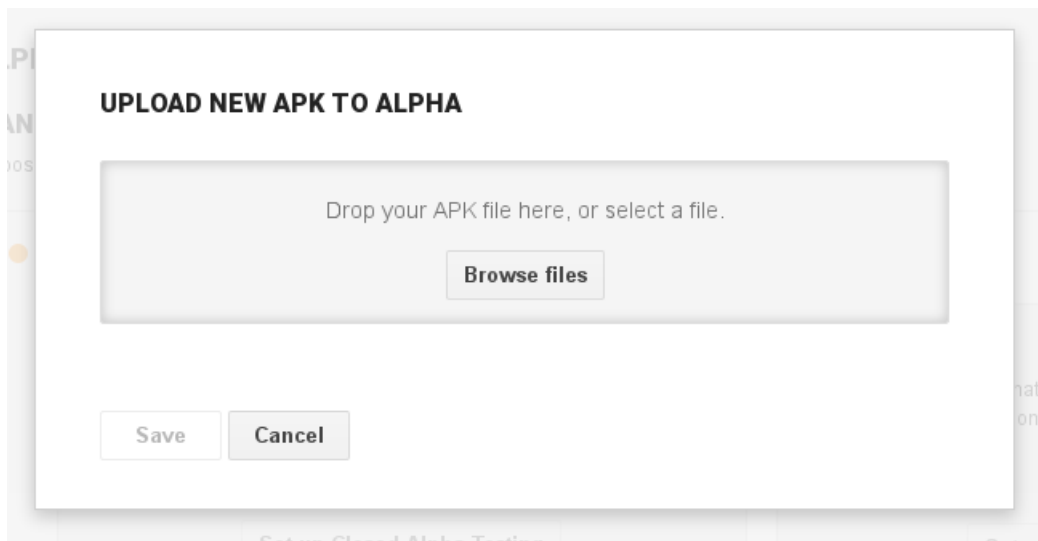
What would you like to start with?

[Upload APK](#) [Prepare Store Listing](#) [Cancel](#)

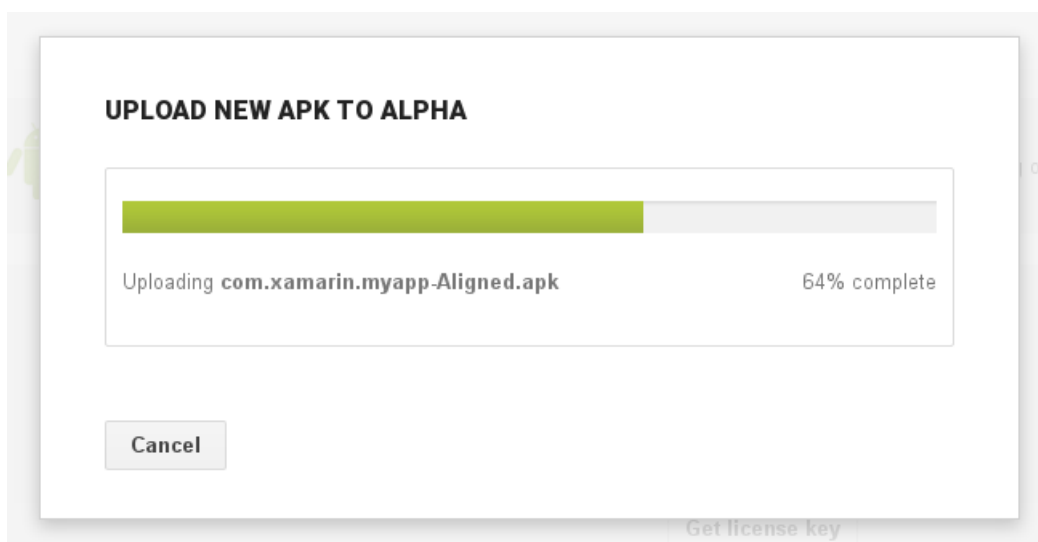
显示如下屏幕时，可发布应用以进行 alpha 测试、Beta 测试或生产。下面的示例选择了“ALPHA 测试”选项卡。由于“MyApp”不使用授权服务，因此在本示例中，不需要单击“获取许可证密钥”按钮。在此处，单击“将第一个 APK 上传到 Alpha”按钮以发布到 Alpha 通道：



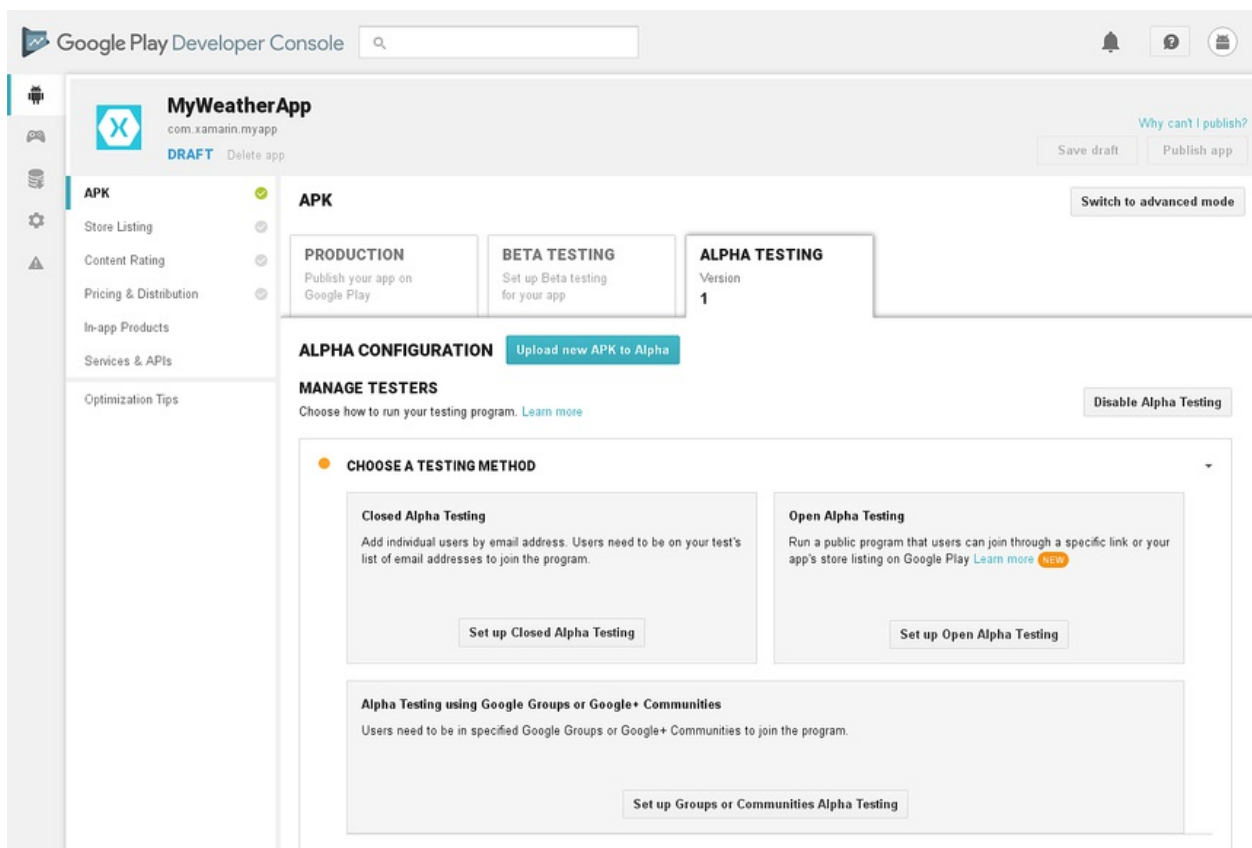
随即显示“将新 APK 上传到 ALPHA”对话框。可通过单击“浏览文件”按钮或拖放 APK 来上传 APK：



请务必上传已准备好进行发布的待分发 APK。下一个对话框指示 APK 上传的进度：



上传 APK 后，可以选择测试方法：



有关应用测试的详细信息，请参阅 Google 的[设置 alpha/beta 测试指南](#)。

上传 APK 后，会将它保存为草稿。如下文所述，向 Google Play 提供更多详细信息之后才能将其发布。

应用商店一览

单击“Google Play 开发者控制台”中的“应用商店一览”，输入 Google Play 将向应用程序的潜在用户显示的信息：

MyApp

com.xamarin.myapplication

DRAFTDelete app

Why can't I publish?

Save draftPublish app

APK

Store Listing

Content Rating

Pricing & Distribution

In-app Products

Services & APIs

Optimization Tips

STORE LISTING

PRODUCT DETAILS

Fields marked with * need to be filled before publishing.

English (United States) – en-USManage translations

Title*English (United States) – en-USMy Awesome App14 of 30 characters

Short description*English (United States) – en-USThis is a Hello World type application developed using Xamarin.Android70 of 80 characters

Full description *English (United States) – en-US

图形资产

向下滚动到“应用商店一览”页的“图形资产”部分：

GRAPHIC ASSETS

If you haven't added localized graphics for each language, graphics for your default language will be used.
[Learn more about graphic assets.](#)

Screenshots *

Default – English (United States) – en-US
JPEG or 24-bit PNG (no alpha). Min length for any side: 320px. Max length for any side: 3840px.
At least 2 screenshots are required overall. Max 8 screenshots per type. Drag to reorder or to move between types.

For your app to be showcased in the 'Designed for tablets' list in the Play Store, you need to upload at least one 7-inch and one 10-inch screenshot. If you previously uploaded screenshots, make sure to move them into the right area below.
[Learn how tablet screenshots will be displayed in the store listing.](#)

Please check out our [Impersonation and Intellectual Property policy](#) to avoid common violations.

Phone

Tablet

Android TV

Android Wear

+

Add screenshot

Drop image here.

在本部分中上传前面已准备好的所有促销资产。提供有关必须提供哪些促销资产，以及应以何种格式提供它们的指导。

分类

“图形资产”部分之后是“分类”部分，选择应用程序类型和类别：

CATEGORIZATION

Application type *Select an application type

Category *Select a category

Content rating *Select a content rating

[Learn more about content rating.](#)

New content rating *You need to fill a rating questionnaire and apply a [content rating](#).

下一部分介绍内容分级。

联系人详细信息

此页的最后一个部分是“联系人详细信息”。此部分用于收集有关应用程序的开发者的联系信息：

CONTACT DETAILS

| | |
|---------|--|
| Website | <input type="text" value="https://github.com/xamarin"/> |
| Email * | <input type="text" value="mm@xamarin.com"/> <small>Please provide an email address where you may be contacted. This address will be publicly displayed with your app.</small> |
| Phone | <input type="text"/> |

PRIVACY POLICY *

If you wish to provide a privacy policy URL for this application, please enter it below. Also, please check out our [User Data policy](#) to avoid common violations.

| | |
|----------------|--|
| Privacy Policy | <input type="text" value="http://..."/> <input type="checkbox"/> Not submitting a privacy policy URL at this time. Learn more |
|----------------|--|

可以在“隐私策略”部分提供应用隐私策略的 URL，如上所示。

内容分级

单击“Google Play 开发者控制台”中的“内容分级”。在此页中指定应用的内容分级。Google Play 要求所有应用程序都指定内容分级。单击“继续”按钮完成内容分级问卷：

CONTENT RATING

The Google Play content rating system for apps and games is designed to deliver reputable, locally relevant ratings to users around the world. The rating system includes official ratings from the International Age Rating Coalition (IARC) and its participating bodies.

Developer responsibilities:

- Complete the content rating questionnaire for each new app submitted to Developer Console, for all existing apps that are active on Google Play, and for all app updates where there has been a change to app content or features that would affect the responses to the questionnaire.
- Provide accurate responses to the content rating questionnaire. Misrepresentation of your app's content may result in removal or suspension.

Your rating will be used to:

- Inform consumers about the age appropriateness of your app.
- Block or filter your content in certain territories or to specific users where legally required.
- Evaluate your app's eligibility for special developer programs.

The content rating questionnaire and the new Content Ratings Guidelines are a condition of your participation in the Google Play store under the Developer Distribution Agreement. [Learn more](#)

Continue



Google Play 上的所有应用程序必须根据 Google Play 分级系统进行分级。除内容分级以外，所有应用程序还必须遵守 Google 的[开发者内容政策](#)。

下表列出了 Google Play 分级系统中的四个级别，并提供有关可能要求或强制分级的功能或内容的一些指导：

- **任何人** – 不可访问、发布或共享位置数据。不可承载用户生成的任何内容。不可实现用户之间的通信。
- **低成熟度** – 访问但不共享位置数据的应用程序。轻度暴力或卡通级暴力描述。
- **中等成熟度** – 涉及毒品、酒精或烟草。赌博主题或模拟赌博。煽动性内容。不敬或粗俗幽默内容。性暗示内容或性相关内容。激烈的幻想暴力。现实的暴力内容。允许用户相互查找。允许用户相互通信。用户位置数据共享。
- **高成熟度** – 侧重酒精、烟草或毒品消费或销售。侧重于性暗示或性相关的内容。图像暴力。

中等成熟度列表中的项的判断具有主观性，这样的话，有可能根据某个准则判断为中等成熟度级别的内容也可能有充分的合理性被判断为高成熟度级别。


定价和分发


单击“Google Play 开发者控制台”中的“定价和分发”。如果应用是付费应用，则在此页中设置价格。或者，可以将应用程序免费分发给所有用户。一旦将应用程序指定为免费，则必须始终保持免费。Google Play 不允许将免费应用程序更改为付费应用(但是，可以使用应用内计费销售免费应用的内容)。Google Play 允许随时将付费应用


更改为免费应用。


在发布付费应用之前，需要有商家帐户。若需要帐户，请单击“设置商家帐户”并按说明进行操作。


PRICING & DISTRIBUTION



Designed for Families


Google Play for Education


Google Play for Work


Android Wear


Android TV


Android Auto

This application is

Paid

Free

To publish paid applications, you need to

set up a merchant account.

[Learn more](#)

管理国家/地区

下一部分，“管理国家/地区”支持对应用可能会分发到哪些国家/地区进行控制：

Countries *

You have not selected any countries.

Manage countries

☐ SELECT ALL COUNTRIES

☐ Albania

☐ Algeria

☐ Angola

☐ Antigua and Barbuda

其他信息

继续向下滚动来指定应用是否包含广告。此外，“设备类别”部分提供分发适用于 Android Wear、Android TV 或 Android Auto 的应用的选项(可选)：

CONTAINS ADS *

Does your application have ads? Also, please check out our [Ads policy](#) to avoid common violations. If yes, users will be able to see the 'ads' label on your application in the Play Store. [Learn more](#)

☒ Yes, it has ads

☐ No, it has no ads

DEVICE CATEGORIES

Android Wear

☐ Distribute your app on Android Wear.
Extend your app to wearables with Android Wear. To submit your app for review, you need to add an Android Wear screenshot on your app's [Store listing](#) page.
To learn more, read the Android Wear [documentation](#) and [distribution guidelines](#).

Android TV

Reimagine your app for the biggest screen in the house with Android TV. To submit your app for review, you need to include a [Leanback launcher intent](#) in your app.
To learn more, read the Android TV [documentation](#) and [distribution guidelines](#).

Android Auto

Bring your app to cars with Android Auto. To submit your app for review, you need to accept the Android Auto [terms and conditions](#).
To learn more, read the Android Auto [documentation](#) and [distribution guidelines](#).

此部分之后是可以选择的其他选项，如选择“家庭专用”和通过 Google Play for Education 分发应用。

许可

“定价和分发”页面的底部是“许可”部分。这是必需的部分，用于声明应用程序满足 [Android 内容准则](#)，并确认应用程序受美国出口法律显示：

CONSENT

Marketing opt-out

- ☐ Do not promote my application except in Google Play and in any Google-owned online or mobile properties. I understand that any changes to this preference may take sixty days to take effect.

Content guidelines *

- ☐ This application meets [Android Content Guidelines](#).

Please check out these [tips on how to create policy compliant app descriptions](#) to avoid some common reasons for app suspension. If your app or store listing is [eligible for advance notice](#) to the Google Play App Review team, [contact us](#) prior to publishing.

US export laws *

- ☐ I acknowledge that my software application may be subject to United States export laws, regardless of my location or nationality. I agree that I have complied with all such laws, including any requirements for software with encryption functions. I hereby certify that my application is authorized for export from the United States under these laws. [Learn more](#)

关于发布 Xamarin.Android 应用的信息远远不止本指南中所包含的内容。有关在 Google Play 中发布应用的详细信息，请参阅[欢迎使用 Google Play 开发者控制台帮助中心](#)。

Google Play 筛选器

当用户浏览 Google Play 网站的应用程序时，他们能够搜索所有已发布的应用程序。用户从 Android 设备浏览 Google Play 时，搜索结果会略有不同。结果将会根据与正在使用的设备的兼容性进行筛选。例如，如果某个应用程序必须发送 SMS 消息，则 Google Play 不会向不能发送 SMS 消息的任何设备显示该应用程序。可通过以下项创建应用于搜索的筛选器：

1. 设备的硬件配置。
2. 应用程序清单文件中的声明。
3. 使用的载波(如果有)。
4. 设备的位置。

可以向应用的清单添加元素，以帮助控制在 Google Play 商店中筛选应用的方式。下表列出了可用于筛选应用程序的清单元素和特性：

- [supports-screen](#) – Google Play 使用此属性，根据屏幕大小确定应用程序是否可部署到设备中。Google Play 假定 Android 可将较小布局调整为较大屏幕，但反之不成立。因此，声明支持标准屏幕的应用程序会在搜索较大屏幕而非较小屏幕时的搜索结果中出现。如果 Xamarin.Android 应用程序在清单文件中不提供 `<supports-screen>` 元素，Google Play 将假定所有属性均具有值 `true`，并且该应用程序支持所有屏幕大小。必须将此元素手动添加到 **AndroidManifest.xml**。
- [uses-configuration](#) – 此清单元素用于请求某些硬件功能，例如键盘类型、导航设备和触摸屏等。必须将此元素手动添加到 ****AndroidManifest.xml**。
- [uses-feature](#) – 此清单元素声明为确保应用程序正常运行，设备所必需的硬件或软件功能。此特性仅提供信息。Google Play 不会向设备显示不符合此筛选条件的应用程序。仍可通过其他方式(手动或下载)安装该应用程序。必须将此元素手动添加到 **AndroidManifest.xml**。
- [uses-library](#) – 此元素指定设备上必须存在的某些共享库(如 Google Maps)。还可使用 `Android.App.UsesLibraryAttribute` 指定此元素。例如：

```
[assembly: UsesLibrary("com.google.android.maps", true)]
```

- [uses-permission](#) – 此元素用于推断应用程序正常运行所必需的某些硬件功能可能未通过 `<uses-feature>` 元素正确声明。例如，如果应用程序请求照相机使用权限，则 Google Play 会假定设备必定装有摄像头，即使没有任何声明摄像头的 `<uses-feature>` 元素。此元素可通过 `Android.App.UsesPermissionsAttribute` 进行设置。例如：

```
[assembly: UsesPermission(Manifest.Permission.Camera)]
```

- [uses-sdk](#) – 该元素用于声明应用程序所需的最低 Android API 级别。此元素可在 Xamarin.Android 项目的 Xamarin.Android 选项中进行设置。
- [compatible-screens](#) – 此元素用于筛选与其指定的屏幕大小和密度不匹配的应用程序。大多数应用程序不应使用此筛选器。此筛选器专用于特定的高性能游戏或需要严格控制应用程序分发的应用程序。上面提到的 `<support-screen>` 特性是首选特性。
- [supports-gl-texture](#) – 此元素用于声明应用程序所需的 GL 纹理压缩构造。大多数应用程序不应使用此筛选器。此筛选器专用于特定的高性能游戏或需要严格控制应用程序分发的应用程序。

有关配置应用清单的详细信息，请参阅 Android [应用清单](#) 主题。

发布到 Amazon 应用商店

2018/10/26 • [Edit Online](#)

通过 Amazon 移动应用分发程序, 移动应用开发人员可在 Amazon 上发布应用程序。本部分简要介绍用于 Android 的 Amazon 应用商店。

23°



17:57

amazon appstore
for Android

Search for Apps



Today's Free App of the Day

Featured New Re



Camping Checklist

Jimbl Software Labs

★★★★★ (88)

\$0.99 FREE



Top

New

Games

Entertain ▶

Top Paid

Top Free

1. Cut the
Rope

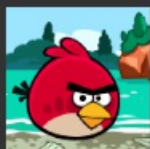
★★★★★

\$0.99

2. Where's My
Water?

★★★★★

\$0.99

3. Angry Birds
Seasons (Ad...

★★★★★

\$0.99

4. Angry Birds
Rio (Ad-Free)

★★★★★

FREE

1. Camping
Checklist

★★★★★

FREE



2. Temple Run

★★★★★

FREE

3. Scramble
With Friends...

★★★★★

FREE

4. Bubble
Birds

★★★★★

FREE



Amazon 不限制 APK 的大小。但是, 如果 APK 大于 30 MB, 则会使用 FTP 进行分发, 而不是 Amazon 移动应用分发门户。

提交应用:二进制信息

将应用程序提交到 Amazon 应用商店的过程与将应用程序提交到 Google Play 的过程类似。通过 Amazon 分发的应用程序需要以下资产:

- **图标** – 这是一个带有透明背景的 114 x 114 .png 文件。这是必需项。
- **缩略图** – 这是上述图标的更大版本。其像素为 512 x 512, 透明背景。此图标也是必需项。
- **屏幕截图** – Amazon 需要至少 3 个、最多 10 个屏幕截图。屏幕截图必须为 1024w x 600h 像素或 800w x 480h 像素。支持 .png 和 .jpg 两种格式。
- **促销图像** – 为在促销位置(如主页)上推出应用程序, 可能需要提交促销图像(可选)。该图像应为横向放置的 .png 或 .jpg 文件, 像素为 1024w x 500h。可能没有任何动画。
- 可能提供 5 个视频的更新内容。

审批过程

应用程序提交后, 需经过审批过程。Amazon 将审查应用程序, 确保其具有产品说明所述的性能, 不会将客户数据置于危险之中, 且不会影响设备操作。审批过程完成后, Amazon 将发出通知并分发应用程序。

独立发布

2018/10/26 • [Edit Online](#)

可在不使用任何现有 Android 市场的情况下发布应用程序。本部分将介绍其他发布方法和 Xamarin.Android 的许可级别。

Xamarin 许可

可使用 4 种许可证来开发、部署和分发 Xamarin.Android 应用：

- **Visual Studio Community** – 面向使用 Windows 的学生、小型团队和 OSS 开发者。
- **Visual Studio Professional** – 面向独立开发者或小型团队(仅限 Windows)。此许可证提供标准或云订阅，支持访问其他 Xamarin University 内容，且无使用限制。
- **Visual Studio Enterprise** – 面向任何规模的团队(仅限 Windows)。此许可包括企业功能(标准或云订阅)。

若要下载社区版或深入了解如何购买 Professional 版和 Enterprise 版，请访问 visualstudio.com。

允许来自未知源的安装

默认情况下，Android 仅允许用户下载和安装来自 Google Play 的应用程序。若要允许来自非应用商店的安装，用户必须先设备上启用未知源设置，才可尝试安装应用程序。可通过“设置”>“安全”找到此设置，如下图所示：



Security

Set up SIM card lock

PASSWORDS

Make passwords visible



DEVICE ADMINISTRATION

Device administrators

View or deactivate device administrators

Unknown sources

Allow installation of non-Market apps



CREDENTIAL STORAGE

Trusted credentials

Display trusted CA certificates

Install from storage

Install certificates from storage

Clear credentials

Remove all certificates

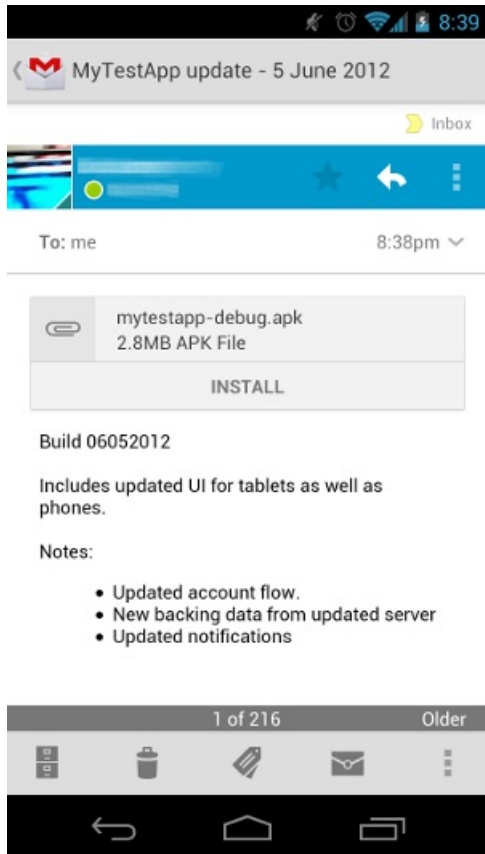


IMPORTANT

无论是否启用此设置，某些网络提供商可能都会阻止安装来自未知源的应用程序。

通过电子邮件发布

若要向用户分发应用程序，一种简单快捷的方式是将发布 APK 附加到电子邮件。用户在 Android 设备上打开电子邮件时，Android 将识别 APK 附件并显示“安装”按钮，如下图所示：



尽管通过电子邮件进行分发非常简单，但其在防止盗版或未授权分发方面提供的保护较少。因此，最好在很少用户接收该应用程序且其肯定不会分发应用程序的情况下采用此方式。

通过 Web 发布

可通过 Web 服务器分发应用程序。为此，需将应用程序上传到 Web 服务器，然后向用户提供下载链接。Android 设备浏览到链接并下载应用程序时，该程序将在下载完成后自动安装。

手动安装 APK

手动安装是安装应用程序的第 3 个选择。若要手动安装应用程序：

1. 向用户分发 **APK 副本** – 例如可通过 CD 或 U 盘分发此副本。
2. (用户)在 **Android 设备上安装应用程序** – 使用命令行 *Android Debug Bridge (adb)* 工具。**adb** 是一种多功能的命令行工具，支持与仿真器实例或 Android 设备进行通信。Android SDK 内附有 **adb**其位于 **/platform-tools/** 目录中。

Android 设备必须使用 USB 接口线连接到计算机。Windows 计算机可能还需要电话供应商提供额外的 USB 驱动程序，使其能够被 adb 识别。本文未介绍其他 USB 驱动程序的安装说明。

发出任何 **adb** 命令之前，有必要了解连接了哪些仿真器实例或设备(如有)。可使用 `devices` 命令查看附加对象的列表，如以下代码段所示：

```
$ adb devices
List of devices attached
0149B2EC03012005device
```

确认已连接设备后, 可通过 **adb** 发出 `install` 命令安装应用程序:

```
$ adb install <path-to-apk>
```

以下代码段演示了将应用程序安装到已连接设备:

```
$ adb install helloworld.apk
3772 KB/s (3013594 bytes in 0.780s)
  pkg: /data/local/tmp/helloworld.apk
Success
```

如已安装应用程序, 则 `adb install` 无法安装 APK 并会报告错误, 如下例所示:

```
$ adb install helloworld.apk
4037 KB/s (3013594 bytes in 0.728s)
  pkg: /data/local/tmp/helloworld.apk
Failure [INSTALL_FAILED_ALREADY_EXISTS]
```

此时必须从设备中卸载该应用程序。首先, 发出 `adb uninstall` 命令:

```
adb uninstall <package_name>
```

以下代码段是一个卸载应用程序的示例:

```
$ adb uninstall mono.samples.helloworld
Success
```

将 Xamarin.Android 安装为系统应用

2018/11/2 • [Edit Online](#)

本指南将讨论系统应用和用户应用之间的区别，以及如何将 Xamarin.Android 应用作为系统应用安装。本指南适用于自定义 Android ROM 映像的作者。它将说明如何创建自定义 ROM。

系统应用

自定义 Android ROM 映像的作者或 Android 设备制造商在分发 ROM 或设备时，可能希望将 Xamarin.Android 应用包含为一个系统应用。系统应用是被认为对设备运行非常重要或提供自定义 ROM 作者始终希望可用的功能的一款应用。

系统应用安装在 `/system/app/` 文件夹中（文件系统的一个只读目录），用户无法将其删除或移动，除非该用户具有根目录访问权限。与此相反，由用户安装的应用程序（通常是从 Google Play 或通过旁加载应用进行安装）称为“用户应用”。用户应用可以被用户删除，并且在许多情况下可以移动到设备上的其他位置（如某类外部存储）。

系统应用的行为与用户应用的完全一致，但具有以下值得注意的例外情况：

- 系统应用可以像正常用户应用一样进行升级。但是，因为应用副本一直存在于 `/system/app/` 中，该应用始终可以回退到原始版本。
- 系统应用可能被授予对用户应用不可用的某些仅限系统的权限。仅限系统权限的一个示例是 `BLUETOOTH_PRIVILEGED`，它允许应用程序与蓝牙设备配对，而无需与任何用户进行交互。

可以将 Xamarin.Android 应用作为系统应用程序分发。除了向自定义 ROM 提供 APK，还有两个共享库，`libmonodroid.so` 和 `libmonosgen 2.0.so`，必须将其手动从 APK 复制到 ROM 映像的文件系统。本指南将说明所涉及的步骤。

限制

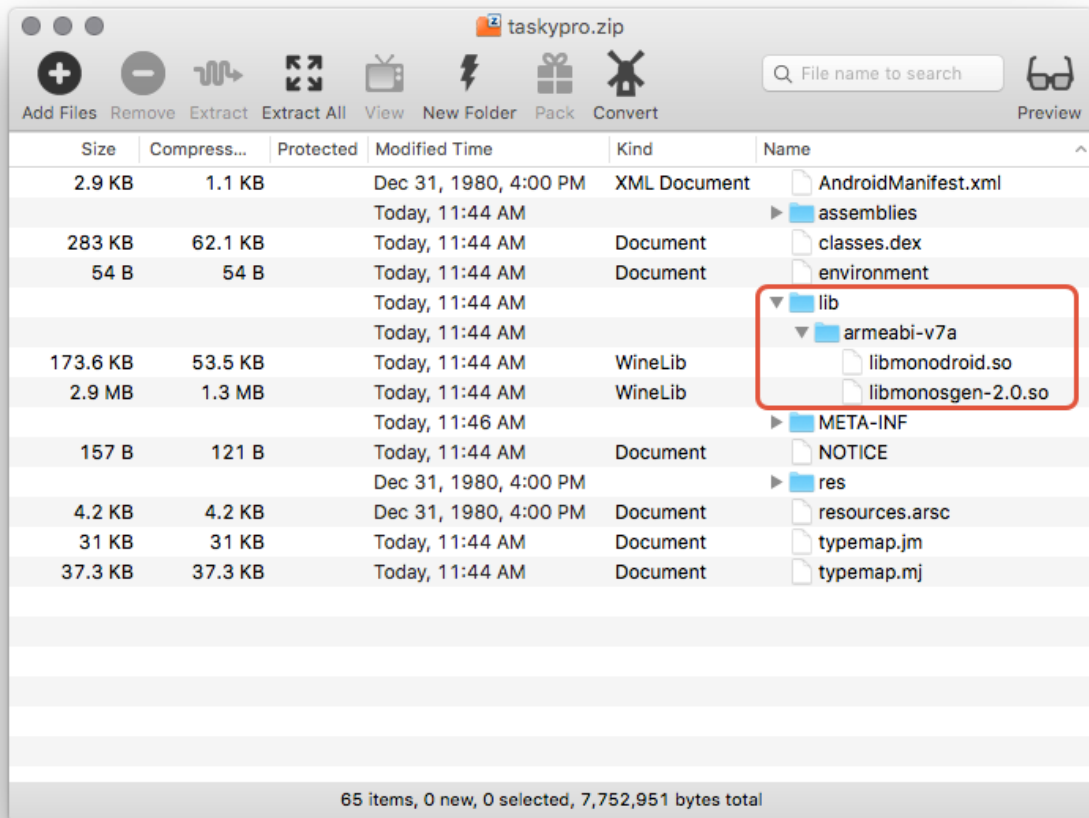
本指南适用于自定义 Android ROM 映像的作者。它将说明如何创建自定义 ROM。

本指南假定你熟悉[打包适用于 Xamarin.Android 的发布 APK](#)，并了解 Android 应用程序的[CPU 体系结构](#)。

将 Xamarin.Android 应用安装为系统应用

以下步骤介绍如何将 Xamarin.Android 应用作为系统应用进行安装。

1. 打包 Xamarin.Android 应用的发布 APK – [发布应用程序](#) 指南对此进行了详述。
2. 从 APK 提取共享库 – 使用任何 ZIP 实用程序，打开 APK 文件并检查 `/lib/` 文件夹的内容。此文件夹的每个应用程序二进制接口 (ABI) 都将有一个受该应用程序支持的子目录；此文件夹内容将包含该特定 ABI 上应用程序所需的所有共享库：



在上面的屏幕截图中，只有一个支持的 ABI (armeabi-v7a) 持有两个应用所需的 .so 文件。请注意，它只用来提取适用于设备或设备 ROM 目标体系结构的 ABI 文件，也就是说，不将 .so 文件从 x86 文件夹复制到 armeabi-v7a 设备或 ROM。

3. 将 .so 文件复制到 /system/lib – 将上一步中从 APK 提取的 .so 文件复制到自定义 ROM 上的 /system/lib/ 文件夹中。
4. 将 APK 文件复制到 /system/app – 最后一步是将 APK 文件复制到 ROM 上的 /system/app 文件夹。

总结

本指南讨论了系统应用和用户应用之间的区别，并介绍了如何将 Xamarin.Android 应用作为系统应用安装。

相关链接

- [发布应用程序](#)
- [CPU 体系结构](#)
- [BLUETOOTH_PRIVILEGED](#)
- [ABI 管理](#)

高级的概念和内部机制

2018/10/26 • [Edit Online](#)

本部分包含这些主题介绍体系结构、API 设计和 Xamarin.Android 的限制。此外，它包含的主题说明其垃圾回收实现以及可在 Xamarin.Android 中的程序集。因为 Xamarin.Android [开放源代码](#)，则还可以通过检查其源代码了解 Xamarin.Android 的内部工作机制。

体系结构

本文介绍 Xamarin.Android 应用程序的基础体系结构。它介绍了如何 Xamarin.Android 应用程序为环境中运行 Mono 执行与一起使用 Android 运行时虚拟机，并说明了此类作为 Android 可调用包装器和托管可调用包装器的重要概念。

API 设计

除了核心是 Mono 的一部分的基类库，Xamarin.Android 提供与各种 Android Api 允许开发人员使用 Mono 创建本机 Android 应用程序的绑定。

有 Xamarin.Android 的核心是互操作引擎与 Java 领域这样桥 C# 的环境和开发人员提供了访问 Java api 从 C# 或其他.NET 语言。

程序集

Xamarin.Android 附带了几个程序集。就像 Silverlight 是桌面.NET 程序集的扩展的子集，Xamarin.Android 也是多个 Silverlight 和桌面.NET 程序集的扩展的子集。

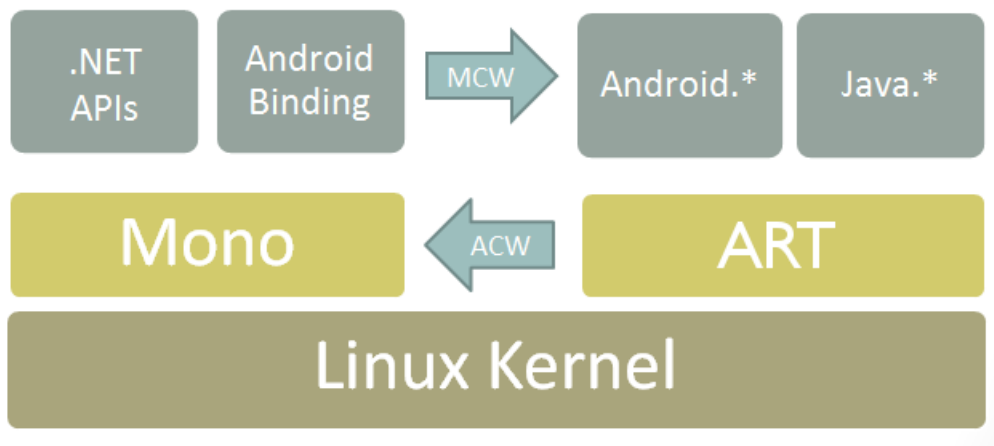
体系结构

2018/11/5 • [Edit Online](#)

Mono 的执行环境中运行 Xamarin.Android 应用程序。此执行环境运行的并行与 Android 运行时（艺术）虚拟机。这两个运行时环境在 Linux 内核顶层运行，并公开到用户代码，允许开发人员访问基础系统的各种 Api。Mono 运行时是用 C 语言编写的。

你可以使用[系统](#)，[System.IO](#)，[System.Net](#)和其余部分的.net 类库以访问基础 Linux 操作系统功能。

在 Android 上，大部分系统机制，例如音频、图形、OpenGL 和电话服务不可用直接到本机应用程序，它们仅公开通过 Android 运行时 Java Api 中的一个驻留[Java.*命名空间](#)或[Android.*命名空间](#)。体系结构是大致如下：



Xamarin.Android 开发人员访问操作系统通过调入他们知道的.NET Api（用于低级别的访问权限）或使用 Android 的 Java api 公开的架起了桥梁的命名空间中公开的类中的各种功能Android 运行时。

Android 类如何与 Android 运行时类进行通信的详细信息请参阅[API 设计文档](#)。

应用程序包

Android 应用程序包是包含 ZIP 容器 .apk文件扩展名。Xamarin.Android 应用程序包具有相同的结构和布局作为常规 Android 包，增加了以下信息：

- （包含 IL）的应用程序程序集都存储在内未压缩程序集文件夹。在过程中启动版本中的生成 .apk是mmap() ed 到过程和程序集是从内存中加载。这将允许更快的应用程序启动，因为程序集不需要在执行之前提取。 - * 注意:* 程序集位置信息，例如[Assembly.Location](#)并 [Assembly.CodeBase](#) *不能依赖* 版本中生成。为不同的文件系统条目不存在，并且必须没有可用的位置。
- 包含 Mono 运行时本机库中有 .apk 。Xamarin.Android 应用程序必须包含本机库所需目标 Android 体系结构，例如armeabi， armeabi-v7a， x86 。Xamarin.Android 应用程序不能在平台上运行，除非它包含相应的运行时库。

Xamarin.Android 应用程序还包含Android 可调用包装器为允许 Android 来调入托管代码。

Android 可调用包装器

- **Android 可调用包装器**都JNI桥用于 Android 运行时需要调用托管的代码的任何时间。Android 可调用包装器是如何虚拟方法可以重写并且可以实现 Java 接口。请参阅[Java 集成概述](#)文档的详细信息。

托管的可调用包装器

托管的可调用包装器是用于托管的代码需要调用 Android 代码，并提供对重写虚拟方法和实现 Java 接口支持的任何时间的 JNI 桥梁。整个 `Android.*` 和相关命名空间都是通过生成的托管可调用包装 `jar` 绑定。托管的可调用包装器负责托管和 Android 的类型之间进行转换和调用 JNI 通过基础的 Android 平台方法。

每个 Java 全局引用，该类可通过创建托管的可调用包装器持有 `Android.Runtime.IJavaObject.Handle` 属性。全局引用用于提供 Java 实例和托管的实例之间的映射。全局引用是一种有限的资源：仿真程序使得仅 2000 个全局引用存在一次，而大多数硬件允许超过 52,000 个全局引用，以同时存在。

若要跟踪时创建和销毁全局引用，可以设置 `debug.mono.log` 系统属性以包含 `gref`。

全局引用可以显式释放通过调用 `Java.Lang.Object.Dispose()` 上托管的可调用包装器。这将删除 Java 实例和托管的实例之间的映射，并允许收集的 Java 实例。如果从托管代码重新访问 Java 实例，将为其创建新的托管可调用包装器。

当释放的托管可调用包装器，如果该实例可以在无意中线程之间共享，以释放该实例将会影响任何其他线程的引用，因此必须小心谨慎。为了最大的安全，仅 `Dispose()` 的实例都通过已分配 `new` 或方法从在 `知道` 始终分配新实例并将可能不缓存的实例会导致意外实例线程间共享。

托管可调用包装器子类

托管的可调用包装器子类是可能所在的“有意义”的所有特定于应用程序逻辑。其中包括自定义 `Android.App.Activity` 子类（如 `Activity1` 默认项目模板中的类型）。（具体而言，这些是任何 `Java.Lang.Object` 来执行此操作的子类不包含 `RegisterAttribute` 自定义特性或 `RegisterAttribute.DoNotGenerateAcw` 是 `false`，这是默认设置。）

如管理托管的可调用包装可调用包装器子类还包含全局引用，可通过访问 `Java.Lang.Object.Handle` 属性。就像与托管的可调用包装器，通过调用全局引用可以显式释放操作 `Java.Lang.Object.Dispose()`。与托管的可调用包装器，不同 `谨慎` 应为释放的这种情况下前，采取 `dispose()` 定义是该实例将中断 Java 实例之间的映射（的实例 Android 可调用包装器）和托管的实例。

Java 激活

当 `Android 可调用包装器` (ACW) 创建从 Java，ACW 构造函数将导致相应 C# 要调用构造函数。例如，对于 ACW `MainActivity` 将包含默认构造函数将调用 `MainActivity` 的默认构造函数。（这是通过 `TypeManager.Activate()` ACW 构造函数内调用。）

没有结果的一个其他构造函数签名：`(IntPtr, JniHandleOwnership)` 构造函数。`(IntPtr, JniHandleOwnership)` 每当 Java 对象公开给托管代码和托管的可调用包装器需要构造管理 JNI 句柄调用构造函数。这通常是自动完成。

有两种情况下，`(IntPtr, JniHandleOwnership)` 上托管的可调用包装器子类必须手动提供构造函数：

1. `Android.App.Application` 子类化。`应用程序` 是特殊；默认值 `应用程序` 构造函数将 `永远不会` 调用，和 `(IntPtr, JniHandleOwnership)` **必须改为提供构造函数**。
2. 从基类构造函数的虚拟方法调用。

请注意，(2) 是一个有漏洞的抽象。在 Java 中，与 C# 中，从一个构造函数调用虚拟方法始终调用派生程度最高的方法实现。例如，`TextView(上下文, AttributeSet, int)` 构造函数调用虚拟方法 `TextView.getDefaultMovementMethod()`，该绑定为 `TextView.DefaultMovementMethod` 属性。因此，如果某种 `LogTextBox` 为 (1) 已子类 `TextView`，(2) 重写 `TextView.DefaultMovementMethod`，和 (3) 激活它的实例类的 XML，通过重写 `DefaultMovementMethod` 之前 ACW 构造函数有机会执行时，它会发生之前将调用属性 C# 机会到构造函数执行。

这通过实例化实例 `LogTextBox` 支持通过 `LogTextView(IntPtr, JniHandleOwnership)` 构造函数时 ACW `LogTextBox` 实例首次进入托管代码中，然后再调用 `(上下文, IAttributeSet, int)` `LogTextBox` 构造函数 `同一个实例` 上 ACW 构造函数执行时。

事件的发生顺序：

1. 布局 XML 加载到 [ContentView](#)。
2. Android 实例化布局的对象图，并实例化的实例 `monodroid.apidemo.LogTextBox`，为 `ACW LogTextBox`。
3. `Monodroid.apidemo.LogTextBox`构造函数执行[android.widget.TextView](#)构造函数。
4. `TextView`构造函数调用 `monodroid.apidemo.LogTextBox.getDefaultMovementMethod()`。
5. `monodroid.apidemo.LogTextBox.getDefaultMovementMethod()` 调用 `LogTextBox.n_getDefaultMovementMethod()`，这样就可调用 `TextView.n_GetDefaultMovementMethod()`，这样就可调用 `Java.Lang.Object.GetObject<TextView>`（处理 `JniHandleOwnership.DoNotTransfer`）。
6. `Java.Lang.Object.GetObject<TextView>()` 检查，以查看是否已经存在相应的 C# 实例处理。如果不存在，则返回它。在此方案中，不存在，这样 `Object.GetObject<T>()` 必须先创建一个。
7. `Object.GetObject<T>()` 寻找 `LogTextBox (IntPtr, JniHandleOwnership)` 构造函数，对其进行调用，之间创建映射处理和创建的实例，并返回所创建的实例。
8. `TextView.n_GetDefaultMovementMethod()` 调用 `LogTextBox.DefaultMovementMethod`属性 getter。
9. 控制权将返回给 `android.widget.TextView`构造函数，完成执行。
10. `Monodroid.apidemo.LogTextBox`构造函数执行时，调用 `TypeManager.Activate()`。
11. `LogTextBox (上下文, IAttributeSet, int)` 构造函数执行 (7) 中创建的同一实例上。
12. 如果 `(IntPtr, JniHandleOwnership)` 找不到构造函数，则将引发 `System.MissingMethodException` (xref: `System.MissingMethodException`)。

过早 dispose () 调用

没有 JNI 句柄和对应的 C# 实例之间的映射。`Java.Lang.Object.Dispose()` 将中断此映射。如果映射已中断后，JNI 句柄将进入托管的代码，它类似于 Java 激活并 `(IntPtr, JniHandleOwnership)` 构造函数将检查并调用。如果构造函数不存在，将引发异常。

例如，给定以下托管可调用 Wrapper 子类：

```
class ManagedValue : Java.Lang.Object {  
  
    public string Value {get; private set;}  
  
    public ManagedValue (string value)  
    {  
        Value = value;  
    }  
  
    public override string ToString ()  
    {  
        return string.Format ("[Managed: Value={0}]", Value);  
    }  
}
```

如果我们创建一个实例，`dispose ()`，并且会导致托管可调用包装器以重新创建：

```
var list = new JavaList<IJavaObject>();  
list.Add (new ManagedValue ("value"));  
list [0].Dispose ();  
Console.WriteLine (list [0].ToString ());
```

该程序会死亡：

```

E/mono ( 2906): Unhandled Exception: System.NotSupportedException: Unable to activate instance of type
Scratch.PrematureDispose.ManagedValue from native handle 4051c8c8 --->
System.MissingMethodException: No constructor found for
Scratch.PrematureDispose.ManagedValue::.ctor(System.IntPtr, Android.Runtime.JniHandleOwnership)
E/mono ( 2906): at Java.Interop.TypeManager.CreateProxy (System.Type type, IntPtr handle,
JniHandleOwnership transfer) [0x00000] in <filename unknown>:0
E/mono ( 2906): at Java.Interop.TypeManager.CreateInstance (IntPtr handle, JniHandleOwnership transfer,
System.Type targetType) [0x00000] in <filename unknown>:0
E/mono ( 2906): --- End of inner exception stack trace ---
E/mono ( 2906): at Java.Interop.TypeManager.CreateInstance (IntPtr handle, JniHandleOwnership transfer,
System.Type targetType) [0x00000] in <filename unknown>:0
E/mono ( 2906): at Java.Lang.Object.GetObject (IntPtr handle, JniHandleOwnership transfer, System.Type
type) [0x00000] in <filename unknown>:0
E/mono ( 2906): at Java.Lang.Object._GetObject[IJavaObject] (IntPtr handle, JniHandleOwnership transfer)
[0x00000

```

如果包含子类 (*IntPtr*, *JniHandleOwnership*) 构造函数, 则新将创建类型的实例。因此, 实例将显示"丢失"所有实例数据, 因为它是一个新实例。(请注意的值为 null)。

```
I/mono-stdout( 2993): [Managed: Value=]
```

仅 *dispose ()* 的当您知道将不再, 使用 Java 对象或子类不包含任何实例数据和托管可调用包装器子类 (*IntPtr*, *JniHandleOwnership*) 提供了构造函数。

应用程序启动

当一种活动、服务时, 等启动时, Android 将首先检查以查看是否已运行托管活动/service/等的进程。如果不存在任何此类进程, 则将创建一个新进程, [AndroidManifest.xml](#)是读取, 和中指定的类型 [/manifest/application/@android:name](#) 加载和实例化属性。接下来, 通过指定的所有类型 [/manifest/application/provider/@android:name](#) 属性值进行实例化, 并且具有其 [ContentProvider.attachInfo%28](#))调用方法。通过添加到此 Xamarin.Android 挂钩 *mono。MonoRuntimeProvider* *ContentProvider*到在生成过程中的 *AndroidManifest.xml*。 *Mono。MonoRuntimeProvider.attachInfo()* 方法负责加载到进程中的 Mono 运行时。在此点之前使用 Mono 的任何尝试将失败。(注意: 这是为什么类型的子类 *Android.App.Application*需要提供 (*IntPtr*, *JniHandleOwnership*) 构造函数, 为应用程序实例创建可以初始化 Mono 之前。)

完成过程初始化后, [AndroidManifest.xml](#) 参考查找以启动活动/service/等的类名称。例如, [/manifest/application/activity/@android:name](#)属性用于确定要加载的活动的名称。为活动, 此类型必须继承 *android.app.Activity*。通过加载指定的类型 *Class.forName()* (这需要的类型为 Java 类型, 因此 Android 可调用包装器), 然后实例化。Android 可调用包装器实例的创建将触发相应的 C# 类型的实例的创建。Android 将随后调用 *Activity.onCreate(Bundle)*, 这将导致相应 *Activity.OnCreate(Bundle)*被调用, 就可以关闭到的资源争用。

可用程序集

2018/10/26 • [Edit Online](#)

Xamarin.iOS、Xamarin.Android 和 Xamarin.Mac 所有附带十几个程序集。就像 Silverlight 是桌面.NET 程序集的扩展的子集, Xamarin 平台也是多个 Silverlight 和桌面.NET 程序集的扩展的子集。

Xamarin 的平台不是与现有程序集编译为不同的配置文件兼容的 ABI。您必须重新编译源代码以生成程序集面向正确的配置文件 (就像您需要重新编译源代码, 以分别面向 Silverlight 和.NET 3.5)。

Xamarin.Mac 应用程序可以编译中三种模式: 另一个使用 Xamarin 的策划移动配置文件、Xamarin.Mac.NET 4.5 Framework 允许你针对现有的完整的桌面程序集, 并且在系统 Mono 中找到一个使用.NET API 的不受支持安装。有关详细信息, 请参阅我们[目标框架文档](#)。

.NET 标准库

除了 iOS、Android 和 Mac 项目都可以使用的 Xamarin 绑定[.NET Standard 库](#)。

可移植类库

此外可以使用 Xamarin 项目[.NET 可移植类库](#), 尽管支持.NET Standard 已弃用此技术。

支持的程序集

这些是中提供的程序集引用管理器 > 程序集 > 框架(Visual Studio 2017) 和编辑引用 > 包(Visual Studio 中为 Mac), 和兼容性与 Xamarin 平台。

| ASSEMBLY | API 兼容性 | XAMARIN IOS | XAMARIN ANDROID | XAMARIN MAC |
|------------------------------|--|-------------|-----------------|-------------|
| FSharp.Core.dll | | ✓ | ✓ | ✓ |
| I18N.dll | 包含 CJK, MidEast, 其他、少见西部 | ✓ | ✓ | ✓ |
| Microsoft.CSharp.dll | | ✓ | ✓ | ✓ |
| Mono.CSharp.dll | | ✓ | ✓ | ✓ |
| Mono.Data.Sqlite.dll | SQLite; 的 ADO.NET 提供程序请参阅限制。 | ✓ | ✓ | ✓ |
| Mono.Data.Tds.dll | TDS 协议的支持;用于 System.Data.SqlClient 内支持 System.Data 。 | ✓ | ✓ | ✓ |
| Mono.Dynamic.Interpreter.dll | | ✓ | | |
| Mono.Security.dll | 加密 Api。 | ✓ | ✓ | ✓ |

| ASSEMBLY | API 兼容性 | XAMARIN IOS | XAMARIN ANDROID | XAMARIN MAC |
|-------------------------|--|-------------|-----------------|-------------|
| monotouch.dll | 此程序集包含 C# 绑定到产品 CocoaTouch API。这是仅在经典 iOS 项目中可用。 | ✓ | | |
| MonoTouch.Dialog-1.dll | | ✓ | | |
| MonoTouch.NUnitLite.dll | | ✓ | | |
| mscorlib.dll | Silverlight | ✓ | ✓ | ✓ |
| OpenTK-1.0.dll | 面向 OpenGL/OpenAL 对象的扩展以提供 iPhone 设备支持的 Api。 | ✓ | ✓ | ✓ |
| System.dll | Silverlight , 加上以下命名空间中的类型: System.Collections.Specialized System.ComponentModel System.ComponentModel.Design System.Diagnostics System.IO System.IO.Compression System.IO.Compression.FileSystem System.Net System.Net.Cache System.Net.Mail System.Net.Mime System.Net.NetworkInformation System.Net.Security System.Net.Sockets System.Runtime.InteropServices System.Runtime.Versioning System.Security.AccessControl System.Security.Authentication System.Security.Cryptography System.Security.Permissions System.Threading System.Timers | ✓ | ✓ | ✓ |

| ASSEMBLY | API 兼容性 | XAMARIN IOS | XAMARIN ANDROID | XAMARIN MAC |
|---|--|-------------|-----------------|-------------|
| System. ComponentModel. Composition.dll | | ✓ | ✓ | ✓ |
| System. ComponentModel. DataAnnotations.dll | | ✓ | ✓ | ✓ |
| System.Core.dll | Silverlight | ✓ | ✓ | ✓ |
| System.Data.dll | .NET 3.5, 使用删除某些功能。 | ✓ | ✓ | ✓ |
| System.Data.Services. Client.dll | 完整的 oData 客户端。 | ✓ | ✓ | ✓ |
| System.IO. Compression | | ✓ | ✓ | ✓ |
| System.IO. Compression. FileSystem | | ✓ | ✓ | ✓ |
| System.Json.dll | Silverlight | ✓ | ✓ | ✓ |
| System.Net.Http.dll | | ✓ | ✓ | ✓ |
| System.Numerics.dll | | ✓ | ✓ | ✓ |
| System.Runtime. Serialization.dll | Silverlight | ✓ | ✓ | ✓ |
| System. ServiceModel.dll | WCF 堆栈中存 在Silverlight | ✓ | ✓ | ✓ |
| System.ServiceModel. Internals.dll | | ✓ | ✓ | ✓ |
| System.ServiceModel. Web.dll | Silverlight, 加上以下 命名空间中的类型: 系统 System.ServiceModel. Channels System.ServiceModel. Description System.ServiceModel. Web | ✓ | ✓ | ✓ |
| System. Transactions.dll | .NET 3.5; 的一部 分System.Data支持。 | ✓ | ✓ | ✓ |
| System.Web. Services.dll | 从.NET 3.5, 配置文件 中删除的服务器功能 的基本 Web 服务。 | ✓ | ✓ | ✓ |

| ASSEMBLY | API 兼容性 | XAMARIN IOS | XAMARIN ANDROID | XAMARIN MAC |
|--|--|-------------|-----------------|-------------|
| System.Windows.dll | | ✓ | ✓ | ✓ |
| System.Xml.dll | .NET 3.5 | ✓ | ✓ | ✓ |
| System.Xml.Linq.dll | .NET 3.5 | ✓ | ✓ | ✓ |
| System.Xml.Serialization.dll | | ✓ | ✓ | ✓ |
| Xamarin.iOS.dll | 此程序集包含 C# 绑定到产品 CocoaTouch API。这仅在统一 iOS 项目中使用。 | ✓ | | |
| Java.Interop.dll | | | ✓ | |
| Mono.Android.dll | | | ✓ | |
| Mono.Android.Export.dll | | | ✓ | |
| Mono.Posix.dll | | | ✓ | |
| System.EnterpriseServices.dll | | | ✓ | |
| Xamarin.Android.NUnitLite.dll | | | ✓ | |
| Mono.CompilerServices.SymbolWriter.dll | 编译器编写器。 | | | ✓ |
| Xamarin.Mac.dll | | | | ✓ |
| System.Drawing.dll | System.Drawing 不支持统一 API 中的 Xamarin.Mac、.NET 4.5 中或移动框架。System.Drawing 支持可添加到 iOS 和 macOS 使用 sysdrawing coregraphics 库 | ✓ | | ✓ |

Xamarin.Android API 设计原则

2018/10/26 • [Edit Online](#)

概述

除了核心是 Mono 的一部分的基类库, Xamarin.Android 提供与各种 Android Api 允许开发人员使用 Mono 创建本机 Android 应用程序的绑定。

有 Xamarin.Android 的核心是互操作引擎与 Java 领域这样桥 C# 的环境和开发人员提供了访问 Java api 从 C# 或其他.NET 语言。

设计原则

以下是一些 Xamarin.Android 绑定我们设计原则

- 遵守.NET Framework 设计准则。
- 开发人员子类的 Java 类。
- 子类应适用于 C# 标准构造。
- 从现有类派生。
- 链接到调用基构造函数。
- 重写方法应通过 C# 重写系统。
- 使常见的 Java 任务很简单和硬 Java 任务成为可能。
- 将 JavaBean 属性公开为 C# 属性。
- 公开强类型的 API:
 - 提高类型安全性。
 - 最小化运行时错误。
 - 获取 IDE intellisense 上返回类型。
 - 允许 IDE 弹出文档。
- 建议 Api 的 IDE 中探索:
 - 利用最大程度减少 Java Classlib 暴露的 Framework 替代方法。
 - 时适当且适用, 而不是单一方法接口公开 C# 委托 (lambda, 匿名方法和 System.Delegate)。
 - 提供一种机制来调用任意 Java 库 ([Android.Runtime.JNIEnv](#))。

程序集

Xamarin.Android 包括大量的程序集构成 *MonoMobile* 配置文件。[程序集](#)页提供了更多信息。

为 Android 平台的绑定都包含在 `Mono.Android.dll` 程序集。此程序集包含正在使用 Android Api 的整个绑定以及 Android 运行时 VM 与通信。

绑定设计

集合

Android Api 利用 java.util 集合广泛地提供列表、集和地图。我们将使用这些元素公开 `System.Collections.Generic` 绑定中的接口。基本映射是：

- `java.util.Set` 映射到系统类型 `ICollection`，帮助器类 `Android.Runtime.JavaSet`。
- `java.util.List` 映射到系统类型 `IList`，帮助器类 `Android.Runtime.JavaList`。
- `< K, V > java.util.Map` 映射到系统类型 `IDictionary < TKey, TValue >`，帮助器类 `Android.Runtime.JavaDictionary < K, V >`。
- `java.util.Collection` 映射到系统类型 `ICollection`，帮助器类 `Android.Runtime.JavaCollection`。

我们提供了帮助程序类来帮助更快地 copyless 这些类型的封送处理。如果可能，我们建议使用这些提供而不是提供框架实现的集合，如 `List<T>` 或 `Dictionary<TKey, TValue>`。 `Android.Runtime` 实现利用本机 Java 集合在内部，因此不需要在复制/粘贴到本机集合传递到 Android API 成员时。

可以将任何接口实现传递到接受该接口的 Android 方法，例如传递 `List<int>` 到 `ArrayAdapter<int>` (上下文中, `int`, `IList<int>`) 构造函数。但是，为所有实现除对于 `Android.Runtime` 实现中，这涉及到复制 Mono Android 运行时 VM 到 VM 的列表。如果列表为更高版本的 Android 运行时中发生更改 (例如通过调用 `ArrayAdapter<T>.Add(T)` 方法)，这些更改将不会在托管代码中可见。如果 `JavaList<int>` 已使用，这些更改都是可见。

措辞不同而已，集合接口的实现代码不以上列出帮助器类es 仅封送 [In]:

```
// This fails:
var badSource = new List<int> { 1, 2, 3 };
var badAdapter = new ArrayAdapter<int>(context, textViewResourceId, badSource);
badAdapter.Add (4);
if (badSource.Count != 4) // true
    throw new InvalidOperationException ("this is thrown");

// this works:
var goodSource = new JavaList<int> { 1, 2, 3 };
var goodAdapter = new ArrayAdapter<int> (context, textViewResourceId, goodSource);
goodAdapter.Add (4);
if (goodSource.Count != 4) // false
    throw new InvalidOperationException ("should not be reached.");
```

属性

Java 方法转换为属性，在适当的时候：

- Java 方法对 `T getFoo()` 并 `void setFoo(T)` 转换为 `Foo` 属性。示例: `Activity.Intent`。
- Java 方法 `getFoo()` 转换为只读的 `Foo` 属性。示例: `Context.PackageName`。
- 只有组的属性不会生成。
- 属性是不如果属性类型将为数组生成。

事件和侦听器

Android Api 基于 Java 和其组件按照挂接事件侦听器的 Java 模式。此模式可能会很麻烦，因为它需要用户创建匿名类，并声明要重写的方法，例如，这是如何将在与 Java 配合使用 Android 中执行操作：

```
final android.widget.Button button = new android.widget.Button(context);

button.setText(this.count + " clicks!");
button.setOnClickListener (new View.OnClickListener() {
    public void onClick (View v) {
        button.setText(++this.count + " clicks!");
    }
});
```

将 C# 使用事件中的等效代码：

```
var button = new Android.Widget.Button (context) {
    Text = string.Format ("{0} clicks!", this.count),
};
button.Click += (sender, e) => {
    button.Text = string.Format ("{0} clicks!", ++this.count);
};
```

请注意，这两个以上的机制可使用 Xamarin.Android。可以实施侦听器接口并将其附加 View.SetOnClickListener，也可以附加创建通过任何常规 C# 模式到 Click 事件的委托。

当侦听器回调方法具有 void 返回时，我们会创建基于 API 元素 [EventHandler<TEventArgs>](#) 委托。我们会生成事件如上面的示例中为这些侦听器类型。但是，如果侦听器回调将返回一个非 void 的和非-布尔不使用值、事件和事件处理。我们改为生成的回调的签名的特定委托，并添加属性而不是事件。原因是要处理委托调用排序并返回处理。此方法反映使用 Xamarin.iOS API 执行的操作。

如果仅自动生成 C# 事件或属性的 Android 事件注册方法：

1. 具有 `set` 前缀，例如 [设置OnClickListener](#)。
2. 具有 `void` 返回类型。
3. 只接受一个参数，参数类型是接口、接口只包含一个方法，并以接口名称结尾 `Listener`，例如 [View.OnClick 侦听器](#)。

此外，如果侦听器接口方法具有返回类型为布尔而不是 **void**，然后生成 [EventArgs](#) 子类将包含 *Handled* 属性。值 *Handled* 作为返回值使用属性 *侦听器方法*，并将默认为 `true`。

例如，Android [View.setOnKeyListener\(\)](#) 方法接受 [View.OnKeyListener](#) 接口，并且 [View.OnKeyListener.onKey](#) (视图、int、KeyEvent) 方法具有 boolean 返回类型。Xamarin.Android 生成相应 [View.KeyPress](#) 事件，即 [EventHandler<View.KeyEventArgs>](#)。KeyEventArgs 类又有 [View.KeyEventArgs.Handled](#) 属性，它的返回值作为 [View.OnKeyListener.onKey\(\)](#) 方法。

我们想要添加的其他方法和 ctor 来公开基于委托的连接的重载。此外，多个回调的侦听器需要一些附加检查，以确定实现单个回调是否合理的因此我们将在发现转换这些。如果没有相应的事件，侦听器必须使用在 C# 中，但请将任何您认为可能具有委托的使用情况与我们的注意力。我们还投入了接口而不使用“侦听器”后缀的某些转换时很明显，他们将从一个委托的替代方法获益。

所有侦听器接口实现 [Android.Runtime.IJavaObject](#) 接口，因为绑定，以便侦听器类必须实现此接口的实现细节。这可以通过实现侦听器接口上的一个子类 [Java.Lang.Object](#) 或任何其他包装 Java 对象，如 Android 活动。

可运行对象

Java 利用 [java.lang.Runnable](#) 接口，以提供一种委派机制。[Java.lang.Thread](#) 类是值得注意的此接口使用者。Android 已经利用了 API 以及中的接口。[Activity.runOnUiThread\(\)](#) 并 [View.post\(\)](#) 是值得注意的示例。

`Runnable` 接口包含单个 void 方法 [run \(\)](#)。它因此适合于 C# 作为中的绑定 [System.Action](#) 委托。我们提供了在绑定中的重载，它接受 `Action` 参数使用的所有 API 成员 `Runnable` 中的本机 API，例如 [Activity.RunOnUiThread\(\)](#) 和 [View.Post\(\)](#)。

我们留下IRunnable作为可运行对象直接传递而不是因为多个类型实现接口，因此可以替换它们的位置中的重载。

内部类

Java 有两种不同类型的**嵌套类**：静态嵌套类和非静态类。

Java 静态嵌套的类的 C# 嵌套类型相同。

非静态嵌套类，也称为**内部类**，大不相同。它们包含对它们的封闭类型的实例的隐式引用，并且不能包含静态成员（在本概述的范围之外的其他差异）。

谈到绑定并使用 C#，静态嵌套的类被视为普通的嵌套类型。内部类，同时，具有两个重大差别：

1. 作为构造函数参数，必须显式提供对包含类型的隐式引用。
2. 从内部类，内部类继承时必须嵌套在类型的继承自基的内部类，包含类型和派生的类型必须提供与 C# 相同的类型的构造函数包含类型。

例如，考虑Android.Service.Wallpaper.WallpaperService.Engine内部类。由于它是一个内部类，因此此WallpaperService.Engine() 构造函数将引用WallpaperService实例（比较和对比到 Java WallpaperService.Engine (构造函数)，这不需要任何参数）。

示例类派生的内部是 CubeWallpaper.CubeEngine:

```
class CubeWallpaper : WallpaperService {
    public override WallpaperService.Engine OnCreateEngine ()
    {
        return new CubeEngine (this);
    }

    class CubeEngine : WallpaperService.Engine {
        public CubeEngine (CubeWallpaper s)
            : base (s)
        {
        }
    }
}
```

请注意如何 CubeWallpaper.CubeEngine 嵌套在 CubeWallpaper，CubeWallpaper 继承自包含类的 WallpaperService.Engine，并 CubeWallpaper.CubeEngine 具有构造函数采用声明的类型- CubeWallpaper 都是在这种情况下-上面指定。

接口

Java 接口可以包含三个集的成员，其中两个从 C# 会导致问题：

1. 方法
2. 类型
3. 字段

Java 接口将转换为两种类型：

1. 一个包含方法声明（可选）的接口。此接口具有相同的名称与 Java 接口，除还具有我前缀。
2. （可选）的静态类，其中包含的任何字段内 Java 接口声明。

嵌套的类型是"重定位"是封闭接口而不是嵌套类型，与封闭的接口名称作为前缀的同级。

例如，考虑android.os.Parcelable接口。Parcelable接口包含方法、嵌套的类型和常量。Parcelable接口方法放入Android.OS.IParcelable接口。Parcelable接口常量放入Android.OS.ParcelableConsts类型。嵌套android.os.Parcelable.ClassLoaderCreator 并android.os.Parcelable.Creator 类型目前不由于我们泛型的支持; 中

的限制绑定如果它们受支持，它们将会显示作

为`Android.OS.IParcelableClassLoaderCreator`并`Android.OS.IParcelableCreator`接口。例如，嵌套`android.os.IBinder.DeathRecipient`作为绑定接口`Android.OS.IBinderDeathRecipient`接口。

NOTE

从 Xamarin.Android 1.9 开始，Java 接口常量是重复在为了简化将 Java 移植的代码中。这有助于改善迁移依赖于的 Java 代码`android` 提供程序常量的接口。

除了上述类型，有四个进一步的更改：

1. 生成 Java 接口与同名的类型，以包含常量。
2. 此外包含接口常量的类型包含来自于实现的 Java 接口的所有常量。
3. 实现 Java 接口包含常量的所有类都获取新的嵌套的 `InterfaceConsts` 类型包含从所有实现的接口的常量。
4. *月成本*类型现已过时。

有关`android.os.Parcelable`接口，这意味着，现在将存在 `Android.OS.Parcelable` 要包含的常量类型。例如，`Parcelable.CONTENTS_FILE_DESCRIPTOR` 常量将作为绑定 `Parcelable.ContentsFileDescriptor` 常量，而不是作为`ParcelableConsts.ContentsFileDescriptor`常量。

对于包含常量的实现包含其他接口但多个常量的接口，现在会生成所有常量的并集。例如，`android.provider.MediaStore.Video.VideoColumns`接口实现`android.provider.MediaStore.MediaColumns`接口。1.9，不过之前，`Android.Provider.MediaStore.Video.VideoColumnsConsts`类型都有无法访问上声明的常量`Android.Provider.MediaStore.MediaColumnsConsts`。因此，Java 表达式`MediaStore.Video.VideoColumns.TITLE`需要绑定到 C# 表达式`MediaStore.Video.MediaColumnsConsts.Title`这很难发现而无需读取很多 Java 文档。在 1.9，等效的 C# 表达式将是 `MediaStore.Video.VideoColumns.Title`。

此外，考虑`android.os.Bundle`类型，实现 Java `Parcelable`接口。因为它实现了接口，例如该接口上的所有常量都是"通过"绑定类型，可访问`Bundle.CONTENTS_FILE_DESCRIPTOR`是完全有效的 Java 表达式。以前，要移植到此表达式C#将需要查看所有接口都实现从哪种类型中看到`CONTENTS_FILE_DESCRIPTOR`原来的位置。从 Xamarin.Android 1.9 开始，实现包含常量的 Java 接口的类将具有嵌套`InterfaceConsts`类型，它将包含继承的接口的所有常量。这将允许翻译`Bundle.CONTENTS_FILE_DESCRIPTOR`到 `Bundle.InterfaceConsts.ContentsFileDescriptor`。

最后，类型与*月成本*如后缀`Android.OS.ParcelableConsts`现在已过时，新引入 `InterfaceConsts` 以外嵌套类型。Xamarin.Android 3.0 中将删除它们。

资源

可以作为应用程序中包含图像、布局说明、二进制 blob 和字符串字典资源文件。各种 Android Api 旨在操作的资源 Id而不是图像处理，字符串或二进制 blob 直接。

例如，示例 Android 应用，其中包含用户界面布局 (`main.xml`)，国际化表字符串 (`strings.xml`) 和一些图标 (`drawable-*/icon.png`) 将保持其资源的应用程序的"资源"目录中：

```
Resources/  
    drawable-hdpi/  
        icon.png  
  
    drawable-ldpi/  
        icon.png  
  
    drawable-mdpi/  
        icon.png  
  
    layout/  
        main.xml  
  
    values/  
        strings.xml
```

本机 Android Api 不直接使用文件名, 但改为对资源 Id。在编译时使用的资源的 Android 应用程序, 生成系统将包分发的资源, 并生成一个名为类 `Resource`, 其中包含为每个包含的资源的令牌。例如, 对于更高版本的资源布局, 这是 R 类将公开:

```
public class Resource {  
    public class Drawable {  
        public const int icon = 0x123;  
    }  
  
    public class Layout {  
        public const int main = 0x456;  
    }  
  
    public class String {  
        public const int first_string = 0xabc;  
        public const int second_string = 0xbcd;  
    }  
}
```

然后, 使用 `Resource.Drawable.icon` 引用 `drawable/icon.png` 文件, 或 `Resource.Layout.main` 引用 `layout/main.xml` 文件, 或 `Resource.String.first_string` 引用字典文件中的第一个字符串 `values/strings.xml`。

常量和枚举

本机 Android Api 具有很多方法, 接受或返回一个整数, 它必须映射到常量字段来确定 int 的意义。使用这些方法, 用户不需要查阅文档以查看哪些常量是适当的值, 这是不太理想。

例如, 考虑 `Activity.requestWindowFeature (int featureId)`。

在这些情况下, 我们尽力一起分组相关的常量是.NET 的枚举, 并将重新映射要改为采用枚举的方法。通过执行此操作, 我们就能够提供 IntelliSense 所选内容的可能的值。

上面的示例将成为: `Activity.RequestWindowFeature (WindowFeatures featureId)`。

请注意, 这是一个大量的手动过程, 以找出哪些常量一起, 属于哪些 Api 使用这些常量。请在 API 中, 将能够更好地表示为一个枚举文件常量用于任何 bug。

垃圾回收

2018/10/31 • [Edit Online](#)

Xamarin.Android 使用 Mono [简单分代垃圾回收器](#)。这是使用这两代标记和清除垃圾回收器和一个 *大型对象空间*，两种类型的集合：

- 次要集合（收集 Gen0 堆）
- （收集 Gen1 和大型对象空间堆）的主要集合。

NOTE

通过显式集合没有 [GC.Collect\(\)](#) 的集合是否 *按需*、基于堆分配。这不是引用计数系统，对象只要有任何未完成的引用将不会收集，或当作用域已退出。GC 将运行时运行次要堆的新分配的内存不足。如果不存在分配，将不运行。

次要集合是低成本且频繁，并且用于收集最近已分配和死对象。之后每隔几 MB 的已分配的对象执行了次要的集合。可以通过调用来手动执行次要集合 [GC.Collect\(0\)](#)

主要集合是成本高昂且频率较低，并且用于回收所有死对象。内存已用尽当前的堆大小（调整大小之前堆）后，会执行主要集合。主要集合可能会通过调用来手动执行 [GC.Collect\(\)](#) 或通过调用 [GC.Collect\(int\)](#) 使用参数 [GC.MaxGeneration](#)。

跨 VM 对象集合

有三个类别的对象类型。

- **托管对象**：执行此操作的类型 *不* 继承自 [Java.Lang.Object](#)，例如 [System.String](#)。这些是正常情况下收集的 GC。
- **Java 对象**：在 Android 运行时的 VM，但不会公开到 Mono VM Java 类型。这些是令人乏味，并且不会进一步讨论。这些是正常情况下收集的 Android 运行时的 VM。
- **对等对象**：类型实现 [IJavaObject](#)，例如所有 [Java.Lang.Object](#) 并 [Java.Lang.Throwable](#) 子类。这些类型的实例具有两个 "halves" *托管对等方* 和一个 *绝对对等方*。托管的对等方是实例的 C# 类。本机的对等是 VM，在 Android 运行时中的 Java 类的实例和 C# [IJavaObject.Handle](#) 属性包含对本机的对等的 JNI 全局引用。

有两种类型的本机的对等方：

- **框架对等方**：知道 nothing Xamarin.Android，例如 "Normal" Java 类型 [android.content.Context](#)。
- **用户对等方**：Android 可调用 [包装器](#) 生成在生成时针对每个应用程序中存在的 [Java.Lang.Object](#) 子类。

因为有两个 Vm Xamarin.Android 进程内的，有两种类型的垃圾回收：

- Android 运行时集合
- Mono 集合

Android 运行时集合的运行正常，但需要特别注意：的 JNI 全局引用视为 GC 根。如果 JNI 全局因此，引用虚拟机对象，该对象保存到 Android 运行时 *不能* 收集一次，即使不符合回收的条件。

Mono 集合是有趣的发生位置。通常情况下收集托管的对象。对等对象将收集通过执行以下过程：

1. Mono 集合的符合条件的所有对等对象具有使用 JNI 弱全局引用替换其 JNI 全局引用。
2. Android 运行时 VM GC 将调用。可能会收集任何本机的对等实例。

- 检查在 (1) 中创建的 JNI 弱全局引用。如果已收集的弱引用, 则收集对等对象。如果具有弱引用 不 已收集, 然后弱引用替换为的 JNI 全局引用, 并且不收集的对等对象。注意: 在 API 14 +, 这意味着, 从返回的值 `IJavaObject.Handle` 垃圾回收后仍可能会更改。

最终结果, 所有这是对等对象的实例将 live, 只要它通过以下任一方式引用的托管代码 (例如存储在中 `static` 变量) 或所引用的 Java 代码。此外, 将超出否则它们将扩展本机的对等方的生存期 live, 如之前的本机的对等方和托管对等都是可回收不可回收的本机的对等方。

对象周期

在 Android 运行时和 Mono VM 内以逻辑方式存在对等对象。例如, `Android.App.Activity` 托管的对等实例将具有相应 `android.app.Activity` framework 对等 Java 实例。继承的所有对象 `Java.Lang.Object` 需要具有两个 Vm 中的表示形式。

具有两个 Vm 中表示形式的所有对象都具有与仅在单个 VM 中存在的对象相比进行了扩展的生存期 (如 `System.Collections.Generic.List<int>`)。调用 GC。收集一定不会收集这些对象, 因为 Xamarin.Android GC 需要确保收集它之前不引用由任一 VM 的对象。

若要缩短对象生存期 `Java.Lang.Object.Dispose()` 应调用。这将会手动 "断开" 两个 Vm, 因为全局引用, 从而允许以更快地进行收集的对象之间的对象上的连接。

自动集合

开头 [版本 4.1.0](#), Xamarin.Android 会自动执行完整 GC gref 阈值时。此阈值为 90% 的已知的最大 grefs 平台: 1800 grefs 在仿真程序 (2000 max) 和 46800 grefs 硬件 (最大 52000) 上的。注意: Xamarin.Android 仅计数通过创建 grefs `Android.Runtime.JNIEnv`, 并且将无法知道有关此过程中创建的任何其他 grefs。这是启发式方法 仅。

执行自动收集时, 类似于以下的消息将被打印到调试日志:

```
I/monodroid-gc(PID): 46800 outstanding GREFs. Performing a full GC!
```

此匹配项是不确定的和不适当的时候 (例如中间图形呈现), 可能会发生。如果你看到此消息, 可能想要执行的显式集合在其他位置, 或者你可能想要尝试 [减少对等对象的生存期](#)。

GC 桥选项

Xamarin.Android 提供了使用 Android 和 Android 运行时进行透明的内存管理。实现作为 Mono 的垃圾回收器调用的扩展 GC 桥。

GC Bridge 的工作原理在 Mono 垃圾回收和图出哪些对等对象需要使用 Android 运行时堆验证其 "实时性" 过程。GC 桥做出该决定, 通过执行以下步骤 (按顺序):

- 引入到它们所代表的 Java 对象无法访问对等对象的 mono 引用关系图。
- 执行 Java GC。
- 验证为真正死对象的对象。

此复杂的过程是一种使类的子类 `Java.Lang.Object` 自由地引用任何对象; 它会删除任何限制的 Java 对象可以绑定到 C#。由于这种复杂性, 桥过程可能会耗费大量资源, 它可能会导致明显的暂停应用程序中。如果应用程序很重要的暂停, 值得调查以下三个 GC 桥实现之一:

- Tarjan** - GC 桥的全新设计基于 [Robert Tarjan 算法和向后引用传播](#)。它有以下我们模拟工作负荷的最佳性能, 但它还具有更大的实验性代码共享。
- 新-重大革新的原始代码修复的二次行为的两个实例**, 但保留核心算法 (基于 [Kosaraju 的算法](#) 强查找连接组件)。

- 旧的原始实现（被视为最稳定的三个）。这是应用程序应使用如果桥 `GC_BRIDGE` 暂停是可接受。

最好地找出哪些 GC Bridge 的工作原理的唯一方法是通过应用程序中进行试验并分析输出。有两种方法收集的数据进行基准测试：

- 启用日志记录-启用日志记录（中所述配置部分）对于每个 GC 桥选项，然后捕获并比较每个设置的日志输出。检查 `GC` 消息的每个选项；具体而言，`GC_BRIDGE` 消息。暂停最多 150ms 年为非交互式应用程序可承受，但上面非常交互式应用程序（如游戏）60 毫秒暂停问题。
- 启用桥记帐-桥记帐将显示指向桥过程中涉及的每个对象的对象的平均成本。按大小排序此信息将提供什么具有多余的对象的容量提示。

若要指定将哪个 `GC_BRIDGE` 应使用应用程序选项，则传递 `bridge-implementation=old`，`bridge-implementation=new` 或 `bridge-implementation=tarjan` 到 `MONO_GC_PARAMS` 环境变量，例如：

```
MONO_GC_PARAMS=bridge-implementation=tarjan
```

默认设置是 **Tarjan**。如果找到一个回归，您可能会发现有必要将此选项设置为旧。此外，您可以选择使用更稳定旧选项时如果 **Tarjan** 不会生成中的性能改进。

帮助 GC

有多种方法来帮助 GC 以减少内存使用和收集时间。

对等方实例的释放

GC 具有不完整的视图的过程和可能不会运行时内存较少的因为 GC 不知道该内存不足。

例如，实例 `Java.Lang.Object` 类型或派生的类型有至少 20 个字节的大小（变动，恕不另行通知，等等，等等）。托管可调用包装器不要将添加其他实例成员，因此后 `Android.Graphics.Bitmap` 实例引用的 10MB blob 的内存，Xamarin.Android 的 gc 便不会知道—GC 将看到的 20 字节对象，将不能以确定其与 Android 使 10 MB 的内存保持活动状态的运行时分配的对象。

很频繁地需要帮助 GC。遗憾的是，`GC.AddMemoryPressure()` 和 `GC.RemoveMemoryPressure()` 不受支持，因此，如果您知道您只需释放大型 Java 分配对象关系图可能需要手动调用 `GC.Collect()` 提示符 GC 释放 Java 端到内存，也可以显式释放 `Java.Lang.Object` 子类，重大托管的可调用包装器和 Java 实例之间的映射。有关示例，请参阅 [Bug 1084](#)。

NOTE

您必须是极释放时请小心 `Java.Lang.Object` 子类实例。

为了尽量减少内存损坏的可能性，应遵守以下原则调用时 `Dispose()`。

多个线程间共享

如果 Java 或托管实例可能会共享多个线程间不应 `Dispose()` d，曾经。例如，`Typeface.Create()` 可能会返回缓存的实例。如果多个线程提供相同的参数，他们将获得同一实例。因此，`Dispose()` 运算结果 `Typeface` 实例从一个线程可能会使其他线程，这可能会导致 `ArgumentException` s 从 `JNIEnv.CallVoidMethod()`（及其他）因为实例已从另一个线程释放。

释放绑定的 Java 类型

如果绑定的 Java 类型的实例，实例可释放的只要实例不能重复使用从托管代码和 Java 实例不能在线程（请参阅以前之间共享 `Typeface.Create()` 讨论）。（在进行此决定可能比较困难。）下一次 Java 实例进入托管代码中，新将为其创建包装器。

谈到绘图和其他大量资源的实例，这是十分有用：


```
using (var d = Drawable.CreateFromPath ("path/to/filename"))
    imageView.SetImageDrawable (d);
```

以上是安全因为对等方的 `Drawable.CreateFromPath()` Framework 对等方, 将引用返回不用户对等。 `Dispose()` 调用的末尾 `using` 块将中断之间的托管关系 `Drawable` 和 framework `Drawable` 实例, 从而使 Java 实例收集只要 Android 运行时需要。这一点 不为安全起见, 如果用户对等引用对等实例; 此处我们使用 "external" 信息知道的 `Drawable` 用户对等方, 不能引用, 因此 `Dispose()` 调用是安全的。

释放其他类型

如果该实例所引用的不是 Java 类型的绑定的类型 (如自定义 `Activity`), 不要调用 `Dispose()` 除非您知道没有 Java 代码将对的调用重写的方法实例。如果不这样做会导致 `NotSupportedException`。

例如, 如果您有一个自定义单击侦听器:

```
partial class MyClickListener : Java.Lang.Object, View.IOnClickListener {
    // ...
}
```

您不应释放此实例中, 如 Java 将尝试在将来调用它的方法:

```
// BAD CODE; DO NOT USE
Button b = FindViewById<Button> (Resource.Id.myButton);
using (var listener = new MyClickListener ())
    b.SetOnClickListener (listener);
```

使用显式检查, 以避免异常

如果已实现 `Java.Lang.Object.Dispose` 重载方法, 应避免触及涉及 JNI 的对象。执行此操作可能会创建双 `dispose` 就可以将代码移植到 (严重) 的情况下尝试访问已被垃圾回收的基础 Java 对象。执行此操作将生成类似于以下异常:

```
System.ArgumentException: ' jobject ' must not be IntPtr.Zero.
Parameter name: jobject
at Android.Runtime.JNIEnv.CallVoidMethod
```

这种情况通常第一个对象的 `dispose` 将导致成员, 才能成为为 null, 且然后此 null 成员上的后续访问尝试会导致引发异常时发生。具体而言, 该对象的 `Handle` (该链接的托管的实例到其基础 Java 实例) 会在第一个 `dispose` 上失效, 但仍将托管的代码尝试访问此基础 Java 实例, 即使它不再可用 (请参阅 [托管可调用包装器](#) Java 实例和托管的实例之间的映射的详细信息)。

若要避免此异常的好方法是显式验证在你 `Dispose` 托管的实例与基础 Java 实例之间的映射是否仍然有效; 这就是方法, 检查以查看是否对象的 `Handle` 为 null (`IntPtr.Zero`) 然后才能访问其成员。例如, 以下 `Dispose` 方法访问 `childViews` 对象:

```
class MyClass : Java.Lang.Object, ISomeInterface
{
    protected override void Dispose (bool disposing)
    {
        base.Dispose (disposing);
        for (int i = 0; i < this.childViews.Count; ++i)
        {
            // ...
        }
    }
}
```

如果初始 `dispose` 通过的原因 `childViews` 具有无效 `Handle`, 则 `for` 循环访问将引发 `ArgumentException`。通过添加

显式 `Handle` 为 null 之前先检查 `childViews` 访问以下 `Dispose` 方法可防止出现异常：

```
class MyClass : Java.Lang.Object, ISomeInterface
{
    protected override void Dispose (bool disposing)
    {
        base.Dispose (disposing);

        // Check for a null handle:
        if (this.childViews.Handle == IntPtr.Zero)
            return;

        for (int i = 0; i < this.childViews.Count; ++i)
        {
            // ...
        }
    }
}
```

减少被引用的实例

每当的实例 `Java.Lang.Object` 类型或子类 GC，整个过程中扫描对象图必须还扫描实例表示。在对象图是"根实例"引用的对象实例的组加上所有内容根实例引用的引用，以递归方式。

请考虑以下类：

```
class BadActivity : Activity {

    private List<string> strings;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        strings.Value = new List<string> (
            Enumerable.Range (0, 10000)
                .Select(v => new string ('x', v % 1000)));
    }
}
```

时 `BadActivity` 是构造的在对象图将包含 10004 实例 (1 x `BadActivity` , 1 x `strings` , 1 x `string[]` 持有 `strings` , 10000 x 字符串实例)，所有的将需要将每当扫描 `BadActivity` 扫描实例。

这可能会产生不利影响上你收集的时间，从而提高了 GC 暂停时间。

可帮助通过 GC 减少这由用户对等实例取得 root 权限的对象图的大小。在上述示例中，这可以通过移动

`BadActivity.strings` 到单独的类，其中不从 `Java.Lang.Object` 继承：

```

class HiddenReference<T> {

    static Dictionary<int, T> table = new Dictionary<int, T> ();
    static int idgen = 0;

    int id;

    public HiddenReference ()
    {
        lock (table) {
            id = idgen ++;
        }
    }

    ~HiddenReference ()
    {
        lock (table) {
            table.Remove (id);
        }
    }

    public T Value {
        get { lock (table) { return table [id]; } }
        set { lock (table) { table [id] = value; } }
    }
}

class BetterActivity : Activity {

    HiddenReference<List<string>> strings = new HiddenReference<List<string>>();

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        strings.Value = new List<string> (
            Enumerable.Range (0, 10000)
                .Select(v => new string ('x', v % 1000)));
    }
}

```

次要集合

可以通过调用来手动执行次要集合[GC. Collect\(0\)](#)。次要集合是低（相比于主要的集合），但执行具有显著固定成本，因此你不想要触发它们过于频繁，应具有的暂停时间为几毫秒。

如果你的应用程序已在其中反复完成相同的操作的“任务周期”，可能会建议你手动任务周期结束后执行一次。示例工作周期包括：

- 单个游戏帧的呈现循环。
- 与给定的应用对话框（打开、填充、关闭）的整个交互
- 一组网络请求以刷新/同步应用数据。

主要的集合

主要集合可能会通过调用来手动执行[GC. Collect \(\)](#)或 `GC.Collect(GC.MaxGeneration)`。

它们应该很少，执行并可能会暂停时遇到的第二个 Android 风格的设备上收集 512MB 堆时。

主要集合应仅手动调用，如果有过：

- 耗时较长的值班末尾周期和较长的暂停时不会向用户出现问题。

- 中被重写 `Android.App.Activity.OnLowMemory()` 方法。

诊断

若要跟踪时创建和销毁全局引用，可以设置 `debug.mono.log` 系统属性以包含 *gref* 和/或 *gc*。

配置

可以通过设置配置 Xamarin.Android 垃圾回收器 `MONO_GC_PARAMS` 环境变量。可能的生成操作设置环境变量 `AndroidEnvironment`。

`MONO_GC_PARAMS` 环境变量是以逗号分隔列表的以下参数：

- `nursery-size` = 大小：设置小堆的大小。大小以字节为单位指定，并且必须是 2 的幂。后缀 `k`，`m` 和 `g` 可用于指定千、庞大和千兆字节为单位，分别。小堆是第一代（两个）。更大的小堆通常会提高程序的速度，但显然会使用更多的内存。默认值小堆大小为 512 kb。
- `soft-heap-limit` = 大小：目标最大托管应用程序的内存占用情况。当内存使用情况低于指定的值时，GC 进行了优化的执行时间（更少的集合）。超出此限制，GC 进行了优化的内存使用（更多集合）。
- `evacuation-threshold` = 阈值：疏散阈值设置以百分比表示。值必须是 0 到 100 范围内的整数。默认值为 66。如果集合的扫描阶段找到的特定堆块类型的占用小于此百分比，它将执行下一步的主要集合中的块类型，从而还原到接近 100% 的空间使用量的复制集合。值为 0 将关闭疏散。
- `bridge-implementation` = 桥接实现：这会设置 GC 桥选项，可帮助解决 GC 性能问题。有三个可能值：*旧*，*新*，*tarjan*。
- `bridge-require-precise-merge`：网桥包含一种优化这可能，在少数情况下，会使对象可 Tarjan 后首次将成为垃圾收集一个 GC。包括此选项会禁用该优化，从而使 Gc 更可预测，但可能会较慢。

例如，若要配置 GC 堆大小限制为 128 MB，添加一个新的文件到你的项目生成操作的 `AndroidEnvironment` 的内容：

```
MONO_GC_PARAMS=soft-heap-limit=128m
```

限制

2018/10/26 • [Edit Online](#)

由于 Android 应用程序需要在生成过程中生成 Java 代理类型，不能生成在运行时的所有代码。

与桌面 Mono 相比 Xamarin.Android 限制如下：

有限的动态语言支持

Android 可调用包装器需要 Android 运行时需要调用托管的代码的任何时间。基于静态分析的 IL 在编译时，会生成 android 可调用包装器。此操作的净结果：您不能使用动态语言（IronPython、IronRuby 等）在任何方案中，子类化的 Java 类型在需要时（包括间接子类化），因为没有任何办法来提取这些动态类型在编译时生成必要的 Android 可调用包装器。

有限的 Java 生成支持

Android 可调用包装器需要按顺序调用托管的代码的 Java 代码生成。默认情况下，Android 可调用包装器将只包含（某些）声明构造函数和方法的重写虚拟的 Java 方法（即它具有 `RegisterAttribute`）或实现一个 Java 接口方法（接口同样具有 `Attribute`）。

4.1 在发布前，无法声明没有其他方法。4.1 版本中，`Export` 并 `ExportField` 自定义属性可用于声明 Java 方法和 Android 可调用包装器中的字段。

缺少构造函数

构造函数仍有些棘手，除非 `ExportAttribute` 使用。生成 Android 可调用包装器构造函数的算法是如果将发出 Java 构造函数：

1. 没有为所有参数类型的 Java 映射
2. 类的基类声明相同的构造函数—这是必需的因为 Android 可调用包装器必须调用相应的基类构造函数；（因为没有任何简单的方法，可以使用任何默认参数确定值应该是什么在 Java 中使用）。

例如，请考虑以下类：

```
[Service]
class MyIntentService : IntentService {
    public MyIntentService (): base ("value")
    {
    }
}
```

虽然此值看上去非常符合逻辑，生成 Android 可调用包装器发布版本中将不包含默认构造函数。因此，如果您尝试启动此服务（例如 `Context.StartService`），它将失败：

```

E/AndroidRuntime(31766): FATAL EXCEPTION: main
E/AndroidRuntime(31766): java.lang.RuntimeException: Unable to instantiate service example.MyIntentService:
java.lang.InstantiationException: can't instantiate class example.MyIntentService; no empty constructor
E/AndroidRuntime(31766):      at android.app.ActivityThread.handleCreateService(ActivityThread.java:2347)
E/AndroidRuntime(31766):      at android.app.ActivityThread.access$1600(ActivityThread.java:130)
E/AndroidRuntime(31766):      at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1277)
E/AndroidRuntime(31766):      at android.os.Handler.dispatchMessage(Handler.java:99)
E/AndroidRuntime(31766):      at android.os.Looper.loop(Looper.java:137)
E/AndroidRuntime(31766):      at android.app.ActivityThread.main(ActivityThread.java:4745)
E/AndroidRuntime(31766):      at java.lang.reflect.Method.invokeNative(Native Method)
E/AndroidRuntime(31766):      at java.lang.reflect.Method.invoke(Method.java:511)
E/AndroidRuntime(31766):      at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:786)
E/AndroidRuntime(31766):      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)
E/AndroidRuntime(31766):      at dalvik.system.NativeStart.main(Native Method)
E/AndroidRuntime(31766): Caused by: java.lang.InstantiationException: can't instantiate class
example.MyIntentService; no empty constructor
E/AndroidRuntime(31766):      at java.lang.Class.newInstanceImpl(Native Method)
E/AndroidRuntime(31766):      at java.lang.Class.newInstance(Class.java:1319)
E/AndroidRuntime(31766):      at android.app.ActivityThread.handleCreateService(ActivityThread.java:2344)
E/AndroidRuntime(31766):      ... 10 more

```

解决方法是声明默认构造函数，修饰其与 `ExportAttribute`，并设置 `ExportAttribute.SuperStringArgument`：

```

[Service]
class MyIntentService : IntentService {
    [Export (SuperArgumentsString = "\"value\"")]
    public MyIntentService (): base("value")
    {
    }

    // ...
}

```

泛型C#类

泛型C#类仅部分受支持。存在以下限制：

- 不能使用泛型类型 `[Export]` 或 `[ExportField]`。尝试执行此操作将生成 `XA4207` 错误。

```

public abstract class Parcelable<T> : Java.Lang.Object, IParcelable
{
    // Invalid; generates XA4207
    [ExportField ("CREATOR")]
    public static IParcelableCreator CreateCreator ()
    {
        ...
    }
}

```

- 泛型方法不能使用 `[Export]` 或 `[ExportField]`：

```

public class Example : Java.Lang.Object
{
    // Invalid; generates XA4207
    [Export]
    public static void Method<T>(T value)
    {
        ...
    }
}

```

- `[ExportField]` 不能使用方法返回 `void` :

```
public class Example : Java.Lang.Object
{
    // Invalid; generates XA4208
    [ExportField ("CREATOR")]
    public static void CreateSomething ()
    {
    }
}
```

- 泛型类型的实例_不得_通过 Java 代码创建。他们仅安全地从托管代码创建:

```
[Activity (Label="Die!", MainLauncher=true)]
public class BadGenericActivity<T> : Activity
{
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);
    }
}
```

泛型部分 Java 支持

Java 绑定的泛型支持是有限的。具体来说, 是派生自另一个 (非实例化) 的泛型类的泛型实例类中的成员将保留作为 `Java.Lang.Object` 公开。例如, [Android.Content.Intent.GetParcelableExtra](#)方法返回 `Java.Lang.Object`。这是因为已清除 Java 泛型。我们有一些类, 不会应用此限制, 但手动调整。

相关链接

- [Android 可调用包装器](#)
- [使用 JNI](#)
- [ExportAttribute](#)
- [超级字符串](#)
- [RegisterAttribute](#)

疑难解答

2018/10/26 • [Edit Online](#)

在本部分中的文档介绍特定于使用 Android 进行故障排除的功能。

疑难解答提示

故障排除提示和技巧。

常见问题

常见问题故障排除问题的 Xamarin.Android。

解决库安装错误

本指南提供有关可能发生同时引用并自动下载 Android 支持库或 Google Play 服务的一些常见错误的解决方法。

对 Android SDK 工具的更改

自 Android SDK Tools 26.0.1 起, Google 已删除的现有 AVD 和 SDK 管理器采用新的命令行工具。

Xamarin.Android 错误参考

错误参考指南中, 显示在 Xamarin Studio 中使用 Xamarin.Android 时可能会遇到的最常见错误

疑难解答指南

2018/11/2 • [Edit Online](#)

获取诊断信息

Xamarin.Android 有许多方法可以跟踪各种 bug 时的外观。这些方法包括：

1. 诊断 MSBuild 输出。
2. 设备部署日志。
3. Android 调试日志输出。

诊断 MSBuild 输出

诊断 MSBuild 可以包含到包构建相关的其他信息，并且可能包含某些包部署信息。

若要在 Visual Studio 中启用诊断 MSBuild 输出：

1. 单击**工具 > 选项...**
2. 在左侧的树视图中，选择**项目和解决方案 > 生成并运行**
3. 在右侧面板中，设置 MSBuild 生成输出详细级别下拉列表中为**诊断**
4. 单击“确定”
5. 清除并重新生成包。
6. 诊断输出将显示在输出面板中。

若要为 Mac/OS x: 启用 Visual Studio 中的诊断 MSBuild 输出

1. 单击**Visual Studio for Mac > 首选项...**
2. 在左侧的树视图中，选择**项目 > 生成**
3. 在右侧面板中，设置日志详细级别下拉列表为**诊断**
4. 单击“确定”
5. 重启 Visual Studio for Mac
6. 清除并重新生成包。
7. 诊断输出将显示错误板中 (视图 > 面板 > 错误)，通过单击生成输出按钮。

设备部署日志

若要启用 Visual Studio 中的设备部署日志记录：

1. **工具 > 选项...>**
2. 在左侧的树视图中，选择**Xamarin > Android 设置**
3. 在右侧面板中，启用 [X]扩展调试日志记录（将 **monodroid.log** 写入桌面）复选框。
4. 日志消息都写入到在您的桌面上 monodroid.log 文件。

Visual Studio for Mac 始终会写入设备部署日志。找到它们会稍有难度;AndroidUtils日志文件创建的每一天 + 进行部署时，例如：**AndroidTools-2012-10-24_12-35-45.log**。

- 在 Windows 中，日志文件将写入到 `%LOCALAPPDATA%\XamarinStudio-{VERSION}\Logs`。
- 在 OS X 上的日志文件写入到 `$HOME/Library/Logs/XamarinStudio-{VERSION}`。

Android 调试日志输出

Android 将多个消息将写入[Android 调试日志](#)。Xamarin.Android 使用 Android 系统属性来控制生成的其他消息到 Android 调试日志。可以通过设置 android 系统属性`setprop`命令内[Android Debug Bridge \(adb\)](#):

```
adb shell setprop PROPERTY_NAME PROPERTY_VALUE
```

系统属性进程在启动期间, 读取, 因此, 必须设置之前启动应用程序或应用程序必须重新启动后更改的系统属性。

Xamarin.Android 系统属性

Xamarin.Android 支持以下系统属性:

- `debug.mono.debug`: 如果一个非空字符串, 这相当于 `*mono-debug*`。
- `debug.mono.env`: 竖线分隔 (|) 的环境变量来导出应用程序启动期间列表之前mono 已初始化。这允许将环境变量设置该控件 mono 日志记录。
 - *请注意*: 因为值是 '|' 分隔, 该值必须包含额外级别的用引号括起来, 作为 `adb shell` 命令将删除组引号。
 - *请注意*: Android 系统属性值不能超过 92 个字符的长度。
 - 示例:

```
adb shell setprop debug.mono.env "'MONO_LOG_LEVEL=info|MONO_LOG_MASK=asm'"
```

- `debug.mono.log`: 以逗号分隔 (,) 应打印到 Android 调试日志的其他消息的组件的列表。默认情况下, 设置执行任何操作。组件包括:
 - *所有*: 打印所有消息
 - *gc*: 与打印 GC 相关的消息。
 - *gref*: 打印弱 (全局) 引用分配和解除分配消息。
 - *lref*: 打印本地引用分配和解除分配消息。*请注意*: 这些是极详细。不要启用, 除非确实需要。
- `debug.mono.trace`: 允许设置[mono-跟踪](#) `=PROPERTY_VALUE` 设置。

Xamarin.Android 不能解决 System.ValueTuple

此错误是由于使用 Visual Studio 不兼容。

- **Visual Studio 2017 Update 1** (版本 15.1 或更低版本) 是仅与兼容**System.ValueTuple NuGet 4.3.0** (或更低版本)。
- **Visual Studio 2017 Update 2** (版本 15.2 或更高版本) 是仅与兼容**System.ValueTuple NuGet 4.3.1** (或更高版本)。

请选择对应于 Visual Studio 2017 安装正确 System.ValueTuple NuGet。

GC 消息

GC 组件消息可以查看通过 `debug.mono.log` 系统属性设置为一个包含 `gc` 的值。

每当 GC 执行, 并提供信息了解多少运行 GC 未生成 GC 消息:

```
I/monodroid-gc(12331): GC cleanup summary: 81 objects tested - resurrecting 21.
```

可以通过设置生成计时信息等的其他 GC 信息 `MONO_LOG_LEVEL` 环境变量为 `debug`：

```
adb shell setprop debug.mono.env MONO_LOG_LEVEL=debug
```

这将导致（很多）其他的 Mono 消息，包括以下三个后果：

```
D/Mono (15723): GC_BRIDGE num-objects 1 num_hash_entries 81226 sccs size 81223 init 0.00ms df1 285.36ms sort
38.56ms dfs2 50.04ms setup-cb 9.95ms free-data 106.54ms user-cb 20.12ms clenapup 0.05ms links
5523436/5523436/5523096/1 dfs passes 1104 6883/11046605
D/Mono (15723): GC_MINOR: (Nursery full) pause 2.01ms, total 287.45ms, bridge 225.60 promoted 0K major 325184K
los 1816K
D/Mono ( 2073): GC_MAJOR: (user request) pause 2.17ms, total 2.47ms, bridge 28.77 major 576K/576K los 0K/16K
```

在中 `GC_BRIDGE` 消息，`num-objects` 是正在考虑此阶段，桥对象的数目和 `num_hash_entries` 是此调用桥代码期间处理的对象的数字。

中 `GC_MINOR` 并 `GC_MAJOR` 消息，`total` 是长时间暂停世界时（没有线程正在执行），而 `bridge` 是在桥处理代码（用来处理 Java VM）中所用的时间量。这一领域不暂停桥处理发生时。

一般情况下的值越大 `num_hash_entries`，则更多时间 `bridge` 需要集合，和较大 `total` 收集所用的时间将为。

全局引用消息

若要启用日志记录，全局引用 loggig (GREF) `debug.mono.log` 系统属性必须包含 `gref`，例如：

```
adb shell setprop debug.mono.log gref
```

Xamarin.Android 使用 Android 全局引用时调用的 Java 实例，需要为 Java 提供的 Java 方法为提供 Java 实例关联的托管的实例之间的映射。

遗憾的是，Android 仿真程序只允许 2000 年全局引用，以同时存在。硬件具有更高版本限制为 52000 全局引用。因此，知道在仿真器上运行应用程序时可能存在问题的下限其中实例来源可能非常有用。

请注意：全局引用计数是 Xamarin.Android，内部，不会（也不能）包括不再考虑其他本机库加载到进程的全局引用。使用全局引用计数为估计值。

```

I/monodroid-gref(12405): +g+ grefc 108 gwrefc 0 obj-handle 0x40517468/L -> new-handle 0x40517468/L from      at
Java.Lang.Object.RegisterInstance(IJavaObject instance, IntPtr value, JniHandleOwnership transfer)
I/monodroid-gref(12405):      at Java.Lang.Object.SetHandle(IntPtr value, JniHandleOwnership transfer)
I/monodroid-gref(12405):      at Java.Lang.Object..ctor(IntPtr handle, JniHandleOwnership transfer)
I/monodroid-gref(12405):      at Java.Lang.Thread+RunnableImplementor..ctor(System.Action handler, Boolean
removable)
I/monodroid-gref(12405):      at Java.Lang.Thread+RunnableImplementor..ctor(System.Action handler)
I/monodroid-gref(12405):      at Android.App.Activity.RunOnUiThread(System.Action action)
I/monodroid-gref(12405):      at Mono.Samples.Hello.HelloActivity.UseLotsOfMemory(Android.Widget.TextView
textView)
I/monodroid-gref(12405):      at Mono.Samples.Hello.HelloActivity.<OnCreate>m__3(System.Object o)
I/monodroid-gref(12405): handle 0x40517468; key_handle 0x40517468: Java Type:
`mono/java/lang/RunnableImplementor`; MCW type: `Java.Lang.Thread+RunnableImplementor`
I/monodroid-gref(12405): Disposing handle 0x40517468
I/monodroid-gref(12405): -g- grefc 107 gwrefc 0 handle 0x40517468/L from      at
Java.Lang.Object.Dispose(System.Object instance, IntPtr handle, IntPtr key_handle, JObjectRefType handle_type)
I/monodroid-gref(12405):      at Java.Lang.Object.Dispose()
I/monodroid-gref(12405):      at Java.Lang.Thread+RunnableImplementor.Run()
I/monodroid-gref(12405):      at Java.Lang.IRunnableInvoker.n_Run(IntPtr jnienv, IntPtr native__this)
I/monodroid-gref(12405):      at System.Object.c200fe6f-ac33-441b-a3a0-47659e3f6750(IntPtr , IntPtr )
I/monodroid-gref(27679): ++ grefc 1916 gwrefc 296 obj-handle 0x406b2b98/G -> new-handle 0xde68f4bf/W from
take_weak_global_ref_jni
I/monodroid-gref(27679): -w- grefc 1915 gwrefc 294 handle 0xde691aaf/W from take_global_ref_jni

```

有四个消息的重要性:

- 全局引用创建: 这些是开头的行 + g + , 并将创建的代码路径提供堆栈跟踪。
- 全局引用析构: 这些是开头的行 -g- , 并可提供用于处理全局引用的代码路径的堆栈跟踪。如果 GC 释放的 gref 中, 将提供没有堆栈跟踪。
- 全局的弱引用创建: 这些是开头的行 + w + 。
- 全局的弱引用析构: 这些是开头的行 -w- 。

中的所有消息, *grefc*值是 Xamarin.Android 已创建, 全局引用的计数时*grefwc*值是弱 Xamarin.Android 已创建的全局引用的计数。处理或obj 句柄NI 句柄值, 后面的字符值为为 '/' 是句柄值的类型: /L的本地引用 /G对于全局引用, 并 /W弱全局引用。

GC 过程的一部分, 作为全局引用 (+ g +) 转换为全局的弱引用 (导致 a + w + 和-g-) 和 Java 端 GC 将启动, 然后检查弱全局引用, 以确定是否收集。如果仍处于活动状态, 周围的弱引用创建新 gref (+ g +、-w-), 否则销毁的弱引用 (-w)。

创建并由 MCW 包装 Java 实例

```

I/monodroid-gref(27679): +g+ grefc 2211 gwrefc 0 obj-handle 0x4066df10/L -> new-handle 0x4066df10/L from ...
I/monodroid-gref(27679): handle 0x4066df10; key_handle 0x4066df10: Java Type:
`android/graphics/drawable/TransitionDrawable`; MCW type: `Android.Graphics.Drawables.TransitionDrawable`

```

GC 是正在执行...

```

I/monodroid-gref(27679): ++ grefc 1953 gwrefc 259 obj-handle 0x4066df10/G -> new-handle 0xde68f95f/W from
take_weak_global_ref_jni
I/monodroid-gref(27679): -g- grefc 1952 gwrefc 259 handle 0x4066df10/G from take_weak_global_ref_jni

```

对象是仍保持活动状态, 为句柄 ! = null

wref 转回 gref

```
I/monodroid-gref(27679): *try_take_global obj=0x4976f080 -> wref=0xde68f95f handle=0x4066df10
I/monodroid-gref(27679): +g+ grefc 1930 gwrefc 39 obj-handle 0xde68f95f/W -> new-handle 0x4066df10/G from
take_global_ref_jni
I/monodroid-gref(27679): -w- grefc 1930 gwrefc 38 handle 0xde68f95f/W from take_global_ref_jni
```

对象已死，句柄作为 == null

wref 是已释放，无法将任何新创建的 gref

```
I/monodroid-gref(27679): *try_take_global obj=0x4976f080 -> wref=0xde68f95f handle=0x0
I/monodroid-gref(27679): -w- grefc 1914 gwrefc 296 handle 0xde68f95f/W from take_global_ref_jni
```

还有一个"有意义"的问题：在运行 Android 4.0 之前的目标，gref 值是否等于 Android 运行时的内存中的 Java 对象的地址。（即，GC 是非移动，保守，收集器，并且它正在处理对这些对象的直接引用。）因此之后 + g + + w +、- g-、+ g +、-w 序列，生成 gref 将具有与原始 gref 值相同的值。这使得通过日志 grepping 非常简单。

Android 4.0 中，但是，已移动的收集器，并且无法再分发对 Android 运行时的直接引用 VM 对象。因此之后，+ g + + w +、-g-、+ g +、-w 序列、gref 值将不同。如果该对象可以幸存，但多个 Gc，它会通过一些 gref 值，因此更难确定其中一个实例已从实际分配。

以编程方式查询

您可以通过查询 GREF 和 WREF 计数 `JniRuntime` 对象。

`Java.Interop.JniRuntime.CurrentRuntime.GlobalReferenceCount` 全局引用计数

`Java.Interop.JniRuntime.CurrentRuntime.WeakGlobalReferenceCount` -弱引用计数

脱机激活

如果您是无法激活 Windows，在 Xamarin.Android 或无法在 Mac OS X 上安装 Xamarin.Android 的完整版本，请参阅[脱机激活](#)页。

从试用版帐户不能升级到独立/企业版

如果你最近购买了 Xamarin.Android 和以前启动了 Xamarin.Android 的试用版，你可能需要完成以下步骤以获取此拾取 Visual Studio for Mac 或 Visual Studio 的许可证更改。

- 关闭 Visual Studio for Mac/Visual Studio
- 为 Android\License\ 为 Windows 从 Mac 上的 ~/Library/MonoAndroid 或 %PROGRAMDATA%\Mono 中删除所有文件
- 重新打开 Visual Studio for Mac/Visual Studio 和生成 Xamarin.Android 项目

这应让您振奋并正在运行。如果继续遇到问题，可能需要尝试[脱机激活](#)以完成激活你的工作站。

接收激活不完整的错误消息

使用 Xamarin.Android for Visual Studio 时，可能出现此问题。若要解决此问题，请将日志发送到以下位置从 *contact@xamarin.com*。

- 日志位置：%localappdata%\Xamarin\日志

收到检索更新信息的错误错误消息

有时，更新将失败并检查更新时，会经常发生此以下错误：

大多数时候, 只需通过注销你的 Xamarin 帐户, 可以解决此错误, 日志记录然后中返回。

若要完成此操作, 请找到以下所选平台, 并按相关步骤:

在 Mac 上:

1. 打开 Visual Studio for Mac
2. 选择 Visual Studio for Mac > 帐户...
3. 单击查看日志
4. 单击登录
5. 输入你的凭据
6. 检查更新

在 PC 上使用 Visual Studio:

1. 打开 Visual Studio
2. 选择工具 > Xamarin 帐户
3. 单击查看日志
4. 单击登录
5. 输入你的凭据
6. 检查更新

如果此错误消息继续出现, 请发送电子邮件**contact@xamarin.com**。

Android 调试日志

[Android 调试日志](#)可能提供有关您要查看的任何运行时错误的更多上下文。

浮点性能太糟糕了 !

或者, "我的应用程序运行速度更快 10 倍与调试版本比发布版本 !"

Xamarin.Android 支持多个设备的 Abi: *armeabi*, *armeabi-v7a*, 并 *x86*。在中指定设备 **Abi 项目属性 > 应用程序选项卡 > 支持的体系结构**。

调试版本中使用的 Android 程序包, 它提供了所有的 Abi, 因此将使用最快的 ABI 针对目标设备。

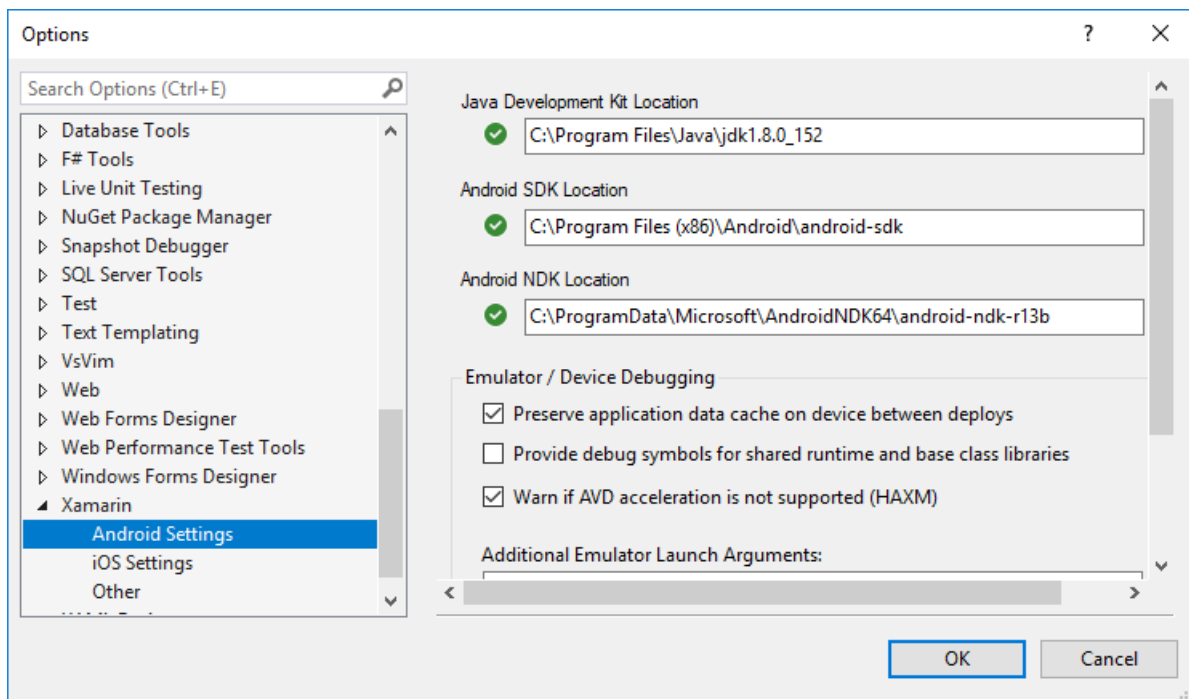
发布版本将仅包括在项目属性选项卡中选择的 Abi。可以选择多个。

armeabi ABI, 默认值, 且是最广泛的设备支持。*但是*, *armeabi* 不支持多 CPU 的设备和硬件浮点型 *among* 其他操作。因此, 使用 *armeabi* 版本运行时应用程序将绑定到单核, 并且将使用软浮点实现。这两种可参与您的应用程序的性能明显变慢。

如果你的应用需要不错的浮点性能 (例如游戏), 则应启用 *armeabi-v7a* ABI。你可能想要仅支持 *armeabi-v7a* 运行时, 尽管这意味着, 仅支持较旧的设备 *armeabi* 将无法运行你的应用。

找不到 Android SDK

将来自 Google 的 Android SDK 的 Windows 提供了 2 个下载。如果您选择 .exe 安装程序, 它将编写告诉 Xamarin.Android 的安装位置的注册表项。如果您选择的 .zip 文件, 并将其解压自己, Xamarin.Android 不知道查找 SDK 的位置。您可以告知 Xamarin.Android 位置 SDK 在 Visual Studio 中通过转到 **工具 > 选项 > Xamarin > Android 设置**:



IDE 不会显示目标设备

有时将尝试应用程序部署到设备，但想要部署到未显示在选择设备对话框中的设备。这可以去度假 Android Debug Bridge 决定。

若要诊断此问题，查找[adb 程序](#)，然后运行：

```
adb devices
```

如果你的设备不存在，然后需要重新启动 Android Debug Bridge 服务器，以便可以找到你的设备：

```
adb kill-server
adb start-server
```

HTC 同步软件可能会阻止[adb 开始服务器](#)无法正常工作。如果[adb 开始服务器](#)命令不会打印出其启动的端口，请退出 HTC 同步软件，请尝试重启 adb 服务器。

无法运行指定的任务可执行文件"keytool"

这意味着你的路径不包含 Java SDK 的 bin 目录所在的目录。检查是否遵循这些步骤，可从[安装指南](#)。

monodroid.exe 或 aresgen.exe 已退出，代码 1

若要帮助你调试此问题，请转到 Visual Studio 和更改 MSBuild 详细级别，为此，请选择：**工具 > 选项 > 项目并解决方案 > 构建和运行 > MSBuild 项目生成输出详细信息**并将此值设置为**正常**。

重新生成，并检查 Visual Studio 输出窗格中，其中应包含完整的错误。

要将包部署的设备上没有足够的存储空间

在不启动 Visual Studio 中的从仿真程序时，将发生这种情况。当从 Visual Studio 外部的仿真程序，需要传递

`-partition-size 512` 选项，例如


```
emulator -partition-size 512 -avd MonoDroid
```

请确保使用正确的模拟器名称, 即[配置模拟器使用的名称](#)。

安装_失败_无效_APK 安装包时

Android 包名称**必须**包含句点 (.)。编辑你的包名称, 以便它包含一个句点。

- 在 Visual Studio:
 - 右键单击你的项目 > 属性
 - 单击左侧的 Android 清单选项卡。
 - 更新包名称字段。
 - 如果看到消息“找到任何 AndroidManifest.xml。单击此项可添加一个。”、单击链接, 然后更新包名称字段。
- 在 Visual Studio for Mac:
 - 右键单击你的项目 > 选项。
 - 导航到生成 / Android 应用程序部分。
 - 更改要包含的包名称字段。'。

安装_失败_MISSING_共享_库安装包时

在此上下文中的“共享的库”, 则**不**本机共享的库 (*libfoo.so*) 文件; 而是必须单独安装在目标设备, 如 Google 地图的库。

Android 包指定正在使用所需的共享的库 `<uses-library/>` 元素。如果**必需**库不存在目标设备上 (例如 `//uses-library/@android:required` 是 *true*, 这是默认设置), 则包的安装将失败, 与**安装_失败_缺少_共享_库**。

若要确定哪些共享的库是必需的请查看生成 **AndroidManifest.xml** 文件 (例如 `obj\调试\android\AndroidManifest.xml`), 并查找 `<uses-library/>` 元素。 `<uses-library/>` 可以在项目中手动添加元素属性 **AndroidManifest.xml** 文件并通过 [UsesLibraryAttribute](#) [自定义特性](#)。

例如, 添加的程序集引用 *Mono.Android.GoogleMaps.dll* 会隐式添加 `<uses-library/>` Google Maps 共享库。

安装_失败_更新_安装包时不兼容

Android 包具有三个要求:

- 它们**必须**包含 '。(请参阅以前的条目)
- 它们**必须**具有唯一的字符串的包名称 (因此在 Android 应用程序名称, 例如 com.android.chrome Chrome 应用中看到的反向 tld 约定)
- 升级程序包时, 包**必须**具有相同的签名密钥。

因此, 设想以下场景:

1. 生成和部署你的应用作为调试应用
2. 更改签名密钥, 例如为用作发布应用程序 (或因为您不喜欢默认提供调试签名密钥)
3. 将应用安装无需首先将其删除, 例如: 调试 > 启动但不调试 Visual Studio 中

在此情况下, 包的安装将失败, 并**安装_失败_更新_不兼容**错误, 因为包名称未发生更改时的签名密钥未。 [Android 调试日志](#) 还将包含如下消息:

```
E/PackageManager( 146): Package [PackageName] signatures do not match the previously installed version; ignoring!
```


若要修复此错误，完全删除应用程序从你的设备然后再重新安装。

安装失败_UID_安装包时，更改

安装 Android 包时，会分配用户 id (UID)。有时，对于当前未知原因，通过已安装的应用程序安装时，安装将失败与

INSTALL_FAILED_UID_CHANGED：

```
ERROR [2015-03-23 11:19:01Z]: ANDROID: Deployment failed
Mono.AndroidTools.InstallFailedException: Failure [INSTALL_FAILED_UID_CHANGED]
  at Mono.AndroidTools.Internal.AdbOutputParsing.CheckInstallSuccess(String output, String packageName)
  at Mono.AndroidTools.AndroidDevice.<>c__DisplayClass2c.<InstallPackage>b__2b(Task`1 t)
  at System.Threading.Tasks.ContinuationTaskFromResultTask`1.InnerInvoke()
  at System.Threading.Tasks.Task.Execute()
```

若要解决此问题，请完全卸载 Android 包，通过从 Android 目标 GUI 安装的应用或使用 adb：

```
$ adb uninstall @PACKAGE_NAME@
```

不要使用 `adb uninstall -k`，因为这会保留应用程序数据，并因此保留在目标设备上发生冲突的 UID。

发布应用程序无法在设备上启动

Android 调试日志输出将包含如下消息：

```
D/AndroidRuntime( 1710): Shutting down VM
W/dalvikvm( 1710): threadid=1: thread exiting with uncaught exception (group=0xb412f180)
E/AndroidRuntime( 1710): FATAL EXCEPTION: main
E/AndroidRuntime( 1710): java.lang.UnsatisfiedLinkError: Couldn't load monodroid: findLibrary returned null
E/AndroidRuntime( 1710):         at java.lang.Runtime.loadLibrary(Runtime.java:365)
```

如果是这样，有的这两个可能的原因：

1. .Apk 不提供目标设备支持的 ABI。例如，.apk 仅包含 armeabi-v7a 二进制文件和目标设备仅支持 armeabi。
2. [Android bug](#)。如果这种情况，卸载该应用、跨您的手指移动，并重新安装该应用。

若要解决问题 (1)，请编辑项目选项/属性和将对所需的 ABI 的支持添加到支持的 Abi 列表。若要确定您需要添加哪些 ABI，请对你的目标设备运行以下 adb 命令：

```
adb shell getprop ro.product.cpu.abi
adb shell getprop ro.product.cpu.abi2
```

输出将包含主（和可选的辅助副本）的 Abi。

```
$ adb shell getprop | grep ro.product.cpu
[ro.product.cpu.abi2]: [armeabi]
[ro.product.cpu.abi]: [armeabi-v7a]
```

OutPath 属性未设置为项目“MyApp.csproj”

这通常意味着您必须将 HP 的计算机和环境变量“平台”MCD 或 HPD 等设置为内容。这与通常设置为 MSBuild 平台属性冲突“任何 CPU”或“x86”。您需要正常 MSBuild 可以从计算机中删除此环境变量：

- 控制面板 > 系统 > 高级 > 环境变量

重启 Visual Studio 或 Visual Studio for Mac, 然后尝试重新生成。功能现在应工作正常。

java.lang.ClassCastException: mono.android.runtime.JavaObject 无法强制转换为...

Xamarin.Android 4.x 不会正确封送嵌套的泛型类型正确。例如, 考虑以下 C#代码使用 [SimpleExpandableListAdapter](#):

```
// BAD CODE; DO NOT USE
var groupData = new List<IDictionary<string, object>> () {
    new Dictionary<string, object> {
        { "NAME", "Group 1" },
        { "IS_EVEN", "This group is odd" },
    },
};

var childData = new List<IList<IDictionary<string, object>>> () {
    new List<IDictionary<string, object>> {
        new Dictionary<string, object> {
            { "NAME", "Child 1" },
            { "IS_EVEN", "This group is odd" },
        },
    },
};

mAdapter = new SimpleExpandableListAdapter (
    this,
    groupData,
    Android.Resource.Layout.SimpleExpandableListItem1,
    new string[] { "NAME", "IS_EVEN" },
    new int[] { Android.Resource.Id.Text1, Android.Resource.Id.Text2 },
    childData,
    Android.Resource.Layout.SimpleExpandableListItem2,
    new string[] { "NAME", "IS_EVEN" },
    new int[] { Android.Resource.Id.Text1, Android.Resource.Id.Text2 }
);
```

问题在于 Xamarin.Android 错误地将嵌套的泛型类型封送。 `List<IDictionary<string, object>>` 正在封送到 [java.lang.ArrayList](#), 但 `ArrayList` 包含 `mono.android.runtime.JavaObject` 实例 (哪个引用 `Dictionary<string, object>` 实例) 而不是某些实现 [java.util.Map](#), 从而导致出现以下异常:

```
E/AndroidRuntime( 2991): FATAL EXCEPTION: main
E/AndroidRuntime( 2991): java.lang.ClassCastException: mono.android.runtime.JavaObject cannot be cast to java.util.Map
E/AndroidRuntime( 2991):         at
android.widget.SimpleExpandableListAdapter.getView(SimpleExpandableListAdapter.java:278)
E/AndroidRuntime( 2991):         at
android.widget.ExpandableListAdapter.getView(ExpandableListAdapter.java:446)
E/AndroidRuntime( 2991):         at android.widget.AbsListView.obtainView(AbsListView.java:2271)
E/AndroidRuntime( 2991):         at android.widget.ListView.makeAndAddView(ListView.java:1769)
E/AndroidRuntime( 2991):         at android.widget.ListView.fillDown(ListView.java:672)
E/AndroidRuntime( 2991):         at android.widget.ListView.fillFromTop(ListView.java:733)
E/AndroidRuntime( 2991):         at android.widget.ListView.layoutChildren(ListView.java:1622)
```

解决方法是使用所提供 [Java 集合类型](#) 而不是 `System.Collections.Generic` 类型“内部”类型。封送处理实例时, 这将导致相应的 Java 类型。(下面的代码是必要的以减少 gref 生存期比要复杂一些。它可以简化为更改原始代码通过 `s/List/JavaList/g` 和 `s/Dictionary/JavaDictionary/g` 如果 gref 生存期不需担心。)

```
// insert good code here
using (var groupData = new JavaList<IDictionary<string, object>> ()) {
    using (var groupEntry = new JavaDictionary<string, object> ()) {
        groupEntry.Add ("NAME", "Group 1");
        groupEntry.Add ("IS_EVEN", "This group is odd");
        groupData.Add (groupEntry);
    }
    using (var childData = new JavaList<IList<IDictionary<string, object>>> ()) {
        using (var childEntry = new JavaList<IDictionary<string, object>> ())
        using (var childEntryDict = new JavaDictionary<string, object> ()) {
            childEntryDict.Add ("NAME", "Child 1");
            childEntryDict.Add ("IS_EVEN", "This child is odd.");
            childEntry.Add (childEntryDict);
            childData.Add (childEntry);
        }
    }
    mAdapter = new SimpleExpandableListAdapter (
        this,
        groupData,
        Android.Resource.Layout.SimpleExpandableListItem1,
        new string[] { "NAME", "IS_EVEN" },
        new int[] { Android.Resource.Id.Text1, Android.Resource.Id.Text2 },
        childData,
        Android.Resource.Layout.SimpleExpandableListItem2,
        new string[] { "NAME", "IS_EVEN" },
        new int[] { Android.Resource.Id.Text1, Android.Resource.Id.Text2 }
    );
}
}
```

这将在未来版本中修复。

出现意外的 NullReferenceExceptions

偶尔[Android 调试日志](#)会提及 nullreferenceexception 抛出的“不会发生”或来自 Mono 前应用就完了 Android 运行时代码：

```
E/mono(15202): Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object
E/mono(15202):   at Java.Lang.Object.GetObject (IntPtr handle, System.Type type, Boolean owned)
E/mono(15202):   at Java.Lang.Object._GetObject[IOnTouchListener] (IntPtr handle, Boolean owned)
E/mono(15202):   at Java.Lang.Object.GetObject[IOnTouchListener] (IntPtr handle, Boolean owned)
E/mono(15202):   at
Android.Views.View+IOnTouchListenerAdapter.n_OnTouch_Landroid_view_View_Landroid_view_MotionEvent_(IntPtr jnienv, IntPtr native__this, IntPtr native_v, IntPtr native_e)
E/mono(15202):   at (wrapper dynamic-method) object:b039cbb0-15e9-4f47-87ce-442060701362
(intptr,intptr,intptr,intptr)
```

或

```
E/mono ( 4176): Unhandled Exception:
E/mono ( 4176): System.NullReferenceException: Object reference not set to an instance of an object
E/mono ( 4176): at Android.Runtime.JNIEnv.NewString (string)
E/mono ( 4176): at Android.Util.Log.Info (string,string)
```

这可能会在 Android 运行时决定中止过程中，其可能会由于多种原因，包括达到目标的 GREF 限制或执行这样的操作“错误”使用 JNI。

若要查看是否这种情况，请从您的过程类似于消息 Android 调试日志：

```
E/dalvikvm( 123): VM aborting
```

由于全局引用用尽而导致中止

Android 运行时的 JNI 层仅支持有限的数量的 JNI 对象引用，以在时间上会在任意给定时间有效。当超过此限制时，事情就会中断。

REF (全局引用) 限制为 2000 年引用的仿真程序中，并在硬件上的 ~ 52000 引用。

您知道正在创建太多 GREFs 时看到消息，例如此 Android 调试日志中：

```
D/dalvikvm( 602): GREF has increased to 1801
```

时达到 GREF 限制，会输出消息如下所示：

```
D/dalvikvm( 602): GREF has increased to 2001
W/dalvikvm( 602): Last 10 entries in JNI global reference table:
W/dalvikvm( 602): 1991: 0x4057eff8 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 1992: 0x4057f010 cls=Landroid/graphics/Point; (28 bytes)
W/dalvikvm( 602): 1993: 0x40698e70 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 1994: 0x40698e88 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 1995: 0x40698ea0 cls=Landroid/graphics/Point; (28 bytes)
W/dalvikvm( 602): 1996: 0x406981f0 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 1997: 0x40698208 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 1998: 0x40698220 cls=Landroid/graphics/Point; (28 bytes)
W/dalvikvm( 602): 1999: 0x406956a8 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): 2000: 0x406956c0 cls=Landroid/graphics/Point; (20 bytes)
W/dalvikvm( 602): JNI global reference table summary (2001 entries):
W/dalvikvm( 602): 51 of Ljava/lang/Class; 164B (41 unique)
W/dalvikvm( 602): 46 of Ljava/lang/Class; 188B (17 unique)
W/dalvikvm( 602): 6 of Ljava/lang/Class; 212B (6 unique)
W/dalvikvm( 602): 11 of Ljava/lang/Class; 236B (7 unique)
W/dalvikvm( 602): 3 of Ljava/lang/Class; 260B (3 unique)
W/dalvikvm( 602): 4 of Ljava/lang/Class; 284B (2 unique)
W/dalvikvm( 602): 8 of Ljava/lang/Class; 308B (6 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 316B
W/dalvikvm( 602): 4 of Ljava/lang/Class; 332B (3 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 356B
W/dalvikvm( 602): 2 of Ljava/lang/Class; 380B (1 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 428B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 452B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 476B
W/dalvikvm( 602): 2 of Ljava/lang/Class; 500B (1 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 548B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 572B
W/dalvikvm( 602): 2 of Ljava/lang/Class; 596B (2 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 692B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 956B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 1004B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 1148B
W/dalvikvm( 602): 2 of Ljava/lang/Class; 1172B (1 unique)
W/dalvikvm( 602): 1 of Ljava/lang/Class; 1316B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 3428B
W/dalvikvm( 602): 1 of Ljava/lang/Class; 3452B
W/dalvikvm( 602): 1 of Ljava/lang/String; 28B
W/dalvikvm( 602): 2 of Ldalvik/system/VMRuntime; 12B (1 unique)
W/dalvikvm( 602): 10 of Ljava/lang/ref/WeakReference; 28B (10 unique)
W/dalvikvm( 602): 1 of Ldalvik/system/PathClassLoader; 44B
W/dalvikvm( 602): 1553 of Landroid/graphics/Point; 20B (1553 unique)
W/dalvikvm( 602): 261 of Landroid/graphics/Point; 28B (261 unique)
W/dalvikvm( 602): 1 of Landroid/view/MotionEvent; 100B
W/dalvikvm( 602): 1 of Landroid/app/ActivityThread$ApplicationThread; 28B
W/dalvikvm( 602): 1 of Landroid/content/ContentProvider$Transport; 28B
W/dalvikvm( 602): 1 of Landroid/view/Surface$CompatibleCanvas; 44B
W/dalvikvm( 602): 1 of Landroid/view/inputmethod/InputMethodManager$ControlledInputConnectionWrapper; 36B
W/dalvikvm( 602): 1 of Landroid/view/ViewRoot$1; 12B
W/dalvikvm( 602): 1 of Landroid/view/ViewRoot$W; 28B
W/dalvikvm( 602): 1 of Landroid/view/inputmethod/InputMethodManager$1; 28B
W/dalvikvm( 602): 1 of Landroid/view/accessibility/AccessibilityManager$1; 28B
W/dalvikvm( 602): 1 of Landroid/widget/LinearLayout$LayoutParams; 44B
W/dalvikvm( 602): 1 of Landroid/widget/LinearLayout; 332B
W/dalvikvm( 602): 2 of Lorg/apache/harmony/xnet/provider/jsse/TrustManagerImpl; 28B (1 unique)
W/dalvikvm( 602): 1 of Landroid/view/SurfaceView$MyWindow; 36B
W/dalvikvm( 602): 1 of Ltouchtest/RenderThread; 92B
W/dalvikvm( 602): 1 of Landroid/view/SurfaceView$3; 12B
W/dalvikvm( 602): 1 of Ltouchtest/DrawingView; 412B
W/dalvikvm( 602): 1 of Ltouchtest/Activity1; 180B
W/dalvikvm( 602): Memory held directly by tracked refs is 75624 bytes
E/dalvikvm( 602): Excessive JNI global references (2001)
E/dalvikvm( 602): VM aborting
```

在上面的示例 (其中, 顺便说一下, 来自[bug 685215](#)) 问题是, 创建的实例过多 `Android.Graphics.Point`; 请参阅[注释 #2](#)有关的修补程序列表此特定的 bug。

通常情况下, 一个有用的解决方案是查找哪种类型具有太多实例分配-`Android.Graphics.Point` 上述转储中的-然后找到其中它们创建在您的源代码和处置它们相应地 (以便其缩短 Java-object 生存期)。这并不总是适当 (#685215 是多线程, 因此简单的解决方案可避免 `Dispose` 调用), 但它是第一件事需要考虑。

可以让[GREF 日志记录](#)若要查看何时创建 GREFs, 以及多少存在。

中止由于 JNI 类型不匹配

如果手动播 JNI 代码, 则有可能的类型不会与匹配正确, 例如如果你尝试调用 `java.lang.Runnable.run` 不会实现的类型 `java.lang.Runnable`。此操作时, 将有一条消息类似于此 Android 调试日志中:

```
W/dalvikvm( 123): JNI WARNING: can't call Ljava/Type;.method on instance of Lanother/java/Type;
W/dalvikvm( 123):             in Lmono/java/lang/RunnableImplementor;.n_run:()V (CallVoidMethodA)
...
E/dalvikvm( 123): VM aborting
```

动态代码支持

动态代码不会进行编译

若要使用的是 C#动态应用程序或库中, 您需要将 `system.core.dll` 的引用、`Microsoft.CSharp.dll` 和 `Mono.CSharp.dll` 添加到你的项目。

在发布版本 **MissingMethodException** 发生动态代码在运行时。

- 很可能你的应用程序项目不具有对 `system.core.dll` 的引用、`Microsoft.CSharp.dll` 或 `Mono.CSharp.dll` 的引用。请确保引用这些程序集。
 - 请注意该动态代码始终成本。如果你需要高效的代码, 请考虑不使用动态代码。
- 在第一个预览中, 除非由应用程序代码显式使用每个程序集中的类型, 否则排除这些程序集。请参阅以下一种解决方法: <http://lists.ximian.com/pipermail/mono-il/009798.html>

与 AOT + LLVM 崩溃构建在 x86 上的项目的设备

部署使用构建的应用时[AOT + LLVM](#)上基于 x86 的设备, 可能会看到异常错误消息如下所示:

```
Assertion: should not be reached at /Users/.../external/mono/mono/mini/tramp-x86.c:124
Fatal signal 6 (SIGABRT), code -6 in tid 4051 (amarin.bug56111)
```

这是一个已知的问题中报告[56111](#)。解决方法是禁用 LLVM。

常见问题

2018/10/19 • [Edit Online](#)

安装和设置

应安装哪些 Android SDK 包？

安装 Android SDK 不会自动包括用于开发的所有最小所需的包。虽然各开发人员需要不同，本指南讨论了通常会使用 Xamarin.Android 进行开发所需的包。

可以在哪里设置 Android SDK 位置？

本指南介绍了 Android SDK，其中应适用于大多数组织；的默认设置以及如何根据需要更改这些默认值在 Visual Studio for Mac 或 Visual Studio。

如何更新 Java Development Kit (JDK) 版本？

本文演示了如何更新 Windows 和 mac 上的 Java 开发工具包 (JDK) 版本

可以使用 Java 开发工具包 (JDK) 版本 9 或更高版本？

Xamarin.Android 需要 JDK 8 或 Microsoft 移动 OpenJDK。本文列出了可能会看到是否安装 JDK 9 或更高版本，以及有关签入的 JDK 版本说明一些常见错误消息。

如何手动安装 Xamarin.Android.Support 包所需的 Android 支持库？

本指南提供有关安装示例步骤 `Xamarin.Android.Support.v4` 支持库上 Windows 和 mac。

在 Windows 上调试 Android 需要哪些 USB 驱动程序？

若要在 Android 设备上调试时开发 Windows；需要安装兼容的 USB 驱动程序。Android SDK 管理器默认情况下添加了用于 Nexus 设备的支持包括"Google USB 驱动程序"。其他设备需要发布的设备制造商的 USB 驱动程序。本指南提供有关查找这些驱动程序还其他测试方法的信息。

是否可以从 Windows VM 连接到在 Mac 上运行的 Android 仿真器？

本指南介绍使用 Android 仿真程序时的方法。

一般问题

如何自动化 Android NUnit 测试项目？

本指南介绍了用于设置 Android NUnit 测试项目，步骤_不_Xamarin.UITest 项目。找不到 Xamarin.UITest 参考线[此处](#)。

如何在 Android.xml 文件中启用 Intellisense？

本指南介绍了如何激活 Visual Studio Intellisense for android.xml 文件。

为何我的 Android 发布版本无法连接到 Internet？

此问题的最常见原因是INTERNET权限将自动包括在调试版本，但必须手动设置为发布版本。本指南介绍如何启用发布版本的权限。

更智能的 Xamarin Android 支持 v4 / v13 NuGet 包

`Support-v4` 和 `Support-v13` 不能使用一起在相同的应用中，即，它们是互相排斥。这是因为 `Support-v13` 实际包含的所有类型和实现 `Support-v4`。如果尝试和引用同一项目中，将会遇到重复的类型错误。

如何解决 PathTooLongException 错误？

此文章介绍了如何解决PathTooLongException生成 Xamarin.Android 项目时可能发生的错误。

不推荐使用

NOTE

下面的文章适用于 Xamarin 的最新版本中已解决的问题。但是，如果最新版本的软件会出现此问题，请提出[新 bug](#)与完整的版本控制信息和完整生成日志输出。

哪个版本的 Xamarin.Android 添加了 Lollipop 支持？

本指南是最初编写 Android L 预览版。Xamarin.Android 4.17 添加 Android L 预览版支持和 Xamarin.Android 4.20 添加了 Android Lollipop 支持。

Android.Support.v7.AppCompat - 未找到与给定名称匹配的资源：属性“android:actionModeShareDrawable”

如果缺少某些所需的 Android SDK 包，在旧版 Xamarin 中可能发生此错误。

调整 Android Designer 的 Java 内存参数

启动时使用的默认内存参数 `java` 处理有关 Android 设计器可能与某些系统配置不兼容。开始使用 Xamarin Studio 5.7.2.7 和 Xamarin 的 Visual Studio 3.9.344 这些设置可以在每个项目上自定义。

Android.Resource.designer.cs 文件不更新

在 Xamarin.Studio 5.1 中的 bug 以前通过部分或完全删除.csproj 文件中的 xml 代码损坏.csproj 文件。这将导致重要部分的 Android 生成系统（例如，更新 Android.Resource.designer.cs）失败。截至 5.1.4 稳定版本在年 7 月 15 日，已修复此错误;但在许多情况下的项目文件必须按本指南中所述手动修复。

应安装哪些 Android SDK 包？

2018/10/26 • [Edit Online](#)

安装 Android SDK 不会自动包括用于开发的所有最小所需的包。虽然各开发人员需要不同，通常会使用 Xamarin.Android 进行开发所需的以下包：

工具

从 SDK 管理器中的工具文件夹中安装最新的工具：

- Android SDK 工具
- Android SDK 平台工具
- Android SDK 生成工具

Android 平台

安装适用于已设置为最小值和目标设置的 Android 版本“平台 SDK”。

示例：

- 目标 API 23
- 最小 API 23

只需为 API 23 安装 SDK 平台

- 目标 API 23
- 最小 API 15

需要安装适用于 API 15 和 23 的 SDK 平台。请注意，不需要安装的最小值和目标之间的 API 级别（即使您是反向移植到这两种 API 级别）。

系统映像

这些只是需要你想要使用来自 Google 的開箱 Android 仿真程序。有关详细信息，请参阅[Android 仿真程序安装程序](#)

附加程序

Android SDK Extras 通常不是必需的;但必须了解它们，因为它们可能根据你的用例很有用。

其他阅读材料

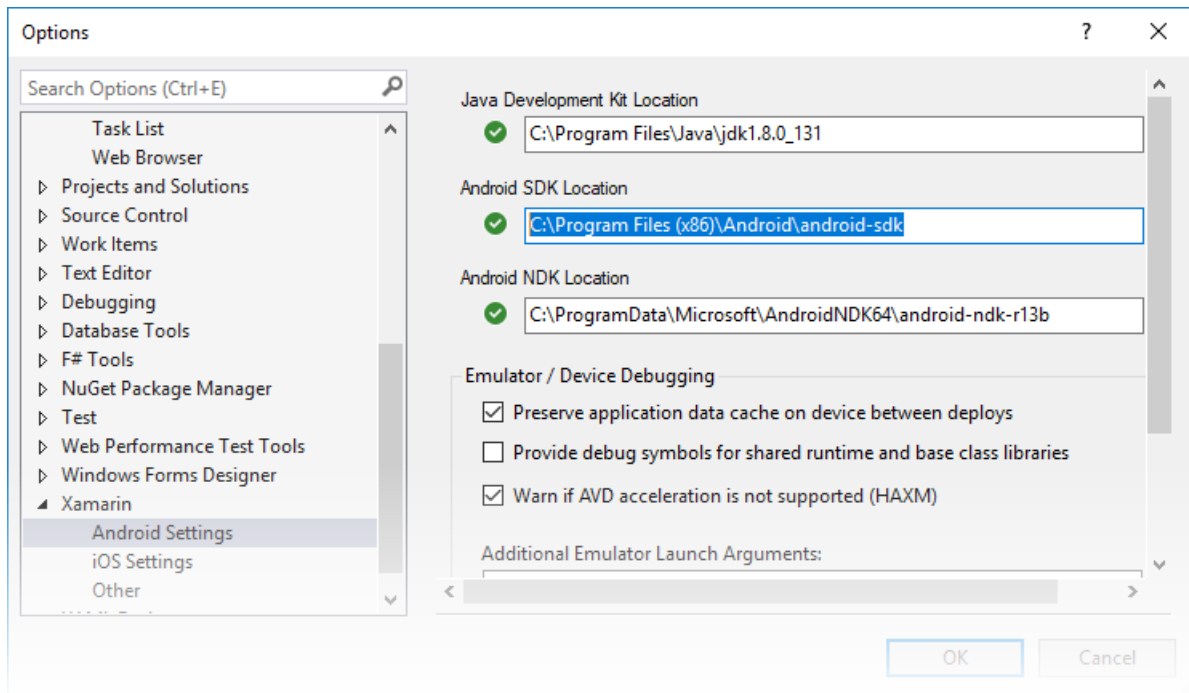
以下指南介绍了这些选项，并进入的详细信息的不同程序包 SDK 管理器具有可用：[Android SDK 管理器安装程序指南](#)

可以在哪里设置 Android SDK 位置？

2018/10/26 • [Edit Online](#)

- [Visual Studio](#)
- [Visual Studio for Mac](#)

在 Visual Studio 中，导航到 **工具 > 选项 > Xamarin > Android** 设置可以查看和设置 Android SDK 位置：



每个路径的默认位置如下所示：

- Java 开发工具包位置：

C:\程序文件\Java\jdk1.8.0_131

- Android SDK 位置：

C:\Program Files (x86)\Android\android-sdk

- Android NDK 位置：

C:\ProgramData\Microsoft\AndroidNDK64\android-ndk-r13b

请注意，NDK 版本号可能不同。而不是，如 **android ndk r13b**，它可能是早期版本，如 **android ndk r10e**。

若要设置 Android SDK 位置，请输入到 Android SDK 目录的完整路径 **Android SDK** 位置框。可以导航到在文件资源管理器中的 Android SDK 位置、将路径从地址栏中，复制并粘贴到此路径 **Android SDK** 位置框。例如，如果你的 Android SDK 位置位于 **c:\用户\用户名\AppData\本地\Android\Sdk**，清除中的旧路径 **Android SDK** 位置中，粘贴该路径中，单击 **确定**。

如何更新 Java 开发工具包 (JDK) 版本？

2018/10/26 • [Edit Online](#)

本文演示了如何更新 Windows 和 mac 上的 Java 开发工具包 (JDK) 版本

概述

Xamarin.Android 使用 Java 开发工具包 (JDK) 将与用于构建 Android 应用程序和运行 Android 设计器的 Android SDK 进行集成。Android SDK (API 24 及更高版本) 的最新版本需要 JDK 8 (1.8)。或者, 也可以安装[Microsoft Mobile OpenJDK 预览版](#)。Microsoft Mobile OpenJDK 将最终替换 Xamarin.Android 开发的 JDK 8。

若要更新到 Microsoft Mobile OpenJDK, 请参阅[Microsoft Mobile OpenJDK 预览版](#)。若要更新到 JDK 8, 请按照下列步骤:

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 从下载 JDK 8 (1.8) [Oracle 网站](#):



Java Platform, Standard Edition

Java SE 8u111 / 8u112
Java SE 8u111 includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release. Java SE 8u112 is a patch-set update, including all of 8u111 plus additional features (described in the release notes).
[Learn more](#) ▶

Important planned change for MD5-signed JARs
Starting with the April Critical Patch Update releases, planned for April 18 2017, all JRE versions will treat JARs signed with MD5 as unsigned. [Learn more and view testing instructions](#).
For more information on cryptographic algorithm support, please check the [JRE and JDK Crypto Roadmap](#).

- [Installation Instructions](#)
- [Release Notes](#)
- [Oracle License](#)
- [Java SE Products](#)
- [Third Party Licenses](#)
- [Certified System Configurations](#)
- [Readme Files](#)
 - [JDK ReadMe](#)
 - [JRE ReadMe](#)

JDK
[DOWNLOAD](#) ⬇

Server JRE
[DOWNLOAD](#) ⬇

JRE
[DOWNLOAD](#) ⬇

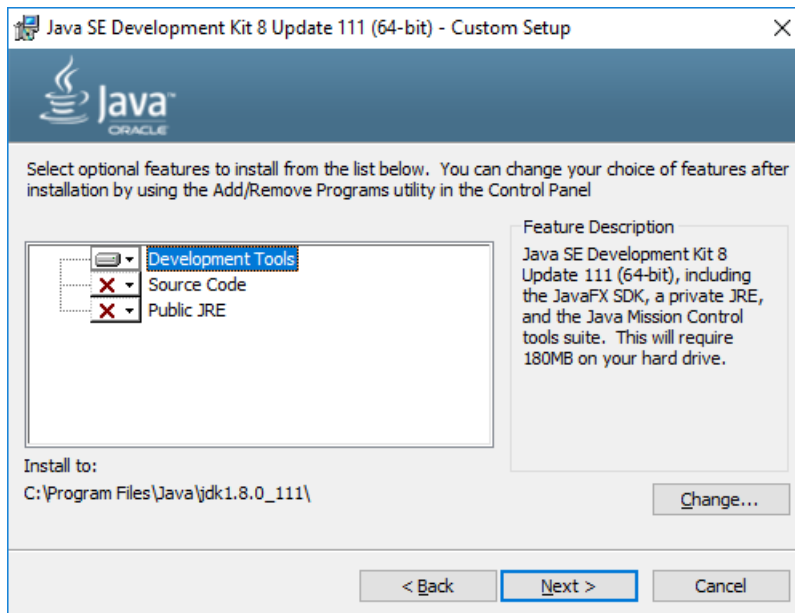
2. 选择要允许的呈现的 64 位版本 [自定义控件](#) Xamarin Android 设计器中:

Java SE Development Kit 8u111
 You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

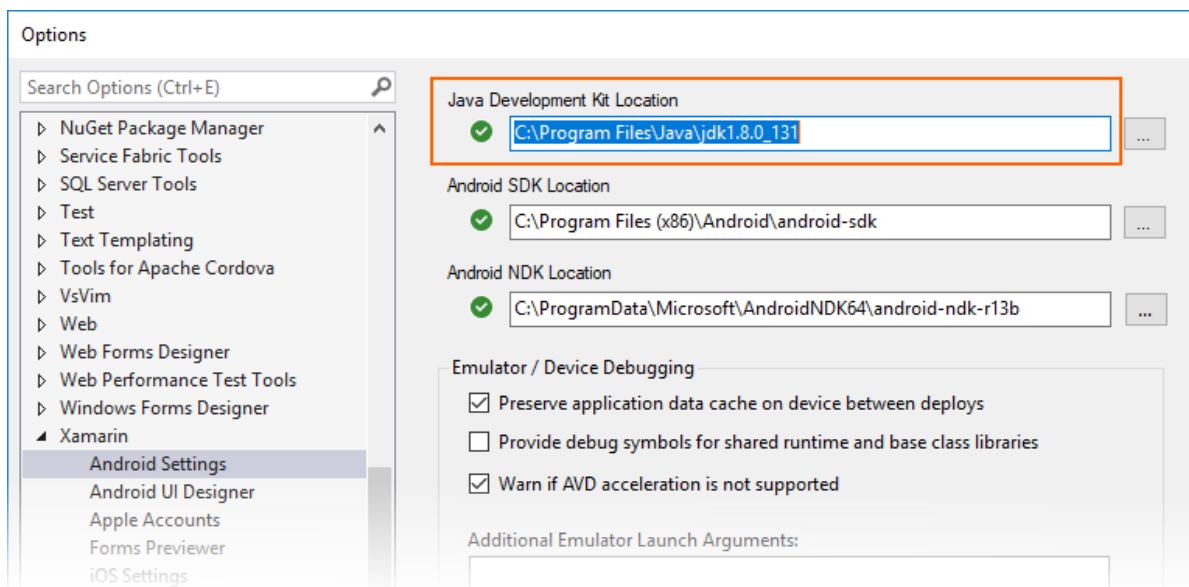
☒ Accept License Agreement ☐ Decline License Agreement

| Product / File Description | File Size | Download |
|-----------------------------|-----------|---|
| Linux ARM 32 Hard Float ABI | 77.78 MB | jdk-8u111-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 74.73 MB | jdk-8u111-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 160.35 MB | jdk-8u111-linux-i586.rpm |
| Linux x86 | 175.04 MB | jdk-8u111-linux-i586.tar.gz |
| Linux x64 | 158.35 MB | jdk-8u111-linux-x64.rpm |
| Linux x64 | 173.04 MB | jdk-8u111-linux-x64.tar.gz |
| Mac OS X | 227.39 MB | jdk-8u111-macosx-x64.dmg |
| Solaris SPARC 64-bit | 131.92 MB | jdk-8u111-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 93.02 MB | jdk-8u111-solaris-sparcv9.tar.gz |
| Solaris x64 | 140.38 MB | jdk-8u111-solaris-x64.tar.Z |
| Solaris x64 | 96.82 MB | jdk-8u111-solaris-x64.tar.gz |
| Windows x86 | 189.22 MB | jdk-8u111-windows-i586.exe |
| Windows x64 | 194.64 MB | jdk-8u111-windows-x64.exe |

3. 运行.exe 和安装开发工具:



4. 打开 Visual Studio 并更新**Java Development Kit** 位置以指向下新的 JDK 工具 > 选项 > **Xamarin** > **Android** 设置 > **Java Development Kit** 位置:



请确保更新的位置后, 重启 Visual Studio。

Xamarin.Android 和 Java 开发工具包 9 或更高版本

2018/10/25 • [Edit Online](#)

本文介绍如何解决 Java 开发工具包 (JDK) 9 或更高版本在 Xamarin.Android 中的错误。

概述

Xamarin.Android 使用 Java 开发工具包 (JDK) 将与用于构建 Android 应用程序和运行 Android 设计器的 Android SDK 进行集成。Android SDK (API 24 及更高版本) 的最新版本需要 JDK 8 (1.8) 或 Microsoft 移动 OpenJDK 预览版。可从 Google 的 Android SDK 工具尚不与 JDK 9 兼容, 因为 Xamarin.Android 不适 JDK 9 或更高版本。

JDK 错误

如果你尝试生成 Xamarin.Android 项目使用的 JDK 8 比更高版本的 JDK 版本, 则会出现显式错误, 指示不支持此版本的 JDK。例如:

```
Building with JDK Version `9.0.4` is not supported. Please install JDK version `1.8.0`. See  
https://aka.ms/xamarin/jdk9-errors
```

若要解决这些错误, 您必须安装 JDK 8 (1.8) 中所述[如何更新 Java 开发工具包 \(JDK\) 版本?](#)。或者, 也可以安装[Microsoft Mobile OpenJDK 预览版](#)Microsoft Mobile OpenJDK Xamarin.Android 开发最终将取代 JDK 8。

正在检查 JDK 版本

您可以检查已通过输入以下命令安装的 Java 版本 (JDK `bin` 目录必须包含在你 `PATH`):

```
java -version
```

如果安装了 JDK 9, 您将看到一条消息, 如下所示:

```
java version "9.0.4"  
Java(TM) SE Runtime Environment (build 9.0.4+11)  
Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11, mixed mode)
```

如果安装 JDK 9 或更高版本, 则必须安装 Java JDK 8 (1.8) 或 Microsoft 移动 OpenJDK 预览版。有关如何安装 JDK 8 的信息, 请参阅[如何更新 Java 开发工具包 \(JDK\) 版本?](#)。有关如何安装 Microsoft Mobile OpenJDK 的信息, 请参阅[Microsoft Mobile OpenJDK 预览版](#)。

请注意, 不需要卸载更高版本的 jdk;但是, 必须确保 JDK 8 而不是更高版本的 JDK 版本, 使用 Xamarin。在 Visual Studio 中, 单击工具 > 选项 > **Xamarin > Android** 设置。如果**Java Development Kit**位置未设置为 JDK 8 位置 (如c:\Program Files\Java\jdk1.8.0_111), 单击更改并将其设置为安装了 JDK 8 的位置。在 Visual Studio for Mac 中, 导航到首选项 > 项目 > **SDK 位置 > Android > Java SDK (JDK)** 然后单击浏览来更新此路径。

使用 JDK 9 的已知的问题

apksigner

没有使用 apksigner 和 JDK 9 中的一个已知的问题 `apksigner.bat` 文件调用 `apksigner.jar` 与 `-Djava.ext.dirs` 而不是 `-classpath` 需要 JDK 9。建议使用 JDK 8 (1.8)。有关如何安装 JDK 8 的信息, 请参阅[如何更新 Java 开发工具包](#)

(JDK) 版本？

如果已安装 JDK 9, 请确保以下路径未设置上你 `PATH` 环境变量, 因为它将仍指向 JDK 9:

`C:\ProgramData\Oracle\Java\javapath`。删除了它, `java-version` 的命令行上应显示 JDK 8。

如何手动安装 Xamarin.Android.Support 包所需的 Android 支持库？

2018/10/26 • [Edit Online](#)

Xamarin.Android.Support.v4 的示例步骤

- [Visual Studio](#)
- [Visual Studio for Mac](#)

下载所需的 Xamarin.Android.Support NuGet 包（例如通过使用 NuGet 包管理器中安装它）。

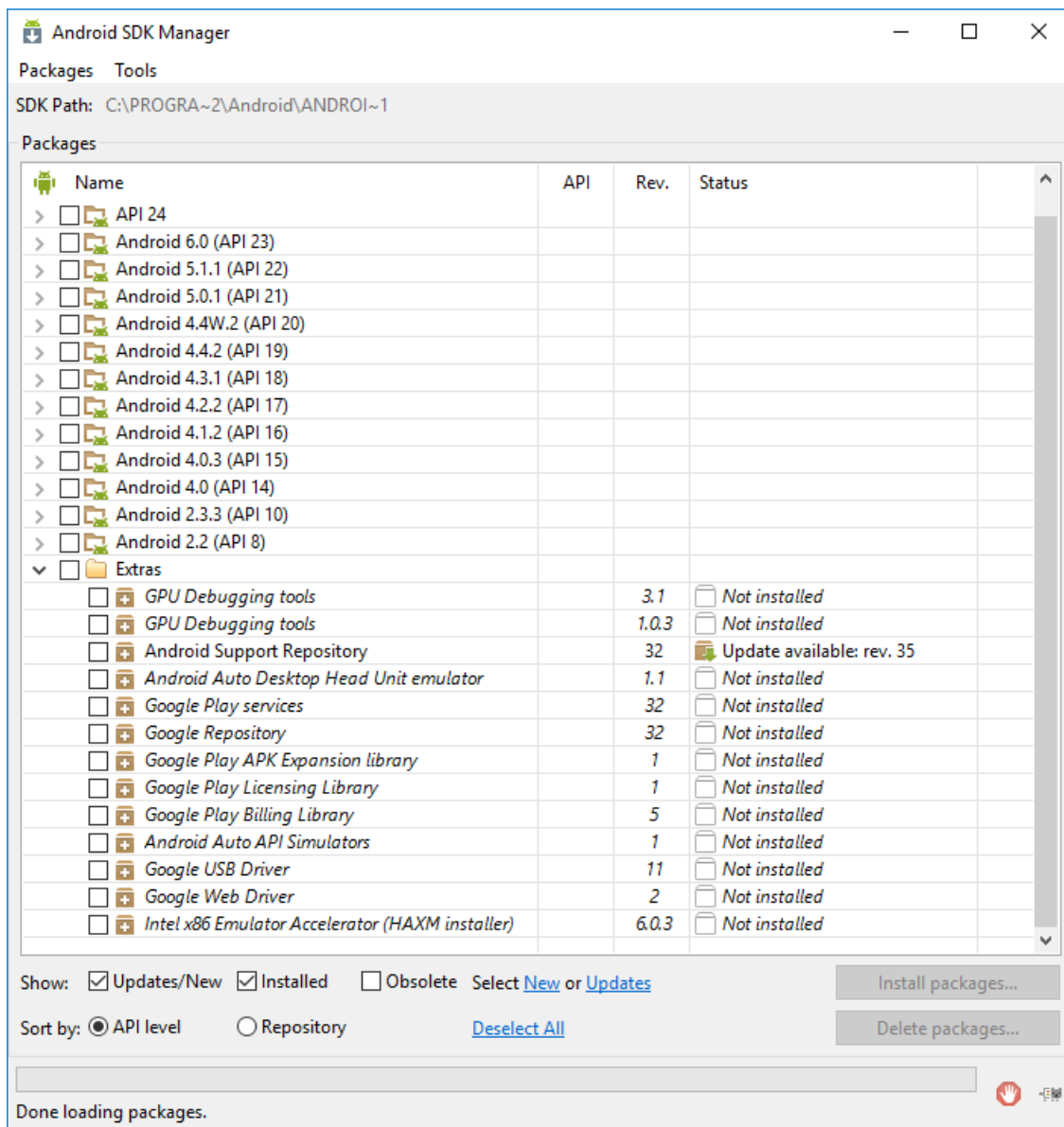
使用 `ildasm` 若要查看哪些版本 **android_m2repository.zip** NuGet 包都需要：

```
ildasm /caverbal /text /item:Xamarin.Android.Support.v4
packages\Xamarin.Android.Support.v4.23.4.0.1\lib\MonoAndroid403\Xamarin.Android.Support.v4.dll | findstr
SourceUrl
```

示例输出：

```
property string 'SourceUrl' = string('https://dl-
ssl.google.com/android/repository/android_m2repository_r32.zip')
property string 'SourceUrl' = string('https://dl-
ssl.google.com/android/repository/android_m2repository_r32.zip')
property string 'SourceUrl' = string('https://dl-
ssl.google.com/android/repository/android_m2repository_r32.zip')
```

下载 **android_m2repository.zip** 从 Google 使用的 URL 返回从 **ildasm**。或者，可以检查哪个版本的 `_Android` 支持存储库_当前已安装 Android SDK 管理器中：



如果版本与匹配所需的 NuGet 包, 然后您无需下载任何新内容。您可以改为重新压缩现有 **m2repository** 目录下的 **extras\android** 中 **_SDK** 路径 (如下所示的 Android 顶部 SDK 管理器窗口)。

计算从返回的 URL 的 MD5 哈希 **ildasm**。设置要使用全部以大写字母, 不留空格的结果字符串的格式。例如, 调整 `$url` 变量, 作为所需, 然后运行下面两行代码 (基于 [原始 C# 从 Xamarin.Android 代码](#)) 在 PowerShell 中:

```
$url = "https://dl-ssl.google.com/android/repository/android_m2repository_r32.zip"
((([System.Security.Cryptography.MD5]::Create()).ComputeHash([System.Text.Encoding]::UTF8.GetBytes($url)) | %{$_.ToString("X02")}) -join "")
```

示例输出:

```
F16A3455987DBAE5783F058F19F7FCDF
```

复制 **android_m2repository.zip** 到 **%LOCALAPPDATA%\Xamarin\zips** 文件夹。重命名要使用从上一个 MD5 哈希计算步骤的 MD5 哈希的文件。例如:

%LOCALAPPDATA%\Xamarin\zips\F16A3455987DBAE5783F058F19F7FCDF.zip

(可选) 将解压缩到的文件 **%LOCALAPPDATA%\Xamarin\Xamarin.Android.Support.v4\23.4.0.0\内容** (创建 **内容\m2repository** 子目录)。如果跳过此步骤, 然后使用库在第一个生成将会稍长因为它将需要完成此步骤。子目录的版本号 (**23.4.0.0** 在此示例中) 不是 NuGet 包版本完全一样。可以使用 **ildasm** 查找正确的版本号:


```
ildasm /caverbal /text /item:Xamarin.Android.Support.v4
packages\Xamarin.Android.Support.v4.23.4.0.1\lib\MonoAndroid403\Xamarin.Android.Support.v4.dll | findstr
/C:"string 'Version'"
```

示例输出：

```
property string 'Version' = string('23.4.0.0')
property string 'Version' = string('23.4.0.0')
property string 'Version' = string('23.4.0.0')
```

其他参考

- [Bug 43245](#) – Inaccurate"下载失败。请下载{0}并将其放{1}目录。"和"请安装包:{0}SDK 安装程序中提供"与 Xamarin.Android.Support 包相关的错误消息

后续步骤

本文档讨论截至 2016 年 8 月的当前行为。本文档中所述的技术不是 xamarin，稳定的测试套件的一部分，因此它在将来可能会损坏。

获取进一步的帮助，请与我们联系，或如果此问题仍即使利用上述信息，请参阅[了可用于 Xamarin 的支持选项？](#)有关联系人选项，建议的信息以及如何如果需要，提交新 bug。

Windows 上调试 Android 需要哪些 USB 驱动程序？

2018/10/26 • [Edit Online](#)

找到 USB 驱动程序

若要在 Android 设备上调试时开发 Windows;需要安装兼容的 USB 驱动程序。Android SDK 管理器默认情况下添加了对如下所述的用于 Nexus 设备支持包括"Google USB 驱动程序": <http://developer.android.com/sdk/win-usb.html>

其他设备需要专门发布的设备制造商的 USB 驱动程序。本指南中包含的最常见的制造商一些链接:
<http://developer.android.com/tools/extras/oem-usb.html>

替代项

具体取决于 manufacturer, 很难跟踪所需的确切 USB 驱动程序。在包括使用 Android 仿真程序或使用外部测试服务的 Windows 开发一些测试 Android 应用的替代方案。其中包括:

- [App Center 测试](#)-云服务在数百个实际 Android 设备上运行测试。
- [适用于 Android 的 Visual Studio 模拟器](#)
- [在 Android Emulator 上调试](#)

它可能连接到 Android 仿真程序运行在 Mac 上从 Windows VM？

2018/10/26 • [Edit Online](#)

若要连接到从 Windows 虚拟机的 Mac 上运行的 Android 仿真程序，请使用以下步骤：

1. 在 mac 上启动仿真程序
2. 终止 `adb` 在 Mac 上的服务器：

```
adb kill-server
```

3. 请注意在仿真程序正在侦听环回网络接口上的 2 个 TCP 端口：

```
lsof -iTCP -sTCP:LISTEN -P | grep 'emulator\|qemu'
```

| | | | | | | | | | |
|-----------|-------|---------|-----|------|--------------------|-----|-----|----------------|----------|
| emulator6 | 94105 | macuser | 20u | IPv4 | 0xa8dacfb1d4a1b51f | 0t0 | TCP | localhost:5555 | (LISTEN) |
| emulator6 | 94105 | macuser | 21u | IPv4 | 0xa8dacfb1d845a51f | 0t0 | TCP | localhost:5554 | (LISTEN) |

奇数端口将用于连接到 `adb`。另请参阅

<http://developer.android.com/tools/devices/emulator.html#emulatorenetworking>。

4. 选项 1: 使用 `nc` 若要转发的入站 TCP 数据包收到外部在端口 5555 上（或您喜欢的任何其他端口）到环回接口上的奇数端口（**127.0.0.1 5555**在此示例中），将转发出站数据包重新另一种方法：

```
cd /tmp
mkfifo backpipe
nc -kl 5555 0<backpipe | nc 127.0.0.1 5555 > backpipe
```

只要 `nc` 命令运行保持在终端窗口中，将按预期方式转发数据包。可以在终端窗口中退出键入控制 C `nc` 命令完成后使用仿真程序。

（选项 1 通常要比容易选项 2，尤其是当系统首选项 > 安全性和隐私 > 防火墙已打开。）

选项 2: 使用 `pfctl` 若要重定向来自端口的 TCP 数据包 `5555`（或您喜欢的任何其他端口）上共享网络接口的环回接口上的奇数端口（`127.0.0.1:5555` 在此示例中）：

```
sed '/rdr-anchor/a rdr pass on vmnet8 inet proto tcp from any to any port 5555 -> 127.0.0.1 port 5555'
/etc/pf.conf | sudo pfctl -ef -
```

此命令将设置端口转发使用 `pf packet filter` 系统服务。换行很重要。请确保以使它们保持不变时复制粘贴。您还需要调整接口名称从 `vmnet8` 如果您使用 Parallels。`vmnet8` 是特殊名称 NAT 设备有关共享网络 VMWare Fusion 中的模式。在 Parallels 中适当的网络接口很可能 `vnic0`。

5. 从 Windows 计算机连接到模拟器：

```
C:\> adb connect ip-address-of-the-mac:5555
```

替换为"ip-地址---mac"与 Mac 的 IP 地址例如作为由列出 `ifconfig vmnet8 | grep 'inet '`。如果需要将为

5555 与您喜欢第 4 步中的其他端口。(注意：一种方法来获取命令行访问权 `adb` 是通过 [工具 > Android > Android Adb 命令提示符](#) Visual Studio 中。)

备用技术使用 `ssh`

如果已启用_远程登录名_在 Mac 上则可以使用 `ssh` 端口转发以连接到模拟器。

1. 在 Windows 上安装 SSH 客户端。一种方法是安装[Git 的 Windows](#)。`ssh` 命令将推出 **Git Bash** 命令提示符。
2. 按照步骤 1-3 上述若要启动仿真程序，请终止 `adb` 在 Mac 上的服务器和识别的仿真程序端口。
3. 运行 `ssh` 若要设置 Windows 上的本地端口之间的双向端口转发的 Windows 上 (`localhost:15555` 在此示例中) 和 Mac 的环回接口上的奇数的仿真程序端口 (`127.0.0.1:5555` 在此示例中):

```
C:\> ssh -L localhost:15555:127.0.0.1:5555 mac-username@ip-address-of-the-mac
```

替换 `mac-username` 与你的 Mac 用户名列出的 `whoami`。替换为 `ip-address-of-the-mac` mac 的 IP 地址

4. 连接到仿真程序在 Windows 上使用的本地端口：

```
C:\> adb connect localhost:15555
```

(注意：一种简单方法获取命令行访问权 `adb` 是通过 [工具 > Android > Android Adb 命令提示符](#) 在 Visual Studio 中。)

小小的警告：如果使用端口 `5555` 对于本地端口 `adb` 会想在 Windows 上本地运行仿真程序。这不会时出现问题导致在 Visual Studio 中，但在 Visual Studio for Mac 会导致应用启动后立即退出。

一种替代方式使用 `adb -H` 尚不支持

从理论上讲，另一种方法是使用 `adb` 的内置功能，可连接到 `adb` 远程计算机上运行的服务器 (请参阅示例 <http://stackoverflow.com/a/18551325>)。但 Xamarin.Android IDE 扩展当前不提供某种方法来配置该选项。

联系信息

本文档讨论截至 2016 年 3 月的当前行为。本文档中所述的技术不是 xamarin，稳定的测试套件的一部分，因此它在将来可能会损坏。

如果您注意到，该技术将不再适用，或者如果您注意到文档中的任何其他错误，随意添加下面的论坛主题的讨论：<http://forums.xamarin.com/discussion/33702/android-emulator-from-host-device-inside-windows-vm>。谢谢！

如何自动化 Android NUnit 测试项目？

2018/10/26 • [Edit Online](#)

NOTE

本指南介绍如何自动化 Android NUnit 测试项目，不是 Xamarin.UITest 项目。找不到 Xamarin.UITest 参考线[此处](#)。

当您创建单元测试应用 (**Android**) Visual Studio 项目中的 (或**Android** 单元测试Visual Studio for Mac 中的项目)，则此项目将不会自动默认情况下运行测试。若要在目标设备上运行 NUnit 测试，可以创建[Android.App.Instrumentation](#)通过使用以下命令启动的子类：

```
adb shell am instrument
```

以下步骤介绍了此过程：

1. 创建名为的新文件**TestInstrumentation.cs**:

```
using System;
using System.Reflection;
using Android.App;
using Android.Content;
using Android.Runtime;
using Xamarin.Android.NUnitLite;

namespace App.Tests {

    [Instrumentation(Name="app.tests.TestInstrumentation")]
    public class TestInstrumentation : TestSuiteInstrumentation {

        public TestInstrumentation (IntPtr handle, JniHandleOwnership transfer) : base (handle,
        transfer)
        {
        }

        protected override void AddTests ()
        {
            AddTest (Assembly.GetExecutingAssembly ());
        }
    }
}
```

在此文件中，[Xamarin.Android.NUnitLite.TestSuiteInstrumentation](#) (从[Xamarin.Android.NUnitLite.dll](#)) 来创建子类化 `TestInstrumentation`。

2. 实现[TestInstrumentation](#)构造函数和[AddTests](#)方法。 `AddTests` 方法控制实际执行哪些测试。
3. 修改 `.csproj` 文件以添加**TestInstrumentation.cs**。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build" ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  ...
  <ItemGroup>
    <Compile Include="TestInstrumentation.cs" />
  </ItemGroup>
  <Target Name="RunTests" DependsOnTargets="_ValidateAndroidPackageProperties">
    <Exec Command="&quot;$( _AndroidPlatformToolsDirectory)adb&quot; $(AdbTarget) $(AdbOptions) shell
am instrument -w $( _AndroidPackage)/app.tests.TestInstrumentation" />
  </Target>
  ...
</Project>
```

4. 使用以下命令以运行单元测试。替换 `PACKAGE_NAME` 与应用程序的包名称 (可以在应用中找到包名称 `/manifest/@package` 属性位于 **AndroidManifest.xml**):

```
adb shell am instrument -w PACKAGE_NAME/app.tests.TestInstrumentation
```

5. 或者, 您可以修改 `.csproj` 文件以添加 `RunTests` MSBuild 目标。这样就可以调用的命令如下所示的单元测试:

```
msbuild /t:RunTests Project.csproj
```

(请注意, 使用此新的目标不是必需的; 早期版本 `adb` 而不是可以使用命令 `msbuild`。)

有关使用详细信息 `adb shell am instrument` 命令以运行单元测试, 请参阅 Android 开发人员[使用 ADB 运行测试](#)主题。

NOTE

与 [Xamarin.Android 5.0](#) release, 默认包名称 Android 可调用包装器将基于要导出的类型的程序集限定名称的 MD5SUM。这样, 相同的完全限定名称, 以提供从两个不同的程序集, 并且不使打包错误。因此, 请确保你使用 `Name` 属性上的

`Instrumentation` 属性生成可读的 ACW/类名称。

必须在使用 ACW 名称 `adb` 上述命令。重命名重构 C# 类将因此需要修改 `RunTests` 命令, 以使用正确的 ACW 名称。



为何无法我的 Android 发布版本连接到 Internet？

2018/10/26 • [Edit Online](#)

原因

此问题的最常见原因是**INTERNET**权限将自动包括在调试版本，但必须手动设置为发布版本。这是因为 Internet 权限用于允许将调试程序附加到进程，针对"DebugSymbols"所述[此处](#)。

修复

若要解决此问题，可能需要在 Android 清单中的 Internet 权限。这可以通过在清单编辑器或该清单的 sourcecode 完成：

- 在编辑器中修复：在 Android 项目，请转到**属性-> AndroidManifest.xml-> 所需的权限**，并检查**Internet**
- 在 Sourcecode 中修复：在源编辑器中打开 AndroidManifest 并将权限标记内的添加 `<Manifest>` 标记：

```
<Manifest>
...
<uses-permission android:name="android.permission.INTERNET" />
</Manifest>
```


更智能的 Xamarin Android 支持 v4 / v13 NuGet 包

2018/10/26 • [Edit Online](#)

有关 Android 支持库

Google 已创建了支持库以使新功能可用于较旧版本的 Android。一般情况下，支持库中将给出的版本号是相互兼容的最低 Android API 级别其名称 (例如：支持 v4 只能在 API 级别 4 及更高版本。有关详细信息这[Stack Overflow 讨论](#))。

两个支持库：`Support-v4` 和 `Support-v13` 不能使用一起在相同的应用中，即，它们是互相排斥。这是因为 `Support-v13` 实际包含的所有类型和实现 `Support-v4`。如果尝试在同一项目中同时引用将会遇到重复的类型错误。

引用的问题

由于 `Support-v4` 变得如此受欢迎，很多第三方库现在依赖于它。它们可能已选择依赖于支持 v13 相反，但更常见依赖_v4_由于这样使用这些第三方库的任何应用的支持一直到 4 的 API 级别的选项。

如果 Xamarin 第三方库引用 `Xamarin.Android.Support.v4.dll` 绑定到 `Support-v4`，任何使用此库的应用还必须引用 `Xamarin.Android.Support.v4.dll`。这就存在问题时的相同应用程序还想要使用的功能的一些 `Xamarin.Android.Support.v13.dll` 绑定到 `Support-v13`。如果引用这两种绑定，将会遇到重复的类型错误。

类型转发 v4 绑定程序集

若要解决此问题，我们创建了一个特殊 `Xamarin.Android.Support.v4.dll` 程序集不具有实现，但只需 `[assembly: TypeForwardedTo (...)]` 转发的所有特性 `Support-v4` 类型中实现 `Xamarin.Android.Support.v13.dll` 程序集。

这意味着开发人员可以引用此_的类型转发_在其应用中将满足对引用的程序集 `Xamarin.Android.Support.v4.dll` 任何第三方库，同时仍允许通过 `Xamarin.Android.Support.v13.dll` 要在应用中使用。

NuGet 协助

虽然开发人员可以手动添加正确的引用必要时，我们将能够使用 NuGet 来帮助选择正确的程序集 (任一普通_v4_绑定或类型转发_v4_程序集) 时安装 NuGet 包。

因此，`Xamarin.Android.Support.v4` NuGet 包现在包含以下逻辑：

如果应用面向 API 级别 13 (Gingerbread 3.2) 或更高版本：

- `Xamarin.Android.Support.v13` NuGet 会自动添加为依赖项
- 的类型转发 `Xamarin.Android.Support.v4.dll` 将在项目中引用

如果您的应用程序面向任何低于 API 级别 13，则会正常 `Xamarin.Android.Support.v4.dll` 在项目中引用的绑定。

我是否必须使用支持 v13？

如果您的应用程序面向 API 级别 13 或更高版本并且你选择使用 `Xamarin Android Support-v4` NuGet 包，则 `Xamarin Android Support v13` NuGet 包是必需的依赖关系。

我们认为非常小 (两个 jar 文件不同的 17 kb) 的应用程序大小增加很值得的兼容性和更少这会导致的问题。

如果您是使用偏激 `Support-v4` 在应用中的目标 API 级别 13 或更高版本，则可以始终手动下载 `.nupkg`、提取和引

用程序集。

如何解决 PathTooLongException 错误？

2018/10/26 • [Edit Online](#)

原因

在 Xamarin.Android 项目中生成的路径名称可能会相当长。例如，可以在生成期间生成如下所示的路径：

C:\Some\Directory\Solution\Project\obj\Debug\library_projects\Xamarin.Forms.Platform.Android\library_project_imports\assets

在 Windows 上 (其中的路径的最大长度是260 个字符)、一个 **PathTooLongException** 无法生成时生成的项目，如果生成的路径超过了最大长度。

修复

从 Xamarin.Android 8.0 开始 `UseShortFileNames` MSBuild 属性可以设置为绕过此错误。当此属性设置为 `True` (默认值是 `False`)，生成过程将使用较短的路径名称以减少生成的可能性 **PathTooLongException**。例如，当 `UseShortFileNames` 设置为 `True`，在以上路径缩短为类似于以下路径：

C:\某些\目录\解决方案\项目\obj\调试\lp\1\jl\资产

若要设置此属性，请将以下 MSBuild 属性添加到项目 **.csproj** 文件：

```
<PropertyGroup>
  <UseShortFileNames>True</UseShortFileNames>
</PropertyGroup>
```

如果设置此标志不能解决 **PathTooLongException** 错误，另一种方法是指定常见的中间输出根通过设置你的解决方案中的项目 `IntermediateOutputPath` 中项目 **.csproj** 文件。尝试使用相对较短的路径。例如：

```
<PropertyGroup>
  <IntermediateOutputPath>C:\Projects\MyApp</IntermediateOutputPath>
</PropertyGroup>
```

有关设置生成属性的详细信息，请参阅[Build 过程](#)。

哪个版本的 Xamarin.Android 添加了 Lollipop 支持？

2018/10/26 • [Edit Online](#)

注意：本指南最初编写 Android L 预览版。

- [Xamarin.Android 4.17](#) 添加 Android L 预览版的支持。
- [Xamarin.Android 4.20](#) 添加了 Android Lollipop 支持。

Xamarin 才主动支持 Xamarin 工具的当前稳定版本。提供以下信息"作为-是"较旧版本的工具。有关 Xamarin 版本的最新信息，请查看[此处](#)。

"缺少 android.jar API 级别 21"Android L 预览版

- [Visual Studio](#)
- [Visual Studio for Mac](#)

以下错误消息（或类似）可能会出现：

```
Error 1 Could not find android.jar for API Level 21.
```

此消息表示未安装 Android SDK 平台的 API 级别 21。可以将其安装在 Android SDK 管理器 (工具 > 打开 Android SDK 管理器...), 或更改 Xamarin.Android 项目以面向已安装的 API 版本。

有几个针对此问题的解决方法：

1. 更改你的项目，以便它面向 API 19 或更低。
2. 文件夹重命名你 android 21 android 21 为 android l。（充其量，这应只用作临时性修补程序，并且它根本不工作很好地。）

%LOCALAPPDATA%\Android\android-sdk\platforms\android-21

3. 暂时降级回 Android API 级别 21"L"预览 [1]:

- a. 删除 **%LOCALAPPDATA%\Android\android sdk\平台\android 21**
- b. 提取 [1] 到 **c:\用户\\AppData\本地\Android\android sdk\平台** 创建 **android-L** 文件夹。

[1] - https://dl-ssl.google.com/android/repository/android-L_r04.zip

Android.Support.v7.AppCompat-未找到资源与给定名称相匹配: attr android:actionModeShareDrawable

2018/10/26 • [Edit Online](#)

1. 请确保下载最新的其他功能, 以及 Android 5.0 (API 21) SDK 通过 Android SDK 管理器。
2. 请确保使用设置为 21 compileSdkVersion 进行编译你的应用程序。为也 21, 可以选择性地设置 targetSdkVersion。
3. 如果你需要一个以前的版本, 如 API 19, 请下载 Nuget 页上找到的各自版本:

<https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/>

请注意: 如果你手动安装此通过程序包管理器控制台, 请确保还安装相同版本的 Xamarin.Android.Support.v4

<https://www.nuget.org/packages/Xamarin.Android.Support.v4/>

堆栈溢出参考: <http://stackoverflow.com/questions/26431676/appcompat-v721-0-0-no-resource-found-that-matches-the-given-name-attr-andro>

请参阅

- [应安装哪些 Android SDK 包?](#)

调整 Android Designer 的 Java 内存参数

2018/11/14 • [Edit Online](#)

启动时使用的默认内存参数 `java` 处理有关 Android 设计器可能与某些系统配置不兼容。

从 Xamarin Studio 5.7.2.7 (和更高版本、Visual Studio for Mac) 和 Visual Studio Tools for Xamarin 3.9.344, 可以基于每个项目自定义这些设置。

新的 Android 设计器属性和相应的 Java 选项

下面的属性名称对应于所指示的 `java` 命令行选项

- **AndroidDesignerJavaRendererMinMemory** -Xms
- **AndroidDesignerJavaRendererMaxMemory** -Xmx
- **AndroidDesignerJavaRendererPermSize** -XX:MaxPermSize
- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 在 Visual Studio 中打开解决方案。
2. 在解决方案资源管理器中选择一个地的每个 Android 项目, 然后单击[显示所有文件](#)两次上每个项目。你可以跳过项目不包含任何 `.axml` 布局文件。此步骤将确保每个项目目录包含 `.csproj.user` 文件。
3. 退出 Visual Studio。
4. 找到 `.csproj.user` 为每个步骤 2 中的项目文件。
5. 编辑每个 `.csproj.user` 文本编辑器中的文件。
6. 添加 any 或 all 中的新 Android 设计器的内存属性 `<PropertyGroup>` 元素。可以使用现有 `<PropertyGroup>` 或新建一个。下面是一个完整示例 `.csproj.user` 文件, 其中包含所有 3 个属性设置为其默认值:

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <ProjectView>ProjectFiles</ProjectView>
  </PropertyGroup>
  <PropertyGroup>
    <AndroidDesignerJavaRendererMinMemory>128m</AndroidDesignerJavaRendererMinMemory>
    <AndroidDesignerJavaRendererMaxMemory>750m</AndroidDesignerJavaRendererMaxMemory>
    <AndroidDesignerJavaRendererPermSize>350m</AndroidDesignerJavaRendererPermSize>
  </PropertyGroup>
</Project>
```

7. 保存并关闭所有更新后的 `.csproj.user` 文件。
8. 重新启动 Visual Studio 并重新打开你的解决方案。

Android Resource.designer.cs 文件不会更新

2018/10/26 • [Edit Online](#)

NOTE

在 Xamarin Studio 5.1.4 和更高版本中解决此问题。但是, 如果 Visual Studio for Mac 发生问题, 请提出[新 bug](#)与完整的版本控制信息和完整生成日志输出。

在 Xamarin.Studio 5.1 中的 bug 以前通过部分或完全删除.csproj 文件中的 xml 代码损坏.csproj 文件。这将导致重要部分的 Android 生成系统(例如, 更新 Android Resource.designer.cs)失败。截至 5.1.4 稳定版本在年 7 月 15 日, 已修复此错误;但在许多情况下的项目文件按下面所述手动修复。

正在修复项目文件的两种可能方法

二者之一:

1. 创建一个全新的 Xamarin.Android 应用程序项目, 将设置所有项目属性以匹配旧的项目, 并添加到项目的所有资源、源文件、等。

或

2. 创建原始项目的.csproj 文件的备份副本, 然后在文本编辑器中, 将其打开, 并从完全生成的.csproj 文件中添加缺少的元素中返回。

如果这没有解决问题

与这些元素一起进行实验之后, 您会注意到, 之后将其添加回元素和重新生成项目, 将更新 Resource.designer.cs 文件, 但然后您可能仍需要关闭并重新打开解决方案以获取代码完成功能来识别Resource.designer.cs 中包含的新类型。

解决库安装错误

2018/11/1 • [Edit Online](#)

在某些情况下，可能会安装 Android 支持库时收到错误。本指南提供了一些常见的错误的解决方法。

概述

同时生成 Xamarin.Android 应用程序项目，Visual Studio 或 Visual Studio for Mac 尝试下载并安装依赖项的库时可能会生成错误。许多这些错误被由于网络连接问题、文件损坏或版本控制问题。本指南介绍了最常见的支持库安装错误，并提供解决这些问题，让您再次生成的应用程序项目的步骤。

下载 m2Repository 时的错误

你可能会看到 **m2repository** 引用的 Android 支持库或 Google Play 服务 NuGet 包时的错误。该错误消息类似于以下内容：

```
Download failed. Please download https://dl-ssl.google.com/android/repository/android_m2repository_r16.zip and extract it to the C:\Users\mgm\AppData\Local\Xamarin\Android.Support.v4\22.2.1\content directory.
```

此示例适用于 **android_m2repository_r16**，但你可能会看到此相同的错误消息的不同版本，如 **android_m2repository_r18** 或 **android_m2repository_r25**。

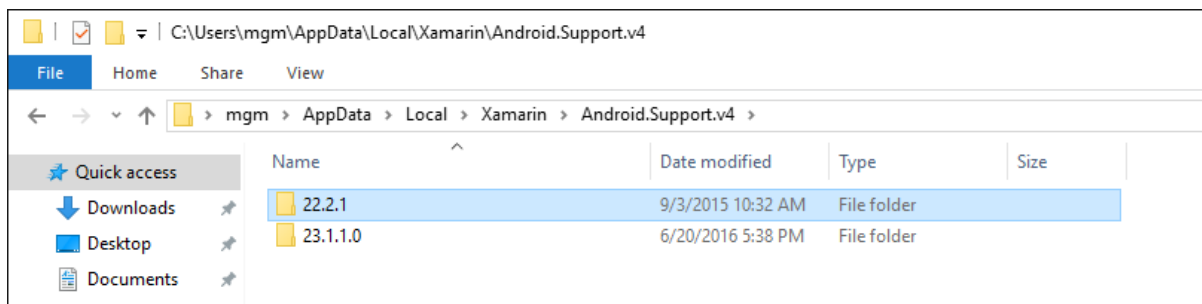
从 m2repository 错误自动恢复

通常情况下，可以通过删除有问题的库，并按照这些步骤重新生成来纠正此问题：

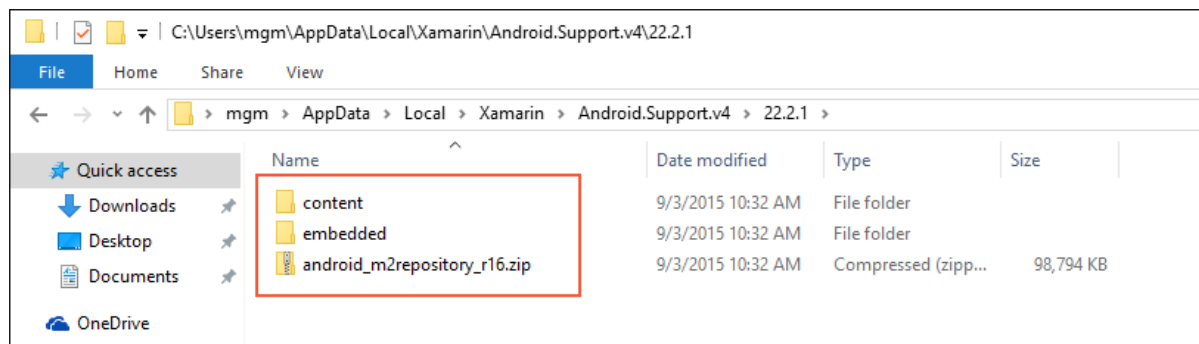
1. 导航到您的计算机上的支持库目录：

- 在 Windows，支持库位于 **c:\用户\用户名\AppData\本地\Xamarin**。
- Mac OS X 上支持库位于 **/Users/用户名/.local/share/Xamarin**。

2. 找到对应的错误消息的库和版本文件夹。例如，上面的错误消息的库和版本文件夹位于 **Android.Support.v4\22.2.1**：



3. 删除版本文件夹的内容。请务必删除 **.zip** 文件并将内容并嵌入子目录在该文件夹中的。如上所示，文件和子目录在此屏幕截图所示的示例错误消息 (内容，嵌入，并 **android_m2repository_r16.zip**) 到删除：



请注意，务必删除整个此文件夹的内容。尽管此文件夹可能最初包含框 **android_m2repository_r16.zip** 文件，此文件可能已部分下载或已损坏。

4. 重新生成项目-这样做将导致生成过程重新下载缺失的库。

在大多数情况下，这些步骤将解决生成错误并允许您继续。如果删除此库不能解决生成错误，必须手动下载并安装**android_m2repository_r_nn.zip**文件中的下一节所述。

手动下载 m2repository

如果你已尝试使用上述自动恢复步骤，并仍然存在生成错误，则可以手动下载**android_m2repository_r_nn.zip**文件（使用 web 浏览器）并安装它根据以下步骤。如果您在开发计算机上没有 internet 访问权限，但无法下载存档使用另一台计算机，则也可以使用此过程。

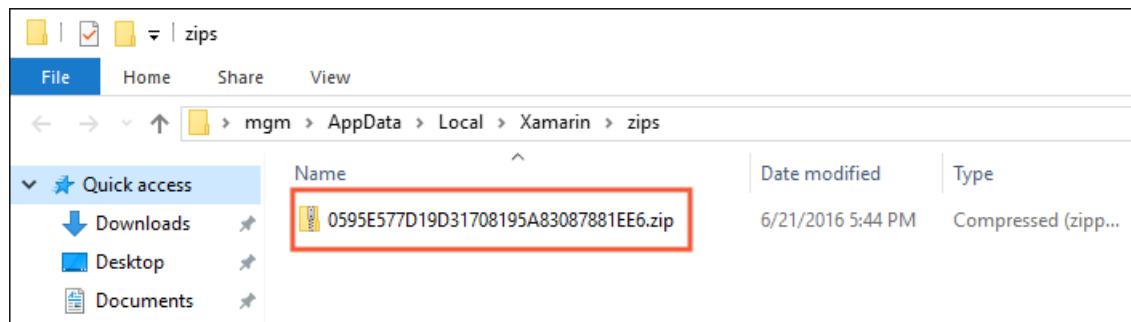
1. 下载**android_m2repository_r_nn.zip**对应的错误消息文件-（以及相应的 MD5 哈希的每个链接下面的列表提供了链接URL）:

- [android_m2repository_r33.zip](#) – 5FB756A25962361D17BBE99C3B3FCC44
- [android_m2repository_r32.zip](#) – F16A3455987DBAE5783F058F19F7FCDF
- [android_m2repository_r31.zip](#) – 99A8907CE2324316E754A95E4C2D786E
- [android_m2repository_r30.zip](#) – 05AD180B8BDC7C21D6BCB94DDE7F2C8F
- [android_m2repository_r29.zip](#) – 2A3A8A6D6826EF6CC653030E7D695C41
- [android_m2repository_r28.zip](#) – 17BE247580748F1EDB72E9F374AA0223
- [android_m2repository_r27.zip](#) – C9FD4FCD69D7D12B1D9DF076B7BE4E1C
- [android_m2repository_r26.zip](#) – 8157FC1C311BB36420C1D8992AF54A4D
- [android_m2repository_r25.zip](#) – 0B3F1796C97C707339FB13AE8507AF50
- [android_m2repository_r24.zip](#) – 8E3C9EC713781EDFE1EFBC5974136BEA
- [android_m2repository_r23.zip](#) – D5BB66B3640FD9B9C6362C9DB5AB0FE7
- [android_m2repository_r22.zip](#) – 96659D653BDE0FAEDB818170891F2BB0
- [android_m2repository_r21.zip](#) – CD3223F2EFE068A26682B9E9C4B6FBB5
- [android_m2repository_r20.zip](#) – 650E58DF02DB1A832386FA4A2DE46B1A
- [android_m2repository_r19.zip](#) – 263B062D6EFAA8AEE39E9460B8A5851A
- [android_m2repository_r18.zip](#) – 25947AD38DCB4865ABEB61522FAFDA0E
- [android_m2repository_r17.zip](#) – 49054774F44AE5F35A6BA9D3C117EFD8
- [android_m2repository_r16.zip](#) – 0595E577D19D31708195A83087881EE6

如果m2repository存档不会显示此表中，可以通过预先计算创建的下载 URL <https://dl-ssl.google.com/android/repository/> 的名称m2repository下载。例如，使用 https://dl-ssl.google.com/android/repository/android_m2repository_r10.zip 若要下载 android_m2repository_r10.zip。

- 文件重命名为上述表中所示的下载 URL 的相应 MD5 哈希。例如，如果您下载android_m2repository_r25.zip，其重命名为0B3F1796C97C707339FB13AE8507AF50.zip。如果下载的文件在下载 URL 的 MD5 哈希表中未显示，则可以使用[online MD5 生成器](#)将 URL 转换为 MD5 哈希字符串。
- 将文件复制到 Xamarin zips文件夹：
 - 在 Windows 中，此文件夹位于c:\用户\用户名\AppData\本地\Xamarin\zips。
 - Mac OS X 上此文件夹位于/Users/用户名/.local/share/Xamarin/zips。

例如，下面的屏幕截图显示了结果时android_m2repository_r16.zip下载并将其重命名为其下载 URL 在 Windows 上的 MD5 哈希：



如果此过程无法解决生成错误，则必须手动下载android_m2repository_r_nn.zip文件，将其解压缩，然后在下一节中所述安装其内容。

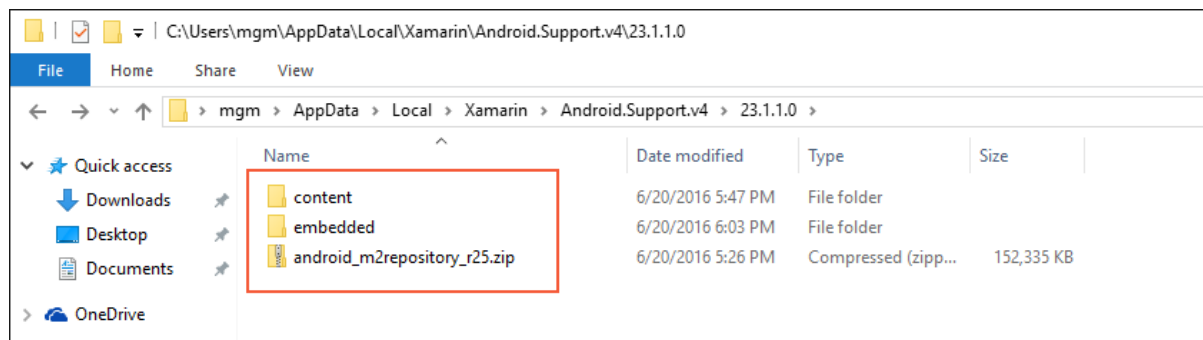
手动下载和安装 m2repository 文件

从恢复的完全手动过程m2repository错误需要下载android_m2repository_r_nn.zip文件（使用 web 浏览器），解压缩它，并将其内容复制到您的计算机上的支持库目录。在以下示例中，我们将恢复该错误消息中：

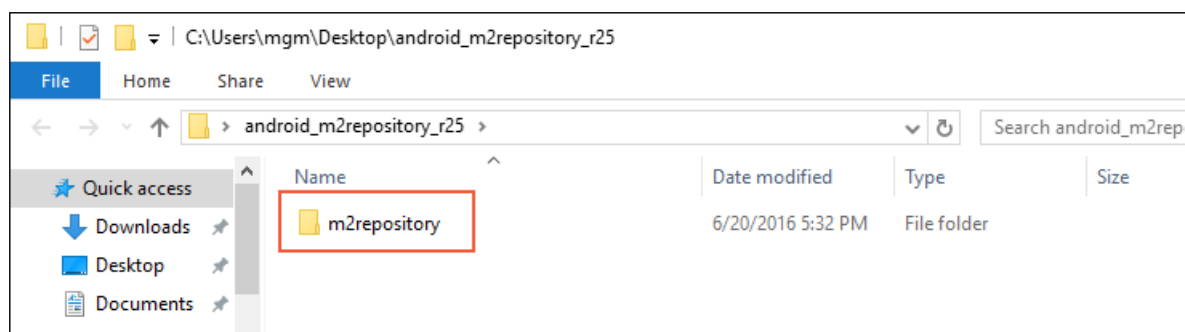
```
Unzipping failed. Please download https://dl-ssl.google.com/android/repository/android_m2repository_r25.zip and extract it to the C:\Users\mgm\AppData\Local\Xamarin\Android.Support.v4\23.1.1\content directory.
```

使用以下步骤下载m2repository并安装它的内容：

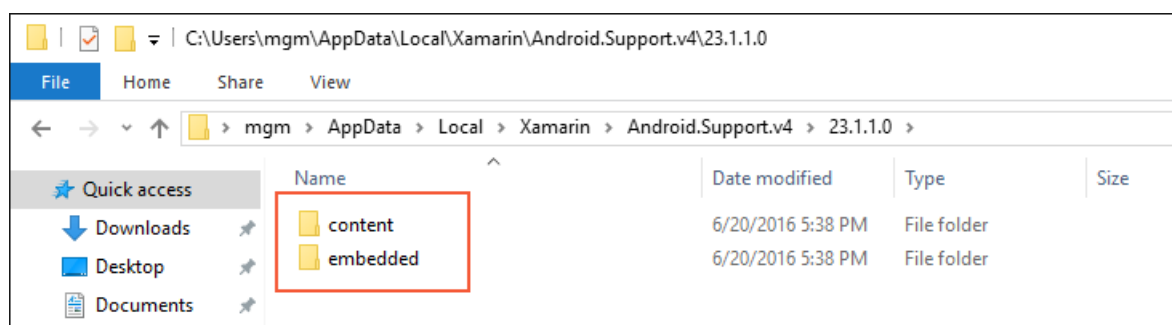
- 删除与该错误消息对应的库文件夹的内容。例如，在上面的错误消息中将删除的内容c:\用户\用户名\AppData\本地\Xamarin\Android.Support.v4\23.1.1.0。如前文所述，必须删除此目录的全部内容：



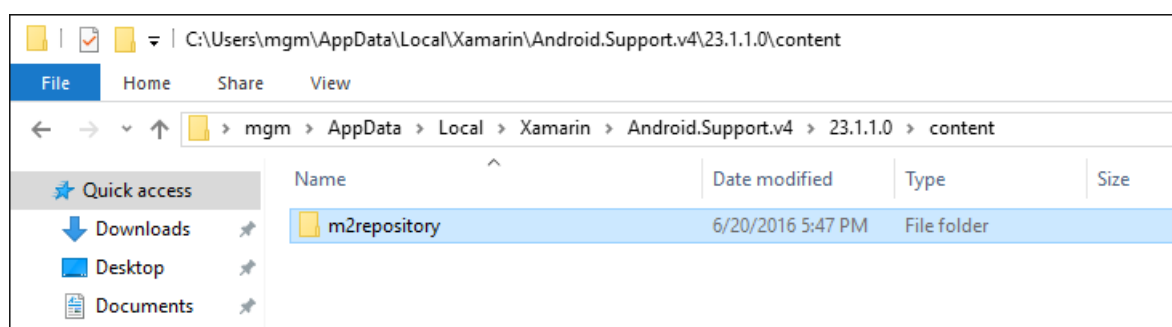
- 下载android_m2repository_r_nn.zip文件从 Google 对应于错误消息（请参阅上一节中的链接中的表）。
- 提取此 .zip存档到任何位置（例如桌面）。这应创建一个目录的名称对应 .zip存档。在此目录中，你应找到名为的子目录m2repository:



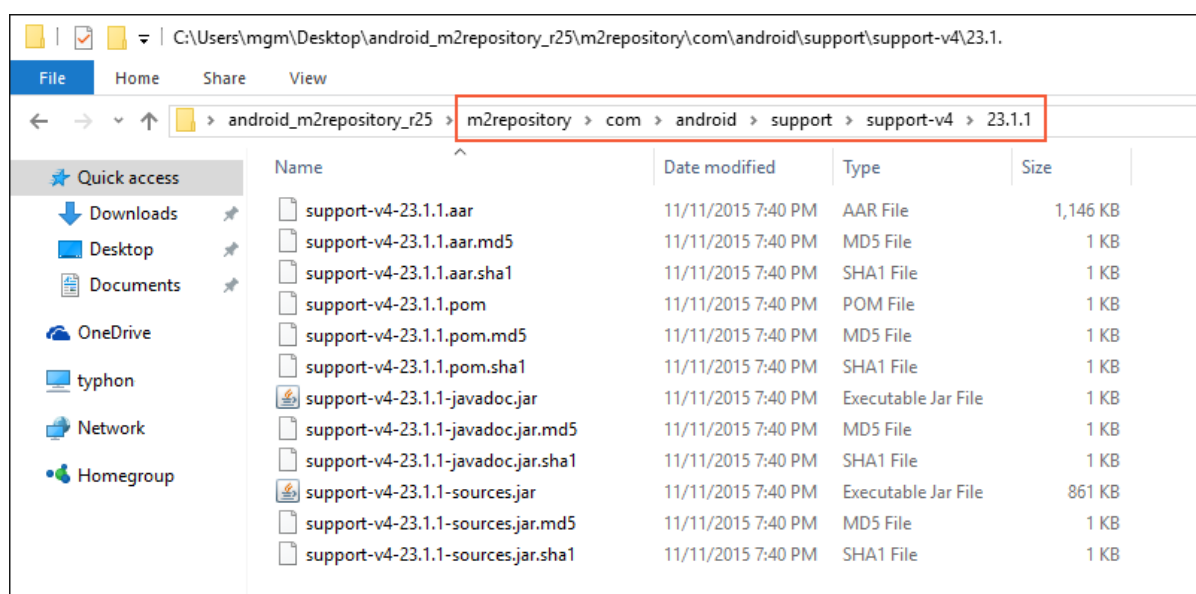
4. 在带有版本的库目录中清除在步骤 1 中，重新创建内容并嵌入子目录。例如，下面的屏幕截图显示了内容并嵌入子目录中创建23.1.1.0文件夹android_m2repository_r25.zip:



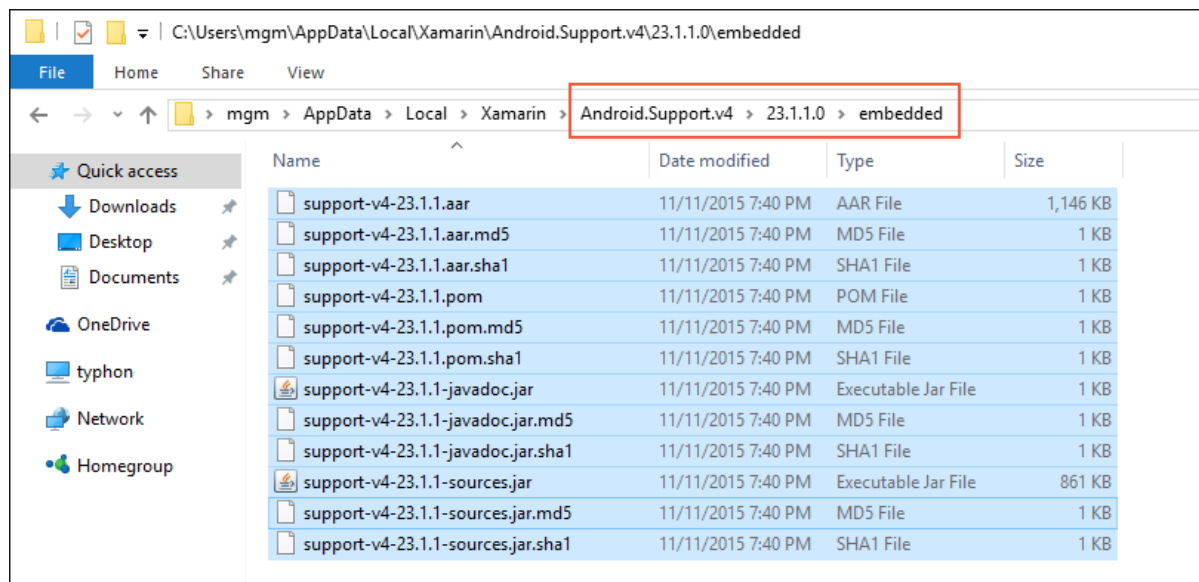
5. 复制m2repository提取 .zip到内容在上一步中创建目录:



6. 在提取 .zip 目录中，浏览到m2repository\com\android\支持\支持 v4和打开相应的文件夹上面创建的版本编号 (在此示例中， 23.1.1):



7. 将所有文件复制到此文件夹中嵌入在步骤 4 中创建目录:



8. 验证通过将复制所有文件。嵌入目录现在应包含文件如 **.jar**, **.aar**, 以及 **.pom**。

9. 将任何提取的内容解压缩 **.aar**文件到嵌入目录。在 Windows 中, 追加 **.zip**扩展 **.aar**文件, 打开它, 然后将复制到内容嵌入目录。在 macOS 上, 将解压缩 **.aar**使用的文件解压缩在终端命令 (例如, 解压缩 **file.aar**)。

此时, 已手动安装缺少的组件和你的项目应正确生成错误。如果没有, 请验证是否已下载**m2repository .zip**存档版本完全对应的版本中的错误消息, 并验证是否已安装在其内容更正的位置, 如上述步骤中所述。

总结

本文介绍了如何从自动下载和安装依赖项库期间可以进行的常见错误中恢复。它介绍了如何删除有问题的库和重新生成项目, 作为一种方式重新下载, 然后重新安装库。它介绍了如何下载库并将其在安装**zips**文件夹。它还介绍了用于手动下载和安装所需的文件作为一种方法来解决, 不能通过自动方式解决更多的步骤。

对 Android SDK 工具的更改

2018/10/26 • [Edit Online](#)

对 Android SDK 管理的已安装的 API 级别和 Avd 的方式的更改。

对 Android SDK 工具的更改

在最新版本的 Android SDK Tools, Google 已删除的现有 AVD 和 SDK 管理器以支持新的 CLI (命令行接口) 工具。**Android**程序已删除, 并在 Visual Studio for Mac 和 Visual Studio Tools for Xamarin 的较旧版本的 Google GUI (图形用户界面) 管理器将不再有效的 Android SDK Tools 版本 25.2.5 过去。例如, 尝试使用**android**通过命令行程序将导致错误消息如下所示:

```
The "android" command is deprecated.  
For manual SDK, AVD, and project management, please use Android Studio.  
For command-line tools, use tools\bin\sdkmanager.bat  
and tools\bin\avdmanager.bat
```

以下各节说明如何管理的 Android SDK 和 Android 虚拟设备使用 Android SDK 25.3.0 及更高版本。

用户界面工具

Visual Studio 和 Visual Studio for Mac 现提供 Xamarin 已停止使用基于 Google GUI 的管理器的替代项:

- 若要下载 Android SDK 工具、平台和其他组件所需的开发 Xamarin.Android 应用程序, 请使用[Xamarin Android SDK 管理器](#)而不是旧版 Google SDK 管理器。
- 若要创建和配置 Android 虚拟设备, 请使用[Android 设备管理器](#)而不是旧版 Google 仿真器管理器。

这些工具在功能上等效于基于 Google GUI 的管理器它们替换。

CLI 工具

或者, 可以使用 CLI 工具来管理和更新你的仿真程序和 Android SDK。以下程序现在构成 Android SDK 工具在命令行接口:

sdkmanager

在添加了: Android SDK Tools 25.2.3 (2016 年 11 月) 和更高版本。

还有一个名为的新程序**sdkmanager**中 **/bin** Android SDK 文件夹。使用此工具来维护 Android SDK 在命令行。有关使用此工具的详细信息, 请参阅[sdkmanager](#)。

avdmanager

在添加了: Android SDK Tools 25.3.0 (2017 年 3 月) 和更高版本。

还有一个名为的新程序**avdmanager**中 **/bin** Android SDK 文件夹。此工具用于为 Android 仿真程序中维护 Avd。有关使用此工具的详细信息, 请参阅[avdmanager](#)。

降级

你可以降级你**Android SDK Tools**通过安装以前版本的 Android sdk 版本[Android 开发人员网站](#)。

使用旧的 GUI

您仍然可以通过运行使用原始 GUI **android**程序内你工具文件夹, 只要你位于**Android SDK Tools**版本**25.2.5**或更低。

相关链接

- [Android SDK 安装](#)
- [Android 设备管理器](#)
- [了解 Android API 级别](#)
- [SDK Tools 发行说明 \(Google\)](#)
- [sdkmanager](#)
- [avdmanager](#)

Xamarin.Android 错误矩阵

2018/10/31 • [Edit Online](#)

错误参考

本文档提供有关从 Xamarin 上的各个错误代码的一些信息。

| 类别 | 描述 |
|--------|-----------------------|
| XA0xxx | mandroid 错误 |
| XA1xxx | 文件复制 / 符号链接 (相关项目) 错误 |
| XA2xxx | 链接器错误 |
| XA3xxx | AOT 错误 |
| XA4xxx | 代码生成错误 |
| XA5xxx | Gcc 高级版和工具链错误 |
| XA6xxx | mandroid 内部工具错误 |
| XA7xxx | 保留 |
| XA8xxx | 保留 |
| XA9xxx | 许可错误 |

错误代码

XA0xxx 错误

| 错误代码 | 描述 |
|--------|---|
| XA0000 | 意外的错误-请填写 bug 报表 。 |
| XA0001 | -devname 提供不需要任何特定于设备的操作。 |
| XA0002 | 无法分析环境变量{0}。 |
| XA0003 | 应用程序名称{0}.exe 与 SDK 或产品程序集 (.dll) 名称冲突。 |
| XA0004 | 新 refcounting 逻辑要求 sgen 太启用。 |
| XA0005 | 输出目录{0}不存在。 |
| XA0006 | 不没有在任何开发平台{0}, 使用--平台 = 来指定 SDK 的平台 |

| 错误代码 | 描述 |
|--------|--|
| XA0007 | 根程序集{0}不存在。 |
| XA0008 | 应提供一个根只有程序集。 |
| XA0009 | 加载程序集时出错：{0}。 |
| XA0010 | 无法分析命令行参数：{0}。 |
| XA0011 | {0} 生成时所针对的较新的运行时 ({1}) 不是 MonoTouch 支持。 |
| XA0012 | 不完整的数据用于完成{0}。 |
| XA0013 | 分析支持需要 sgen 太启用。 |
| XA0014 | iOS{0}不支持面向 ARMv6 的构建应用程序。 |
| XA0020 | 无法确定 mandroid 路径。 |
| XA0100 | EmbeddedNativeLibrary{0}是无效的 Android 应用程序项目中。 请改为使用 AndroidNativeLibrary。 |

XA1xxx 错误

| 错误代码 | 描述 |
|--------|---|
| XA1001 | 在指定目录中找不到应用程序。 |
| XA1002 | 无法创建符号链接，复制文件。 |
| XA1003 | 无法终止该应用程序{0}。您可能需要手动终止应用程序。 |
| XA1004 | 无法获取安装的应用程序的列表。 |
| XA1005 | 无法终止该应用程序{0}上的设备{1}: {2}。您可能需要手动终止应用程序。 |
| XA1006 | 无法安装该应用程序{0}上的设备{1}: {2}。 |
| XA1007 | 未能启动应用程序{0}上的设备{1}: {2}。您仍可以通过点击它手动启动该应用程序。 |
| XA1008 | 无法启动模拟器：{0}。 |
| XA1009 | 无法将复制程序集 '{0}'到{1}: {2}。 |
| XA1010 | 无法加载程序集 '{0}': {1}。 |
| XA1011 | 无法将添加缺少的资源文件:{0}。 |
| XA1101 | 无法启动应用。 |

| 错误代码 | 描述 |
|--------|---|
| XA1102 | 未能附加到应用程序（若要终止它）：{0}。 |
| XA1103 | 无法分离。 |
| XA1104 | 发送数据包失败：{0}。 |
| XA1105 | 意外的响应类型。 |
| XA1106 | 无法在设备上获取应用程序的列表：请求已超时。 |
| XA1107 | 应用程序启动失败。 |
| XA1201 | 无法加载模拟器：{0}。 |
| XA1301 | 本机库 '{0}({1})' 已被忽略，因为它不匹配当前生成 architecture(s) ({2})。 |

XA2xxx 错误

| 错误代码 | 描述 |
|--------|--------------------------------|
| XA2001 | 无法链接程序集。 |
| XA2002 | 无法解析引用：{0}。 |
| XA2003 | 选项{0}由于禁用了链接都将被忽略。 |
| XA2004 | 额外的链接器定义文件{0}找不到。 |
| XA2005 | 定义从{0}无法分析。 |
| XA2006 | 对元数据项目引用{0}(在中定义{1}) 从{2}无法解析。 |

XA3xxx 错误

这些是 AOT 错误。

| 错误代码 | 描述 |
|--------|--|
| XA3001 | 可以不 AOT 的程序集{0}。 |
| XA3002 | AOT 限制：方法{0}必须是静态的因为它用 [MonoPInvokeCallback] 修饰。 |
| XA3003 | 冲突-调试和-llvm 选项。软调试被禁用。 |

XA4xxx 错误

这些是代码生成错误。

| 错误代码 | 描述 |
|--------|---|
| XA4001 | 主模板中找不到 expanded{0}。 |
| XA4101 | 在注册机构无法生成类型的签名{0}。 |
| XA4102 | 在注册机构找到无效的类型{0}中方法的签名{2}。使用{1}相反。 |
| XA4103 | 在注册机构找到无效的类型{0}中方法的签名{2}：类型实现 INativeObject，但没有采用两个构造函数 (IntPtr, bool) 参数。 |
| XA4104 | 在注册机构不能封送类型的返回值{0}中方法的签名{1}。 |
| XA4105 | 在注册机构不能封送的类型参数{0}中方法的签名{1}。 |
| XA4106 | 在注册机构不能封送结构的返回值{0}中方法的签名{1}。 |
| XA4107 | 在注册机构不能封送的类型参数{0}中方法的签名{1}。 |
| XA4108 | 在注册机构不能获取托管类型的 ObjectiveC 类型{0}。 |
| XA4109 | 未能编译生成的注册机构代码。请提出 bug 报表 。 |
| XA4110 | 在注册机构不能封送类型的输出参数{0}中方法的签名{1}。 |
| XA4111 | 在注册机构不能生成的签名类型{0}中方法{1}。 |
| XA4200 | 对于 class 类型只能生成 ACW 的。 |
| XA4201 | 无法确定类型的 JNI 名称{0}。 |
| XA4203 | 指定的类型名称必须是完全限定的。 |
| XA4204 | 无法解析接口类型{0}。是否缺少程序集引用？ |
| XA4205 | [ExportField] 只能在不带参数的方法。 |
| XA4206 | 不能在泛型类型上使用 [导出]。 |
| XA4207 | 不能在泛型类型上使用 [ExportField]。 |
| XA4208 | [Java.Interop.ExportFieldAttribute] 不能返回 void 的方法上使用。 |
| XA4209 | 未能创建类 JavaTypeInfo:{0}由于{1}。 |
| XA4210 | 需要使用 ExportAttribute 或 ExportFieldAttribute 时将添加对 Mono.Android.Export.dll 的引用。 |
| XA4211 | AndroidManifest.xml //uses-sdk/@android:targetSdkVersion '{0}' is \$(TargetFrameworkVersion) 小于{1}。使用 API-{1} ACW 编译。 |

XA5xxx 错误

| 错误代码 | 描述 |
|--------|---|
| XA5101 | 缺少{0}编译器。请安装 Android NDK。 |
| XA5102 | 无法从程序集为本机代码的转换。请提出 bug 报表 。 |
| XA5103 | 未能将文件编译{0}。请提出 bug 报表 。 |
| XA5201 | 本机链接失败。请查看提供给 gcc 高级版的用户标志：{0} |
| XA5202 | 本机链接失败。请查看生成日志。 |
| XA5303 | 本机链接警告：{0}。 |
| XA5300 | Android SDK 找不到或未完全安装。 |
| XA5301 | 缺少去除工具。请安装 Xcode 命令行工具组件。 |
| XA5302 | 缺少 dsymutil 工具。请安装 Xcode 命令行工具组件。 |
| XA5203 | 未能生成调试符号（dSYM 目录）。请查看生成日志。 |
| XA5204 | 无法将最小化最终二进制文件。请查看生成日志。 |
| XA5205 | 缺少 aapt 工具。请安装 Android SDK 生成工具包。 |
| XA5206 | {0}。Android 资源目录{1}不存在。 |
| XA5207 | {0}。Java 库文件{1}不存在。 |
| XA5208 | 下载失败。请下载{0}并将其放{1}目录。 |
| XA5209 | 解压缩失败。请下载{0}并将其解压缩到{1}目录。 |
| XA5210 | {0}。本机库文件{1}不存在。 |

XA6xxx 错误

| 错误代码 | 描述 |
|--------|---------------------------------|
| XA6001 | Cecil 的正在运行的版本不支持最小化的程序集。 |
| XA6002 | 不删除程序集{0}。 |
| XA6003 | UnauthorizedAccessException 消息。 |

XA9xxx 错误

这些错误代码是许可和激活错误。

XA9000

原因：许可证已过期

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 警告 | 警告 | 警告 | 警告 | 警告 |

XA9001

原因：试用版已过期

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 警告 | 警告 | 警告 | 警告 | 警告 |

XA9002

原因：AndroidJavaSource

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 确定 | 确定 | 确定 | 确定 |

原因：AndroidJavaLibrary

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 确定 | 确定 | 确定 | 确定 |

如果绑定程序集具有嵌入.jar, 这是捕获在包时, 不生成时间。

原因：AndroidNativeLibrary

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 确定 | 确定 | 确定 | 确定 |

XA9003

原因：System.Runtime.Serialization

检查期间：包

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

原因：System.ServiceModel.Web

检查期间：包

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

原因：Mono.Data.Tds

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

这被引用 **System.Data.dll**, 这允许的

原因：Mono.Android.Export

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 确定 | 确定 | 确定 | 确定 |

XA9004

原因：-分析

检查期间：包

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

XA9005

原因：大小限制 (32 kb)。

检查期间：包

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 确定 | - | - | - |

XA9006

原因：System.Data.SqlClient 命名空间。

检查期间：包

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

XA9008

原因：从命令行生成。

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 确定 | 确定 | 确定 |

XA9009

原因：缺少序列号。

检查期间: 纠结

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9010

原因: 无效 ProductId。

检查期间: 生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

等效于 XA9018。

XA9011

原因: 未能更新许可证文件 (到新的文件格式)。

检查期间: 激活

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9012

原因: 无 internet

检查期间: 激活

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9013

原因: 未知的错误

检查期间: 激活

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9014

原因: 无效激活代码

检查期间: 激活

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9017

原因: 激活服务器不会返回有效的许可证。

检查期间: 激活

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| 错误 | 错误 | 错误 | 错误 | 错误 |

XA9018

原因：许可证无效

检查期间：生成

| STARTER | 独立 | BUSINESS(TRIAL) | 业务 | 企业 |
|---------|----|-----------------|----|----|
| - | - | 错误 | - | - |

Android 穿戴设备

2018/10/26 • [Edit Online](#)

Android 穿戴设备是 android 的专为智能手表等可穿戴设备版本。本部分包括有关如何安装和配置穿戴设备开发, 创建第一个穿戴设备设备, 并可以引用以创建你自己 Wear 应用的示例的列表的分步演练所需的工具的说明。

入门

引入了 Android Wear、介绍如何安装和配置计算机进行穿戴设备开发和提供了有助于创建和运行仿真程序或在穿戴设备上的第一个 Android Wear 应用的步骤。

用户界面

介绍了 Android Wear 特定控件, 并提供指向演示如何使用这些控件的示例。

平台功能

在本部分中的文档介绍特定于 Android Wear 的功能。此处, 您会发现本主题介绍如何创建 WatchFace。

屏幕大小

预览并优化您的用户界面为可用的屏幕尺寸。

部署和测试

介绍如何将 Android Wear 应用部署到 Android Wear 设备或 Android 仿真程序配置的损耗。它还包括调试提示和有关如何设置开发计算机和 Android 设备之间的蓝牙连接的信息。

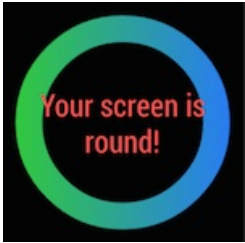

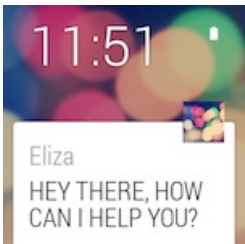

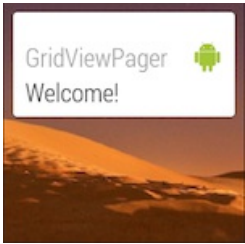
Wear Api

Android 开发人员站点提供有关关键 Wear Api 详细的信息, 如[可穿戴活动](#), [意向](#), [身份验证](#), [复杂性](#), [呈现的复杂情况](#), [通知](#), [视图](#), 并且[WatchFace](#)。

示例

您可以找到大量[示例](#)使用 Android Wear (或直接转到[github](#))。



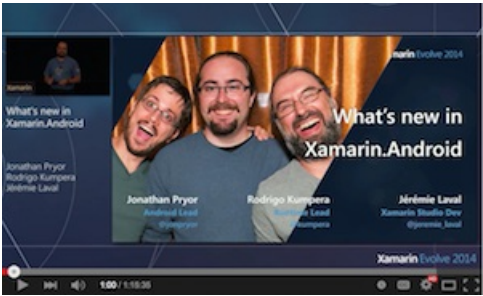
| 示例 | 描述 | 屏幕快照 |
|------------------------------|---|---|
| SkeletonWear | 可穿戴项目, 包括 GridViewPager 和交互式通知的基础知识的简单示例。 |  |

| 示例 | 描述 | 屏幕快照 |
|---------------------------------|--|---|
| WatchViewStub | 检测到屏幕形状并自动加载正确的布局的 WatchViewStub 控件的简单演示。请参阅的 WatchViewStub 工作原理 Resources/layout/main_activity.xml 布局。 |  |
| RecipeAssistant | Wear 通知页, 方案步骤形式的演示。RecipeService.cs 中创建通知。 |  |
| ElizaChat | 与"个人助理"交互的有趣的示例调用 Eliza, 穿戴设备交互式通知用于创建使用内置的响应的会话。 |  |
| GridViewPager | GridViewPager 实现 2D 导航模式, 其中在用户轻扫垂直, 然后水平导航选项和内容。 |  |
| WatchFace | WatchFace 是自定义的观察人脸与模拟样式小时、分钟和第二个指针。此示例演示如何创建监视人脸服务用于绘制当前时间和句柄的环境模式和可见性更改事件。它包括一个广播的接收器, 侦听的时区更改并自动将相应地更新时间。 |  |

视频

请查看[这些视频链接](#)讨论 Xamarin.Android 使用穿戴设备的支持：

| 描述 | 屏幕快照 |
|----|------|
|----|------|

| 描述 | 屏幕快照 |
|---|---|
| <p>Android L 和很多 – Android L 开发者预览版引入了大量的新 Api, 开发人员能够充分利用, 包括 Material Design、通知和新动画, 仅举几例。</p> |  |
| <p>C#是在我和我的眼睛: Google 玻璃效果和 Android Wear – 可穿戴计算可能看起来像未来 (或检查器小工具集), 但许多人已经立即利用未来 ! C#开发人员知道这和已有的工具和技能, 以利用可穿戴设备 (从发展 2014) 的功能。</p> |  |
| <p>什么是 Xamarin.Android 中的新增功能 – Android L、Android Wear、Android TV、Android Auto、Material Design 和将来使用。此 mean 到您为 Xamarin 开发人员 ? 从发展 2014年。</p> |  |

开始使用 Android 穿戴设备

2018/10/26 • [Edit Online](#)

在本部分中的指南介绍 Android Wear、介绍如何安装和配置计算机进行穿戴设备开发和提供帮助您创建并运行第一个 Android Wear 应用的步骤。

穿戴设备简介

提供的 Android Wear 的基本概述，描述其主要功能、列出了一些更受欢迎的 Android Wear 设备，并提供指向有关其他参考资料的基本 Google Android Wear 文档。

设置和安装

提供的安装步骤和准备你的计算机和设备的 Android Wear 开发所需的配置详细信息。

你好，穿戴设备

本演练提供了创建，用于处理按钮单击事件并单击计数器显示在穿戴设备上的小 Android Wear 项目的分步说明。

Android 穿戴设备简介

2018/10/26 • [Edit Online](#)

Google 的 Android Wear 的引入, 已不再限制为只是手机和平板电脑谈到开发优秀的 Android 应用程序。对 Android Wear Xamarin.Android 的支持使您可以运行 C# 上手腕代码! 本简介提供的 Android Wear 的基本概述, 描述其主要功能, 并提供了 Android Wear 2.0 中提供的功能的概述。它列出了一些更受欢迎的 Android Wear 设备, 并提供有关其他参考资料的基本 Google Android Wear 文档的链接。

概述

Android 穿戴设备在各种设备, 包括第一代 Motorola 360、LG 的 G 观看和 Samsung 齿轮 Live 上运行。此外具有其他功能, 包括内置的 GPS 和脱机音乐播放发布了第二个生成, 包括 Sony 的 SmartWatch 3。对于 Android Wear 2.0, Google 已与协作 LG 的两个新监视: LG 监视运动和 LG 监视样式。



Xamarin.Android 5.0 及更高版本支持 Android Wear (API 20) 我们 Android 4.4W 通过支持和 NuGet 包, 它将添加其他特定于穿戴设备的 UI 控件。Xamarin.Android 5.0 及更高版本还包括用于打包穿戴设备应用程序的功能。也将适用于 Android Wear 2.0 本指南后面所述 NuGet 程序包。

Android 穿戴设备基础知识

Android 穿戴设备具有不同于手持设备的 Android 应用的用户界面范例。Wear 应用的第一波次旨在扩展一起提供一些方法, 但开头的 Android Wear 2.0 中的手持设备应用, 穿戴设备应用, 可以在单独使用。Wear 应用部署, 它是与辅助掌上电脑应用程序一起打包。由于大多数 Wear 应用都依赖于在手持辅助应用程序时, 它们需要以某种方式与手持应用进行通信。以下各节描述这些使用方案, 并概述 Android Wear 的基本功能。

使用方案

Android Wear 的第一个版本主要关注当前手持通过扩展应用程序增强通知和手持设备的应用程序和可穿戴应用之间同步数据。因此, 这些方案是相对比较简单实现。

可穿戴通知

支持 Android Wear 的最简单方法是特性的利用共享掌上电脑和可穿戴设备之间的通知。通过使用支持 v4 通知 API 并 `WearableExtender` 类 (提供 [Xamarin Android 支持库](#)), 可以利用平台, 如收件箱设置卡样式的本机功能或语音输入。RecipeAssistant 示例提供了代码示例演示如何向 Android Wear 的设备发送的通知的列表。

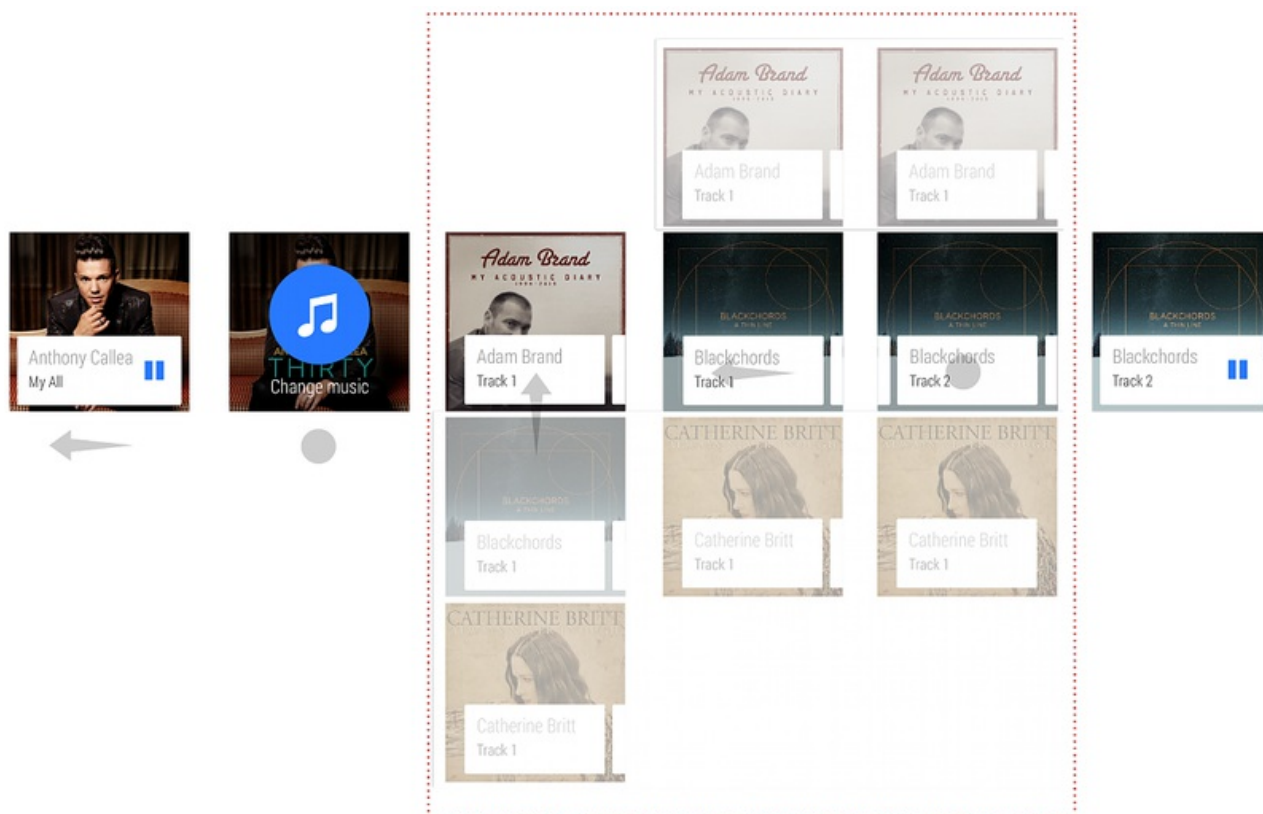
配套应用程序

另一种策略是创建一个可穿戴设备上本机运行并对与辅助掌上电脑应用程序的完整应用程序。此方法的一个典型示例是 [测验](#) 示例应用中, 该示例演示了如何创建测验的手持设备上运行并可穿戴设备上询问测验问题。

用户界面

穿戴设备的主要导航模式是一系列的垂直排列的卡。每个卡可以具有关联的分层出同一行的操作。 `GridViewPager` 类提供此功能, 它遵循相同的适配器概念 `ListView`。您通常将相关联 `GridViewPager` 与 `FragmentGridPagerAdaptor`

(或 `GridPagerAdaptor`), 用于表示每个行和列的单元格作为 `Fragment` :



Wear 还使操作按钮构成的一项非常重大的使用彩色圆圈使用下方的小说明文本（如上面所示）。[GridViewPager](#) 示例演示如何使用 `GridViewPager` 和 `GridPagerAdapter` 穿戴设备应用中。

Android Wear 2.0 添加到穿戴设备用户界面导航抽屉、操作抽屉和内联操作按钮。有关 Android Wear 2.0 用户界面元素有关的详细信息，请参阅 [Android剖析](#) 主题。

通信

Android 穿戴设备提供了两个不同的通信 Api，以促进穿戴设备应用和手持的配套应用程序之间的通信：

数据 API – 此 API 是类似于可穿戴设备和手持设备之间同步的数据存储。Android 负责将可穿戴和手持设备之间的更改传播时将执行此操作最佳选择。可穿戴设备超出范围时，它将队列更高版本的时间的同步。此 API 的主入口点是 `WearableClass.DataApi`。有关此 API 的详细信息，请参阅 [Android同步数据项](#) 主题。

消息 API – 此 API 使您能够使用较低级别通信路径：小负载发送单向无需在掌上电脑和可穿戴应用之间的同步。此 API 的主入口点是 `WearableClass.MessageApi`。有关此 API 的详细信息，请参阅 [Android发送和接收消息](#) 主题。

可以选择要注册以接收这些消息通过的每个 API 侦听器接口回调或，或者，派生的应用程序中实现的服务 `WearableListenerService`。此服务将自动通过 Android Wear 实例化。[FindMyPhone](#) 示例演示如何实现 `WearableListenerService`。

部署

每个可穿戴应用不会随其自己嵌入主应用程序的 APK 的 APK 文件部署。此打包在 Xamarin.Android 5.0 及更高版本，会自动处理，但必须手动执行的 Xamarin.Android 版本早于版本 5.0。[使用打包](#) 介绍更多详细信息中的部署。

深入学习

熟悉 Android Wear 的最佳方式是生成和测试你的第一个应用。以下列表提供了建议的阅读顺序，以帮助您快速掌握：

1. [设置和安装](#) 安装和配置用于构建 Xamarin.Android 穿戴设备应用程序开发环境提供了详细的说明。
2. 已安装所需的包并配置仿真器或设备后，请参阅[你好，穿戴设备](#)的分步说明了如何创建一个小型的 Android

Wear 项目该句柄按钮单击, 并显示单击在穿戴设备上的计数器。

3. [部署和测试](#)提供更详细的有关配置和部署到仿真程序和设备, 包括如何将应用部署到在穿戴设备通过蓝牙说明信息。
4. [使用屏幕大小](#)介绍了如何预览和优化您的用户界面穿戴设备的设备上各种可用的屏幕大小。
5. [使用打包](#)说明手动打包穿戴设备适用于 Google Play 上的分发的应用程序的步骤。

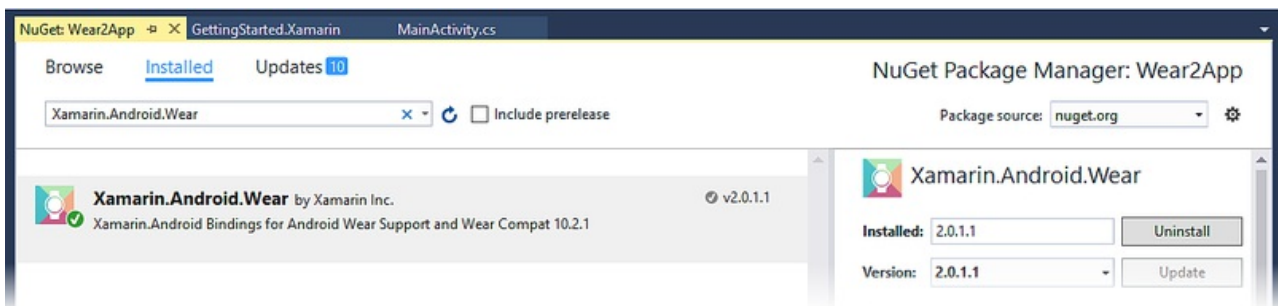
创建第一个 Wear 应用后, 你可能想要尝试为 Android Wear 构建自定义手表表盘。[创建表盘](#)开发压下数字监视人脸服务, 可以增强其功能到模拟样式表盘具有额外功能的更多代码后跟提供分步说明和示例代码。

Android Wear 2.0

Android Wear 2.0 引入了各种新特性和功能, 如 [复杂性](#), 曲线布局、导航和操作抽屉和扩展的通知。此外, Wear 2.0 使您可以构建的独立手持应用于的独立应用程序。[新手腕手势功能](#), 可以与您的应用程序的单手交互。以下部分重点介绍了这些特性, 并提供链接, 以帮助你开始使用应用程序中使用它们。

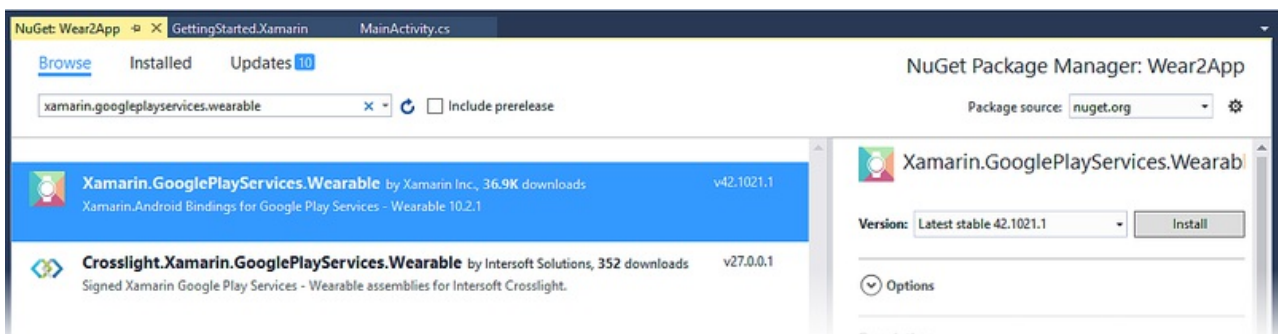
安装 Wear 2.0 包

若要生成使用 Xamarin.Android Wear 2.0 应用, 必须添加**Xamarin.Android.Wear v2.0**包到你的项目 (单击浏览选项卡):



此 NuGet 包包含绑定的 Android 支持可穿戴和穿戴设备兼容性的库。

除了**Xamarin.Android.Wear**, 我们建议你安装**Xamarin.GooglePlayServices.Wearable** NuGet:

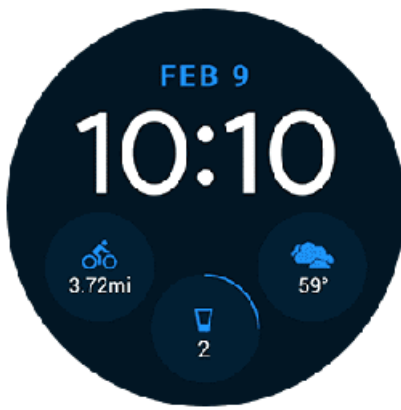


Wear 2.0 的主要功能

Android Wear 2.0 是自 2014年中其初始启动后到 Android Wear 史无前例的重大更新。以下部分重点介绍 Android Wear 2.0 中, 主要功能并提供链接以帮助您了解如何在应用中使用这些新功能。

复杂情况

[复杂情况](#)是人脸小组件, 您可以快速查看而无需往下轻扫手表表盘的小监视。复杂性是类似于桌面样式仪表板小组件; 它们显示信息, 例如天气、电池寿命、日历事件和 fitness 应用统计信息:



有关复杂的详细信息，请参阅 Android [监视人脸复杂性](#) 主题。

导航和操作抽屉

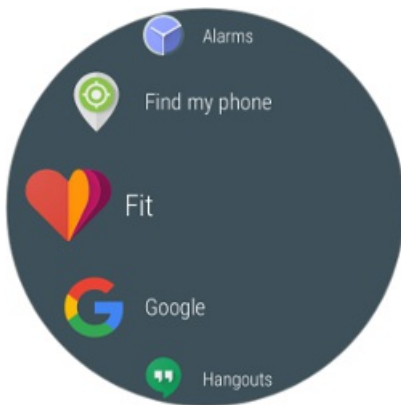
Wear 2.0 中包含两个新抽屉。*导航抽屉*，其中显示在屏幕的顶部，用户可以应用视图间导航了（如左下图所示）。*操作抽屉*，其中显示在屏幕（如右侧所示）的底部，用户可以从操作列表中选择。



有关这两个新的交互式抽屉的详细信息，请参阅 Android [Wear 导航和操作](#) 主题。

曲线的布局

Wear 2.0 引入了用于显示曲线的布局倒圆角穿戴设备上的新功能。具体而言，新 `WearableRecyclerView` 类适用于在倒圆角的显示器上显示垂直的项的列表：



`WearableRecyclerView` 扩展了 `RecyclerView` 类，以支持曲线的布局和循环滚动手势。有关详细信息，请参阅 Android [WearableRecyclerView API 文档](#)。

独立的应用程序

Android Wear 2.0 应用程序可独立于掌上电脑应用。这意味着，例如，智能手表可以继续提供完整的功能即使伴侣手持设备处于关闭还是远从可穿戴设备状态。有关此功能的详细信息，请参阅 Android [独立的应用程序](#) 主题。

手腕手势

手腕手势可使用户与您的应用程序交互而无需使用触摸屏-用户能够响应应用程序与单个手的形状。支持两个手腕手势：

- [笔锋手腕出](#)
- [笔锋手腕中](#)

有关详细信息，请参阅 [Android 手腕手势](#) 主题。

有许多详细 Wear 2.0 功能，例如内联操作、智能回复、远程输入、扩展的通知和通知新的桥接模式。有关 Wear 2.0 的新功能的详细信息，请参阅 [Android API 概述](#)。

设备

下面是可以运行 Android Wear 的设备的一些示例：

- [Motorola 360](#)
- [LG G 监视](#)
- [LG G 监视 R](#)
- [Samsung 齿轮实时](#)
- [Sony SmartWatch 3](#)
- [ASUS ZenWatch](#)

其他阅读材料

查看 Google 的 Android Wear 文档：

- [有关 Android 穿戴设备](#)
- [Android 穿戴设备应用程序设计](#)
- [android.support.wearable 库](#)
- [Android Wear 2.0](#)

总结

本简介提供 Android Wear 的概述。它所述的 Android Wear 的基本功能，并包含 Android Wear 2.0 中引入的功能的概述。它提供基本读取以帮助开发人员开始使用 Xamarin.Android 穿戴设备开发的链接，它列出当前市场上的 Android Wear 设备的一些示例。

相关链接

- [安装和设置](#)
- [入门](#)

设置和安装

2018/10/26 • [Edit Online](#)

本文将指导完成安装步骤和准备你的计算机和设备的 Android Wear 开发所需的配置详细信息。本文结束时，您将有一个有效的 Xamarin.Android 穿戴设备安装到 Visual Studio 中集成 Mac 和/或 Microsoft Visual Studio 中，并且就可以准备好开始构建您的第一个 Xamarin.Android 穿戴设备应用程序。

要求

以下被所创建的基于 Xamarin 的 Android Wear 应用：

- **Visual Studio 或 Visual Studio for Mac** – 你如果使用的 Visual Studio, Visual Studio 2015 Professional 或更高版本。
- **Xamarin.Android** – Xamarin.Android 4.17 或更高版本必须安装并配置与 Visual Studio 或 Visual Studio for mac。
- **Android SDK** -Android SDK 5.0.1 (API 21) 或更高版本必须安装通过 Android SDK 管理器。
- **Java 开发人员工具包** – Xamarin Android 开发需要 [JDK 1.8](#) 如果您是开发的 API 级别 24 或更高版本 (JDK 1.8 还支持 API 级别低于 24)。

你可以继续使用 [JDK 1.7](#) 如果您是开发专门针对 API 级别 23 或更早版本。

IMPORTANT

Xamarin.Android 不支持 JDK 9。

安装

安装 Xamarin.Android 后，执行以下步骤，以便你可以生成和测试 Android Wear 应用：

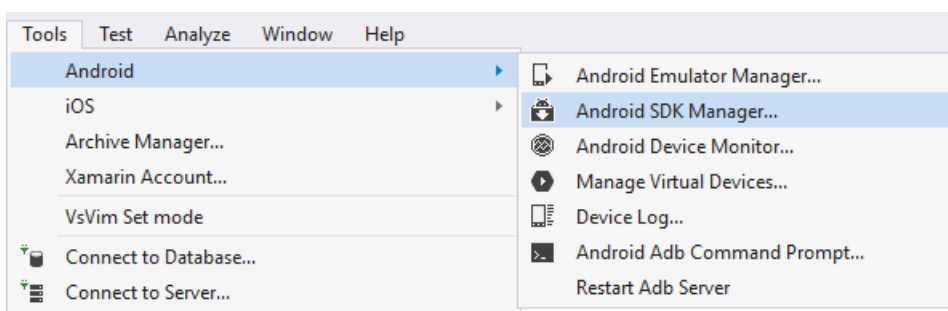
1. 安装所需的 Android SDK 和工具。
2. 配置测试设备。
3. 创建第一个 Android Wear 应用。

以下各节中介绍这些步骤。

安装 Android SDK 和工具

启动 **Android SDK 管理器**：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



























请确保您有以下 Android SDK 和安装工具：

- Android SDK Tools v 24.0.0 或更高版本，并
- Android 4.4W (API20)，或
- Android 5.0.1 (API21) 或更高版本。

如果没有最新 SDK 和安装工具，下载所需的 SDK 工具并 API 位（可能需要进行滚动一点，找到它们-API 选择如下所示）：

- [Visual Studio](#)
- [Visual Studio for Mac](#)

|  Name | API | Rev. | Status |
|--|-----|------|---|
| ▼ <input type="checkbox"/>  Android 5.0.1 (API 21) | | | |
| <input checked="" type="checkbox"/>  SDK Platform | 21 | 2 |  Installed |
| <input type="checkbox"/>  Android TV ARM EABI v7a System Image | 21 | 3 | <input type="checkbox"/> Not installed |
| <input type="checkbox"/>  Android TV Intel x86 Atom System Image | 21 | 3 | <input type="checkbox"/> Not installed |
| <input checked="" type="checkbox"/>  Android Wear ARM EABI v7a System Image | 21 | 3 |  Installed |
| <input checked="" type="checkbox"/>  Android Wear Intel x86 Atom System Image | 21 | 3 |  Installed |
| <input checked="" type="checkbox"/>  ARM EABI v7a System Image | 21 | 4 |  Installed |
| <input type="checkbox"/>  Intel x86 Atom_64 System Image | 21 | 4 |  Installed |
| <input type="checkbox"/>  Intel x86 Atom System Image | 21 | 4 |  Installed |
| <input checked="" type="checkbox"/>  Google APIs ARM EABI v7a System Image | 21 | 18 |  Installed |
| <input type="checkbox"/>  Google APIs Intel x86 Atom_64 System Image | 21 | 18 |  Installed |
| <input type="checkbox"/>  Google APIs Intel x86 Atom System Image | 21 | 18 |  Installed |
| <input checked="" type="checkbox"/>  Google APIs | 21 | 1 |  Installed |

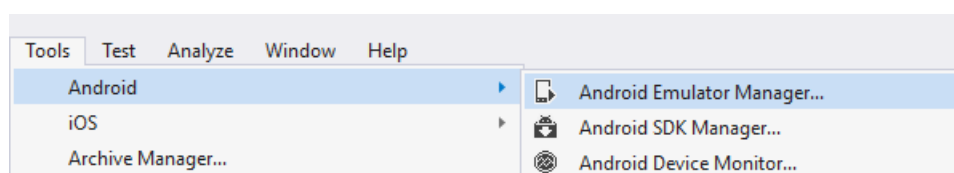
配置

可以使用之前测试你的应用，必须配置 Android Wear 仿真器或实际的 Android Wear 设备。

Android 穿戴设备仿真程序

可以使用 Android Wear 仿真程序之前，必须配置 Android Wear Android 虚拟设备 (AVD) 使用 **Google 仿真器管理器**：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



有关设置 Android Wear 仿真程序的详细信息，请参阅[仿真器上调试 Android Wear](#)。

Android 穿戴设备

如果具有 Android Wear Smartwatch 如 Android Wear 设备，则可以调试而不是使用仿真程序此设备上的应用。有关使用穿戴设备进行开发的信息，请参阅[穿戴设备的设备上调试](#)。

创建第一个 Android Wear 应用

请按照[你好，穿戴设备](#)生成首个监视应用的说明。

打包应用程序

Android 穿戴设备应用程序始终使用辅助 Android 手机应用程序分发。

作为您主要的 Android 应用程序的引用添加你的 Android Wear 应用程序时它会自动被假定为 Android Wear 的项目，并将为您生成所有必需的 XML 和元数据。此外，它将验证包和版本号，因此，您可以轻松地提供您的应用

程序, 到 Google Play 与匹配。

若要了解有关打包 Wear 应用的详细信息, 请参阅[使用打包](#)。

相关链接

- [SkeletonWear](#) (示例)

你好, 穿戴设备

2018/10/26 • [Edit Online](#)

创建第一个 Android Wear 应用并在穿戴设备仿真程序或设备上运行。本演练提供了创建, 用于处理按钮单击事件并单击计数器显示在穿戴设备上的小 Android Wear 项目的分步说明。其中介绍了如何调试使用穿戴设备仿真程序或在穿戴设备通过蓝牙连接到 Android 手机的应用。它还提供有关 Android Wear 的一组调试提示。



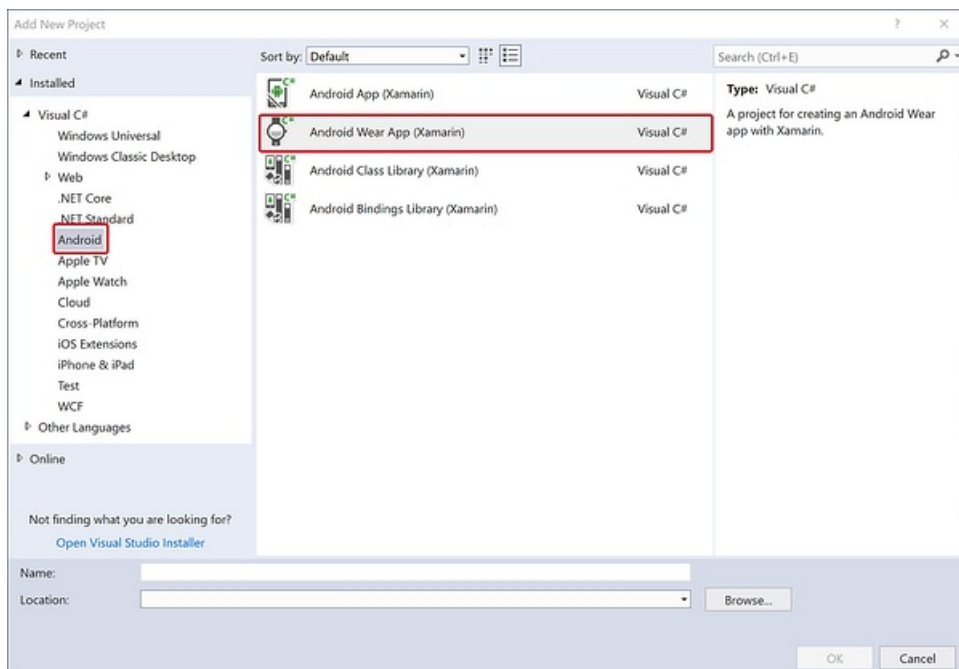
第一个 Wear 应用

请按照以下步骤创建第一个 Xamarin.Android Wear 应用操作:

1.创建新的 Android 项目

创建一个新Android Wear 应用程序:

- [Visual Studio](#)
- [Visual Studio for Mac](#)

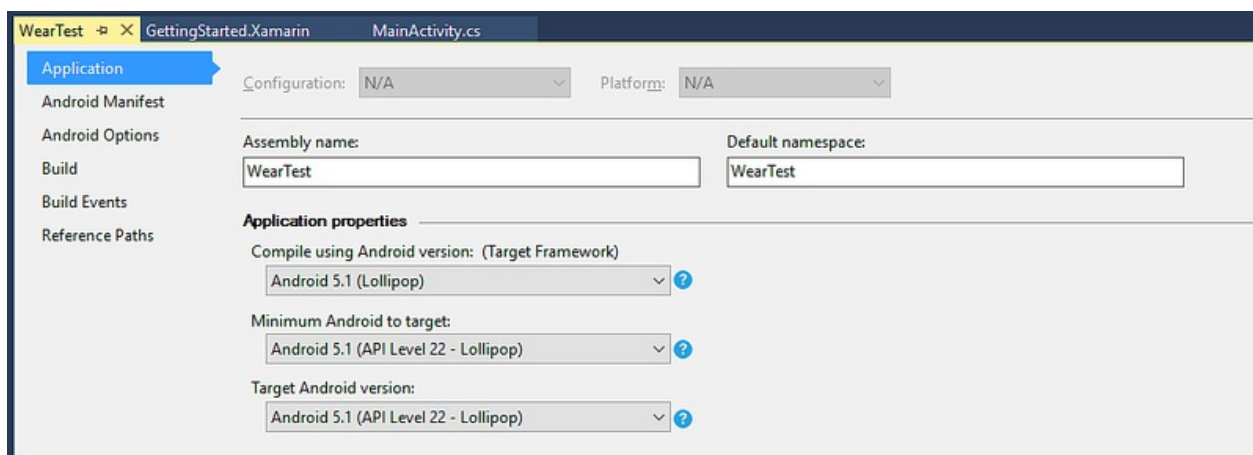


此模板会自动包括**Xamarin Android 可穿戴库** NuGet（及其依赖项）以便将有权访问特定于穿戴设备的小组件。如果看不到穿戴设备模板，请查看[安装和设置指南](#)，请仔细检查已安装受支持的 Android SDK。

2.选择正确目标框架

- [Visual Studio](#)
- [Visual Studio for Mac](#)

模板的最低 **Android** 目标设置为**Android 5.0 (Lollipop)** 或更高版本：



设置目标框架的详细信息，请参阅[了解 Android API 级别](#)。

3.编辑Main.axml布局

配置要包含的布局 `TextView` 和一个 `Button` 示例：

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ScrollView
        android:id="@+id/scroll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#000000"
        android:fillViewport="true">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginBottom="2dp"
                android:text="Main Activity"
                android:textSize="36sp"
                android:textColor="#006600" />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginBottom="2dp"
                android:textColor="#cccccc"
                android:id="@+id/result" />
            <Button
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:onClick="showNotification"
                android:text="Click Me!"
                android:id="@+id/click_button" />
        </LinearLayout>
    </ScrollView>
</FrameLayout>

```

4.编辑MainActivity.cs源

添加代码以递增计数器并将其显示每次单击按钮时:

```

[Activity (Label = "WearTest", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
{
    int count = 1;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        SetContentView (Resource.Layout.Main);

        Button button = FindViewById<Button> (Resource.Id.click_button);
        TextView text = FindViewById<TextView> (Resource.Id.result);

        button.Click += delegate {
            text.Text = string.Format ("{0} clicks!", count++);
        };
    }
}

```

5.仿真器或设备安装程序

下一步是设置要部署并运行应用的仿真程序或设备。如果你尚不熟悉的部署和运行过程 Xamarin.Android 应用一般情况下, 请参阅[Hello, Android 快速入门](#)。

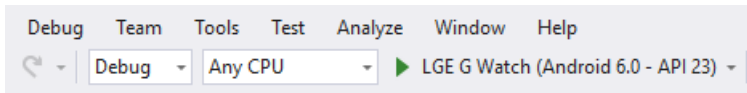
如果没有如 Android Wear Smartwatch 的 Android Wear 设备，你可以在仿真器上运行应用。有关调试的仿真程序上 Wear 应用的信息，请参阅[仿真器上调试 Android Wear](#)。

如果具有 Android Wear Smartwatch 如 Android Wear 设备，你可以而不是使用仿真程序在设备上运行应用。有关在穿戴设备上调试的详细信息，请参阅[穿戴设备的设备上调试](#)。

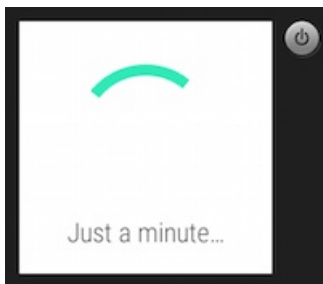
6.运行 Android Wear 应用

Android Wear 设备应出现在设备下拉菜单中。请确保选择正确的 Android Wear 设备或 AVD 开始调试之前。选择设备之后，单击播放按钮以将应用部署到仿真器或设备。

- [Visual Studio](#)
- [Visual Studio for Mac](#)

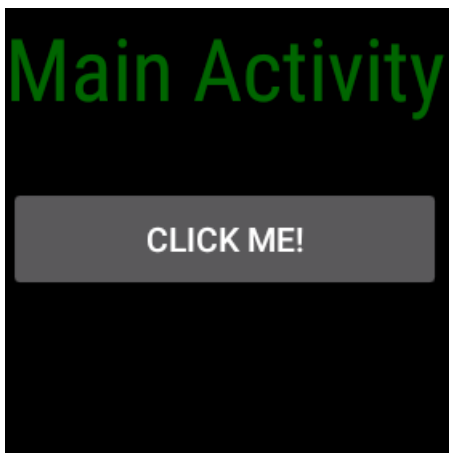


你可能会看到只需一分钟的时间... 在第一个消息（或某些其他插播式屏幕）：

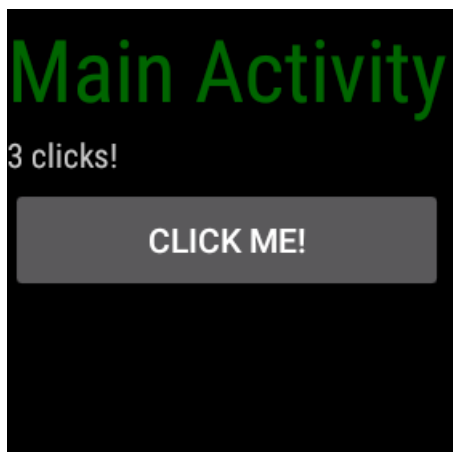


如果使用的监视仿真程序，可能需要一段时间才能启动该应用程序。当使用蓝牙时，花费更多时间来部署应用，通过 USB 相比。（例如，它需要 5 分钟内将此应用部署到蓝牙连接到 Nexus 5 电话 LG G Watch。）

应用可以成功部署后，在穿戴设备的屏幕应显示如下所示的屏幕：



点击单击我！在穿戴设备并查看每个点击计数增量的表面上的按钮：



后续步骤

请查看[Wear 示例](#)包括辅助电话应用程序使用 Android Wear 应用。

当准备好分发应用, 请参阅[使用打包](#)。

相关链接

- 单击[我的应用程序（示例）](#)

用户界面

2018/10/26 • [Edit Online](#)

以下部分介绍各种工具和用于组成 Android Wear 应用中的用户界面的构建基块。

控件

介绍了 Android Wear 特定控件，并提供指向演示如何使用这些控件的示例。

Android 穿戴设备控件

2018/10/26 • [Edit Online](#)

Android Wear 应用可以使用的正则 Android 应用, 包括已使用许多相同的控件 `Button`, `TextView`, 边距和图像绘图。布局控件包括 `ScrollView`, `LinearLayout`, 和 `RelativeLayout` 也可用。

此页链接到 Android Wear 特定控件从[可穿戴 UI 库](#)在通过 Xamarin 项目中可用[可穿戴支持](#) NuGet 包。这些控件包括:

- **GridViewPager** - 创建二维的导航界面, 其中用户滚动鼠标向下后行中进行选择 (有关详细信息, 请参阅[GridViewPager](#)):



Wear 应用其他重要控件包括:

- `BoxInsetLayout` (请参阅[处理屏幕尺寸](#)),
- `WatchViewStub` (请参阅[处理屏幕尺寸](#)),
- `CardFrame` (请参阅[Android 创建卡](#)),
- `CardScrollView` (请参阅[Android 创建卡](#)),
- `WearableListView` (请参阅[列出了 Android 创建](#))。

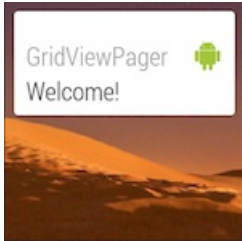
相关链接

- [Android.Support.Wearable docs](#)

GridViewPager

2018/10/26 • [Edit Online](#)

[GridViewPager](#) 示例演示如何实现 Android wear 2D 选取器导航模式。



首先添加 [Xamarin Android Wear](#) 支持到你的项目的 NuGet 包。

布局 XML 如下所示：

```
<android.support.wearable.view.GridViewPager xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:keepScreenOn="true" />
```

创建 [GridPagerAdapter](#) (或如子类 [FragmentGridPagerAdapter](#)) 视图将显示为用户提供导航。

[示例适配器](#) 演示如何实现所需的方法, 包括重写 `RowCount`, `GetColumnCount`, `GetBackground`, 和 `GetFragment`

接通适配器所示：

```
pager.Adapter = new SimpleGridPagerAdapter (this, FragmentManager);
```

相关链接

- [Google 的 2D 选取器文档](#)
- [android.support.wearable docs](#)
- [GridViewPager \(示例\)](#)

平台功能

2018/10/26 • [Edit Online](#)

在本部分中的文档介绍特定于 Android Wear 的功能。此处, 您会发现本主题介绍如何创建 WatchFace。

创建表盘

实现自定义的观察人脸服务的 Android Wear 分步演练。提供了有关构建压下数字监视人脸服务说明, 然后添加更多代码以创建具有额外功能的模拟样式表盘。

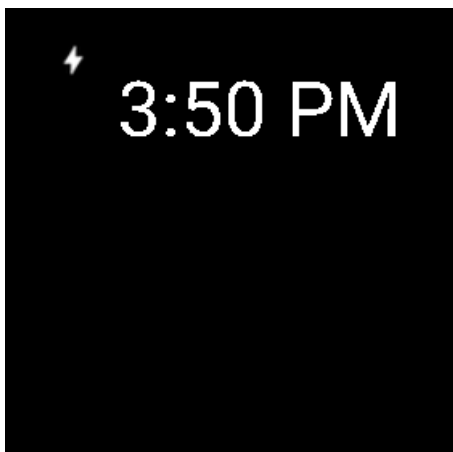
创建表盘

2018/10/26 • [Edit Online](#)

本指南介绍如何实现自定义的观察人脸服务针对 Android Wear 1.0。用于构建压下数字监视人脸服务后,跟更多代码以创建模拟样式表盘提供分步说明。

概述

在此演练中,创建一个基本的监视人脸服务来演示创建自定义的 Android Wear 1.0 手表表盘的基础知识。初始监视的人脸服务显示以小时和分钟显示的当前时间的简单数字监视:



此数字手表表盘的开发和测试后,将其升级到更复杂的三个手与模拟手表表盘添加更多的代码:



观看人脸服务是捆绑在一起且作为 Wear 1.0 应用程序的一部分安装。在以下示例中, `MainActivity` 包含没有什么比 Wear 1.0 应用程序模板中的代码,以便监视人脸服务可以打包并部署到智能手表应用的一部分。实际上,此应用将充当纯粹获取监视人脸服务加载到 Wear 1.0 设备(或仿真器)的工具进行调试和测试。

要求

若要实现监视人脸服务,需要以下各项:

- Android 5.0 (API 级别 21) 或更高版本在穿戴设备或仿真程序上。
- [Xamarin Android Wear 支持库](#) 必须添加到 Xamarin.Android 项目。

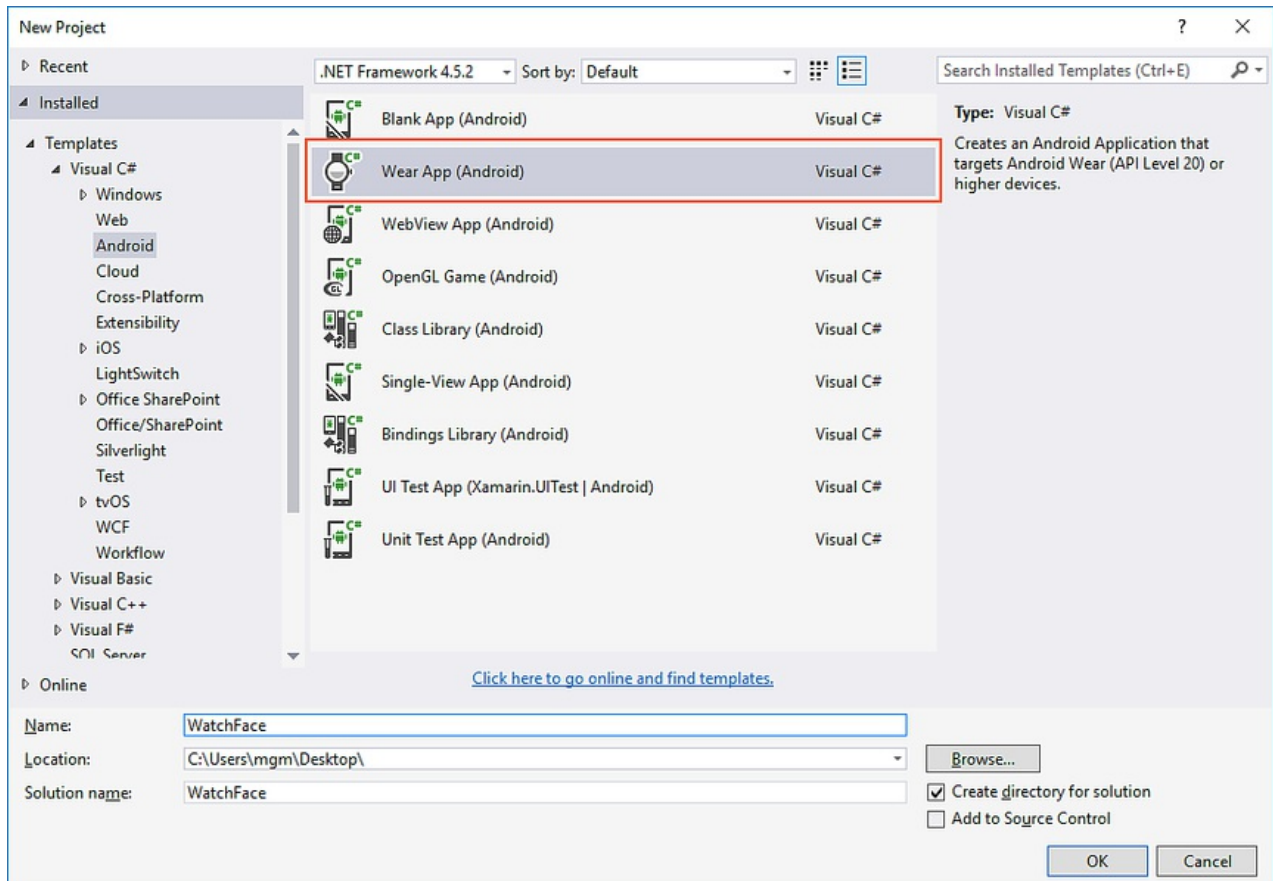
虽然 Android 5.0 是最低 API 级别用于实现监视的人脸服务,Android 5.1 或更高版本建议。Android Wear 运行 Android 5.1 (API 22) 的设备或更高版本允许穿戴设备应用来控制显示的内容在屏幕上时在设备处于低功耗环境模

式。当设备离开低功耗环境模式时，它是在交互式模式。有关这些模式的详细信息，请参阅[使您的应用程序可见](#)。

启动应用程序项目

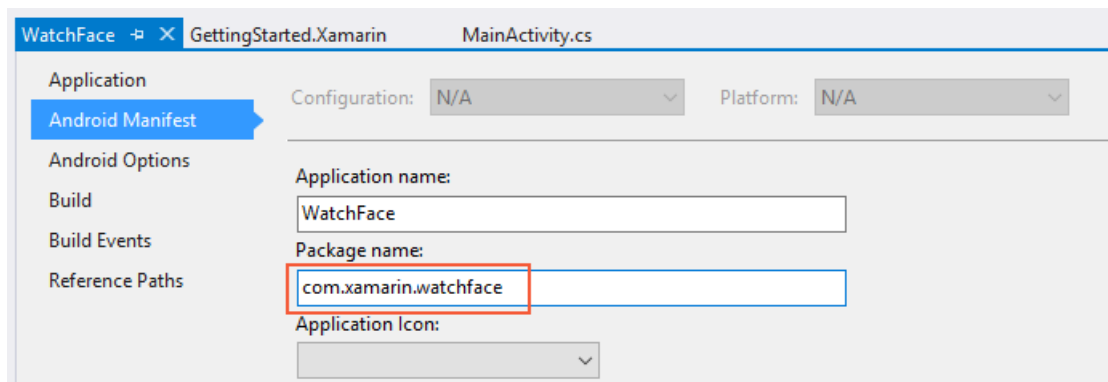
创建一个名为新的 Android Wear 1.0 项目 **WatchFace** (有关创建新的 Xamarin.Android 项目的详细信息，请参阅[Hello, Android](#)):

- [Visual Studio](#)
- [Visual Studio for Mac](#)



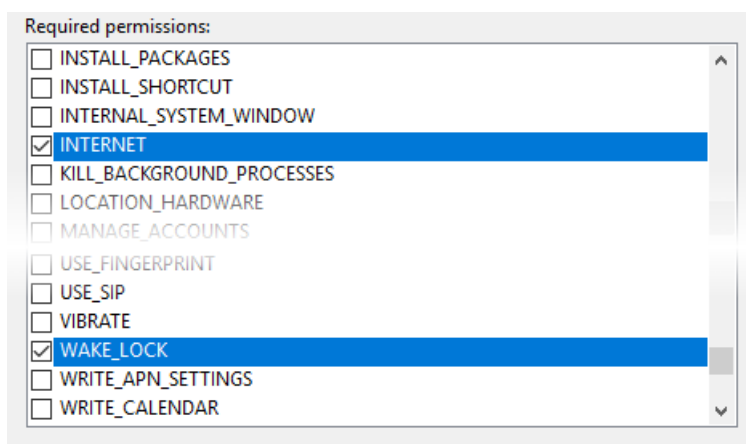
将包名称设置为 `com.xamarin.watchface` :

- [Visual Studio](#)
- [Visual Studio for Mac](#)



- [Visual Studio](#)
- [Visual Studio for Mac](#)

此外，向下滚动并启用 **INTERNET** 并 **WAKE_LOCK** 权限：

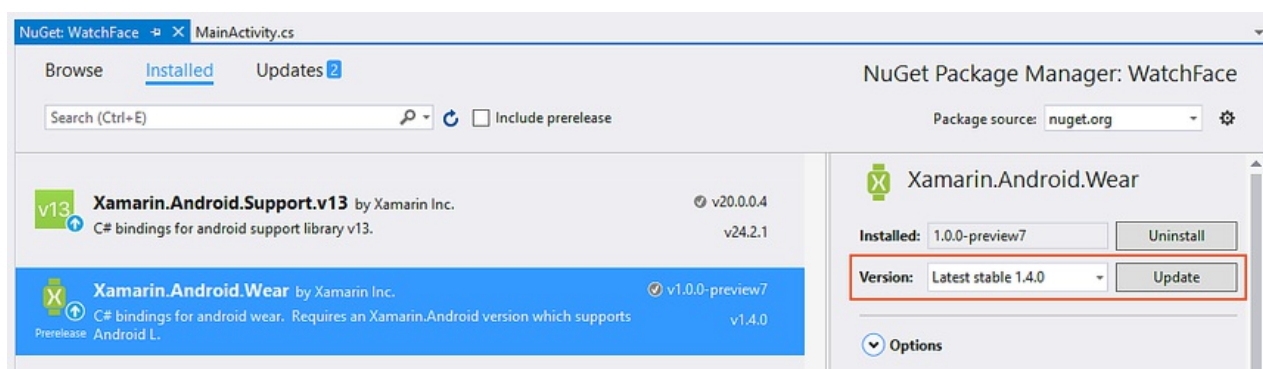


接下来，下载[preview.png](#) –这将添加到绘图稍后在本演练的文件夹。

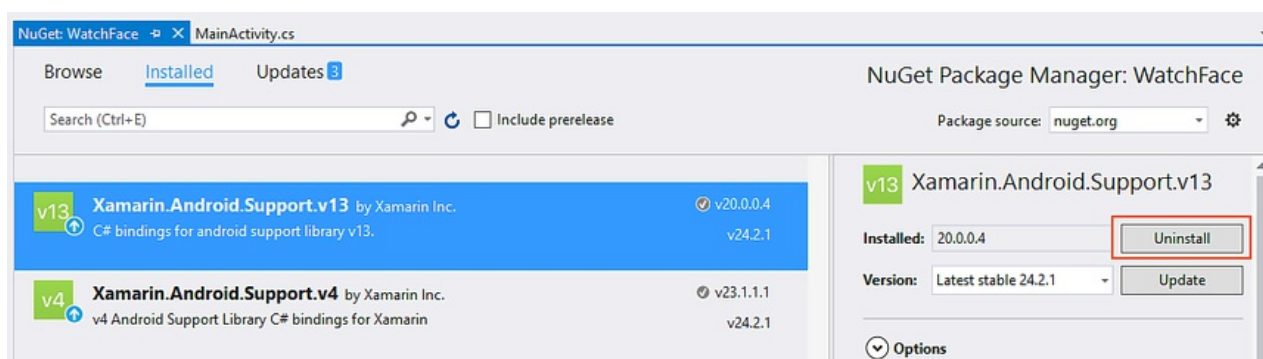
添加 Xamarin.Android 穿戴设备包

- [Visual Studio](#)
- [Visual Studio for Mac](#)

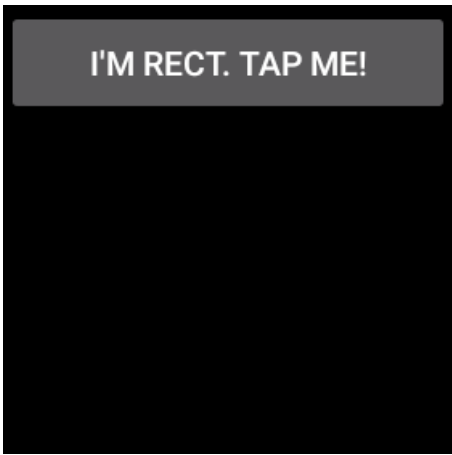
启动 NuGet 包管理器 (在 Visual Studio 中，右键单击引用中解决方案资源管理器，然后选择**管理 NuGet 包...**)。更新到最新稳定版本的项目 **Xamarin.Android.Wear**：



接下来，如果 **Xamarin.Android.Support.v13** 是安装，将其卸载：



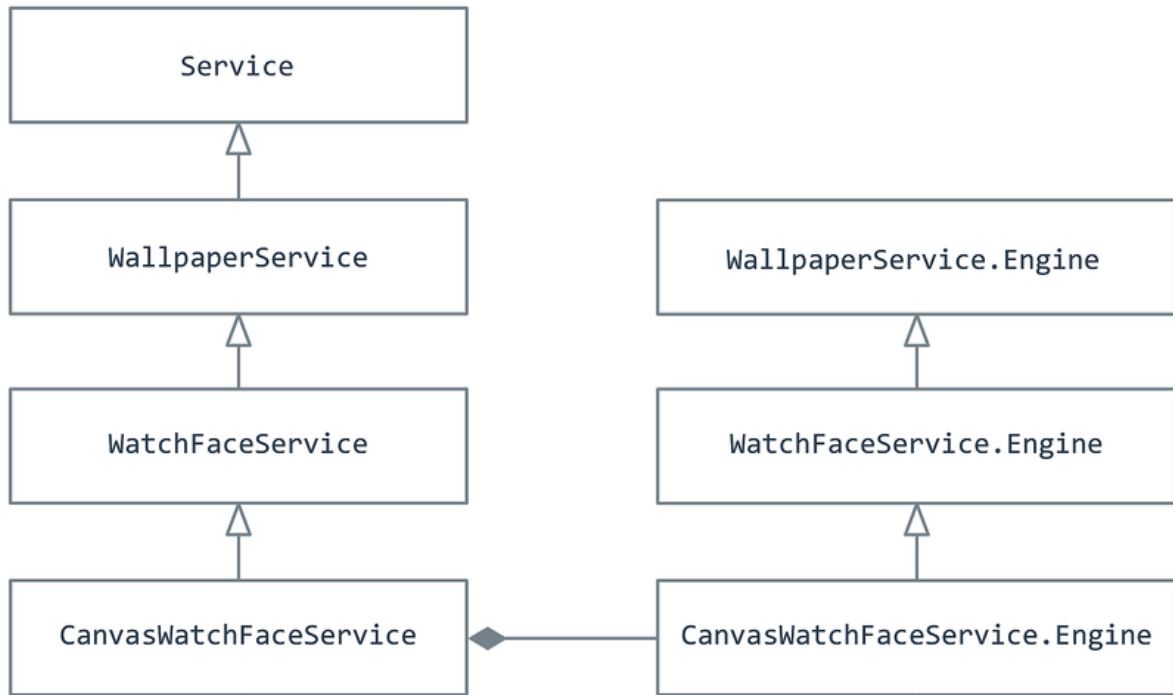
生成并在穿戴设备或仿真器上运行应用 (有关如何执行此操作的详细信息，请参阅[Getting Started](#)指南)。在穿戴设备上看到以下应用屏幕：



此时，基本 Wear 应用不具有监视人脸功能，因为尚未提供监视人脸服务实现。接下来将添加此服务。

CanvasWatchFaceService

Android 穿戴设备实现观看通过人脸 `CanvasWatchFaceService` 类。 `CanvasWatchFaceService` 派生自 `WatchFaceService`，该类本身派生自 `WallpaperService` 以下关系图中所示：



`CanvasWatchFaceService` 包括嵌套 `CanvasWatchFaceService.Engine`；它实例化 `CanvasWatchFaceService.Engine` 执行绘制手表表盘的实际工作的对象。 `CanvasWatchFaceService.Engine` 派生自 `WallpaperService.Engine` 上图中所示。

不在此图中所示是 `Canvas` 该 `CanvasWatchFaceService` 用于绘制手表表盘-这 `Canvas` 通过传入 `OnDraw` 方法如下所述。

在以下部分中，将通过执行以下步骤创建自定义的观察人脸服务：

1. 定义一个名为类 `MyWatchFaceService` 派生自 `CanvasWatchFaceService`。
2. 内 `MyWatchFaceService`，创建一个名为的嵌套的类 `MyWatchFaceEngine` 派生自 `CanvasWatchFaceService.Engine`。
3. 在中 `MyWatchFaceService`，实现 `CreateEngine` 方法实例化 `MyWatchFaceEngine` 并将其返回。

4. 在中 `MyWatchFaceEngine` , 实现 `OnCreate` 方法创建的监视人脸样式并执行任何其他初始化任务。
5. 实现 `OnDraw` 方法的 `MyWatchFaceEngine` 。每当手表表盘需要重绘时调用此方法 (即失效)。`OnDraw` 为绘制 (和重绘) 监视人脸元素, 如小时、分钟和第二个指针的方法。
6. 实现 `OnTimeTick` 方法的 `MyWatchFaceEngine` 。 `OnTimeTick` 每分钟 (在环境和交互模式) 或日期/时间已更改时至少一次调用。

有关详细信息 `CanvasWatchFaceService` , 请参阅 Android [CanvasWatchFaceService](#) API 文档。同样, [CanvasWatchFaceService.Engine](#) 介绍手表表盘的实现。

添加 `CanvasWatchFaceService`

- [Visual Studio](#)
- [Visual Studio for Mac](#)

添加名为的新文件 **MyWatchFaceService.cs** (在 Visual Studio 中, 右键单击 **WatchFace** 中解决方案资源管理器, 单击 **添加 > 新建项...**, 然后选择类)。

此文件的内容替换为以下代码:

```
using System;
using Android.Views;
using Android.Support.Wearable.Watchface;
using Android.Service.Wallpaper;
using Android.Graphics;

namespace WatchFace
{
    class MyWatchFaceService : CanvasWatchFaceService
    {
        public override WallpaperService.Engine OnCreateEngine()
        {
            return new MyWatchFaceEngine(this);
        }

        public class MyWatchFaceEngine : CanvasWatchFaceService.Engine
        {
            CanvasWatchFaceService owner;
            public MyWatchFaceEngine (CanvasWatchFaceService owner) : base(owner)
            {
                this.owner = owner;
            }
        }
    }
}
```

`MyWatchFaceService` (派生自 `CanvasWatchFaceService`) 是手表表盘"主计划"。`MyWatchFaceService` 实现只有一个方法, `OnCreateEngine` , 其实例化并返回 `MyWatchFaceEngine` 对象 (`MyWatchFaceEngine` 派生自 `CanvasWatchFaceService.Engine`)。实例化 `MyWatchFaceEngine` 对象必须作为返回 `WallpaperService.Engine` 。封装 `MyWatchFaceService` 对象传递到构造函数。

`MyWatchFaceEngine` 是实际观察人脸实现-它包含绘制手表表盘的代码。它还处理系统事件, 例如屏幕更改 (环境/交互模式中, 屏幕关闭, 等等)。

实现引擎 `OnCreate` 方法

`OnCreate` 方法初始化手表表盘。以下字段添加到 `MyWatchFaceEngine` :

```
Paint hoursPaint;
```

这 `Paint` 对象将用于绘制 watch 表盘上的当前时间。接下来, 添加以下方法 `MyWatchFaceEngine` :

```
public override void OnCreate(ISurfaceHolder holder)
{
    base.OnCreate (holder);

    SetWatchFaceStyle (new WatchFaceStyle.Builder(owner)
        .SetCardPeekMode (WatchFaceStyle.PEEK_MODE_SHORT)
        .SetBackgroundVisibility (WatchFaceStyle.BACKGROUND_VISIBILITY_INTERRUPTIVE)
        .SetShowSystemUiTime (false)
        .Build ());

    hoursPaint = new Paint();
    hoursPaint.Color = Color.White;
    hoursPaint.TextSize = 48f;
}
```

`OnCreate` 不久后调用 `MyWatchFaceEngine` 已启动。设置了 `WatchFaceStyle` (它可以控制在穿戴设备与用户交互的方式), 并实例化 `Paint` 将用于显示时间的对象。

对调用 `SetWatchFaceStyle` 执行以下操作:

1. 集 `peek` 模式到 `PEEK_MODE_SHORT`, 这将导致通知, 以显示为小的"速览"卡上显示。
2. 将背景可见性设置为 `BACKGROUND_VISIBILITY_INTERRUPTIVE`, 这会导致背景的查看卡以显示只是暂时是否它表示中断通知。
3. 禁用默认系统 UI 时间从手表表盘上进行绘制, 以使自定义手表表盘可以改为显示时间。

有关这些和其他监视人脸样式选项的详细信息, 请参阅 Android [WatchFaceStyle.Builder](#) API 文档。

之后 `SetWatchFaceStyle` 完成后, `OnCreate` 实例化 `Paint` 对象 (`hoursPaint`), 并将其颜色设置为白色, 其文本大小以适合 48 像素 (`TextSize` 必须指定以像素为单位)。

实现引擎 `OnDraw` 方法

`OnDraw` 方法可能是最重要 `CanvasWatchFaceService.Engine` 方法-是, 实际绘制观看人脸元素, 如数字和时钟人脸指针的方法。在以下示例中, watch 表盘上绘制的时间字符串。添加以下方法 `MyWatchFaceEngine` :

```
public override void OnDraw (Canvas canvas, Rect frame)
{
    var str = DateTime.Now.ToString ("h:mm tt");
    canvas.DrawText (str,
        (float)(frame.Left + 70),
        (float)(frame.Top + 80), hoursPaint);
}
```

当调用 Android `OnDraw`, 它将传入 `Canvas` 实例, 并可以在其中绘制将人脸的边界。在上面的代码示例中, `DateTime` 用于计算的当前时间以小时和分钟数 (采用 12 小时格式)。生成的时间字符串在画布上绘制使用 `Canvas.DrawText` 方法。字符串将显示 70 像素通过从左边的缘和 80 像素下的上边缘。

有关详细信息 `OnDraw` 方法, 请参阅 Android [onDraw](#) API 文档。

实现引擎 `OnTimeTick` 方法

Android 定期调用 `OnTimeTick` 方法来更新通过手表表盘所显示的时间。它在至少一次, 每分钟 (在环境和交互模式下), 或已更改的日期/时间或时区时调用。添加以下方法 `MyWatchFaceEngine` :

```
public override void OnTimeTick()
{
    Invalidate();
}
```

此实现 `OnTimeTick` 只需调用 `Invalidate`。 `Invalidate` 方法计划 `OnDraw` 重绘手表表盘。

有关详细信息 `OnTimeTick` 方法，请参阅 Android [onTimeTick](#) API 文档。

注册 CanvasWatchFaceService

`MyWatchFaceService` 必须在中注册 **AndroidManifest.xml** 关联穿戴设备应用程序。若要执行此操作，添加以下 XML 到 `<application>` 部分：

```
<service
    android:name="watchface.MyWatchFaceService"
    android:label="Xamarin Sample"
    android:allowEmbedded="true"
    android:taskAffinity=""
    android:permission="android.permission.BIND_WALLPAPER">
    <meta-data
        android:name="android.service.wallpaper"
        android:resource="@xml/watch_face" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview"
        android:resource="@drawable/preview" />
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
        <category android:name="com.google.android.wearable.watchface.category.WATCH_FACE" />
    </intent-filter>
</service>
```

此 XML 将执行以下操作：

1. 集 `android.permission.BIND_WALLPAPER` 权限。此权限使更改在设备上的系统壁纸的监视人脸服务权限。请注意，此权限必须设置在 `<service>` 部分，而不是在外部 `<application>` 部分。
2. 定义 `watch_face` 资源。此资源是声明一个简短的 XML 文件 `wallpaper` 资源（在下一节中将创建此文件）。
3. 声明名为 `drawable` 映像 `preview`，将显示的监视选取器选择屏幕。
4. 包括 `intent-filter` 让 Android 知道 `MyWatchFaceService` 将显示手表表盘。

完成为基本代码 `WatchFace` 示例。下一步是添加所需的资源。

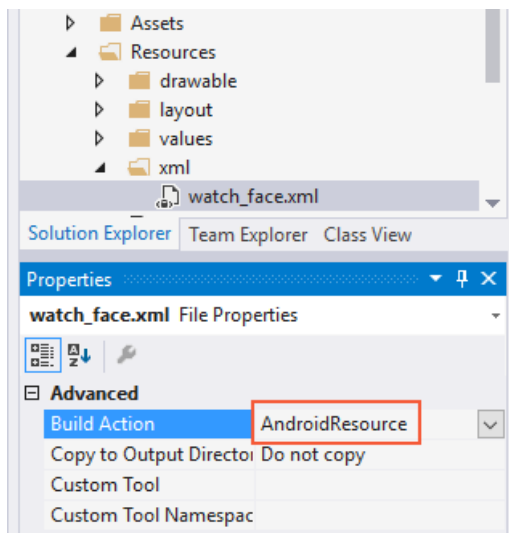
添加资源文件

您可以运行监视服务之前，必须添加 `watch_face` 资源和预览图像。首先，创建新的 XML 文件在 **Resources/xml/watch_face.xml** 并将其内容替换为以下 XML：

```
<?xml version="1.0" encoding="UTF-8"?>
<wallpaper xmlns:android="http://schemas.android.com/apk/res/android" />
```

将此文件的生成操作设置为 **AndroidResource**：

- [Visual Studio](#)
- [Visual Studio for Mac](#)



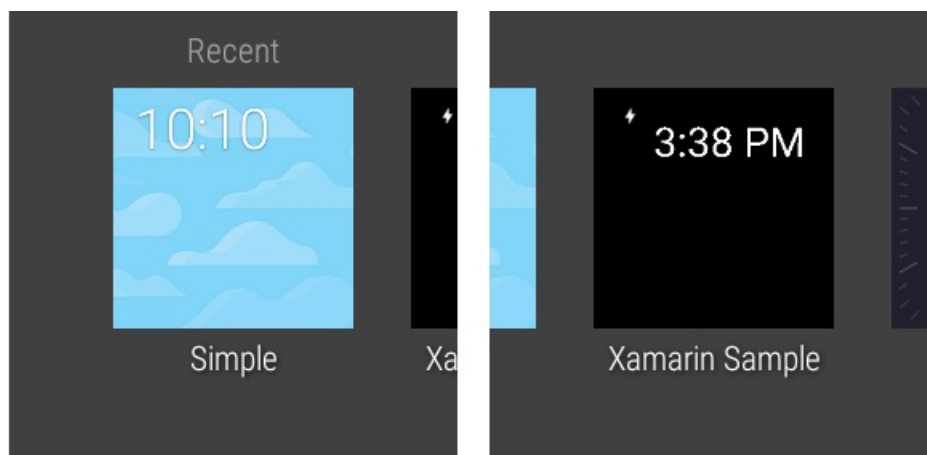
此资源文件定义了一个简单 `wallpaper` 用于手表表盘的元素。

如果尚未这样做，下载 [preview.png](#)。安装在 `Resources/drawable/preview.png`。请确保添加到此文件 `WatchFace` 项目。在穿戴设备上观看人脸选取器中向用户显示此预览图像。若要创建你自己的手表表盘的预览图像，可以在运行时执行手表表盘的屏幕截图。（有关获取从穿戴设备的设备的屏幕截图的详细信息，请参阅[拍摄屏幕快照](#)）。

试试看！

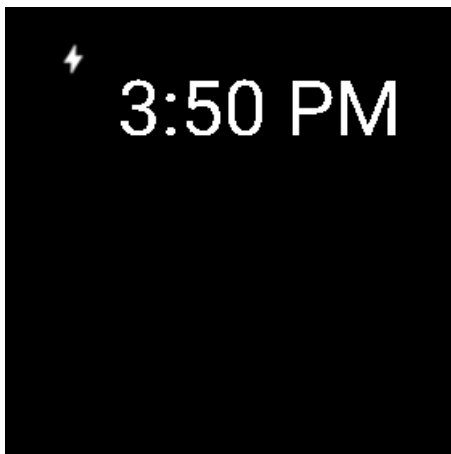
生成并部署到在穿戴设备的应用程序。您应该可以看到 Wear 应用屏幕像以前一样显示。执行以下操作来启用新的手表表盘：

1. 轻扫到右侧，直到您看到监视屏幕的背景。
2. 触摸并在屏幕的背景上任意位置保存两秒钟。
3. 轻扫，从左到右，可以浏览各种表盘。
4. 选择 **Xamarin 示例** 观看（显示在右侧）的人脸：



5. 点击 **Xamarin 示例** 表盘以将其选中。

这会更改要使用自定义的观察人脸服务实现到目前为止在穿戴设备表盘：



这是因为应用程序实现是因此最小种相对简陋手表表盘 (例如, 它不包括监视人脸背景, 它不会调用 `Paint` 抗锯齿方法以改善外观)。但是, 它实现所需创建自定义手表表盘的基本功能。

在下一步部分中, 此手表表盘将升级到更复杂的实现。

升级手表表盘

在本演练的其余部分 `MyWatchFaceService` 升级以显示模拟样式表盘, 它可进行扩展以支持更多的功能。将添加以下功能, 若要创建已升级的手表表盘:

1. 指示以模拟小时、分钟和第二个手的时间。
2. 在可见性的更改作出反应。
3. 对环境的模式和交互模式之间的更改做出响应。
4. 读取基础穿戴设备的属性。
5. 自动更新所在的时区更改发生的时间。

实现下面的代码更改之前, 下载 [drawable.zip](#), 将其解压缩, 并移动到已解压缩的.png 文件资源/`drawable` (覆盖前一 `preview.png`)。添加到新的.png 文件 `WatchFace` 项目。

更新引擎功能

下一步是升级 `MyWatchFaceService.cs` 到绘制模拟表盘和支持新功能的实现。内容替换为 `MyWatchFaceService.cs` 中的监视人脸代码的模拟版本 `MyWatchFaceService.cs` (可以剪切并粘贴此源的现有 `MyWatchFaceService.cs`)。

此版本的 `MyWatchFaceService.cs` 将更多的代码添加到现有方法并包括其他重写的方法来添加更多的功能。以下部分提供的源代码的 guided 教程。

OnCreate

已更新 `OnCreate` 方法配置监视人脸样式与之前一样, 但它还包括一些附加步骤:

1. 将背景图像设置为 `xamarin_background` 驻留在资源 `Resources/drawable-hdpi/xamarin_background.png`。
2. 初始化 `Paint` 绘制小时手、分针和第二个指针的对象。
3. 初始化 `Paint` 对象, 用于绘制手表表盘的边缘周围小时计时周期数。
4. 该调用将创建一个计时器 `Invalidate` (重绘) 方法, 以便第二个指针都将重绘每隔一秒。请注意, 此计时器是必要的因为 `OnTimeTick` 调用 `Invalidate` 仅一次每隔一分钟。

此示例包含一个 `xamarin_background.png` 映像; 但是, 你可能想要创建你的自定义手表表盘将支持每个屏幕密度的不同的背景图像。

OnDraw

已更新**OnDraw**方法绘制模拟样式表盘使用以下步骤：

1. 获取当前时间，它现在保存在 `time` 对象。
2. 确定的绘图图面和其中心的边界。
3. 绘制背景，缩放以适应设备绘制背景。
4. 绘制 12 计时周期围绕时钟（对应于时钟表面上的小时数）的人脸。
5. 计算角度、旋转和每个监视指针的长度。
6. 监视的表面上绘制每个指针。请注意是否所监视的环境模式不绘制第二个指针。

OnPropertiesChanged

调用此方法以通知 `MyWatchFaceEngine` 穿戴设备（如低位环境模式和刻录中保护）的属性。在 `MyWatchFaceEngine`，此方法只检查的低位环境模式（在低位环境模式下，屏幕支持较少的位用于每种颜色）。

有关此方法的详细信息，请参阅 Android [onPropertiesChanged](#) API 文档。

OnAmbientModeChanged

在穿戴设备进入或退出环境模式时，调用此方法。在 `MyWatchFaceEngine` 实现中，在环境的模式下时手表表盘禁用抗锯齿。

有关此方法的详细信息，请参阅 Android [onAmbientModeChanged](#) API 文档。

OnVisibilityChanged

调用此方法是每当监视变得可见还是隐藏。在 `MyWatchFaceEngine`，此方法注册/注销的时区接收方（如下所述）根据的可见性状态。

有关此方法的详细信息，请参阅 Android [onVisibilityChanged](#) API 文档。

时区功能

新**MyWatchFaceService.cs**还包括功能更新时区更改（例如在旅行跨时区）的当前时间。结尾附近**MyWatchFaceService.cs**，区域更改的时间 `BroadcastReceiver` 定义用于处理时区更改意向对象：

```
public class TimeZoneReceiver: BroadcastReceiver
{
    public Action<Intent> Receive { get; set; }
    public override void OnReceive (Context context, Intent intent)
    {
        if (Receive != null)
            Receive (intent);
    }
}
```

`RegisterTimezoneReceiver` 并 `UnregisterTimezoneReceiver` 会调用方法 `OnVisibilityChanged` 方法。
`UnregisterTimezoneReceiver` 调用时手表表盘的可见性状态更改为隐藏。当再次可见时手表表盘
`RegisterTimezoneReceiver` 调用 (请参阅 `OnVisibilityChanged` 方法)。

引擎 `RegisterTimezoneReceiver` 方法将一个处理程序声明此时区接收器 `Receive` 事件; 此处理程序更新 `time` 对象所跨时区的新时间：

```
timeZoneReceiver = new TimeZoneReceiver ();
timeZoneReceiver.Receive = (intent) => {
    time.Clear (intent.GetStringExtra ("time-zone"));
    time.SetToNow ();
};
```

创建并注册为时区接收方意向筛选器：

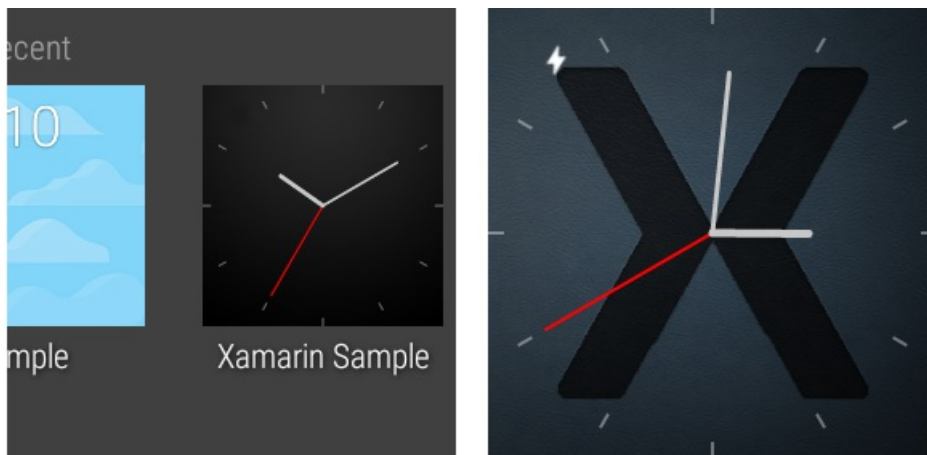
```
IntentFilter filter = new IntentFilter(Intent.ActionTimeZoneChanged);
Application.Context.RegisterReceiver (timeZoneReceiver, filter);
```

`UnregisterTimeZoneReceiver` 方法注销时区接收方：

```
Application.Context.UnregisterReceiver (timeZoneReceiver);
```

运行改进的手表表盘

生成并再次将应用部署到在穿戴设备。从监视人脸选取器作为之前选择手表表盘。在监视选取器中的预览将显示在左侧，并在右侧显示新的手表表盘：



在此屏幕截图，第二个指针每秒一次移动。在穿戴设备上运行此代码时，第二个指针将消失时监视输入环境的模式。

总结

在此演练中，自定义 Android Wear 1.0 watchface 是实现和测试。`CanvasWatchFaceService` 和 `CanvasWatchFaceService.Engine` 引入了类，并实现引擎类的基本方法来创建简单的数字表盘。此实现了更新，采用更多的功能，以创建模拟的表盘，和其他方法在实现以处理更改可见性、环境模式和设备属性之间的差异。最后，时区广播的接收器已实现，以便监视会自动更新何时跨越一个时区的时间。

相关链接

- [创建表盘](#)
- [WatchFace 示例](#)
- [WatchFaceService.Engine](#)

使用屏幕大小

2018/10/26 • [Edit Online](#)

Android Wear 设备可以有矩形或圆角显示，这也可以是不同的大小。



确定屏幕上键入

Wear 支持库提供了一些控件可帮助您检测和适应不同的屏幕的形状，如 `WatchViewStub` 和 `BoxInsetLayout`。

请注意的其他支持库控件 (如 `GridViewPager`) 自动检测屏幕形状本身并不应如下所述控件的子级添加。

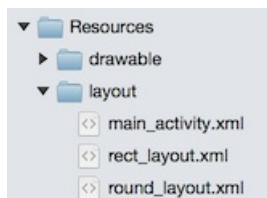
WatchViewStub

请参阅[WatchViewStub](#)示例，了解有关如何检测屏幕上，键入并显示每个类型不同的布局。

主要的布局文件包含 `android.support.wearable.view.WatchViewStub` 引用不同的布局矩形的圆角屏幕，即使用 `app:rectLayout` 和 `app:roundLayout` 属性：

```
<android.support.wearable.view.WatchViewStub
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/stub"
    app:rectLayout="@layout/rect_layout"
    app:roundLayout="@layout/round_layout" />
```

该解决方案包含有关每个样式将在运行时选择不同的布局：



BoxInsetLayout

而不是生成每个屏幕上，键入不同的布局，还可以创建适应矩形或圆角的屏幕的单一视图。

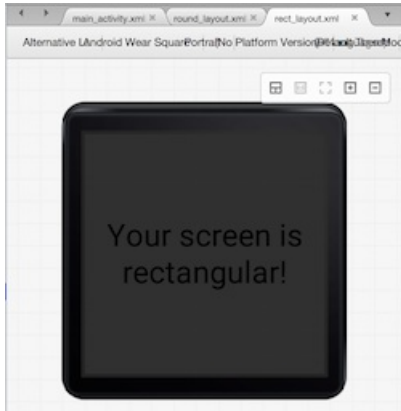
这[Google 示例](#)演示如何使用 `BoxInsetLayout` 矩形的圆角屏幕上使用相同的布局。

Wear UI 设计器

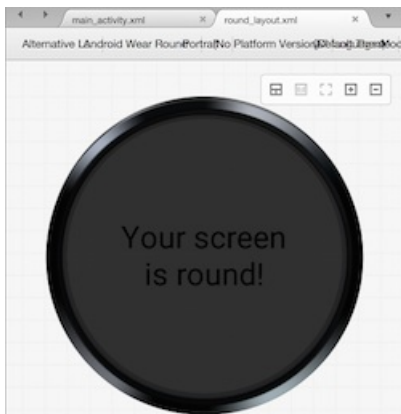
Xamarin Android 设计器支持矩形的圆角屏幕：



矩形样式中的设计图面将如下所示：



舍入样式中的设计图面将如下所示：

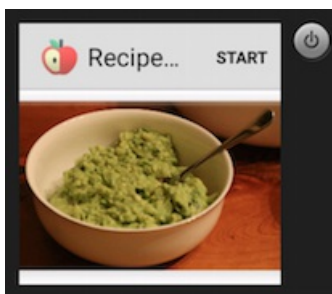


穿戴设备模拟器

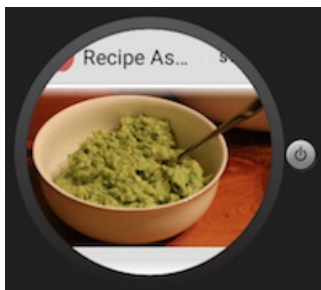
Google 仿真器管理器包含设备定义，用于同时为屏幕类型。您可以创建矩形和圆形仿真程序来测试您的应用程序。



为矩形屏幕如下所示在仿真程序：



为圆角屏幕, 它会呈现如下:



视频

适用于 Android 穿戴设备的全屏应用从developers.google.com。

部署和测试

2018/10/26 • [Edit Online](#)

本部分介绍如何测试 Android Wear 应用在 Android Wear 设备上（或配置为 Wear 的 Android 仿真程序上）。它还包括调试提示和有关如何设置开发计算机和 Android 设备之间的蓝牙连接的信息。您的应用程序准备就绪后，最后一个主题将介绍如何为部署准备您的应用程序。

调试的仿真程序上的 Android 穿戴设备

如何调试 Android SDK 仿真程序上的 Xamarin.Android 穿戴设备应用程序。

在穿戴设备上调试

如何配置 Android 设备，以便 Xamarin.Android 穿戴设备应用程序可以部署到它直接从 Visual Studio 或 Visual Studio for mac。

打包 Wear 应用

如何打包 Xamarin.Android 穿戴设备适用于 Google Play 上的分发的应用程序。

调试的仿真程序上的 Android 穿戴设备

2018/10/26 • [Edit Online](#)

这些文章介绍如何调试的仿真程序上的 *Xamarin.Android* 穿戴设备应用程序。

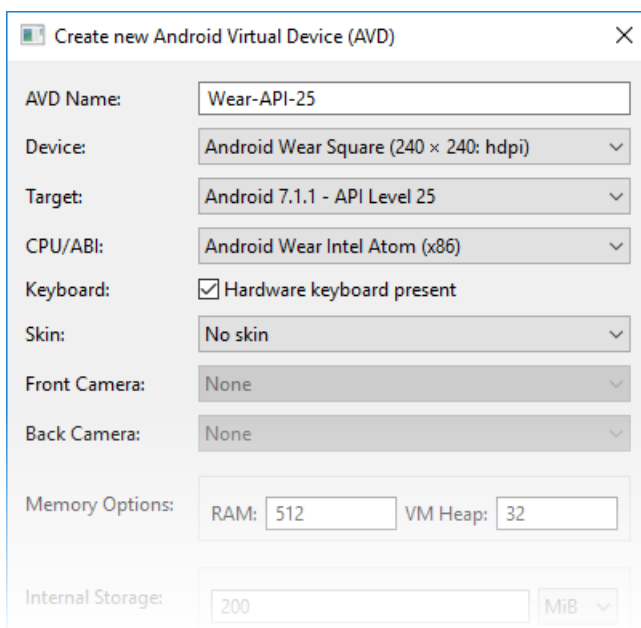
调试穿戴设备仿真程序概述

开发 Android Wear 应用程序需要运行该应用程序，在物理硬件上或使用仿真器或模拟器。使用硬件是最好的方法，但并不总是最实用的方法。在许多情况下，它可以是更简单且更具成本效益模拟/仿真 Android Wear 硬件使用仿真器，如下所述。如果你尚不熟悉的部署和运行过程 Android Wear 应用，请参阅[你好，穿戴设备](#)。

配置 Android 仿真器

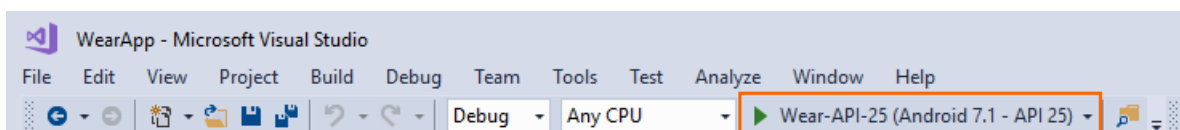
若要在仿真器上运行穿戴设备应用，必须安装 Android SDK Android 仿真器，并将其配置为 Android Wear。有关整体的 Android SDK 仿真程序安装和配置信息，请参阅[Android 仿真程序安装程序](#)。

当创建 Wear 虚拟设备时，选择 Android Wear 设备配置文件 (如**Android Wear 正方形**)。为改进性能，使用 Wear **x86** CPU/ABI 在此示例中所示：



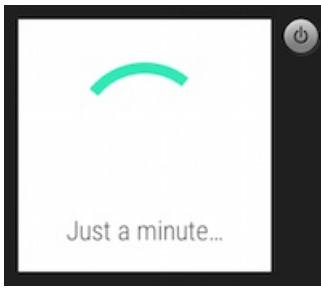
启动 Wear 虚拟设备

创建 Android Wear 虚拟设备后，你可以选择它从设备下拉列表菜单在 IDE 中开始调试之前。如果你的虚拟设备不可用的设备下拉列表中，验证你的项目是 Android Wear 应用程序项目（不 Android 应用程序项目），且，其目标 API 级别设置为相同的 API 级别设置为虚拟设备。例如：



Android 仿真程序启动后，则 Xamarin.Android 将 Wear 应用部署到仿真程序。仿真器会使用配置的虚拟设备映像运行此应用。

不会感到惊讶，如果看到此错误（或另一个插播式屏幕）在第一个。观看仿真程序可能需要一段时间才能启动：



可以一直运行仿真器;无需关闭仿真器并在每次运行应用时重启。

总结

本指南介绍了如何配置 Android 仿真器以穿戴设备开发和启动 Wear 虚拟设备以进行调试。

在穿戴设备上调试

2018/10/26 • [Edit Online](#)

本文介绍如何调试在穿戴设备上的 Xamarin.Android 穿戴设备应用程序。

概述

如果具有 Android Wear Smartwatch 如 Android Wear 设备，你可以而不是使用仿真程序在设备上运行应用。(如果你尚不熟悉的部署和运行过程 Android Wear 应用，请参阅[你好，穿戴设备](#)。)

准备在穿戴设备：

使用以下步骤启用 Android Wear 设备上调试：

1. 打开设置 Android Wear 设备上的菜单。
2. 滚动到底部的菜单和点击有关。
3. 点击内部版本号七倍。
4. 上设置菜单中，点击开发人员选项。
5. 确认 ADB 调试已启用。

通过 USB 调试

如果在穿戴设备有 USB 端口，可以在穿戴设备连接到计算机，将部署到它，并运行/调试应用程序是使用 Android 手机 (有关详细信息，请参阅[设备上调试](#))。

通过蓝牙调试

如果在穿戴设备不具有 USB 端口，可以应用部署到在穿戴设备通过蓝牙通过应用程序的调试输出路由到 Android 手机连接到您的计算机。

准备你的手机

使用以下步骤准备你的电话以进行蓝牙连接到在穿戴设备：

1. 如果尚未这样做，设置手机以供 Xamarin.Android 开发中所述[设置设备进行开发](#)。
2. 下载并安装免费 [Android Wear](#) 来自 Google Play 商店的应用。

将设备连接

使用以下步骤连接到你的手机穿戴设备：

1. 在手机上，将充当蓝牙中间 (上文中配置)，启动 Android Wear 应用程序。
2. 点击设置图标。
3. 启用调试通过蓝牙。您应该看到显示的屏幕上的 Android Wear 应用的以下状态：

```
Host: disconnected
Target: connected
```

4. 通过 USB 连接到您的计算机的电话。在计算机上，输入以下命令：

```
adb forward tcp:4444 localabstract:/adb-hub
adb connect 127.0.0.1:4444
```

如果端口 4444 不可用，可以使用有权访问的任何其他可用端口。

请注意：如果你重启 Visual Studio 或 Visual Studio for Mac，则必须运行这些命令以建立与在穿戴设备。

5. 当在穿戴设备提示您时，请确认你允许**ADB** 调试。在 Android Wear 应用中，您应看到状态更改为：

```
Host: connected
Target: connected
```

6. 完成上述步骤后，运行 `adb devices` 显示手机和 Android Wear 设备的状态：

```
List of devices attached
127.0.0.1:4444    device
019ad61df0a69399 device
```

此时，您可以将应用部署到在穿戴设备。

对于拍摄屏幕快照

您可以通过输入以下命令来执行在穿戴设备的屏幕截图：

```
adb -s 127.0.0.1:4444 shell screencap -p /sdcard/DCIM/screencap.png
```

将屏幕快照复制到您的计算机，通过输入以下命令：

```
adb -s 127.0.0.1:4444 pull /sdcard/DCIM/screencap.png
```

通过输入以下命令来删除设备上的屏幕截图：

```
adb -s 127.0.0.1:4444 shell rm /sdcard/DCIM/screencap.png
```

卸载应用

可以从在穿戴设备卸载应用，通过输入以下命令：

```
adb -s 127.0.0.1:4444 uninstall <package name>
```

例如，若要删除的包名称应用 `com.xamarin.wear-test`，输入以下命令：

```
adb -s 127.0.0.1:4444 uninstall com.xamarin.wear-test
```

有关调试 Android Wear 设备通过蓝牙的详细信息，请参阅[调试通过蓝牙](#)。

调试包含辅助电话应用程序的 Wear 应用

与 Google Play 上的分发辅助 Android 手机应用程序打包在 android Wear 应用 (有关详细信息，请参阅[使用打包](#))。但是，您仍开发穿戴设备应用和其辅助应用程序分开。释放通过 Google Play 应用商店应用程序时，将与辅助应用程序一起打包 Wear 应用，并自动安装在可能的情况。

若要调试穿戴设备应用程序与辅助应用程序：

1. 生成并部署到手机的辅助应用程序。
2. 右键单击穿戴设备项目并将其设置为默认启动项目。
3. Wear 项目部署到可穿戴设备。
4. 运行和调试 Wear 应用在设备上。

总结

本文介绍如何配置 Android Wear 穿戴设备通过蓝牙，Visual Studio 中调试的设备，以及如何调试使用辅助电话应用程序的穿戴设备应用。它还提供有关调试通过蓝牙 Wear 应用的常见调试的提示。

打包 Wear 应用

2018/10/26 • [Edit Online](#)

Android Wear 应用都打包在一起的完整的 Android 应用的 Google Play 上的分发。

自动打包

从 Xamarin Android 5.0 开始, Wear 应用自动打包为掌上电脑应用中资源时您创建从手持项目穿戴设备项目到项目引用。可以使用以下步骤来创建此关联:

- [Visual Studio](#)
- [Visual Studio for Mac](#)

1. 如果 Wear 应用尚未手持式解决方案的一部分, 右键单击解决方案节点, 然后选择**添加 > 添加现有项目...**
2. 导航到 **.csproj** 文件的穿戴设备应用, 选择它, 然后单击**打开**。Wear 应用程序项目现在应可以看到在手持设备的解决方案中。
3. 右键单击**引用**节点, 然后选择**添加引用**。
4. 在中**引用管理器**对话框中, 启用穿戴设备项目 (单击以添加一个复选标记), 然后单击**确定**。
5. 更改包名称为穿戴设备项目, 使其匹配手持项目的包名称 (包名称可以更改**下属性 > Android** 清单)。

请注意, 您将获得**XA5211**如果 Wear 应用的包名称与手持应用的包名称不匹配错误。例如:

```
Error XA5211: Embedded wear app package name differs from handheld app package name (com.companyname.mywearapp != com.companyname.myapp). (XA5211)
```

若要更正此错误, 请更改 Wear 应用的包名称, 使其匹配掌上电脑应用的包名称。

当您单击**生成 > 生成所有**, 这种关联触发穿戴设备项目自动打包到主 Handheld (Phone) 项目。Wear 应用自动生成并包含为掌上电脑应用中的资源。

Wear 应用程序项目生成的程序集不用作 Handheld (Phone) 项目中的程序集引用。相反, 生成过程将执行以下操作:

- 验证包名匹配。
- 生成的 XML 并将其添加到手持项目, 以将它与 Wear 应用相关联。例如:

```
<!-- Handheld (Phone) Project.csproj -->
<ProjectReference Include="..\MyWearApp\MyWearApp.csproj">
  <Project>{D80E1FEF-653B-448C-B2AA-609C74E88340}</Project>
  <Name>MyWearApp</Name>
  <IsAppExtension>True</IsAppExtension>
</ProjectReference>
```

- 将作为 Wear 应用添加**原始**到手持项目的资源。

手动打包

可以之前的版本 5.0 中, 在 Xamarin.Android 中编写 Android Wear 应用程序, 但是, 必须执行这些手动打包说明

将应用分发：

1. 请确保你的可穿戴项目和 Handheld (Phone) 项目具有相同的版本数和包名称。
2. 手动构建该项目可穿戴视为**版本生成**。
3. 手动添加发布。**APK**从步骤 (2) 到**资源/原始** Handheld (Phone) 项目的目录。
4. 手动添加新的 XML 资源**Resources/xml/wearable_app_desc.xml**在手持设备项目中，是指可穿戴设备**APK**步骤 (3) 中：

```
<wearableApp package="wearable.app.package.name">
  <versionCode>1</versionCode>
  <versionName>1.0</versionName>
  <rawPathResId>NAME_OF_APK_FROM_STEP_3</rawPathResId>
</wearableApp>
```

5. 手动添加 `<meta-data />` 到手持项目的元素**AndroidManifest.xml** `<application>` 指的是新的 XML 资源的元素：

```
<meta-data android:name="com.google.android.wearable.beta.app"
  android:resource="@xml/wearable_app_desc"/>
```

另请参阅 Android 开发人员站点[手动 packing 说明](#)。