

说明：此系列教程翻译自 [Google Android 开发者官网](#) 的 [Training](#) 教程，利用 Chrome 浏览器的自动翻译功能作初译，然后在一些语句不顺或容易造成误解的地方作局部修正。方便英文不好的开发者查看。如有错漏之处，欢迎大家指出修正。

同时欢迎大家关注我的技术博客

http://blog.csdn.net/it_magician。大家的支持是我最大的动力。

第一章 起始篇

1 构建你的第一个应用程序

欢迎到 Android 应用程序开发中心！

这节课教你如何建立你的第一个 Android 应用程序。您将学习如何创建一个 Android 项目，并运行调试版本的应用程序。您还将学习 Android 应用程序设计的一些基础知识，包括如何建立一个简单的用户界面和处理用户输入。

在你启动这个类之前，确保你有你的开发环境搭建。您需要：

1. 下载 Android SDK。
2. 为 Eclipse 安装 ADT 插件（如果你会使用 Eclipse IDE）。
3. 下载最新的 SDK 使用 SDK 管理器的工具和平台。

如果你还没有已经完成了这些任务，开始下载 [Android SDK](#) 的安装步骤。一旦你设定完成后，你准备开始这节课。

这节课使用教程格式，逐步建立一个小的 Android 应用程序，教你一些关于 Android 开发的基本概念，所以按照每一步是很重要的。

1.1 创建一个 Android 项目

Android 项目包含的所有文件，包括你的 Android 应用程序的源代码。Android SDK 的工具，可以很容易地开始新的 Android 项目与一组默认项目目录和文件。

这一节说明如何创建一个新的项目，无论是使用 Eclipse ADT 插件或使用 SDK 命令行工具。

注意：你应该已经安装了 Android SDK，如果你使用 Eclipse，你应该也有安装 [ADT 插件](#)（21.0.0 或更高版本）。如果你没有这些，按照指南[安装了 Android SDK](#)，然后再开始这一节。

使用 Eclipse 创建一个项目

1. 在工具栏上点击新建 .
2. 在出现的窗口中，打开 **Android** 的文件夹，选择“**Android 应用项目**”，并单击“**下一步**”。

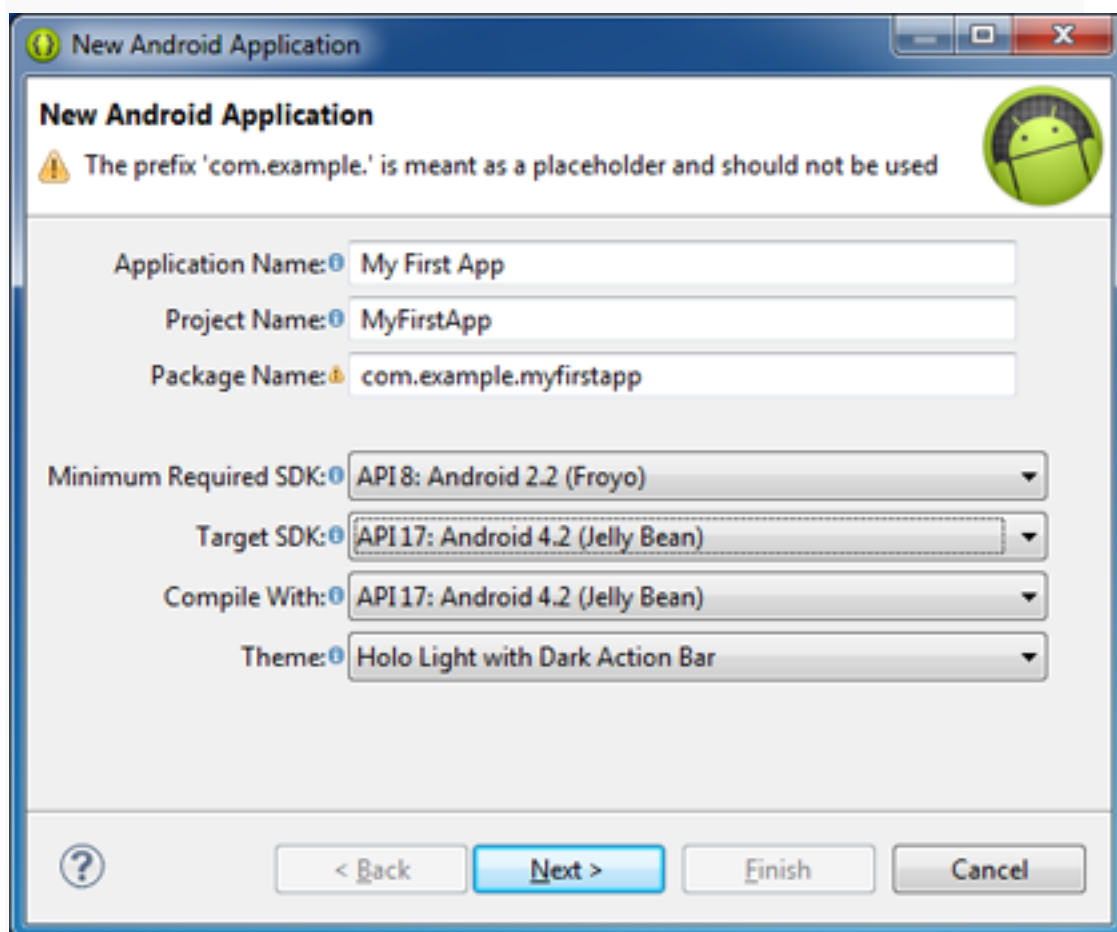


图 1。新的 Android 应用程序在 Eclipse 项目向导。

3. 填写的形式出现：

- **应用程序名称** 显示给用户的是应用程序的名称。对于这个项目，使用“我的第一个应用程序。”
- **项目名称** 是您的项目目录的名称和在 Eclipse 中可见的名称。
- **软件包名称** 是您的应用程序的包的命名空间（在 Java 编程语言中也遵循相同的包规则）。你的包的名称必须是在 Android 系统上安装的所有软件包中唯一的。出于这个原因，它通常最好的做法是，使用与您的组织或出版实体的反向域名开头的名称。对于这个项目，你可以使用一些像“com.example.myfirstapp”但是，您不能使用“com.example”命名空间在 Google Play 中发布您的应用程序。
- **最低要求** 是您的应用程序支持的 Android SDK 最低版本，表示使用 [API 的级别](#)。为了支持尽可能多的设备，你应该设定一个最低版本，可以让您的应用程序提供其核心功能集。如果您的应用程序的任何特性只能在新版本的 Android 或者判定它不是应用程序的核心功能集,您可以只在支持它的版本中启用这些特性（参考[支持不同的平台版本](#)）。对于这个项目保留这些默认设置即可。
- **目标 SDK** 表明您测试您的应用程序的 Android 的最高版本（也叫 [API 级别](#)）。

随着新版本的 Android 变得可用，您应该在新版本上测试应用程序并更新此值，以符合最新的 API 级别，以充分利用新的平台功能。

- **编译** 平台版本对您编译您的应用程序。默认情况下，此设置为您的 SDK 提供的最新版本的 Android。（应该是 Android 4.1 或更高版本，如果你没有这样的版本，您必须使用 [SDK 管理器](#) 安装一个）。您仍然可以建立你的应用程序来支持旧版本，但设置构建目标到最新版本可以让您启用新功能和优化您的应用程序在最新的设备上获得最好的用户体验。
- **主题** 指定申请您的应用程序的 Android UI 风格。你可以暂时不用管它。

单击“下一步”。

4. 在下一个项目配置界面，保留默认选项，并单击“下一步”。
5. 下一个界面，可以帮助您创建一个你的应用程序的启动图标。

您可以通过几个方面来自定义图标，该工具会针对所有的屏幕密度单独生成一个图标。在您发布您的应用程序之前，你应该确保你的图标符合[图解](#)（Iconography）设计指南中定义的规范。

单击“下一步”。

6. 现在您可以在活动模板中选择一个来开始构建应用程序。

对于这个项目，选择 **BlankActivity**，并单击“下一步”。

7. 保留其默认状态中的活动的所有细节，并单击“完成”。

现在你的 **Android** 项目设置一些默认的文件，你准备好开始构建应用程序。继续[下一课](#)。

使用命令行工具创建项目

如果你不使用 **Eclipse IDE ADT** 插件，可以改为使用 **SDK** 命令行工具来创建你的项目：

1. 改变当前目录到 **Android SDK** 的 `tool/` 路径。
2. 执行：

```
android list targets
```

它会打印你已经下载你的 **SDK** 中可用的 **Android** 平台的列表。找到你要编译你的应用程序的平台，请记住 **target id**。我们建议您尽可能的选择最高版本。您仍然可以建立你的应用程序来支持旧版本，但设置构建目标到最新版本，可以让你在最新设备上优化你的应用程序。

如果你没有看到任何 **target** 列出来，你需要使用 **Android SDK** 管理器工具来安装一些。请参阅[添加平台和软件包](#)。

3. 执行：

```
android create project --target<target-id> --name MyFirstApp \  
--path <path-to-workspace>/MyFirstApp --activity MainActivity \  
--package com.example.myfirstapp
```

从 **target** 列表中的 **id**(在上一步得到)更换 `<target-id>`，并更换 `<path-to-workspace>` 为你要保存你的 **Android** 项目的位置。

现在你的 **Android** 项目设置了几个默认的配置，你准备好开始构建应用程序。继续[下一课](#)。

提示： 把 `platform-tools/` 以及 `tool/` 目录添加到你的 **PATH** 环境变量中。

1.2 运行你的应用程序

如果你是按照[上一课](#)创建一个 Android 项目，它包含一些默认设置的“Hello World”的源文件，现在您可以立即运行这个应用程序。

如何运行你的应用程序依赖于两件事情：你是否有台个真实的 Android 设备，你是否使用 Eclipse。这堂课将告诉您，通过 Eclipse 或命令行工具，如何在真实的设备和 Android 模拟器中安装和运行您的应用程序。

在你运行你的应用程序之前，你应该注意在 Android 项目中的几个目录和文件：

Android Manifest.xml 文件

[manifest 文件](#)描述了应用程序的基本特征，并定义每个组件。在这个文件中您将学习到各种声明，随着你读到更多的课程。

manifest 应包括最重要的元素之一的是的<uses-sdk> 元素。使用的 android: minSdkVersion 和 android: targetSdkVersion 属性，可以声明您的应用程序对不同的 Android 版本的兼容性。对于你的第一个应用程序，它应该看起来像这样：

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="8"
android:targetSdkVersion="17" />
    ...
</manifest>
```

你总是应该设置 尽可能高的 android: targetSdkVersion 并在相应的平台版本中测试您的应用程序。欲了解更多信息，请阅读[支持不同的平台版本](#)。

SRC /

为您的应用程序的主源文件目录。默认情况下，它包含一个活动（Activity）类，当你的应用程序使用应用图标启动时就是此类在运行。

res/

包含[应用程序资源](#)的几个子目录。下面是其中几个：

drawable-hdpi/

专为高密度 (hdpi) 屏幕绘图对象 (如位图) 的目录。其它的绘图目录包含其它屏幕密度专用资产 (assets)。

layout/

定义你的应用程序的用户界面文件的目录。

values/

其它各种 XML 文件的目录，该目录包含一系列的资源，如字符串和颜色定义。

当你建立并运行默认的 **Android** 应用程序，默认的[活动](#)类启动并加载布局文件中说的“Hello World”。其结果是没有什么令人兴奋的，但重要的是你了解如何运行你的应用程序，然后开始开发。


在一台真实的设备上运行

如果你有一台真实的 **Android** 设备，下面是你安装和运行您的应用程序的方法：

1. 使用 **USB** 线将设备接入到你的开发机器上。如果你正在 **Windows** 开发，你可能需要为您的设备安装相应的 **USB** 驱动程序。安装驱动程序的帮助，请参阅 [OEM 的 USB 驱动程序](#) 文件。
2. 在您的设备上启用 **USB 调试**。
 - 在运行 **Android 3.2** 或以上版本的大多数设备上，你可以在这里找到选项 **设置>应用程序>开发**。
 - 在 **Android 4.0** 和更高版本，它是在“**设置**”>“**开发人员选项**”。

注意：在 **Android 4.2** 和更高版本中，**开发人员选项**默认是隐藏的。为了使其可用，进入 **设置>关于手机** 并点击 **生成数 (build number)** 七次。返回到前一个画面，找到 **开发人员选项**。

从 **Eclipse** 中运行的应用程序：

1. 打开你的项目的一个文件，并 从工具栏中单击“**运行**”.
2. 在出现的窗口中**运行**，选择 **Android 的应用程序**，并单击“**确定**”。

Eclipse 从连接的设备上安装应用程序，并启动它。

或运行您的应用程序的命令行：

1. 更改你的 Android 项目的根目录，然后执行：

```
ant debug
```

2. 请确保 Android SDK 的 `platform-tools/` 目录已经包含在你的 `PATH` 环境变量，然后执行：

```
adb install bin/MyFirstApp-debug.apk
```

3. 在设备上，找到 `MyFirstActivity` 并打开它。

这就是你如何在设备上构建并运行 Android 应用程序的方法！要开始开发，继续[下一课](#)。

在模拟器上运行

无论你使用 Eclipse 或命令行，要在模拟器上运行你的应用，你需要首先创建一个 [Android 虚拟设备](#) (AVD)。一个 AVD 就是一个 Android 模拟器，它可以让你模拟不同配置的设备。

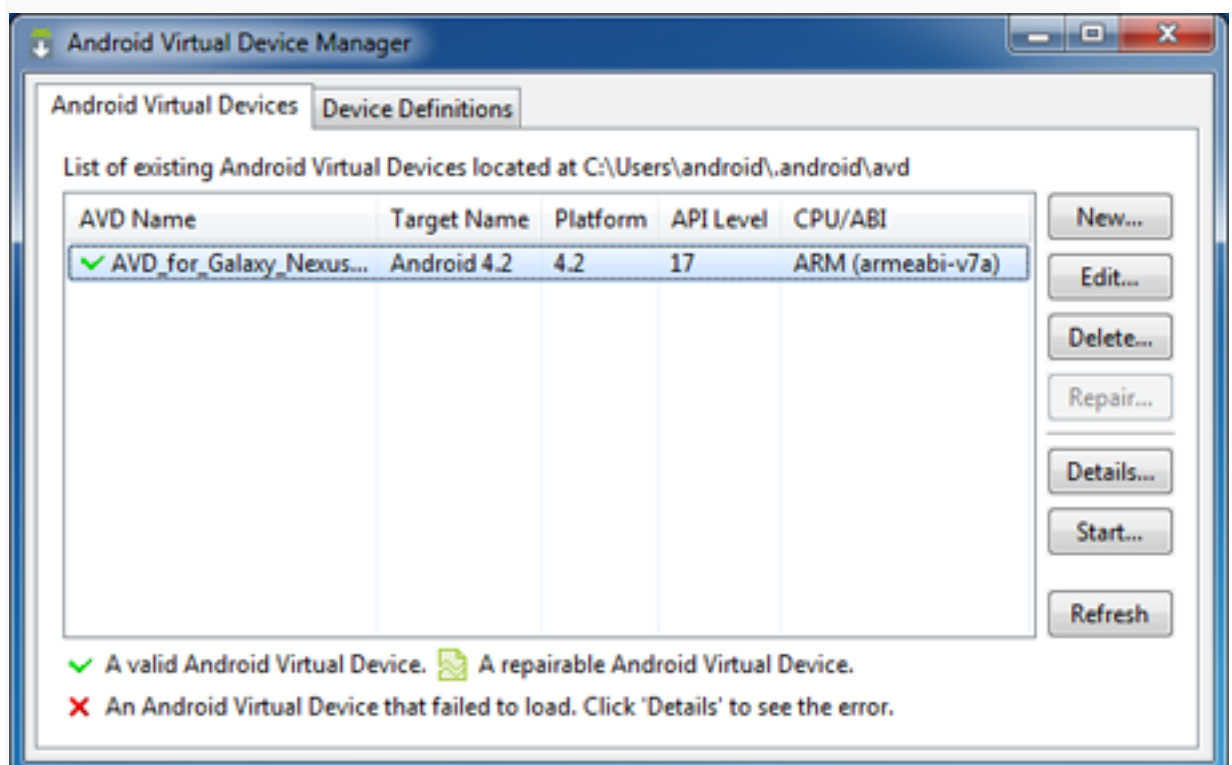


图 1AVD 管理器列出了几个虚拟设备。

要创建一个 AVD：

1. 启动 Android 虚拟设备管理：

- a. 在 Eclipse 中，从工具栏点击 Android 虚拟设备管理器 。
- b. 在命令行，更改目录到 `<SDK>/tool/` 执行：

```
android AVD
```

2. 在 *Android* 的虚拟设备管理器“面板”中，单击“新建”。
3. 填写 AVD 的细节。给它一个名字，一个平台目标，一个 SD 卡的大小和皮肤（HVGA 是默认值）。
4. 单击“创建 AVD”。
5. 从 *Android* 虚拟设备管理器中选择新的 AVD，单击“开始”。
6. 模拟器启动后，模拟器的屏幕解锁。

从 Eclipse 中运行的应用程序：

1. 打开你的项目的文件之一，并从工具栏中单击“运行” .
2. 在出现的窗口中运行，选择 **Android** 的应用程序，并单击“确定”。

Eclipse 会在 AVD 上安装应用程序，并启动它。

或者从命令行中运行你的应用程序：

1. 转到你的 **Android** 项目的根目录，然后执行：

```
ant debug
```

2. 请确保 Android SDK 的 `platform-tools/` 目录已经包含在你的 `PATH` 环境变量，然后执行：

```
adb install bin/MyFirstApp-debug.apk
```


3. 在模拟器上，找到 *MyFirstActivity* 并打开它。

这就是你如何在模拟器上构建和运行你的 Android 应用程序的方法！要开始开发，继续[下一课](#)。

1.3 构建一个简单的用户界面

Android 应用程序的图形用户界面，建立在视图（**View**）和视图组（**ViewGroup**）对象的层次结构上。视图对象通常是 UI 小部件（**widget**），如按钮或文本字段；而视图组对象是无形的视图容器，定义子视图的布局，如在一个网格或垂直的列表中。

Android 提供了一个 XML 对照表来对应视图和视图组的子类，这样你就可以在 XML 中使用的 UI 元素的层次结构来定义你的 UI。

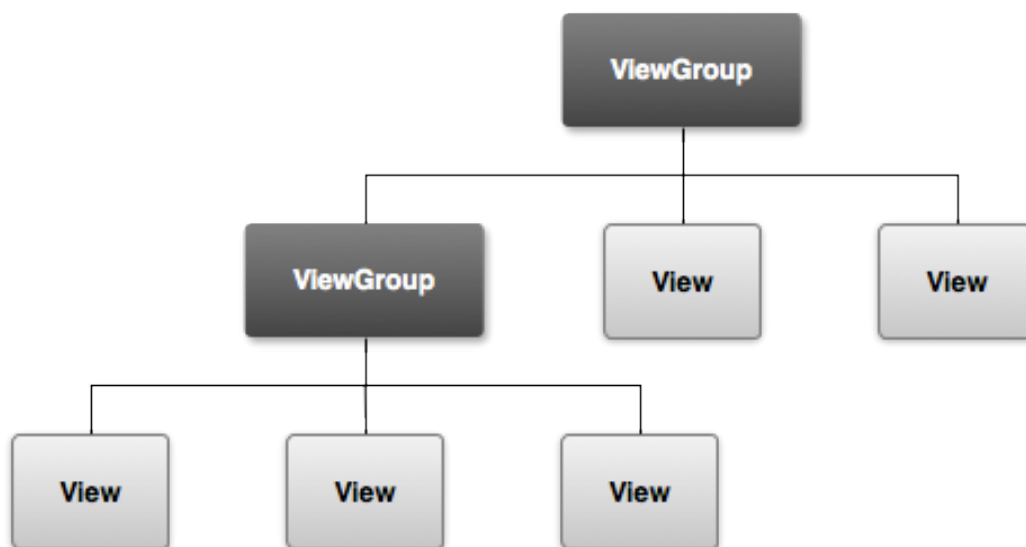


图 1。分支上的视图组对象是如何布局 and 包含其它视图对象的示意图

可替换的布局

在 XML 中声明 UI 布局，而不是在运行时代码中声明，是由于一些非常有用的原因。但它特别重要的原因是，这样你就可以为不同的屏幕大小创建不同的布局。例如，您可以创建两个版本的布局，并告诉系统在“小”屏幕上使用一个而在其它“大”屏幕上使用另一个。欲了解更多信息，请参阅[支持不同的设备](#)。

在本课中，您将在 XML 中创建一个包含一个文本字段和一个按钮的布局。在下面一课，当按钮被按下时，你会响应把文本字段中的内容发送到另一个活动。

创建一个线性布局

从 `res/layout/` 目录打开 `activity_main.xml` 文件。

注：在 Eclipse 中，当你打开一个布局文件，首先会显示图形布局编辑器。这是一个编辑器，帮助您使用所见即所得的工具构建布局。在这一课中，你会直接使用 XML，所以点击 `activity_main.xml` 的标签在屏幕底部打开 XML 编辑器。

当你创造了这个项目时，你选择的的 `BlankActivity` 模板包括 `activity_main.xml` 的文件与一个 `RelativeLayout` 根视图和一个 `TextView` 子视图。

首先，删除 `<TextView>` 元素和改变 `<RelativeLayout>` 元素 `<LinearLayout>`。然后添加 `android:orientation` 属性，将其设置为“**横向**”。结果看起来像这样：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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"
    android:orientation="horizontal" >
</LinearLayout>
```

`LinearLayout` 是一个视图组 (`ViewGroup` 的一个子类)，它根据 `android:orientation` 属性，将子视图按垂直或水平方向布局。`LinearLayout` 的每一个子视图将以 XML 中的顺序出现在屏幕上。

另外两个属性，`android:layout_width` 和 `android:layout_height`，对所有的视图都是必须的，用来指定它们的大小。

因为 `LinearLayout` 是在布局中的根视图，它应该充满整个应用程序屏幕区域，通过设定的宽度和高度为“`match_parent`”即可实现。此值声明视图的宽度或高度应该展开来匹配父视图的宽度或高度。

对于布局属性的更多信息，请参阅[布局指南](#)。

添加一个文本字段

要创建一个用户可编辑的文本字段，在 `<LinearLayout>` 元素中添加一个 `<EditText>`。

像每一个视图对象那样，你必须定义某些 XML 属性来指定的 `EditText` 对象的性质。以下是它在 `<LinearLayout>` 元素中的声明方法：

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

关于资源对象

资源对象仅仅是一个简单唯一的整数名称，它与某个应用程序资源关联起来，如位图，布局文件或字符串。

每个资源都有一个相应的资源对象，这些对象在项目的 `gen/R.java` 文件中定义。您可以使用 `R` 类中的对象名称来指向你的资源，例如你需要为 `android:hint` 属性指定一个字符串值。您还可以创建任意的资源 ID，使用 `android:id` 属性关联到一个视图，然后允许你通过它从其它代码来引用该视图。

每次编译您的应用程序时，SDK 工具会生成 `R.java` 文件。您永远不应该手动修改这个文件。

欲了解更多信息，请阅读“提供资源”指南。

关于这些属性：

`android:id`

它为视图提供了一个唯一的标识符，你可以用它在您的应用程序代码中引用该对象，如读取和操纵的对象（你会在下一课中看到）。

当你从 XML 中引用任何资源对象时，`at` 符号（`@`）都是必须的。它后面是资源类型（在本例中是 `id`），斜线，然后是资源名称（`edit_message`）。

只有当你首次定义资源 ID 时，资源类型之前的加号（`+`）才是必要的。当你编译应用程序，SDK 工具使用 ID 名称在项目的 `gen/R.java` 文件中创建一个新的资源 ID，这个资源 ID 指向 `EditText` 元素。一旦资源 ID 以这种方式声明过，其它对此 ID 的引用都不需要加号。只有当指定一个新的资源 ID 时使用加号是必要的，而对于字符串或布局这些具体资源是不需要的。更多信息参见资源对象。

`android:layout_width` 和 `android:layout_height`

不使用特定尺寸的宽度和高度，而用“`wrap_content`”指定视图应该实际大小来适应内容。如果改为使用“`match_parent`”，那 `EditText` 元素会填满屏幕，因为它要匹配父元素 `LinearLayout` 的大小。更多信息请参阅[布局指南](#)。

`android:hint`

这是一个文本字段为空时要显示的默认的字符串。不使用硬编码的字符串值，“`@string/edit_message`”的值指向一个单独文件中定义的字符串资源。因为它指向的是一个具体的资源（不只是一个标识符），它并不需要加号。但是，因为你还没有定义字符串资源，你首先会看到一个编译错误。你会在下一节中通过定义字符串来修正这个问题。

注：此字符串资源使用相同的名称作为元素的 ID： `edit_message`。然而，资源总是根据资源类型划分的（如 `ID` 或字符串），所以使用相同的名称不会发生冲突。

添加字符串资源

当需要在用户界面中添加文本时，你应该总是指定每个字符串作为一种资源。字符串资源，让您在一个单一的位置管理所有 UI 文本，这可以更容易找到并更新文本。外部化字符串还允许您为不同语言的每个字符串资源提供替代定义，实现应用程序本地化。

默认情况下，你的 Android 项目包括 `res/value/strings.xml` 字符串资源文件。添加一个新的字符串名为“`edit_message`”，并将其值设置为“Enter a message”（你可以删除“`hello_world`”字符串）。

在这个文件中，顺便为将要添加的按钮添加一个“Send”字符串，叫“`button_send`”。

`strings.xml` 的结果看起来像这样：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

对于使用字符串资源为您的应用程序本地化为其它语言的更多信息，请参阅[支持不同的设备](#) 课程。

添加一个按钮

现在添加一个 `<button>` 到布局中，紧随 `<EditText>` 元素：

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send" />
```

高度和宽度都设置为“`wrap_content`”，所以按钮是适应文本需要的实际大小。此按钮并不需要的 `android:id` 属性，因为它不会在活动代码中被引用。

输入框中填写屏幕宽度

根据目前设计的布局，`EditText` 和 `Button` 部件只需要以适合其内容的大小显示，如在图 2 中所示。

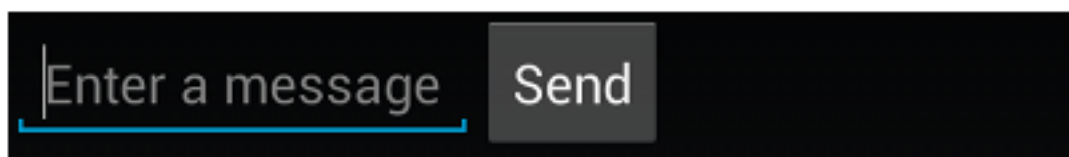


图 2。 `EditText` 和 `Button` 部件的宽度设置为 “`wrap_content`” 。

对于按钮这工作得很好，但文本字段则不一样，因为用户可能会输入较长的东西。所以，较好的做法应该是把文本字段填充在全部未使用的屏幕宽度上。你可以在 `LinearLayout` 内使用权重属性做到这一点，即指定 `android:layout_weight` 属性。

权重值是一个数字，它指定每个视图相对于同级视图对剩余空间所占的比例。这有点像在饮料配方中各种成分的数量：“2 份伏特加，1 份咖啡利口酒”是指饮料中三分之二是伏特加。例如，如果你给一个视图的权重值为 2 而另外一个的权重值为 1，那么总和是 3，所以第一个视图填充的剩余空间的 2/3，第二个视图填充剩下的部分。如果您添加了第三个视图，并给它一个权重为 1，那么现在第一个视图（权重为 2）将得到 1/2 的剩余空间，而剩下的两个各占 1/4。

所有视图的默认权重为 0，所以如果你只对某一个视图指定的权重值大于 0，等其它所有视图分配好它们需要的空间后，这个视图就会填充所有的剩余空间。因此，要在你的布局中给 `EditText` 元素填充所有剩余空间，只需要把它的权重设为 1，让按钮保持没有权重。

```
<EditText
    android:layout_weight="1"
    ... />
```

为了提高布局的效率,当你指定 `EditText` 部件权重时,你应该把它的宽度设为零(0DP)。把宽度设为零可以提高布局的性能,因为使用“`wrap_content`”作为宽度,系统计算的宽度最终是无关紧要的, 因为权重值会要求通过计算另一个宽度(即 `Button` 的宽度,译者注)来填充剩余的空间。(这句比较绕口,附上原文: `Setting the width to zero improves layout performance because using "wrap_content" as the width requires the system to calculate a width that is ultimately irrelevant because the weight value requires another width calculation to fill the remaining space.`)

```
<EditText  
    android:layout_weight="1"  
    android:layout_width="0dp"  
    ... />
```

图 3 显示了当你将所有的权重赋给 `EditText` 元素时的结果。



图 3 `EditText` 部件获得所有布局的权重,所以填充了 `LinearLayout` 中的剩余空间。

完整的布局文件现在看起来应该像下面这样:

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    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"  
    android:orientation="horizontal">  
    <EditText android:id="@+id/edit_message"  
        android:layout_weight="1"  
        android:layout_width="0dp"
```

```
        android:layout_height="wrap_content"

        android:hint="@string/edit_message" />

<Button


        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="@string/button_send" />

</LinearLayout>
```

SDK 工具创建项目时产生的默认的活动类指定此布局，所以你现在可以运行应用程序来查看结果：

- 在 Eclipse 中， 从工具栏中单击“运行”。
- 或从一个命令行，切换到你的 Android 项目的根目录并执行：

```
• ant debug
  adb install bin/MyFirstApp-debug.apk
```

接下来的一课中，你可以学习如何响应按钮按下，读取的文本字段的内容，启动另一个活动，等等。

1.4 启动另一个活动

完成[上一课](#)后，你有一个应用程序可以显示带有文本字段和一个按钮的活动（单屏）。在这一课中，您将在 `MainActivity` 中添加一些代码，使得用户点击“发送”按钮时能启动一个新的活动。

响应“发送”按钮

为了响应按钮的点击事件，打开 `activity_main.xml` 布局文件并添加 `android:onClick` 属性到 `<Button>` 元素：


```
<Button

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="@string/button_send"

    android:onClick="sendMessage" />
```

`android:onClick` 属性的值“`sendMessage`”，是用户单击按钮时系统调用的你的活动中的方法名称。

打开 `MainActivity` 类（位于项目的 `src/` 目录），并添加相应的方法：

```
/** Called when the user clicks the Send button */
public void sendMessage(View view) {
    // Do something in response to button
}
```

这需要你导入[视图](#)类：

```
import android.view.View;
```

提示：在 Eclipse 中，按 `Ctrl + Shift + O` 导入缺少的类（`CMD + SHIFT + O` 在 Mac）。

为了让系统匹配这个方法和 `android:onClick` 的方法名，签名必须完全如上所示。具体来说，该方法必须：

- 公共的
- 有一个 `void` 返回值
- 有一个[视图](#)作为唯一的参数（被点击的是视图）

接下来，您将填写这个方法读取该文本字段的内容，并提供该文本到另一个活动。

建立一个 Intent

一个[意图](#)是提供单独的组件（如两个活动）之间的运行时绑定的对象。[意图](#)表示一个应用程序“想做一些事情。” 您可以使用意图实现多种任务，但多数情况下，他们用来启动另一个活动。

在 `SendMessage()` 这个方法里面，创建一个启动活动的名称为 `DisplayMessageActivity` 的意图：

```
Intent intent = new Intent(this, DisplayMessageActivity.class);
```

这里使用的构造函数有两个参数：

- [上下文](#)作为第一个参数（这是因为[活动](#)类是[上下文](#)的一个子类）
- 应用程序的组成部分的类，它是系统要传达的[意图](#)（在这种情况下，应启动的活动）

向其它应用程序发送意图

在这一课中创建的意图是，称为显式意图，因为它指定意图需要的特定的应用程序组件。然而，意图也可以是隐式的，在这种情况下，[意图](#)并没有指定所需的组件，但允许在设备上安装的任何应用程序响应这个意图，只要这些应用程序满足元数据（[meta-data](#)）的动作([action](#)) 的规格，那些动作由不同的意图参数来规定。欲了解更多信息，请参阅[教程与其它应用程序交互](#)。

注：如果您使用的是 IDE，比如 Eclipse，对 `DisplayMessageActivity` 的引用 将增加一个错误，因为这个类尚不存在。现在忽略这个错误，你很快就会创建这个类。

意图不仅可以让你启动另一个活动，它还可以把一束数据带到这个活动。在 `SendMessage()` 方法中，使用 [findViewById\(\)](#) 方法得到 `EditText` 元素，并把它的文本值添加到意图：

```
Intent intent = new Intent(this, DisplayMessageActivity.class);
EditText editText = (EditText) findViewById(R.id.edit_message);
String message = editText.getText().toString();
intent.putExtra(EXTRA_MESSAGE, message);
```

注意： 您现在需要导入语句 `android.content.Intent` 和 `android.widget.EditText`。马上您将定义 `EXTRA_MESSAGE` 常量。

[意向](#)可以键-值对的方式带上各种数据类型，叫 **extra**。[putExtra\(\)](#) 方法将键名作为第一个参数，值作为第二个参数。

为了下一个活动能查询 **extra** 数据，你的意图 **extra** 应该使用公共常量作为键名。所以在 **MainActivity** 类的顶部添加 **EXTRA_MESSAGE** 的定义：

```
public class MainActivity extends Activity {  
    public final static String EXTRA_MESSAGE =  
    "com.example.myfirstapp.MESSAGE";  
    ...  
}
```

使用你应用的包名的前缀来命名意图的键名，通常是一个很好的做法。这确保它们是唯一的，如果你的应用程序还要与其它应用程序交互的话。

启动的第二个活动

要启动一个活动，调用 **startActivity()** 方法，向它传递你的[意图](#)。系统接收这个调用，并启动意图中 所指定的[活动](#)的一个实例。

加上这些新的代码，“发送”按钮调用的完整的 **SendMessage()** 方法现在看起来像这样：

```
/** Called when the user clicks the Send button */  
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
    String message = editText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
    startActivity(intent);  
}
```

为了让它可用，现在你需要创建 `DisplayMessageActivity` 类了。

创建第二个活动

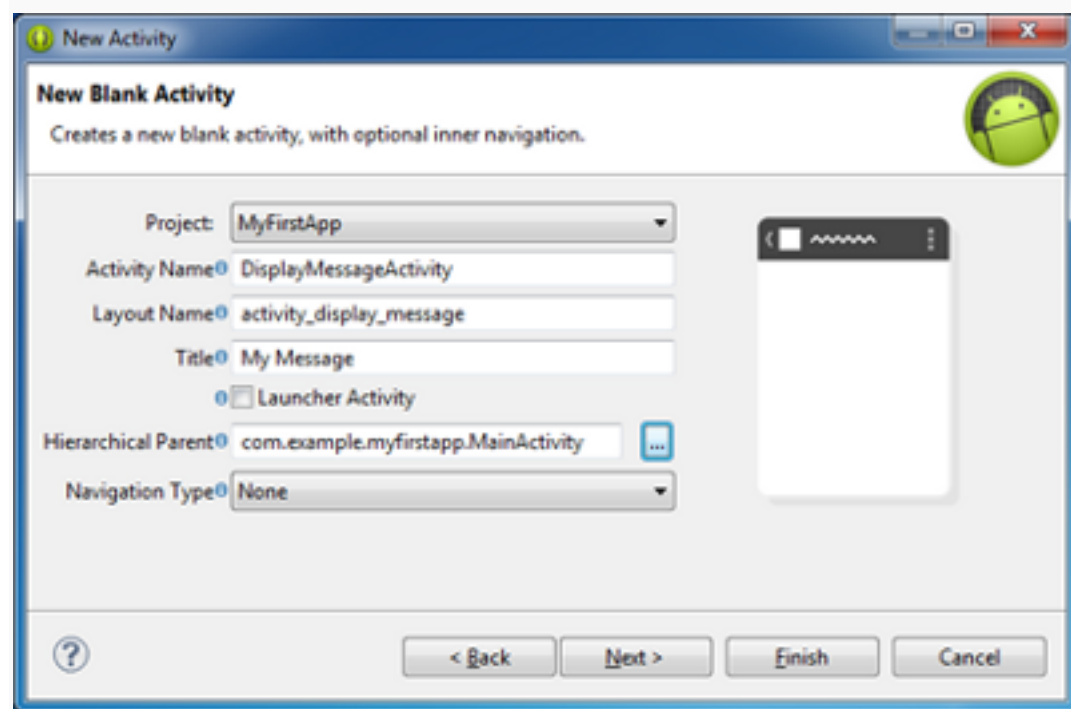


图 1：Eclipse 中的新建活动的向导。

要使用 Eclipse 创建一个新的活动：

1. 在工具栏上点击新建 .
2. 在出现的窗口中，打开 **Android** 的文件夹，并选择 **Android 活动**。单击“下一步”。
3. 选择 **BlankActivity**，并单击“下一步”。
4. 填写在活动的详细信息：
 - 项目 : MyFirstApp
 - 活动名称 : DisplayMessageActivity
 - 布局名称 : activity_display_message
 - 标题 : My Message
 - 谱系父 com.example.myfirstapp.MainActivity 的

○ 导航类型：无

单击“完成”。

如果你使用不同的 IDE 或命令行工具，在项目的 `src/` 目录创建一个新的文件名为 `DisplayMessageActivity.java`，在原来的 `MainActivity.java` 文件旁边。

打开的 `DisplayMessageActivity.java` 文件。如果你使用 Eclipse 来创建这个活动：

- 类已经包含了一个必需[的 `onCreate\(\)` 方法的实现](#)。
- 还有一个 [`onCreateOptionsMenu\(\)` 方法的实现](#)，但在这个应用程序你不需要它，所以你可以删除它。
- 还有一个 [`onOptionsItemSelected\(\)` 方法的实现，它](#)处理实施操作栏（action bar）中的向上的行为。请保留它原来的样子。

[ActionBar 的](#) API 是因为仅适用于 [HONEYCOMB](#)（API 级别 11）和较高的平台，您必须在 [`getActionBar\(\)` 方法](#)周围添加一个条件来检查当前的平台版本。此外，您必须[为 `onCreate\(\)` 方法添加 `@SuppressWarnings\("NewApi"\)` 标签](#)，避免[线头（lint）](#)错误。

`DisplayMessageActivity` 类现在看起来应该是这样的：

```
public class DisplayMessageActivity extends Activity {

    @SuppressWarnings("NewApi")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_message);

        // Make sure we're running on Honeycomb or higher to use ActionBar
        APIs

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            // Show the Up button in the action bar.
            getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }
}
```

```

@Override

public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {

        case android.R.id.home:

            NavUtils.navigateUpFromSameTask(this);

            return true;

        }

    return super.onOptionsItemSelected(item);

}
}

```

如果您使用 Eclipse 以外的 IDE，用上面的代码更新您的 `DisplayMessageActivity` 类。

[活动](#)的所有子类都必须实现 `onCreate()` 方法。当创建一个新的活动实例时，系统会调用这个方法。在这个方法中，你必须通过 `setContentView()` 方法定义活动的布局，并为你的活动组件进行初始设置。

注：如果您使用的是 Eclipse 以外的 IDE，你的项目不包含 `setContentView()` 需要的 `activity_display_message` 布局。那也是可以的，因为你稍后将更新此方法，并将不使用该布局。

添加标题字符串

如果你使用 Eclipse，你可以跳到[下一章节](#)，因为模板为新的活动提供了标题字符串。

如果您使用的是 Eclipse 以外的 IDE，在 `strings.xml` 文件中为新的活动添加标题：

```

<resources>

    ...

    <string name="title_activity_display_message">My Message</string>

</resources>

```

将它添加到 manifest

所有的活动都必须使用 `<activity>` 元素在 manifest 文件 `AndroidManifest.xml` 中声明。

当你使用 Eclipse 工具创建活动，它创建了一个默认项。如果你使用不同的 IDE，你需要自己添加 manifest 文件。它应该看起来像这样：

```
<application ... >

    ...

    <activity

        android:name="com.example.myfirstapp.DisplayMessageActivity"

        android:label="@string/title_activity_display_message"

        android:parentActivityName="com.example.myfirstapp.MainActivity" >

        <meta-data

            android:name="android.support.PARENT_ACTIVITY"

            android:value="com.example.myfirstapp.MainActivity" />

        </activity>

    </application>
```

`android:parentActivityName` 属性声明此活动在应用程序的逻辑层次的父活动的名称。该系统使用此值来实现默认的导航行为，如在 Android 4.1（API 级别 16）和更高的[向上导航](#)。通过使用[支持库](#)并添加如上所示的`<meta-data>`元素，您也可以为旧版本的 Android 提供相同的导航行为。

注意：你的 Android SDK 中应该已经包含了最新的 Android 支持库。它是 ADT 束附带的，但如果你使用不同的 IDE，你应该在[添加的平台和软件包](#)步骤中已经安装过。在 Eclipse 中使用模板时，支持库会自动添加到您的应用程序项目（你可以在 *Android Dependencies* 下看到库的 JAR 文件列表）。如果你不使用 Eclipse，你需要按照[建立支持库](#)指南手动将库添加到您的项目，然后再回到这里。

如果你正在使用 Eclipse 开发，可以运行现在的应用程序，但没有什么发生。点击“发送”按钮，启动第二个活动，但它使用模板所提供的一个默认“Hello world”的布局。你很快就会更

新活动来显示一个自定义的文本视图，所以如果你使用不同的 IDE，应用程序尚未编译也不要担心。

接收意图

每一个[活动](#)都是由一个[意图调用](#)，不管用户是怎么导航到那里。你可以通过 [getIntent\(\)](#) 方法得到调用您的活动的[意图](#)和它包含的数据。

在 `DisplayMessageActivity` 类 [onCreate\(\)](#) 方法，得到意图并提取提交到 `MainActivity` 的消息：

```
Intent intent = getIntent();  
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

显示信息

在屏幕上显示信息，创建一个 `TextView` 部件和使用 [setText\(\)](#) 方法设置文本。然后把 [TextView](#) 传到 `setContentView()` 将它作为动布局的根视图。

完整的 [onCreate\(\)](#) 方法 `DisplayMessageActivity` 现在看起来像这样：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    // Get the message from the intent  
    Intent intent = getIntent();  
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```



```
// Create the text view

TextView textView = new TextView(this);

textView.setTextSize(40);

textView.setText(message);


// Set the text view as the activity layout

setContentView(textView);

}
```

现在，您可以运行应用程序。当它打开时，在文本字段中键入信息，点击发送，然后信息会出现在第二个活动中。

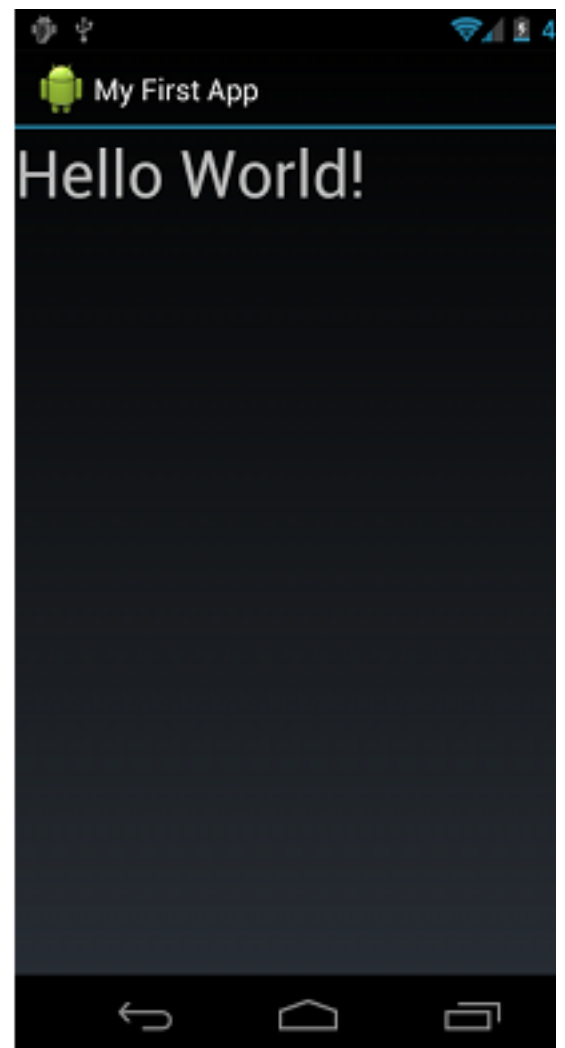
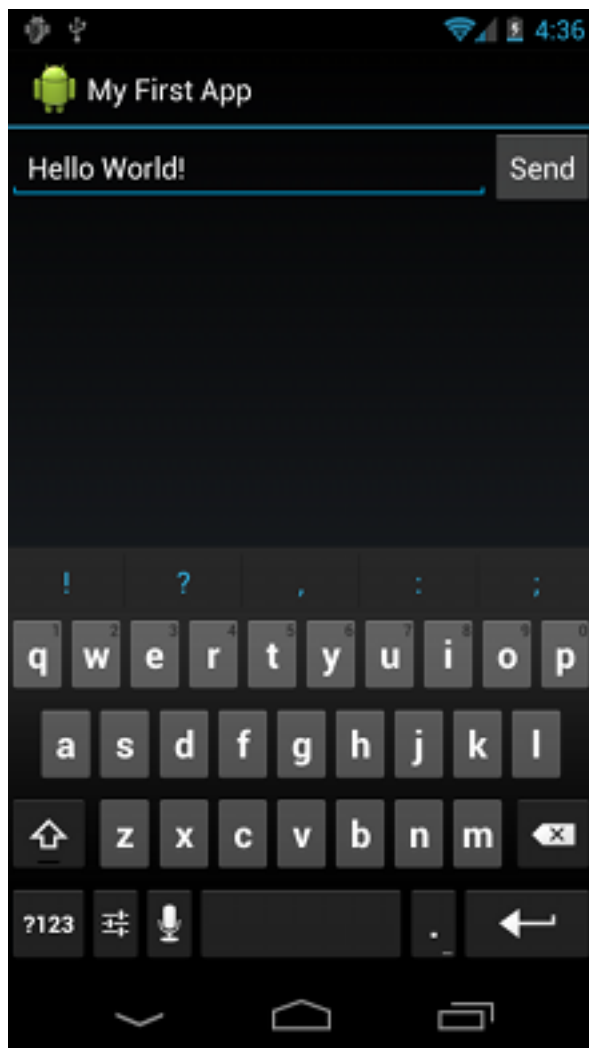


图 2。最终的应用程序里的两个活动，运行环境 Android 4.0。

就这样，你已经建立了你的第一个 **Android** 应用程序！

要了解更多关于构建 **Android** 应用程序，继续跟随我们的基本训练课程。下一课是管理[活动的生命周期](#)。

2 管理活动的生命周期

当用户浏览、离开、重新回到你的应用程序，你的应用程序的 [Activity](#) 实例在其生命周期的不同状态之间转换。例如，当您的活动的第一次启动时，它来到系统的前台，并接收用户关注。在这个过程中，**Android** 系统对活动调用了一系列的生命周期方法，在活动中设置了用户界面和其它组件。如果用户执行一个动作启动另一个活动或切换到另一个应用程序，在活动移动到后台时（此时活动不再可见，但实例及其状态仍然完好），系统会调用它的另外一组生命周期方法。

在生命周期回调方法中，你可以声明当用户离开并重新进入活动时你的活动的行为方式。例如，如果你正在构建一个流媒体视频播放器，当用户切换到另一个应用程序时，你可能会暂停视频并终止网络连接。当用户返回时，你可以重新连接到网络，并允许用户从相同的位置恢复播放视频。

这个课程讲解了每个 **Activity** 实例接收的重要生命周期回调方法，以及你如何使用它们让你的活动以用户期望的方式执行，并在活动不需要时不消耗系统资源。

课程

[启动活动](#)

学习活动的生命周期，用户如何启动你的应用程序，以及如何执行基本活动的创建的基本知识。

[暂停和恢复活动](#)

了解当你的活动被暂停（部分遮盖）和恢复时发生了什么，以及在这些状态变化中你应该做什么。

[停止和重新启动活动](#)

了解当用户完全离开你的活动和回到它时发生了什么。

[重建活动](#)

了解活动被销毁时会发生了什么，以及你如何能在必要时重建活动的状态。

2.1 启动活动

不像其它的用 `main()` 方法启动应用程序的编程范式，Android 系统启动 **Activity** 实例的代码，是通过调用特定的回调方法，这些方法对应其生命周期特定阶段的。有一组有序的回调方法启动活动，也有一组有序的回调方法拆除活动。

这个课程提供了最重要的生命周期方法的概述，并告诉您如何处理创建一个您的活动新的实例的第一个生命周期回调方法。

了解生命周期回调

在一个活动的生命过程中，系统按照一种类似阶梯金字塔的顺序，调用一组生命周期方法。就是说，活动的生命周期的每个阶段是金字塔上的一个单独的步骤。系统创建了一个新的活动实例时，每一个回调方法就把活动状态向顶部移动一步。金字塔的顶部是活动在前台运行的一个点，用户可以在这里与它进行交互。

当用户开始离开活动时，为了拆除活动，系统会调用其它的方法把活动状态从金字塔向下移动。在某些情况下，活动将只从金字塔向下移动一部分并等待（如当用户切换到另一个应用程序），活动可以从用户离开的地方重新回到顶部（如果用户返回到活动）并恢复。

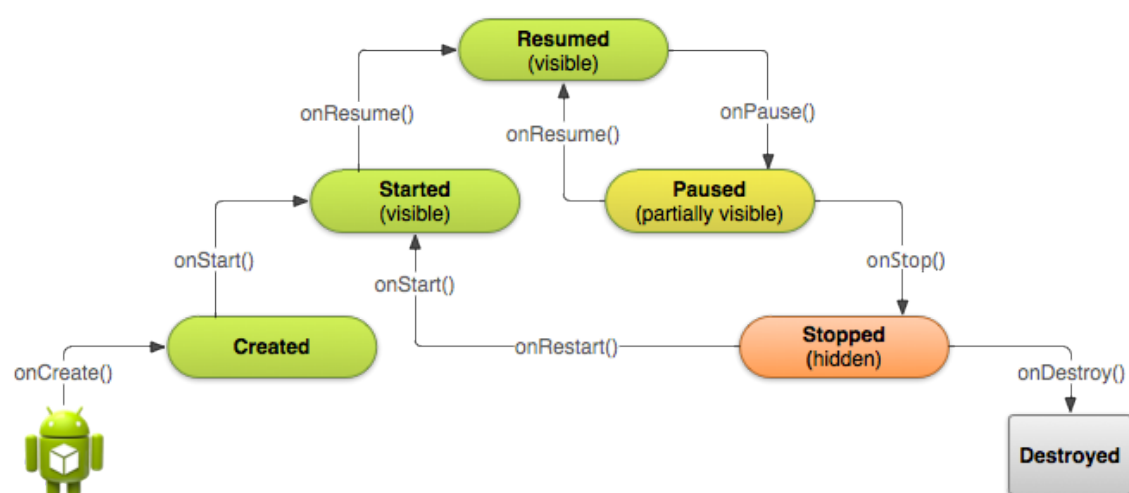


图 1 简要说明了活动的生命周期，表现为一个阶梯金字塔。它显示出，对于每个让活动移动到恢复状态的回调，都有对应的回调方法让活动状态往下降。活动也可以从暂停和停止状态返回到恢复状态。

根据您的活动的复杂性，您可能并不需要实现所有的生命周期方法。然而，重要的是你了解它们的每一个，并实现那些用来确保您的应用程序按照用户期望的方式执行的方法。正确地实现您的活动的生命周期方法，确保您的应用程序的行为在以下几个方面表现很好：

- 当使用您的应用程序时，如果用户收到一个电话，或切换到另一个应用程序，你的应用程序不会崩溃。
- 当用户没有活跃地使用它时，不消耗宝贵的系统资源。
- 当用户离开你的应用程序并在以后的时间返回到它时，不失去他们的进度。
- 当屏幕在横向和纵向之间旋转时，不会崩溃或失去用户的进度。

在你将要学习的以下课程中，有几种情况下，一个活动会像图 1 所示那样在不同状态之间转换。然而，其中只有三个状态是静态的。即，在相当长的一段时间内，活动可以停留在三种状态中的某一个：

恢复

在这种状态下，该活动是在前台，并且用户可以与它进行交互。（有时也被称为“运行”状态。）

暂停

在这种状态下，活动被另一个活动部分遮蔽——另一个在前台的活动是半透明或不覆盖整个屏幕的。暂停的活动不接收用户输入，也不能执行任何代码。

停止

在这种状态下，该活动是完全隐藏并对用户是不可见的，它被认为是在后台中。停止时，活动实例及其所有状态信息（如成员变量）都会被保留，但它不能执行任何代码。

其它状态（创建和启动）是瞬时的，通过调用下一个生命周期回调方法，系统将迅速从这些状态移动到下一个状态。也就是说，在系统调用 `onCreate()` 方法后，会马上调用 `onStart()` 方法，紧接着是 `onResume()` 方法。

这就是基本的活动生命周期。现在，你要开始学习一些特定的生命周期行为。

指定你的应用程序的启动活动

当用户从主屏幕中选择你的应用程序图标后，系统调用您的应用程序中的[活动 onCreate\(\)](#) 方法，这个活动是你已经声明为“启动”（或“主”）的活动。这个活动作为您的应用程序的用户界面的主要切入点。

您可以在 **Android manifest** 文件中定义使用哪个活动为主要活动，[AndroidManifest.xml 文件](#)在你的项目根目录下。

你的应用程序的主活动，必须在 **manifest** 文件中通过[<intent-filter>](#)标签声明，[<intent-filter>](#)标签包含主操作与启动类别。例如：

```
<activity android:name=".MainActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

注意：当你用 **Android SDK** 工具创建一个新的 **Android** 项目时，默认的项目文件包括一个用这个过滤器声明的 **Activity** 类。

如果主活动或启动类别之一没有在你的所有活动中的任意一个里声明，那么你的应用程序的图标将不会出现在主屏幕的应用程序列表上。

创建一个新的实例

大多数应用程序包括几个不同的活动，允许用户执行不同的操作。无论是用户点击你的应用程序图标生成的主活动，还是为了响应用户操作而启动的不同活动，系统都[通过调用 onCreate\(\)](#) 方法创建每一个新的 **Activity** 实例。

您必须实现 [onCreate\(\)](#) 方法来执行基本的应用程序的启动逻辑，这些启动逻辑在整个生命活动应该只发生一次。比如，可以在你的 [onCreate\(\)](#) 方法的实现中，定义用户界面，并可能实例化一些类范围的变量。

例如，下面的例子的 `onCreate()` 方法展示了一些代码，这些代码为活动进行一些基本的设置，如用户界面（在 XML 布局文件中定义）声明，定义成员变量和配置的某些 UI。

```
TextView mTextView; // Member variable for text view in the layout

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project
    res/layout/main_activity.xml file
    setContentView(R.layout.main_activity);

    // Initialize member TextView so we can manipulate it later
    mTextView = (TextView) findViewById(R.id.text_message);

    // Make sure we're running on Honeycomb or higher to use ActionBar
    APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // For the main activity, make sure the app icon in the action
        bar
        // does not behave as a button
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```

注意：使用 `SDK_INT` 的方式防止旧系统执行新的 API 工作只在 Android 2.0（API 5 级）和更高版本可用。旧版本会遇到运行时异常。

`onCreate()` 方法执行完毕后，系统快速地先后调用 `onStart()` 和 `onResume()` 方法。您的活动从来没有驻留在创建或启动状态。从技术上说，`onStart()` 被调用时活动变得对用户

可见，但紧接着 `onResume()` 方法就被调用，然后活动停留在恢复状态，直到发生某些事情改变它，如当收到一个电话，用户导航到其它活动，或者设备屏幕关闭。

在接下来的其它的课程中，你会看到其它启动方法 `onStart()` 和 `onResume()`，用于从暂停或停止状态恢复活动时，在您的活动的生命周期是如何有用的。

注：`onCreate()` 方法包含的参数叫做 `savedInstanceState`，它会在后面的课程“重建一个活动”中讨论。

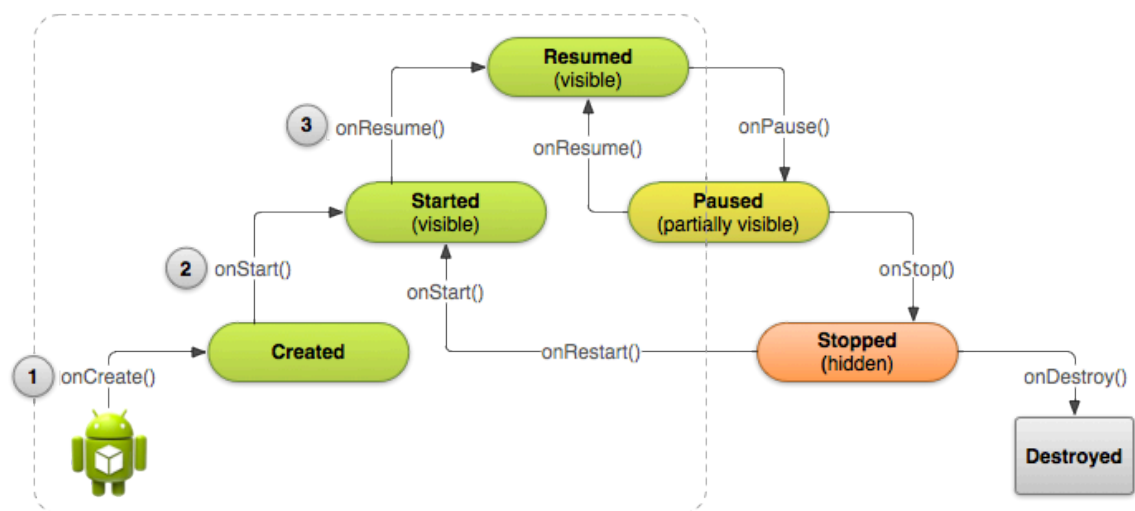


图 2。另一个活动的生命周期结构的展示，它突出了创建一个新活动时，系统依次调用的三个主要回调方法：`onCreate()`、`onStart()`和 `onResume()`。当这个有序的回调完成时，活动到达恢复状态，此时用户可以与活动进行交互直到他们切换到另一个不同的活动

销毁活动

活动的第一个生命周期回调是 `onCreate()` 方法，而它的最后一个回调则是 `onDestroy()` 方法。系统对你的活动调用这个方法，作为您的活动实例被完全从系统内存中移除的最终信号。

大多数应用程序并不需要实现这个方法，因为本地类的引用会被活动销毁，而活动应在 `onPause()` 和 `onStop()` 方法中执行大部分清理。但是，如果你的活动包含在 `onCreate()` 方法里创建的后台线程，或其它长时间运行的资源，而这些资源如果没有正确关闭可能造成内存泄漏的话，你应该在 `onDestroy()` 方法里杀掉它们。


```
override  
public void onDestroy() {  
    super.onDestroy(); // Always call the superclass  
  
    // Stop method tracing that the activity started during onCreate()  
    android.os.Debug.stopMethodTracing();  
}
```

注：系统总是在已经调用 `onPause()` 和 `onStop()` 方法之后才调用 `onDestroy()` 方法，除了一种情况：当你在 `onCreate()` 方法中调用 `finish()` 方法。在某些情况下，比如您的活动作为一个临时的决策者来启动另一个活动时，你可能会从活动的 `onCreate()` 方法中调用 `finish()` 方法来销毁活动。在这种情况下，系统会立即调用 `onDestroy()` 方法而不用调用任何其它生命周期方法。

2.2 暂停和恢复活动

在应用程序的正常使用过程中，位于前台的活动有时会被其它可视组件遮挡，导致活动暂停。例如，当打开一个半透明的活动（如一个对话框样式的活动）时，之前的活动就会暂停。只要活动仍然是部分可见的，但又不是当前的焦点活动，它会保持暂停。

然而，一旦活动被完全遮挡和不可见时，它就会停止（在下一课中讨论）。

在您的活动进入暂停状态时，系统调用活动的 `onPause()` 方法，它允许你停止正在进行的行动，这些行动不应该暂停时继续（比如视频），或保存任何应永久保存的信息，以防用户接着离开您的应用程序。如果用户从暂停状态返回到你的活动，系统会恢复它并调用 `onResume()` 方法。

注意：当你的活动接到一个 `onPause()` 方法的调用，它可能表明该活动将被暂停一会儿，然后用户可能重新返回到您的活动。然而，它通常还是用户离开你的活动的第一个迹象。

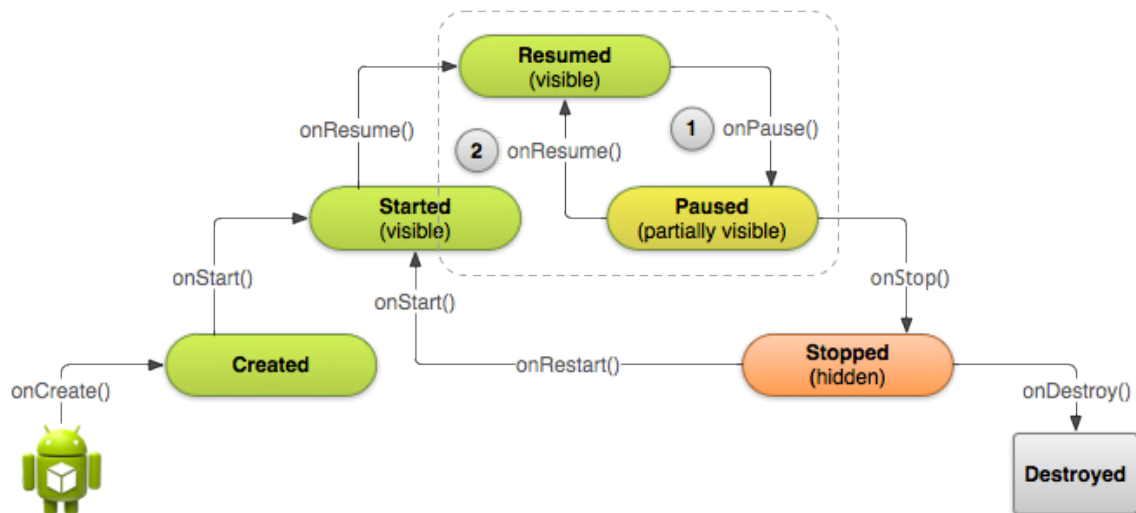


图 1。半透明的活动遮挡你的活动时，系统调用 `onPause()` 方法，而活动在暂停状态（1）处等待。如果用户返回时，活动仍处于暂停状态时，系统会调用 `onResume()` 方法（2）。

暂停你的活动

当系统为您的活动调用 `onPause()` 方法，它在技术上意味着你的活动仍是部分可见，但最常见的是一个迹象是，用户正在离开活动，而活动很快将进入停止状态。通常你应该为以下目的使用 `onPause()` 回调方法：

- 停止动画或其它正在进行的可能消耗 CPU 的行动。
- 提交未保存的更改，但只在用户希望在他们离开时永久保存这样的变化的情况下（如电子邮件草稿）。
- 释放系统资源，如广播接收器，处理传感器（如 GPS），或任何可能会影响电池寿命的。当您的活动被暂停时用户并不需要他们的资源，。

例如，如果您的应用程序使用 `Camera` 类，`onPause()` 方法是一个好释放它的地方。

```
@Override
public void onPause() {
    super.onPause(); // Always call the superclass method first
```

```
// Release the Camera because we don't need it when paused
// and other activities might need to use it.

if (mCamera != null) {
    mCamera.release()
    mCamera = null;
}
}
```

一般情况下，你不应该使用 [onPause\(\)](#) 来存储用户的变化（如表单中填写的个人信息）到永久存储。你唯一要用 [onPause\(\) 方法](#) 坚持永久存储用户更改，是当你确保用户期望那些更改被自动保存（如起草一封电子邮件时）。然而，你应该避免在 [onPause\(\) 方法中](#) 执行 CPU 密集型工作，如写入数据库，因为它会减缓到下一个活动可视化转化（你应该在 [onStop\(\) 方法](#) 执行期间重载关闭操作）。

你应该保持 [onPause\(\)](#) 方法中的一系列操作相对的简单，让用户在你的活动实际上是被停止时能快速转到下个目标。

注： 当你的活动被暂停时，[活动](#) 实例继续驻留在内存中，并在活动恢复时重新召回。在回到恢复状态时，你不需要重新初始化任何在回调方法中创建的组件。

恢复你的活动

当用户从暂停状态恢复您的活动时，系统会调用 [onResume\(\)](#) 方法。

注意系统在您的活动每次来到前台时调用此方法，包括活动第一次被创建的时候。因此，你应该实现 [onResume\(\)](#) 来初始化那些在 [onPause\(\)](#) 期间释放的组件，并执行任何其它每次进入恢复状态必须发生的初始化，（例如只用于活动获得用户焦点时的开始动画和初始化组件）。

下面例子的 [onResume\(\) 方法](#) 是对应上面例子的 [onPause\(\)](#) 的，所以它初始化活动暂停时释放的相机对象。

```
@Override
public void onResume() {
    super.onResume(); // Always call the superclass method first

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

2.3 停止和重新启动活动

正确地停止与重新启动您的活动，在活动生命周期中是一个重要的过程，它确保您的用户感知到您的应用永远是活动的，并没有丢失他们的进度。有几个关键场景，您的活动将会停止并重新启动：

- 用户打开最近应用程序窗口，并从你的应用程序切换到另一个应用程序。你的应用程序中目前在前台的活动就停止。如果用户通过主屏幕启动图标或最近使用的应用程序窗口返回到您的应用程序，活动将重新启动。
- 用户在你的应用程序执行一个动作，开启一个新的活动。在第二个活动创建时，当前的活动就停止。如果用户接着按下“返回”按钮时，第一个活动会重新启动。
- 用户在他或她的手机上使用您的应用程序时接到一个电话。

[活动](#)类提供了两个生命周期方法 [onStop\(\)](#) 和 [onRestart\(\)](#)，让你在活动停止和重新启动时专门处理它。与标识部分 UI 被遮挡的暂停状态不同的是，停止状态表明用户界面是不再可见，用户的焦点是在另一个活动（或另一个的应用程序）。

注：由于在你的[活动](#)实例停止时，系统保留它在系统内存中，你可能并不需要实现 [onStop\(\)](#) 和 [onRestart\(\)](#) 方法（或 [onStart\(\)](#) 方法都不用）。对于大多数比较简单的活动，活动可以很好地停止和重新启动，你可能只需要使用 [onPause\(\)](#) 方法暂停正在进行的行动，并断开系统资源。

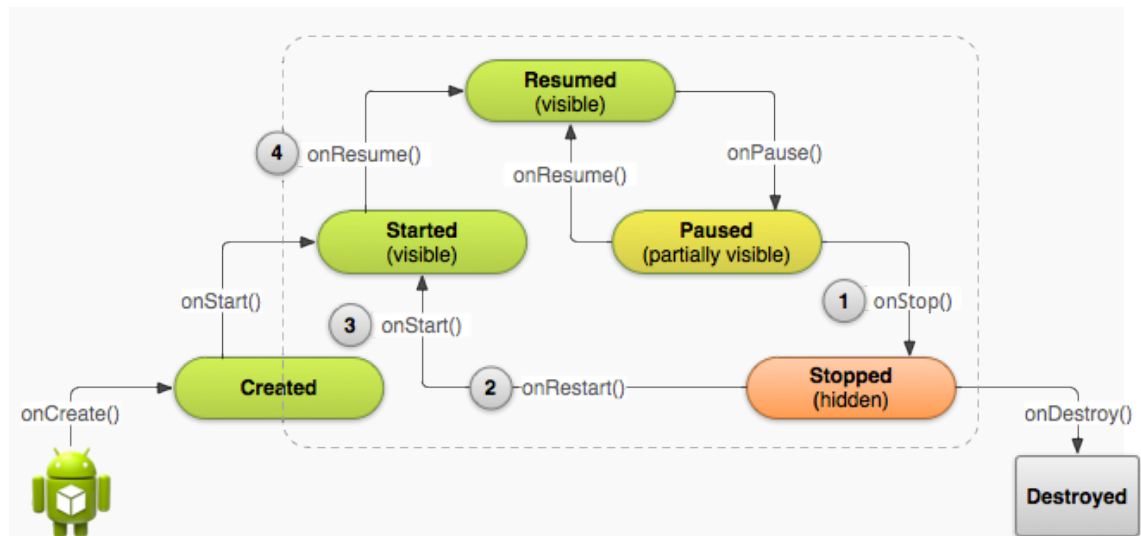


图 1：当用户离开你的活动时，系统调用 `onStop()` 停止活动（1）。如果用户在活动停止时返回，系统调用 `onRestart()`（2），紧接着调用 `onStart()`（3）和 `onResume()`（4）。请注意，无论在什么样的情况下导致活动停止，系统总是调用 `onStop()` 前调用 `onPause()`。

停止你的活动

当您的活动收到 `onStop()` 方法的调用时，它不再可见，用户不使用它时应该释放几乎所有不需要的资源。一旦您的活动停止了，如果需要回收系统内存，系统可能会销毁这个活动实例。在极端的情况下，系统可能只是杀掉了你的应用程序的过程而并不调用活动最终的 `onDestroy()` 回调方法，所以使用 `onStop()` 来释放可能会导致内存泄漏的资源对你很重要。

虽然 `onPause()` 方法在 `onStop()` 之前调用，你应该使用 `onStop()` 执行大量的更密集的 CPU 关闭操作，比如将信息写入到数据库。

例如，这里有一个 `onStop()` 方法的实现，它把草稿的内容保存到永久存储中：

```
@Override
protected void onStop() {
    super.onStop(); // Always call the superclass method first
```

```

        // Save the note's current draft, because the activity is stopping
        // and we want to be sure the current note progress isn't lost.
        ContentValues values = new ContentValues();
        values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
        values.put(NotePad.Notes.COLUMN_NAME_TITLE,
getCurrentNoteTitle());

        getResolver().update(
            mUri,    // The URI for the note to update.
            values,  // The map of column names and new values to apply
to them.

            null,    // No SELECT criteria are used.
            null     // No WHERE columns are used.

        );
    }
}

```

当你的活动停止后，[活动](#)对象是驻留在内存中，在活动恢复时被召回。在恢复状态你不需要重新初始化任何回调方法中创建的组件。该系统还为布局中的每个[视图](#)跟踪当前状态，所以如果用户输入文本到一个 **EditText** 部件，该内容将被保留，所以你就不需要保存和恢复它。

注：即使系统在你的活动停止时销毁了它，系统仍然在一个 **Bundle**（键-值对集合）中保留[视图](#)对象的状态（比如一个 **EditText** 中的文本），并在用户导航回到同一个活动实例时恢复它们（[下一课中](#)会更多地谈论关于使用 **Bundle** 保存其它状态数据，以防你的活动被销毁并重新创建）。

启动/重新启动你的活动

当你的活动从停止状态回到前台，它会接收到 [onRestart\(\)](#) 方法的调用。每次当您的活动变得可见时，系统还会调用 [onStart\(\)](#) 方法（无论是重新启动还是第一次创建）。然而，

[onRestart\(\)](#)方法只有在活动从停止状态恢复时被调用，所以你可以用它来执行一些特殊的恢复工作，那些可能只有当活动之前是停止而不是被销毁时才必需的恢复。

一个应用程序要使用 [onRestart\(\)](#) 在还原活动的状态是很少见的，所以这个方法没有适用于普遍应用的指南。然而，因为您的 [onStop\(\)](#) 方法基本上会清理所有的活动的资源，你需要在活动重新启动时重新实例化它们。但是，当你第一次创建活动（没有现成的活动实例的时候），你也需要实例化它们。基于这个原因，你通常会配对地使用 [OnStart\(\)](#) 回调方法和 [onStop\(\)](#) 方法，因为活动创建时和活动从停止状态重新启动时，[系统都会调用 OnStart\(\) 方法](#)。

例如，由于用户可能在回到您的应用程序之前已经远离它很长一段时间了，[OnStart\(\)](#) 方法是一个很好的验证所需的系统功能已经启用的地方：

```
@Override
protected void onStart() {
    super.onStart(); // Always call the superclass method first

    // The activity is either being restarted or started for the first
    time

    // so this is where we should make sure that GPS is enabled
    LocationManager locationManager =
        (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);

    boolean gpsEnabled =
        locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

    if (!gpsEnabled) {
        // Create a dialog here that requests the user to enable GPS, and
        use an intent

        // with the
        android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS action

        // to take the user to the Settings screen to enable GPS when they
        click "OK"
```

```

    }

}

@Override
protected void onRestart() {
    super.onRestart(); // Always call the superclass method first

    // Activity being restarted from stopped state
}

```

当系统销毁您的活动时，它对您的[活动](#)调用 [onDestroy\(\)](#) 方法。因为你通常应该已经在 [onStop\(\)](#) 方法中释放了你的大部分资源，当接收到 [onDestroy\(\)](#) 调用的时候，大多数应用程序没有太多的东西需要做了。这个方法是你清理可能会导致内存泄漏的资源的最后机会，所以你应该确保额外的线程被销毁，其它长期运行的操作（如方法跟踪）要停止。

2.4 重建活动

在以下几个场景中，由于正常的应用程序的行为，你的活动会被销毁：当用户按下“[返回](#)”按钮，或您的活动调用 [finish\(\)](#) 方法表示自身的销毁。当活动目前已停止而且在很长一段时间没有使用，系统也会销毁你的活动。或者如果前台活动需要更多的资源，系统必须关闭后台进程回收内存。

当用户按下[返回](#)或[活动](#)结束自己，系统对该活动实例的概念就会永远消失，因为这些行为表明不再需要该活动了。然而，如果由于系统的限制（而不是正常的应用程序的行为）让系统销毁了活动，即使实际的 [活动](#) 实例已经不存在了，系统仍会记得它曾经存在，当用户导航回来的时候，系统会使用一组之前保存的数据创建一个新的实例，这些数据描述了活动被销毁时的状态。系统用来还原到之前状态所保存的数据，被称为“实例状态”，它是存储在 [Bundle](#) 对象中的一个键-值对集合。

注意： 用户每次旋转屏幕时你的活动都会被销毁并重新创建。当屏幕改变方向，系统会销毁并重建前景活动，是因为屏幕的配置已经改变，你的活动可能需要加载替换资源（如布局）。

默认情况下，系统采用 **Bundle** 实例状态保存活动布局中每个视图对象的信息（`EditText` 对象中输入的文本值）。所以，如果你的活动实例被销毁并重新创建，不需要你写任何代码，布局的状态就可以恢复到之前的状态。然而，你的活动可能有更多你想要恢复的状态信息，如跟踪用户在活动中进度的成员变量。

注：为了让 Android 系统还原活动中视图的状态，每个视图必须有一个唯一的 ID，由 `android:id` 属性提供。

为了保存额外的活动状态的数据，你必须重写 `onSaveInstanceState()` 回调方法。系统在用户离开你的活动时调用这个方法，并把 **Bundle** 对象传递给它，**Bundle** 对象在您的活动意外销毁的事件中保存。如果系统稍后必须重建活动实例，它传递相同的对象到 `onRestoreInstanceState()` 和 `onCreate()` 方法。

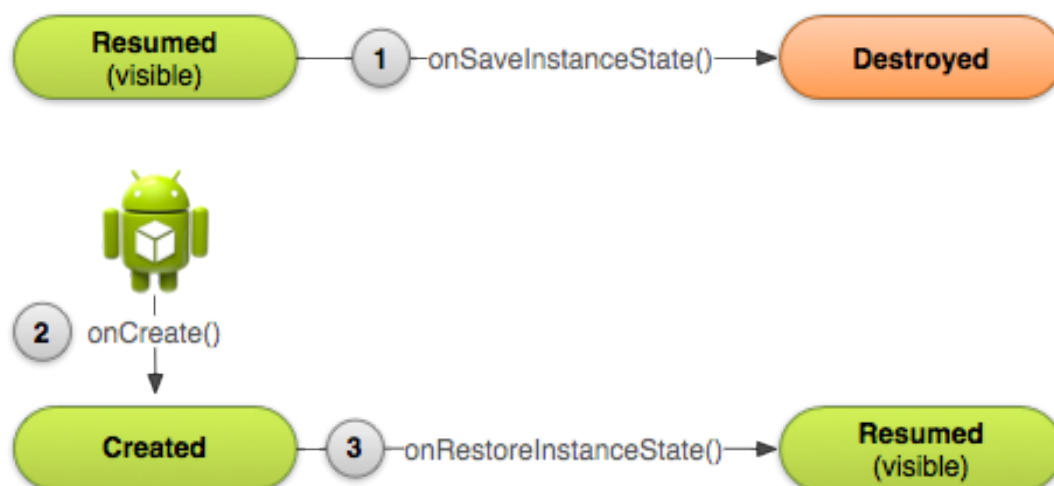


图 2。系统开始停止你的活动时，它会调用 `onSaveInstanceState()` 方法（1），所以你可以指定你想保存的状态数据，以防必须重新创建活动实例。如果活动被销毁而必须重新创建相同的实例，系统传递在（1）中定义的状态数据到 `onCreate()` 方法（2）和 `onRestoreInstanceState()` 方法（3）。

保存您的活动状态

活动开始停止时，系统调用 [onSaveInstanceState\(\)](#)，这样可以以键-值对集合的方式保存您的活动状态信息。这种方法的默认实现保存活动的视图所有层次结构的状态信息，如 `EditText` 部件的文本或某个 `ListView` 的滚动位置。

为了保存您的活动额外的状态信息，你必须实现 [onSaveInstanceState\(\) 方法并把键-值对](#) 添加到 `Bundle` 对象。例如：

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

注意：始终调用超类的实现 [onSaveInstanceState\(\)](#)，这样默认实现可以保存视图的层次状态。

还原活动状态

当你的活动在销毁后被重建时，你可以从系统传递给你的活动的 `Bundle` 还原保存的状态。[onCreat\(\)](#) 和 [onRestoreInstanceState\(\)](#) 回调方法获得相同包含实例的状态信息的 `Bundle`。

因为系统创建你的活动一个新的实例或重建之前那个都会调用 [onCreate\(\)](#) 方法，在你尝试读取状态 **Bundle** 时你必须先检查它是否为空。如果它是空的，那么系统是在创建活动的一个新实例，而不是还原之前被销毁的一个实例。

例如，下面是你在 `onCreate()` 中还原一些状态数据[的方法](#)：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass
    first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new
        instance
    }
    ...
}
```

相比于在 [onCreate\(\)](#) 中还原状态，您还可以选择实现 [onRestoreInstanceState\(\)](#) 方法，系统会在 [OnStart\(\)](#) 方法后调用它。系统只有在有保存的状态可以还原时才会调用 [onRestoreInstanceState\(\)](#) 方法，所以你不需要检查包 **Bundle** 是否为空：

```
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
}
```

```
mCurrentScore = savedInstanceState.getInt(STATE_SCORE);  
mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);  
}
```

注意：始终调用超类的实现 [onRestoreInstanceState\(\)](#)，这样默认实现可以还原视图的层次状态。。

要了解更多关于由于运行时重新启动事件而导致重建您的活动的信息(例如当屏幕旋转时)，请参阅[处理运行时更改](#)。

3 支持不同的设备

遍布在世界各地的 Android 设备有许多形状和大小。通过范围广泛的设备类型，您的应用程序有机会得到一个数量巨大的受众。为了能够在 Android 上尽可能地成功，你的应用需要适应不同的设备配置。你应该考虑的一些重要变化包括不同的语言、屏幕尺寸和 Android 平台的版本。

这节课教你如何使用基本的平台功能——利用替代资源和其它功能，让你的应用只使用一个单一的应用程序包（APK），却可以在各种兼容 Android 设备上提供优化的用户体验。

课程

支持不同的语言

了解如何利用替代字符串资源支持多国语言。

支持不同的屏幕

了解如何为不同的屏幕大小和密度优化用户体验。

支持不同的平台版本

了解如何使用新版本的Android可用的API，仍能同时继续支持旧版本的Android。

3.1 支持不同的语言

从您的应用程序代码中提取 UI 字符串并把它们存放在一个外部文件，总是一个很好的做法。通过每个 Android 项目中的资源目录，Android 使得这个做法的变得简单。

如果你使用了 Android SDK 工具创建你的项目（参考[创建一个 Android 项目](#)），该工具会在项目的顶层创建一个 `res/` 目录。在这个 `res/` 目录下，是各种资源类型的子目录。还有一些默认的文件，如 `res/value/strings.xml`，它保存字符串的值。

创建场景目录和String文件

要添加更多语言的支持，在 `res/` 目录下创建另外的 `values` 目录，这些目录包含一个连字符和 ISO 国家代码。例如，`values-es/` 是目录含有以“es”作为语言代码的环境下的简单资源。**Android** 根据设备在运行时的区域设置加载合适的资源。

一旦你决定了你要支持的语言，创建的资源子目录和字符串资源文件。例如：

```
MyProject/
    res/
        values/
            strings.xml
        values-es/
            strings.xml
        values-fr/
            strings.xml
```

为每个区域添加字符串值到相应的文件。

Android 系统在运行时，根据用户的设备上目前的语言环境设置，使用相应的一组字符串资源。

例如，以下是一些不同语言的不同字符串资源文件。

英语（默认语言环境）/`values/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

西班牙语/`values-es/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Aplicación</string>
```

```
<string name="hello_world">Hola Mundo!</string>

</resources>
```

法国/values-fr/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <string name="title">Mon Application</string>

    <string name="hello_world">Bonjour le monde !</string>

</resources>
```

注：您可以对任何资源类型的使用场景限定符（或任何配置限定符），例如，你要提供你的位图的本地化版本。如需了解更多信息，请参阅[本地化](#)。

使用字符串资源

你可以在你的源代码和其它 XML 文件，使用`<String>`的元素的 `name` 属性定义的资源名称引用字符串资源。

在你的源代码，你可以通过语法 `R.string.<string_name>` 引用一个字符串资源。有很多这种接受字符串资源的方法。

例如：

```
// Get a string resource from your app's Resources
String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

在其它的 XML 文件中，你可以在 XML 属性需要得到一个字符串资源时，通过@string/<string_name> 引用一个字符串值。

例如：

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/hello_world" />
```

3.2 支持不同的屏幕

对 Android 设备屏幕进行分类，一般使用两个属性：尺寸和密度。你应该预料到你的应用程序将被安装在不同范围的尺寸和密度屏幕的设备上。因此，您应该包含一些可替换资源，针对不同的屏幕大小和密度，优化你的应用程序的外观。

- 有四种普遍的尺寸：小(small)，正常(normal)，大(large)，超大(xlarge)
- 也有4种普遍的密度：低 (ldpi)，中 (mdpi)，高 (hdpi)，超高 (xhdpi)

声明你要在不同的屏幕中使用的布局和位图，你必须在不同的目录中放置这些可替换的资源，就像你为不同的语言字符串所做的那样。

另外要注意，屏幕方向（横向或纵向）被认为是一个屏幕大小的变化，所以很多应用程序应该针对每个方向修改布局，来优化用户体验。

创建不同的布局

为了优化在不同的屏幕尺寸上用户体验，，你应该为每个你想支持的屏幕尺寸创建一个独一无二的布局 XML 文件。每个布局应该保存到适当的资源目录，与一个 - <screen_size> 后缀来命名。例如，大屏幕唯一的布局应保存在 res/layout-large/目录下。

注： Android 自动拉伸你的布局来正确地适应屏幕。因此，你的不同屏幕尺寸的布局，不需要担心 UI 元素的绝对大小，而是把重点放在影响用户体验的布局结构上（如重要视图相对于同级视图的大小或位置）。

例如，这个项目包含一个默认的布局和一个可替换的大屏幕布局：

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

文件名必须是完全一样的，但它们的内容是不同的，以便为相应的画面尺寸提供优化的用户界面。

只需像往常一样在你的应用程序中引用这个布局文件：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

根据运行您的应用程序的设备屏幕尺寸，系统从布局目录加载适当的布局文件。关于 Android 是如何选择适当的资源的更多信息，可参考[提供资源](#) 指南。

再举一个例子，这里是一个有横向替换布局的项目：

```
MyProject/
    res/
        layout/
            main.xml
        layout-land/
            main.xml
```

默认情况下，纵向使用 `layout/main.xml` 文件。

如果你想提供一个特殊的横向布局，包括在大屏幕上，那么你需要同时使用 `larger` 和 `land` 标识符：

```
MyProject/
    res/
        layout/                # default (portrait)
            main.xml
        layout-land/           # landscape
            main.xml
        layout-large/          # large (portrait)
            main.xml
        layout-large-land/     # large landscape
            main.xml
```

注：Android 3.2 及以上的系统，支持一个先进的指定屏幕尺寸的方法，这个方法允许你根据与密度无关的最小宽度和高度像素，为不同的屏幕尺寸指定资源文件。这节课不包括这项新技术。欲了解更多信息，请阅读[多屏幕的设计](#)。

创建不同的位图

你应该总是对各个普遍的密度桶（bucket）提供正确缩放的位图资源：低，中，高和超高密度。这可以帮助你在所有的屏幕密度上实现良好的图形质量和性能。

要生成这些图像，你的原始资源应该是矢量格式的，并为各个密度生成的图像使用下面的拉伸比例：

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0（基线）
- ldpi: 0.75

这意味着，如果你为 xhdpi 设备生成一个 200x200 的图像，你应该为其它密度产生相同的资源：hdpi 是 150X150，mdpi 是 100x100，ldpi 是 75X75。

然后，将这些文件放在适当的绘制资源目录中：

```
MyProject/  
    res/  
        drawable-xhdpi/  
            awesomeimage.png  
        drawable-hdpi/  
            awesomeimage.png  
        drawable-mdpi/  
            awesomeimage.png  
        drawable-ldpi/  
            awesomeimage.png
```

任何时候你引用`@drawable/awesomeimage`，系统根据屏幕的密度选择合适的位图。

注意：低密度（ldpi）的资源并非总是必要的。当你提供 hdpi 的资产，系统会把它们缩小一半来正确地适应 ldpi 屏幕。

关于您的应用程序创建图标资产的更多的提示和指引，请参阅 [意象设计指南](#)。

3.3 支持不同的平台版本

虽然最新版本的 Android 往往为您的应用程序提供了最好的 API，但你应该继续支持旧版本的 Android，直至更多的设备得到更新。这节课告诉你如何利用最新的 API，同时继续支持旧版本。

根据访问谷歌 Play Store 的 Android 设备数量，在[平台版本](#)仪表板上定期更新，来显示运行每个 Android 版本的激活设备的分布。一般来说，支持约 90% 激活设备是一个很好的做法，同时针对您的应用程序到最新版本。

提示：为了提供跨 Android 版本的最好的特性和功能，你应该在你的应用程序使用 Android Support Library，它允许您使用最近的几个平台上的旧版本的 API。

指定最小和目标API级别

[AndroidManifest.xml](#) 文件描述了你的应用程序的细节，并确定它支持的 Android 版本。具体来说，`<uses-sdk>` 元素的 `minSdkVersion` 和 `targetSdkVersion` 属性识别与您的应用程序兼容的 API 最低级别，以及设计和测试您的应用程序的最高的 API 级别。

例如：

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15"
/>
    ...
</manifest>
```

随着新版本的 Android 被发布，一些风格和行为可能会发生变化。为了让您的应用程序以充分利用这些变化，并确保你的应用适合每个用户的设备的风格，你应该设置 `targetSdkVersion` 的值匹配最新的 Android 版本。

检查系统版本在运行

Android 在 [Build](#) 常量类中为各平台版本提供了一个唯一的代码。在你的应用中使用这些代码构建条件，确保在依赖 API 级别较高的代码只在那些 API 在系统中可用时才会执行。

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

```
}  
  
}
```

注：当解析 XML 资源，Android 会忽略当前设备不支持的 XML 属性。所以，你可以放心地使用那些仅支持较新版本的 XML 属性，不用担心遇到这些代码时破坏旧版本。例如，如果你设置了 `targetSdkVersion="11"`，默认情况下，你的应用在 Android 3.0 和更高版本中会包含 [ActionBar](#)。然后要在操作栏中添加菜单项，你需要在你的菜单资源 XML 中设置的 `android:showAsAction="ifRoom"`。在跨版本的 XML 文件中这样做是安全的，因为旧版本的 Android 根本无视 `showAsAction` 属性（也就是说，你并不需要一个独立的版本放在 `res/menu-v11/` 目录下）。

使用平台的样式和主题

Android 提供给应用程序的外观和感觉的底层操作系统的用户体验主题。这些主题可以应用到您的应用程序 `manifest` 文件内。您的应用程序通过使用这些内置的风格和主题，自然会按照最新的外观和感觉的 Android 每一个新版本。

为了使您的活动看起来像一个对话框：

```
<activity android:theme="@android:style/Theme.Dialog">
```

为了使您的活动有一个透明的背景：

```
<activity android:theme="@android:style/Theme.Translucent">
```

要应用在 `/res/values/styles.xml` 中的自己的自定义主题：

```
<activity android:theme="@style/CustomTheme">
```

主题应用到整个应用程序（所有活动），添加 `android:theme` 属性到 `<application>` 元素的：

```
<application android:theme="@style/CustomTheme">
```

有关创建和使用主题的更多信息，请阅读的[样式和主题](#)指南。

4 用碎片构建一个动态的用户界面

要在 Android 上创建一个动态的和多面板(multi-pane)的用户界面，你需要把 UI 组件和活动行为封装成模块，让模块可以添加到或者抽离你的各个活动。您可以使用[碎片 \(Fragment\)](#)类创建这些模块，这行为有点像一个嵌套的活动，可以定义自己的布局和管理自己的生命周期。

当一个碎片指定它自己的布局，它可以与活动内的其它碎片用不同的组合进行配置，来为不同的屏幕尺寸修改您的布局配置（小的屏幕可能会一次显示一个碎片，但一个大屏幕可以显示两个或更多）。

这个课程，向您展示如何使用碎片创建一个动态的用户体验，为不同屏幕尺寸的设备优化您的应用程序的用户体验，同时全部继续支持运行低至 **Android 1.6** 版本的设备。

课程

使用 Android 支持库 (Android Support Library)

了解如何通过捆绑 Android 支持库到您的应用程序，在早期版本的 Android 使用最新的框架 API。

创建一个碎片

了解如何构建一个碎片以及在其回调方法中实现基本的行为。

构建一个灵活的用户界面

了解如何使用为不同的屏幕提供不同的碎片配置的布局构建您的应用程序。

与其它碎片通信

了解如何设置从碎片到活动和其它碎片的通信路径。

4.1 使用支持库

Android 的[支持库](#)提供了一个 API 库的 JAR 文件，它可让您的应用程序在早期版本的 Android 上运行时，使用一些最新的 Android 的 API。例如，支持库提供了[碎片](#)的一个版本，您可以在 Android 1.6（API 4 级）和较高的版本中使用它。

这节课展示如何使用支持库设置您的应用程序，来使用碎片建立一个动态的应用程序 UI。

用支持库设置项目

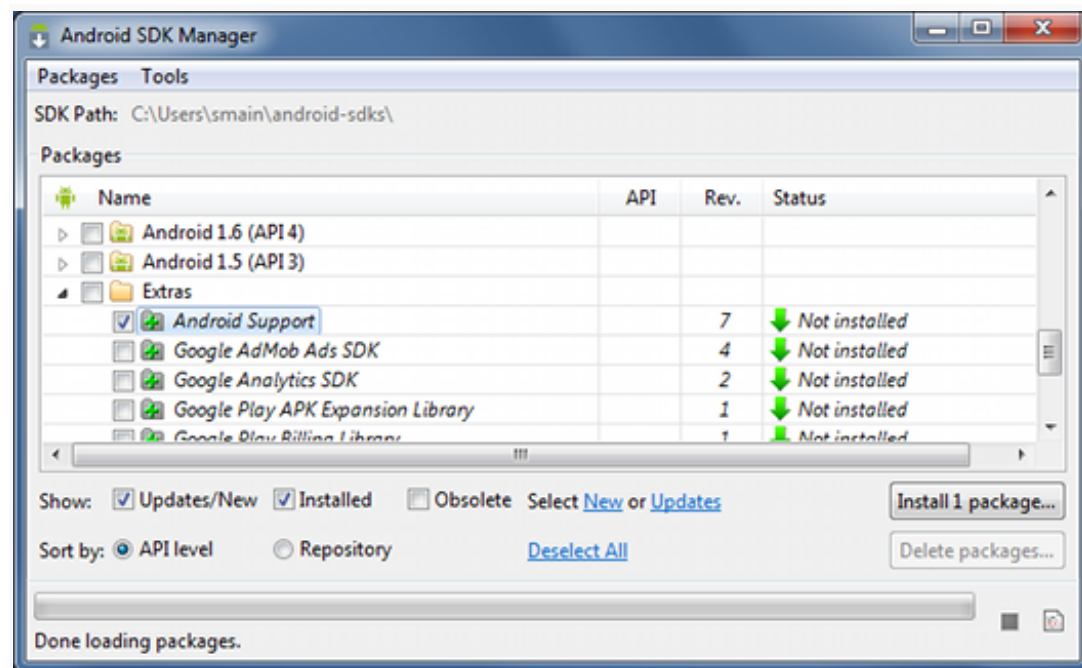


图 1。选中 Android 支持包的 Android SDK 管理器。

要设置项目：

1. 使用SDK管理器下载Android支持包。
2. 在你的Android项目的根级别创建一个`libs`目录。
3. 找到您要使用的JAR库文件，并将其复制到`lib/`目录。

例如，支持 API 级别 4 的库位于

`<sdk>/extras/android/support/v4/android-support-v4.jar`。

4. 更新manifest文件设置最低API级别为4和目标API级别为最新版本：

```
<uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
```


导入支持库API

支持库包括在最新版本的 Android 中添加或平台中根本不存在的各种 API，它只是在你开发特定应用程序功能时给你提供额外的支持。

你可以在 [android.support.v4.*](#) 的平台文档中找到支持库的所有 API 参考文档。

警告： 确保你没有意外地在旧系统版本上使用新的 API，保证你从

[android.support.v4.app](#) 包导入的 [Fragment](#) 类和相关 API：

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
...
```

在使用支持库创建活动来承载碎片时，你还必须扩展(`extend`)[FragmentActivity](#) 类，而不是传统的 [Activity](#) 类。在下一课中，你会看到碎片和活动的示例代码。

4.2 创建一个碎片

你可以把一个碎片考虑成活动的一个模块化的部分，它有自己的生命周期，接收自己的输入事件，您可以在活动运行时添加或删除它（有点像一个您可以在不同的活动中重用的“子活动”）。这节课展示如何使用支持库扩展 [Fragment](#) 类，使您的应用程序对运行低于 Android 1.6 版本系统的设备保持兼容。

注： 如果你由于其它原因决定你的应用程序需要的最低 API 级别为 11 或更高的，你并不需要使用支持库，可以改用框架的构置 [Fragment](#) 类和相关 API。只是要知道，这节课的重点是使用支持库的 API，它使用一个特定的软件包签名和有时与平台中包含的版本相比略有不同的 API 名称。

创建一个Fragment类

要创建一个碎片，扩展 [Fragment](#) 类，然后重写关键的生命周期的方法来插入你的应用程序逻辑，类似于你处理 [Activity](#) 类的方法。

创建碎片时的一个区别是，你必须使用 `onCreateView()` 回调来定义布局。事实上，这是为了让一个碎片运行你唯一需要的回调。例如，下面是一个简单的指定自己布局的碎片：

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container,
false);
    }
}
```

就像一个活动，一个碎片应实现其它让您管理它自己的状态生命周期回调方法，比如它从活动中添加或删除，以及活动在它的生命周期状态之间的过渡。例如，当活动的 `onPause()` 方法被调用时，在活动的任何碎片也会接收到 `onPause()` 调用。

关于碎片生命周期和回调方法的更多信息，可参看《[Fragment 开发指南](#)》。

使用XML添加碎片到活动

虽然碎片是可重用的、块化的 UI 组件，但是 `Fragment` 类的每个实例都必须与父 `FragmentActivity` 关联。你可以在你的活动布局 XML 文件内定义每个碎片来实现这种关联。

注意：`FragmentActivity` 是在支持库中提供的一种特殊的活动，用来在版本在 **API 级别 11** 更旧的系统中处理碎片。如果最低系统版本是 **API 级别 11** 或更高，那么你可以使用一个普通的 `Activity`。

下面是一个例子，当设备屏幕上被认为是“large”时（在目录名用 `largr` 标识），在布局文件中添加两个碎片到活动。

`res/layout-large/news_articles.xml`:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment
android:name="com.example.android.fragments.HeadlinesFragment"
        android:id="@+id/headlines_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment
android:name="com.example.android.fragments.ArticleFragment"
        android:id="@+id/article_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

提示：有关为不同的屏幕尺寸创建布局的更多信息，请阅读[支持不同的屏幕尺寸](#)。

下面是活动应用此布局的方法：

```
import android.os.Bundle;

import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.news_articles);

    }

}
```

注意：当您通过在布局 XML 文件中定义碎片来添加碎片到活动布局时，您不能在运行时移除碎片。如果您打算在用户交互中替换碎片，您必须在活动第一次启动时添加碎片到活动时，这会在下一课中展示。

4.3 构建一个灵活的用户界面

当设计你的应用程序要支持大范围的屏幕尺寸时，你可以在不同的布局配置中重用碎片，来根据可用的屏幕空间优化用户体验。

例如，在手持设备上，它可能是适应来在一个单窗格用户界面一次只显示一个碎片。相反地，你可能要设置碎片并排地在具有更宽的屏幕尺寸的平板电脑上显示更多的信息给用户。



图 1 两个碎片，在不同的屏幕尺寸上同一个活动以不同的配置显示。在大屏幕上，两个碎片适合并排，但在手机设备上，适合在一次只有一个碎片，所以碎片必须在当用户浏览时互相代替。

[FragmentManager](#) 类提供了一些方法，使您可以在活动运行时添加、移除和替换碎片，来创建一个动态的体验。

在运行时添加一个碎片到活动

不是在布局文件中定义活动的碎片——像[上一课](#)所示那样使用<fragment>元素——您可以在活动的运行期间添加一个碎片到活动。如果你打算在活动的生命过程中改变碎片，这是必要的。

执行添加或删除碎片之类的事务，您必须使用 [FragmentManager](#) 创建一个 [FragmentTransaction](#)，它提供添加、删除、替换以及执行其它碎片事务的 API。

如果你的活动允许碎片被移除或者替换，你应该在活动 [的 onCreate\(\)](#) 方法添加初始的碎片到活动。

处理碎片——尤其是那些你在运行时添加的碎片——的一个重要原则是，碎片在布局中必须有一个容器[视图](#)，碎片的布局将驻留在该视图中。

下面的布局是在[上节课](#)展示的一次只显示一个碎片的布局的一个替代方案。为了用另一个碎片替换一个碎片，活动的布局包含一个空的[FrameLayout](#) 作为碎片容器。

注意文件名跟在上节课中的布局文件是一样的，但布局目录不会有 `large` 标识符，所以这个布局在设备屏幕小于 `large` 时使用，因为屏幕同一时间不适应两个碎片。

`res/layout/news_articles.xml:`

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

在您的活动内使用支持库 API 调用 [getSupportFragmentManager\(\)](#) 得到一个 [FragmentManager](#)。然后调用 [beginTransaction\(\)](#) 创建一个 [FragmentTransaction](#) 并调用 [add\(\)](#) 添加一个碎片。

您可以使用相同的 [FragmentTransaction](#) 对活动执行多个碎片事务。当你准备好提交变化时，你必须调用 [commit\(\)](#) 方法。

例如，下面添加一个碎片到之前的布局的方法：

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);

        // Check that the activity is using the layout version with
        // the fragment_container FrameLayout
        if (findViewById(R.id.fragment_container) != null) {

            // However, if we're being restored from a previous state,
            // then we don't need to do anything and should return or else
            // we could end up with overlapping fragments.
            if (savedInstanceState != null) {
                return;
            }

            // Create an instance of ExampleFragment
            HeadlinesFragment firstFragment = new HeadlinesFragment();

            // In case this activity was started with special instructions
```

```

from an Intent,

        // pass the Intent's extras to the fragment as arguments
        firstFragment.setArguments(getIntent().getExtras());

        // Add the fragment to the 'fragment_container' FrameLayout
        getSupportFragmentManager().beginTransaction()

            .add(R.id.fragment_container,

firstFragment).commit();

    }

}

}

```

由于碎片在运行时已被添加到 [FrameLayout](#) 容器——而不是在活动的布局使用 `<fragment>` 元素定义它——活动可以移除碎片并用另一个不同的碎片替换它。

用另一个碎片替换碎片

替换一个碎片的过程跟添加一个类似，但需要 [replace\(\)](#) 方法，而不是 [add\(\)](#) 。

请记住，当你执行碎片事务时，如替换或移除一个碎片，它通常应当允许用户导航回去并“撤销”变化。为了让用户可以在碎片事务导航回去，你必须在提交 [FragmentTransaction](#) 之前调用 [addToBackStack\(\)](#)。

注意：当您移除或替换一个碎片，并添加事务到后退堆栈，被移除的碎片是停止的（而不是被销毁）。如果用户导航回去恢复碎片，它会重新启动。如果你不添加事务到后退堆栈，那么碎片在移除或替换时会被销毁。

用另一个碎片替换碎片的示例：

```

// Create fragment and give it an argument specifying the article it should
show

ArticleFragment newFragment = new ArticleFragment();

Bundle args = new Bundle();

```

```
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentManager transaction =
getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack so the user can navigate back
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

[addToBackStack\(\)](#) 方法需要一个可选的字符串参数来指定事务一个唯一的名称。这个名字是没有必要的，除非你打算使用 [FragmentManager.BackStackEntry](#) API 来执行高级的碎片操作。

4.4 与其它碎片通信

为了重用碎片 UI 组件，你应该把它们每个建立一个完全独立的、模块化的组件，让它定义自己的布局和行为。一旦你定义了这些可复用的碎片，您可以将它们与活动关联起来，并将它们连接到应用程序逻辑来实现整体复合 UI。

通常情况下，你会想让一个碎片与另一个通信，例如根据用户事件改动内容。所有碎片到碎片的通信是通过相关联的活动进行的。两个碎片永远不应该直接通信。

定义一个接口

为了让碎片与其活动通信，你可以在 `Fragment` 类中定义一个接口，并在活动内实现它。碎片在其 `onAttach()` 生命周期方法捕获接口实现，然后可以调用接口中的方法来与活动通信。

下面是一个活动到碎片通信的例子：


```

public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Container Activity must implement this interface
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // This makes sure that the container activity has implemented
        // the callback interface. If not, it throws an exception
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                + " must implement OnHeadlineSelectedListener");
        }
    }

    ...
}

```

现在碎片可以通过调用 `onArticleSelected()` 方法（或其它接口中的方法）给活动分发消息，使用 `OnHeadlineSelectedListener` 接口的 `mCallback` 实例。

例如，当用户点击一个列表项时碎片的下述方法会被调用。碎片使用回调接口来分发事件到父活动。

```
@Override

public void onListItemClick(ListView l, View v, int position, long
id) {

    // Send the event to the host activity

    mCallback.onArticleSelected(position);

}
```

实现的接口

为了从碎片接收事件回调，装载碎片的活动必须实现碎片类中定义的接口。

例如，以下活动实现了上面例子的接口。

```
public static class MainActivity extends Activity

    implements HeadlinesFragment.OnHeadlineSelectedListener{

    ...

    public void onArticleSelected(int position) {

        // The user selected the headline of an article from the
HeadlinesFragment

        // Do something here to display that article

    }

}
```

分发信息到碎片

载体活动可以通过使用 [findFragmentById\(\)](#) 捕捉 [Fragment](#) 实例，来分发消息到一个碎片，然后直接调用该碎片的公共方法。

例如，想象一下上面展示的活动可能包含的另一个碎片，这个碎片用来显示由上面的回调方法返回的数据指定的项目。在这种情况下，活动可以传递在回调方法中的收到的信息到另一个将显示该项目的碎片：

```
public static class MainActivity extends Activity
    implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the
HeadlinesFragment

        // Do something here to display that article

        ArticleFragment articleFrag = (ArticleFragment)
            getSupportFragmentManager().findFragmentById(R.id.arti
cle_fragment);

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layout...

            // Call a method in the ArticleFragment to update its content
            articleFrag.updateArticleView(position);
        } else {
            // Otherwise, we're in the one-pane layout and must swap
frags...

            // Create fragment and give it an argument for the selected
article

            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);
```

```
        FragmentTransaction transaction =  
getSupportFragmentManager().beginTransaction();  
  
        // Replace whatever is in the fragment_container view with this  
fragment,  
        // and add the transaction to the back stack so the user can  
navigate back  
        transaction.replace(R.id.fragment_container, newFragment);  
        transaction.addToBackStack(null);  
  
        // Commit the transaction  
        transaction.commit();  
    }  
}  
}
```

5 保存数据

大多数 Android 应用程序需要保存数据，即使只为了不丢失用户的进度，在 `onPause()` 中保存应用程序的状态信息。大部分不平凡的应用程序也需要保存用户设置，而一些应用程序必须在文件和数据库中管理大量信息。本课程介绍你在 Android 的主要数据存储选项，包括：

- 在一个共享的首选项文件中保存键-值对简单数据类型
- 在 Android 的文件系统中保存任意文件
- 使用 SQLite 进行数据库管理

课程

保存键-值集

了解使用共享首选项文件存储少量的信息键-值对的。

保存文件

了解保存基本文件，如要存储的数据一般都是按顺序阅读的长序列。

将数据保存在 SQL 数据库

了解如何使用 SQLite 数据库读取和写入结构化的数据。

5.1 保存键-值集

如果你有一个相对较小的键-值的集合想要保存，你应该使用 `SharedPreferences` API。`SharedPreferences` 对象指向一个包含键-值对文件，并提供简单的方法来读取和写入它们。每个 `SharedPreferences` 文件由框架管理，可以设为私有或共享。

本课程向您展示如何使用 `SharedPreferences` API 来存储和检索简单的值。

注：`SharedPreferences` API 只能读取和写入键-值对，你不应该将它与 `Preference` 的 API 混淆，`Preference` 帮助你为你的应用程序设置建立一个用户界面（虽然它们保存应用程序设置是通过 `SharedPreferences` 来实现）。对于使用的 `Preference` API 的信息，请参阅“[设置](#)”指南。

获取 SharedPreferences 的句柄

您可以通过调用以下两种方法之一来创建一个新的或访问一个现有的共享偏好文件：

- `getSharedPreferences()` -如果你需要使用多个不同文件名的共享偏好文件,可以使用这个方法,用第一个参数指定文件名。您可以在您的应用程序的任何 [上下文 \(Context\)](#) 中调用这个方法。
- `getPreferences()` -如果你只需要为活动 (Activity) 使用一个共享偏好文件,可以在活动使用此方法。因为它检索的默认共享偏好文件从属于活动,所以你不需要提供一个文件名。

例如,下面的代码在一个碎片 ([Fragment](#)) 里执行。它访问由资源字符串 `R.string.preference_file_key` 所标识的共享文件,并使用私有的模式,所以只有您的应用程序才能访问该文件。

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
    getString(R.string.preference_file_key),
    Context.MODE_PRIVATE);
```

(在新版的 API 中,已经不需要通过 `getActivity()` 方法得到 `context` 对象,可直接调用 `getSharedPreferences()` 方法,译者注)

为共享偏好文件命名时,你应该使用一个应用程序能唯一识别的名称,如

```
"com.example.myapp.PREFERENCE_FILE_KEY"
```

另外,如果你的活动只需要一个共享的偏好设置文件,您可以使用 `getPreferences()` 方法:

```
SharedPreferences sharedPref =
    getActivity().getPreferences(Context.MODE_PRIVATE);
```

注意：如果您以 `MODE_WORLD_READABLE` 或 `MODE_WORLD_WRITEABLE` 方式创建了一个共享的偏好文件，任何其它知道该文件标识符的应用程序都可以访问您的数据。

写入共享首选项

要写入一个共享的喜好文件，对您的 `SharedPreferences` 调用 `edit()` 方法创建一个 `SharedPreferences.Editor`。

通过 `putInt()` 和 `putString()` 等方法传递你想要写的键和值。然后调用 `commit()` 方法来保存更改。例如：

```
SharedPreferences sharedPref =
    getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

读取共享首选项

从一个共享的偏好文件检索值，可以调用 `getInt()` 的 `getString()` 方法，提供你想要的值的键，和一个当键不存在时默认返回值（可选的）。例如：

```
SharedPreferences sharedPref =
    getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue =
    getResources().getInteger(R.string.saved_high_score_default);
long highScore =
    sharedPref.getInt(getString(R.string.saved_high_score),
        defaultValue);
```

5.2 保存文件

Android 使用的文件系统，这是在其它平台上的基于磁盘的文件系统类似。这节课介绍了如何使用 Android 文件系统的 **File API** 来读取和写入文件。

一个 **File** 对象适合以从头到尾非跳跃的方式读取或写入大量的数据。例如，它适合图像文件或任何在网络上交换的东西。

这节课在您的应用程序显示了如何执行基本的文件相关的任务。这节课假定您是熟悉 **Linux** 文件系统的基础知识和 **java.io** 中的标准文件输入/输出 **API**。

选择内部或外部存储

所有的 **Android** 设备有两个文件存储区：“内部”和“外部”存储。这些名称来自 **Android** 的早期，那时大多数设备提供内置的非易失性存储器（内部存储器），加上一个可移动存储介质如微型 **SD** 卡（外部存储）。某些设备把永久存储空间划分为“内部”和“外部”分区，所以，即使没有可移动存储介质，都会有两个存储空间，并且不管外部存储是否可移动，**API** 的行为都是相同的。下面的列表总结了每个存储空间的事实。

内部存储：

- 它总是可用的。
- 默认情况下，这里保存的文件只有您的应用程序能访问。
- 当用户卸载你的应用程序，系统将从内部存储删除你的应用程序的所有文件。

当你要确信无论是用户还是其它的应用程序都不可以访问您的文件的时候，内部存储是最好的选择。

外部存储：

- 它并不总是可用的，因为用户可以用 **USB** 存储设备作为外部存储，并在某些情况下，把它从装置中取出。
- 这是全局可读，所以保存在这里的文件可能会在你的控制之外被读取。

- 当用户卸载您的应用程序时，只有当你之前是用 `getExternalFilesDir()` 方法将你的应用程序的文件保存在目录，系统才会从这里删除它们。

不需要访问限制的文件和要与其它应用程序共享的文件，或者允许用户用电脑访问时，外部存储是最好的地方。

提示： 虽然默认情况下应用程序安装到内部存储的，你可以在你的 `manifest` 文件指定 `android:installLocation` 属性，让您的应用程序可以安装在外部存储。当 APK 大小是非常大的并且用户有一个比内部存储大得多的外部存储时，他们偏向于这个选项。欲了解更多信息，请参阅[应用程序安装位置](#)。

获取外部存储的权限

要写入到外部存储，您必须在您的 `manifest` 文件中请求 `WRITE_EXTERNAL_STORAGE` 权限：

```
<manifest ...>
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

注意： 目前，所有的应用程序都无需特殊权限就能够读取外部存储。然而，这将在未来的版本中改变。如果你的应用程序需要读取外部存储（但不需要写入），那么您将需要声明 `READ_EXTERNAL_STORAGE` 权限。要确保你的应用继续按预期方式工作，你应该在更改生效前现在就声明此权限。

```
<manifest ...>
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE" />
    ...
</manifest>
```

但是，如果你的应用程序使用 `WRITE_EXTERNAL_STORAGE` 权限，那么它同时隐含有权限读取外部存储。

你不需要任何权限在内部存储上保存文件。你的应用程序总是有权限在其内部存储目录读取和写入文件。

在内部存储上保存文件

保存一个文件到内部存储时，通过以下两种方法之一，你可以获取相应的目录作为 `File` 对象：

`getFilesDir()`

返回一个表示您的应用程序的内部目录的 `File` 对象。

`getCacheDir()`

返回一个你的应用程序的临时缓存文件的内部目录 `File` 对象。确保每个文件一旦不再需要时删除它们，并在任何给定的时间内对使用的内存施加一个合理的大小限制，如 1MB。如果系统开始在低存储情况下运行，它可能会在没有警告的情况下删除您的缓存文件。

要在这些目录中创建一个新的文件，你可以使用 `File()` 构造函数，传递文件由上述指定你的内部存储目录的方法之一提供的 `File` 对象。例如：

```
File file = new File(context.getFilesDir(), filename);
```

或者，您可以调用 `openFileOutput()` 得到一个 `FileOutputStream` 写入到内部目录的文件中。例如，这里是写一些文字到一个文件的方法：

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {
```

```

        outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
        outputStream.write(string.getBytes());
        outputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

或者，如果你需要缓存一些文件，你应该使用 `createTempFile()` 代替。例如，下面的方法从 URL 中提取文件名，并用该名字在您的应用程序的内部缓存目录中创建一个文件：

```

public File getTempFile(Context context, String url) {
    File file;
    try {
        String fileName = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(fileName, null,
context.getCacheDir());
        catch (IOException e) {
            // Error while creating file
        }
        return file;
    }
}

```

注： 您的应用程序的内部存储目录由您的应用程序的包名指定在 Android 文件系统中一个特殊的位置。从技术上说，另一个应用程序可以读取你的内部文件，如果你设置了文件模式是可读的。然而，其它应用程序还需要知道你的应用程序包名和文件名。其它应用程序不能浏览您的内部目录也没有读或写访问权限，除非你明确地设置文件可以读写。所以只要你对内部存储中的文件使用 `MODE_PRIVATE`，它们对其它的应用程序就从来不可用。

在外部存储上保存文件

由于外部存储可能不可用,例如,当用户安装存储到 PC 上或移除了提供外部存储的 SD 卡,你一定要在访问它前确认它可用。您可以调用 `getExternalStorageState()` 查询外部存储状态。如果返回的状态等于 `MEDIA_MOUNTED`,那么你可以阅读和写入文件。例如,可以用下列方法来确定存储可用:

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

虽然外部存储是对用户和其它应用程序是可修改的,有两类文件你可能会保存在这里:

公共文件

自由提供给其它应用程序和用户的文件。当用户卸载你的应用程序,这些文件应该对用户仍然可用。

例如,由你的应用程序拍摄的照片或其它下载的文件。

私有文件

原本属于您的应用程序并应该在用户卸载您的应用程序时删除的文件。由于它们是在外部存

储中，尽管这些文件在技术上是可由用户和其它应用程序来访问，但它们实际上是在您的应用程序外不提供用户价值的文件。当用户卸载你的应用程序时，系统会删除你的应用程序的外部私有目录中的所有文件。

例如，您的应用程序下载的额外资源或临时的媒体文件。

如果你想在外部存储保存公共文件，使用 `getExternalStoragePublicDirectory()` 方法来获得一个 `File` 对象表示外部存储中的相应目录。该方法需要一个参数指定你要保存的文件类型，使它们可以与其它公共文件在逻辑上组织好，比如 `DIRECTORY_MUSIC` 或 `DIRECTORY_PICTURES`。例如：

```
public File getAlbumStorageDir(String albumName) {  
    // Get the directory for the user's public pictures directory.  
    File file = new File(Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES), albumName);  
    if (!file.mkdirs()) {  
        Log.e(LOG_TAG, "Directory not created");  
    }  
    return file;  
}
```

如果你想保存对您的应用程序是私有的文件，你可以通过调用 `getExternalFilesDir()`，并传递给它一个名字，表示你想要的目录类型，来获得相应的目录。这种方式创建每个目录都会添加到一个父目录，该父目录封装了你的应用程序的所有外部存储文件，当用户卸载您的应用程序时系统会删除这件文件。

例如，这里有一个方法，你可以使用它来创建一个个人相册目录：

```
public File getAlbumStorageDir(Context context, String albumName) {  
    // Get the directory for the app's private pictures directory.  
    File file = new File(context.getExternalFilesDir(  
        Environment.DIRECTORY_PICTURES), albumName);  
    if (!file.mkdirs()) {
```

```
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

如果没有预先定义的子目录名称匹配你的文件，你可以调用 `getExternalFilesDir()`，并传递 `null`。它会返回外部存储上您的应用程序的私有目录的根目录。

请记住，`getExternalFilesDir()` 在这样的目录下创建一个目录：当用户卸载您的应用程序时该目录会被删除。如果你保存的文件，要在用户卸载后您的应用程序还能保留，例如，你的应用程序是一个摄像头而用户将要保留照片，你应该使用 `getExternalStoragePublicDirectory()` 代替。

无论你对要共享的文件使用 `getExternalStoragePublicDirectory()` 还是对您的应用程序是私有的文件使用 `getExternalFilesDir()`，使用像 `DIRECTORY_PICTURES` 这样的 API 常量作为目录名，对你都是相关重要的。这些目录的名称，确保这些文件由系统妥善处理。例如，保存在 `DIRECTORY_RINGTONES` 的文件被系统媒体扫描仪归类为铃声，而不是音乐。

查询空闲空间

如果你提前知道你要保存多少数据，调用 `getFreeSpace()` 或 `getTotalSpace()` 你可以知道是否有足够的空间可用，而不是引起一个 `IOException`。这些方法分别提供了当前可用空间和存储容量总空间。此信息也有助于避免填充的存储量超过某个临界值。

然而，系统并不能保证你可以完全写入 `getFreeSpace()` 表示的字节数。如果返回的数字比你想要保存的数据的大小超过几 MB，或者如果文件系统还不到 90%，那么它进行保存可能是安全的。否则，你可能不应该写入到存储中。

注：在您保存你的文件之前，你并不需要检查可用空间。相反，你可以马上尝试写入文件，然后捕获一个 `IOException`，如果它发生了。您可能需要这样做，如果你不知道你需要的空间究竟有多大。例如，如果你把文件转换为 PNG 图片或 JPEG 来保存之前，改变它的编码，那么你不会事先知道文件的大小。

删除文件

你应该总是删除您不再需要的文件。删除一个文件，最简单的方法是对打开的文件的引用本身调用 `delete()` 方法。

```
myFile.delete();
```

如果该文件被保存在内部存储，你还可以使用 `Context` 调用 `DeleteFile()` 定位并删除一个文件：

```
myContext.deleteFile(fileName);
```

注：当用户卸载你的应用程序，`Android` 系统将删除以下内容：

- 你保存在内部存储上的所有文件
- 你使用 `getExternalFilesDir()` 保存在外部存储上的所有文件。

然而，你应该手动删除所有定期用 `getCacheDir()` 创建的缓存文件，还定期删除不再需要的其它文件。

5.3 将数据保存在 SQL 数据库

对于重复或结构化的数据，如联系人信息，将它们保存到数据库是理想选择。这节课假定您熟悉一般的 `SQL` 数据库，并帮助您开始在 `Android` 上使用 `SQLite` 数据库。在 `Android` 上，你需要使用数据库的 API 都在 [android.database.sqlite](#) 包提供。

定义一个模式（**Schema**）和合同（**Contract**）

SQL 数据库的主要原则之一是模式：数据库是如何组织的正式声明。该模式体现在你用来创建数据库的 SQL 语句中。您可能会发现它有利于创建一个同伴（companion）类，作为合同类为人所知，后者用系统和自文档(self-documenting)的方式明确指定您的模式的布局。

合同类是定义 URI、表和列名等常量的一个容器。合同类，允许您在同一个包中的其它所有类里使用相同的常量。这可以让你在一个地方改变某个列名，它就传遍你的代码。

组织合同类的一个好方法就是把你的整个数据库的全局定义放到类的根层次。然后为每个枚举列的表创建一个内部类。

注：通过实现 [BaseColumns](#) 接口，内部类可以继承一个称为 `_ID` 的主键字段，某些 Android 类，如光标适配器，会期望你的内部类有这个字段。它不是必需的，但这个可以帮你的数据库与 Android 框架和谐地工作。

例如，这个碎片定义为一个单一的表的表名和列名：

```
public static abstract class FeedEntry implements BaseColumns {
    public static final String TABLE_NAME = "entry";
    public static final String COLUMN_NAME_ENTRY_ID = "entryid";
    public static final String COLUMN_NAME_TITLE = "title";
    public static final String COLUMN_NAME_SUBTITLE = "subtitle";
    ...
}
```

为了防止有人意外地实例化合同类，给它一个空的构造方法。

```
// Prevents the FeedReaderContract class from being instantiated.
private FeedReaderContract() {}
```

使用 **SQL Helper** 创建一个数据库，

一旦你定义完你的数据库的样子，你应该实现创建和维护数据库和表的方法。下面是一些典型的创建和删除一个表的语句：

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedReaderContract.FeedEntry.TABLE_NAME + " (" +
    FeedReaderContract.FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedReaderContract.FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE +
    COMMA_SEP +
    FeedReaderContract.FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE +
    COMMA_SEP +
    ... // Any other options for the CREATE command
    " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + TABLE_NAME_ENTRIES;
```

就像您保存在设备[内部存储](#)的文件那样，Android 的存储您的数据库在与应用程序关联的私有磁盘空间。您的数据是安全的，因为默认情况下，这个区域对其它应用程序是不可访问的。

一组有用的 API，可在 `SQLiteOpenHelper` 类中找到。当你使用这个类来获取到你的数据库的引用时，系统执行潜在的长时间运行的创建和更新数据库的操作，只会在需要的时候，而不是在应用程序启动时。你所要做的全部就是调用 [getWritableDatabase\(\)](#) 或 [getReadableDatabase\(\)](#)。

注：因为他们可以长时间运行，确保您是在后台线程调用 [getWritableDatabase\(\)](#) 或 [getReadableDatabase\(\)](#)，比如用 [AsyncTask](#) 或 [IntentService](#)。

要使用 [SQLiteOpenHelper](#)，创建一个子类重写 [onCreate\(\)](#)，[onUpgrade\(\)](#) 和 [onOpen\(\)](#) 回调方法。您可能还想要实现 [onDowngrade\(\)](#) 方法，但它不是必需的。

例如，这里是 SQLiteOpenHelper 使用上面展示的命令的一个实现：

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {  
    // If you change the database schema, you must increment the database  
    version.  
  
    public static final int DATABASE_VERSION = 1;  
    public static final String DATABASE_NAME = "FeedReader.db";  
  
    public FeedReaderDbHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(SQL_CREATE_ENTRIES);  
    }  
  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {  
        // This database is only a cache for online data, so its upgrade  
policy is  
        // to simply to discard the data and start over  
        db.execSQL(SQL_DELETE_ENTRIES);  
        onCreate(db);  
    }  
  
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {  
        onUpgrade(db, oldVersion, newVersion);  
    }  
}
```

要访问数据库，实例化 SQLiteOpenHelper 的子类：

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
```

将信息存入数据库

通过传递一个 [ContentValues](#) 对象到 [insert\(\)](#) 方法，可以把数据插入到数据库中：

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedReaderContract.FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedReaderContract.FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedReaderContract.FeedEntry.COLUMN_NAME_CONTENT,
content);

// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
    FeedReaderContract.FeedEntry.TABLE_NAME,
    FeedReaderContract.FeedEntry.COLUMN_NAME_NULLABLE,
    values);
```

[insert\(\)](#) 方法的第一个参数很简单就是表名。第二个参数提供列名，如果 [ContentValues](#) 为空 (empty) 框架会为该列插入 NULL 值 (如果你把列名设为 "null", 那么当它没有值时框架将不会插入一行)。

从数据库中读取信息

使用 [query\(\)](#) 方法从数据库中读取，传递你的选择子句和所需的列。该方法结合了 [insert\(\)](#) 和 [update\(\)](#) 的元素，除了那些列是定义你想获取的数据，而不是要插入的数据。查询结果在一个 [Cursor](#) 对象中返回给你。

```

SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedReaderContract.FeedEntry._ID,
    FeedReaderContract.FeedEntry.COLUMN_NAME_TITLE,
    FeedReaderContract.FeedEntry.COLUMN_NAME_UPDATED,
    ...
};

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedReaderContract.FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
    FeedReaderContract.FeedEntry.TABLE_NAME, // The table to query
    projection,                               // The columns to return
    selection,                                // The columns for the WHERE
    clause
    selectionArgs,                            // The values for the WHERE
    clause
    null,                                     // don't group the rows
    null,                                     // don't filter by row groups
    sortOrder                                 // The sort order
);

```

为了看光标中的一行，使用其中一个 [Cursor](#) 的移动方法，在开始读取值之前你必须始终调用这些方法。一般来说，你应该从调用 [moveToFirst\(\)](#) 开始，让“读取位置”定位到结果中的第一项。对于每一行，你可以通过调用一个 [Cursor](#) 的 [get](#) 方法来读取列的值，如 [getString\(\)](#) 或 [getLong\(\)](#)。对于每一个 [get](#) 方法，你必须传递你想要的列的索引位置，

你可以通过调用 [getColumnIndex\(\)](#) 或 [getColumnIndexOrThrow\(\)](#) 得到列的索引。例如：

```
cursor.moveToFirst();

long itemId = cursor.getLong(
    cursor.getColumnIndexOrThrow(FeedReaderContract.FeedEntry._ID)
);
```

从数据库中删除信息

要删除表中的行，你需要提供的选择条件来确定哪些行。数据库 API 提供了一种机制来创建可防止 SQL 注入的选择条件。该机制将选择规范分解成一个选择子句和选择参数。[子句定义列，也允许你组合列来试验。参数是试验绑定到子句的值。](#)（原文：The clause defines the columns to look at, and also allows you to combine column tests. The arguments are values to test against that are bound into the clause.）因为结果与一个常规的 SQL 语句处理不一样，它对 SQL 注入是免疫的。

```
// Define 'where' part of query.

String selection = FeedReaderContract.FeedEntry.COLUMN_NAME_ENTRY_ID +
    " LIKE ?";

// Specify arguments in placeholder order.

String[] selectionArgs = { String.valueOf(rowId) };

// Issue SQL statement.

db.delete(table_name, selection, selectionArgs);
```

更新数据库

当你需要修改你的数据库数值的一个子集时，使用 [update\(\)](#) 方法。[更新表内容结合使用 \[insert\\(\\)\]\(#\) 方法的语法和带 **where** 的 \[delete\\(\\)\]\(#\) 方法的语法。](#)（原文：Updating the table

combines the content values syntax of `insert()` with the `where` syntax of `delete()`

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// New value for one column
ContentValues values = new ContentValues();
values.put(FeedReaderContract.FeedEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the ID
String selection = FeedReaderContract.FeedEntry.COLUMN_NAME_ENTRY_ID +
    " LIKE ?";
String[] selectionArgs = { String.valueOf(rowId) };

int count = db.update(
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

6 与其它应用程序交互

一个 Android 应用程序通常有多个[活动](#)。每一项活动都将显示一个用户界面，允许用户执行某种特定任务（如查看地图或者照片）。为了把用户从一个活动带到另一个，你的应用必须使用 [Intent](#) 定义您的应用程序做某些事情的“意图”。当你用某个方法，如 [startActivity\(\)](#)，给系统传递一个 [意图](#)，系统就使用[意图](#)来识别并启动相应的应用程序组件。使用意图，甚至可以让你的应用程序启动在另一个的应用程序中包含的活动。

一个[意图](#)可以是显式的，用以启动一个特定的组件（一个特定的[活动](#)实例），或者是隐式的，用以启动可以处理预期动作的任何组件（如“拍摄照片”）。

这节课，告诉你如何使用[意向](#)与其它的应用程序执行一些基本的交互，如启动另一个应用程序，从该应用程序接收结果，以及使您的应用程序能够响应来自其它应用程序的意图。

课程

[把用户带到另一个应用程序](#)

展示如何创建隐式的意图，启动其它应用程序，来执行一个动作。

[从活动获取结果](#)

展示如何启动另一个活动，并从活动中接收结果。

[允许其它应用程序启动你的活动](#)

展示了如何通过定义意图过滤器声明您的应用程序接受的隐式意图，使你的应用程序中的活动对其它应用程序开放使用。

6.1 把用户带到另一个应用程序

Android 的最重要的特点之一是应用程序把用户带到另一个应用程序的能力，这种能力基于它想执行的“操作”。例如，如果您的应用程序有一个企业的地址，而你想在地图上显示该地址，你不必在您的应用程序中建立一个活动来显示地图。相反，你可以使用[意图](#)创建一个请求来查看该地址。Android 系统接着就会启动一个能够在地图上显示地址的应用程序。

正如在第一节课“[建立你的第一个应用程序](#)”中所说的，你必须使用意图，在自己的应用程序的活动之间进行导航。你通常是使用一个显式意图，它定义了你想要启动的组件的确切的类名。然而，当你想有一个单独的应用程序执行一个操作，如“查看地图”，你必须使用一个隐式意图。

这节课告诉你如何为一个特定的操作创建一个隐式意图，以及如何用它来启动另一个应用程序中的一个活动来执行该操作。

建立一个隐式意图

隐式意图不声明要启动组件的类名，而是声明要执行的操作。该操动指定你想做的事情，如查看，编辑，发送，或获取某些东西。意图往往还包括与操作相关的数据，如你要查看的地址，或您要发送的邮件信息。根据您要创建的意图，数据可能是一个 [Uri](#)，其它几种数据类型之一，或意图可能并不需要任何的数据。

如果您的数据是一个 [Uri](#)，有一个简单的 `Intent()` 构造函数，你可以用它来定义操作和数据。

例如，下面是如何创建一个意图使用 [Uri](#) 数据指定电话号码来发起电话呼叫的方法：

```
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

当您的应用程序通过调用 `startActivity()` 唤起这个意图，电话应用程序用给定的电话号码发起呼叫。

这里有一对另外的意图，以及它们的行动和 [Uri](#) 数据对：

- 查看地图：

```
// Map point based on address
Uri location =
Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+
California");
```



```
// Or map point based on latitude/longitude

// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); //
// z param is zoom level

Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

- 查看一个网页:

```
Uri webpage = Uri.parse("http://www.android.com");

Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

其它类型的隐式意图需要“额外的(extra)”数据，这些数据提供了不同的数据类型，如字符串。您可以使用各种 `putExtra()` 方法添加一个或多个 **extra** 数据。

默认情况下，系统将根据包含的 **Uri** 数据决定意图所需要的合适的 **MIME** 类型。如果你的意图不包含 **Uri**，你通常应该使用 `setType()` 指定意图的相关数据的类型。设置 **MIME** 类型，还能指定哪些类型的活动会收到该意图。

下面是一些更多的添加 **extra** 数据来指定所要求的操作的意图，：

- 发送带附件的电子邮件:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);

// The intent does not have a URI, so declare the "text/plain" MIME
type

emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);

emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]
{"jon@example.com"}); // recipients

emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
emailIntent.putExtra(Intent.EXTRA_STREAM,
Uri.parse("content://path/to/email/attachment"));

// You can also attach multiple items by passing an ArrayList of
Uris
```

- 创建一个日历事件:

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT,
Events.CONTENT_URI);

Calendar beginTime = Calendar.getInstance().set(2012, 0, 19, 7,
30);

Calendar endTime = Calendar.getInstance().set(2012, 0, 19, 10, 30);

calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME
, beginTime.getTimeInMillis());

calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
endTime.getTimeInMillis());

calendarIntent.putExtra(Events.TITLE, "Ninja class");

calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

注：此意图日历事件仅在 API 级别 14 和更高版本中支持。

注：你尽可能具体地定义你的意图是很重要的。例如，如果你想使用 `ACTION_VIEW` 意图显示图像，你应该指定 MIME 类型为 `image/*`。这可以防止那些能“查看”其它类型的数据的应用程序（例如地图应用程序），被意图触发。

验证有一个应用程序能接收意图

尽管 Android 平台保证某些意图将由某个内置的应用程序解决（如电话、电子邮件或日历应用程序），你应该总是在调用意图之前包括验证的步骤。

注意：如果你调用一个意图，而设备上没有任何的应用程序可以处理这个意图，你的应用程序将会崩溃。

要想验证是否有一个有效的活动可以响应的意图，可调用 `queryIntentActivities()` 得到一个能够处理你的意图的活动列表。如果返回的列表不为空，你可以放心地使用的意图。例如：

```
PackageManager packageManager = getPackageManager();  
List<ResolveInfo> activities =  
packageManager.queryIntentActivities(intent, 0);  
boolean isIntentSafe = activities.size() > 0;
```

如果 `isIntentSafe` 为真，那么至少有一个应用程序将响应该意图。如果它为假，那么就不会有任何的应用程序来处理这个意图。

注：您的活动第一次启动时你应该执行此检查，以防你需要在用户尝试使用它之前禁用该意图的功能。如果你知道一个特定的应用程序可以处理这个意图，你也可以提供一个链接供用户下载此应用程序（请参阅[如何链接您的产品到 Google Play](#)）。

用意图启动一个活动

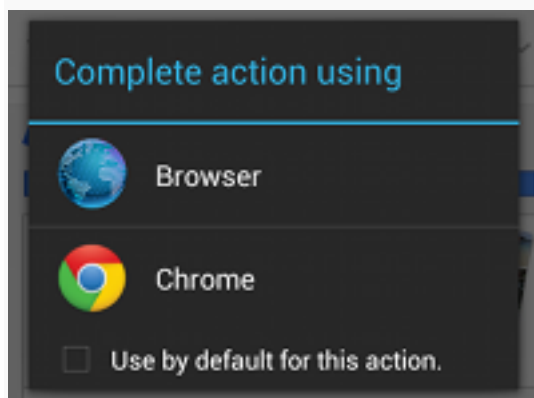


图 1。一个以上的应用程序可以处理意图时出现选择对话框时的示例。

一旦你创建了你的意图，并设置了 `extra` 信息，调用 `startActivity()` 将它发送到系统。如果系统识别到多个活动可以处理这个意图，它会显示一个对话框供用户选择使用哪个应用程序，如图 1 所示。如果只有一个活动能处理意图时，系统立即启动它。

```
startActivity(intent);
```

下面是一个完整的例子，显示了如何创建一个意图来查看地图，确认存在一个应用程序来处理这个意图，然后启动它：

```
// Build the intent
Uri location =
Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities =
packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

显示一个应用程序选择器

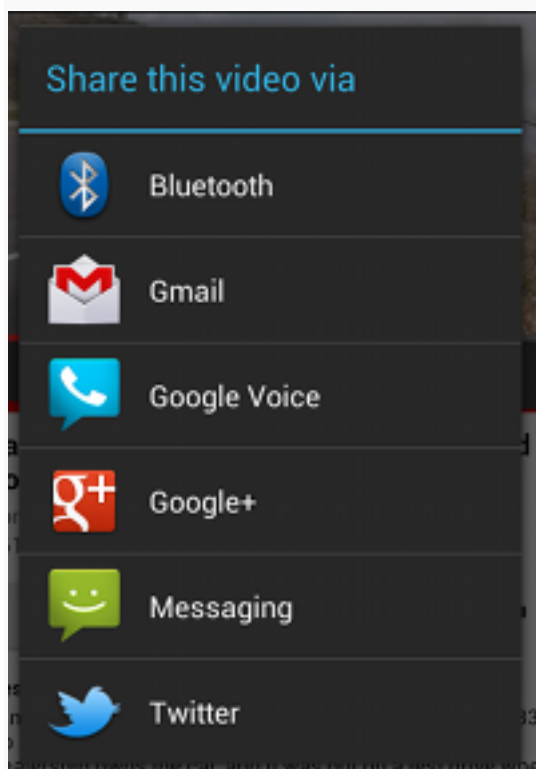


图 2 当你使用 `createChooser()` 来确保总是向用户显示回应你的意图的应用程序列表时，出现选择对话框的示例。

请注意，当你传递你的意图到 `startActivity()` 来启动一个活动时，有一个以上的应用程序响应你的意图，用户可以选择默认使用的应用程序（通过选择对话框底部的一个复选框；参见图 1）。当用户通常希望每次使用相同的应用程序来执行操作时，如打开一个网页时（用户可能只使用一个 **Web** 浏览器），或拍摄照片（用户可能更喜欢一个照相机），这是一种不错的做法。然而，如果要执行的动作可以被多个应用程序处理，而用户可能更喜欢每一次用不同的应用程序，如“共享”操作，用户可能有几个应用程序能够共享项目，你应该明确地显示一个选择对话框，强制用户每次选择使用哪款应用执行这个操作（用户不能为操作选择一个默认的应用程序）。

要显示选择器，使用 `createChooser()` 创建一个意图，并把它传递给 `startActivity()`。例如：

```
Intent intent = new Intent(Intent.ACTION_SEND);  
...
```

```
// Always use string resources for UI text. This says something like "Share
this photo with"

String title = getResources().getText(R.string.chooser_title);

// Create and start the chooser

Intent chooser = Intent.createChooser(intent, title);

startActivity(chooser);
```

这将显示一个对话框，包含响应传递给 `createChooser()` 方法的意图的应用程序列表，并使用提供的文本作为对话框的标题。

6.2 从活动获取结果

启动另一个活动不必是单向的。您也可以启动另一个活动，并接收一个结果回来。为了接收一个结果，调用 `startActivityForResult()`（而不是 `startActivity()`）。

例如，您的应用程序可以启动照相机应用程序，并接收拍摄的照片作为结果。或者，您可能启动通讯录应用程序，让用户选择一个联系人，您会收到联系人详情作为结果。

当然，响应的活动必须设计为返回一个结果。当它这样做时，它会发送结果作为另一个 `Intent` 对象。你的活动在 `onActivityResult()` 回调方法中接收它。

注：当你调用 `startActivityForResult()` 时，您可以使用显式或隐式意图。当启动一个自己的活动来接收一个结果时，你应该使用一个显式意图，以确保您收到预期的结果。

启动活动

在你为了结果启动活动时使用的意图对象，并没有什么特别，但你确实需要传递另外一个整数参数到 `startActivityForResult()` 方法。

整数参数是一个“请求码”，它确定您的要求。当您接收到结果意图，回调方法提供了同样的请求码，使您的应用程序可以正确识别结果，并决定如何处理它。

例如，以下是启动允许用户选择一个联系人的活动的方法，：

```

static final int PICK_CONTACT_REQUEST = 1; // The request code

...

private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
Uri.parse("content://contacts"));

    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user only
contacts w/ phone numbers

    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}

```

接收结果

当用户完成后续活动和返回结果，系统调用活动的 `onActivityResult()` 方法。这个方法有三个参数：

- 你传递给 `startActivityForResult()` 的请求代码。
- 第二个活动指定的结果代码。如果操作成功这可能是 `RESULT_OK`，或者如果用户退出或操作由于某种原因失败了，就是 `RESULT_CANCELED`。
- 一个附带结果数据的意图。

例如，以下是你处理“选择一个联系人”意图的结果的方法：

```

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // The user picked a contact.
            // The Intent's data Uri identifies which contact was selected.

```

```

        // Do something with the contact here (bigger example below)
    }
}
}

```

在这个例子中，Android 通讯录或联系人应用程序返回的结果意图，提供了一个内容 Uri 标识用户选择的联系人。

为了成功地处理结果，你必须明白结果意图将会是什么格式。当返回结果的活动是自己的活动之一时，这样做是很容易的。Android 平台包含的应用程序提供自己的 API，依靠这些 API 你可以得到特定的结果数据。例如，联系人应用程序（在一些比较旧的版本是通讯录应用程序）总是返回内容 URI 的结果，该 URI 标识选定的联系人，而相机应用程序在 “data”extra 中返回一个 Bitmap（见捕捉照片一课）。

福利：读取联系人数据

上面显示如何从联系人应用程序得到结果的代码 并没有深入到如何实际从结果读取数据的细节，因为它需要关于内容提供者的更先进的讨论。但是，如果你很好奇，下面就有更多的代码，显示了如何查询结果数据来获得选定的联系人的电话号码：

```

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent
data) {

    // Check which request it is that we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {

        // Make sure the request was successful
        if (resultCode == RESULT_OK) {

            // Get the URI that points to the selected contact
            Uri contactUri = data.getData();

            // We only need the NUMBER column, because there will be only
            one row in the result

            String[] projection = {Phone.NUMBER};

```



```

        // Perform the query on the contact to get the NUMBER column
        // We don't need a selection or sort order (there's only one
result for the given URI)

        // CAUTION: The query() method should be called from a separate
thread to avoid blocking

        // your app's UI thread. (For simplicity of the sample, this
code doesn't do that.)

        // Consider using CursorLoader to perform the query.
Cursor cursor = getContentResolver()
        .query(contactUri, projection, null, null, null);
cursor.moveToFirst();

// Retrieve the phone number from the NUMBER column
int column = cursor.getColumnIndex(Phone.NUMBER);
String number = cursor.getString(column);

// Do something with the phone number...
    }
}
}

```

注：在 Android 2.3（API 级别 9）之前，在[联系供应者](#)（如上面所示）上查询，需要您的应用程序声明 `READ_CONTACTS` 权限（见[安全性和权限](#)）。然而，从 Android 2.3 开始，通讯录/联系人应用程序授予您的应用程序一个临时的从联系供应商读取的权限，当它返回你一个结果。临时权限仅适用于特定联系人的请求，所以你不能查询意图的 `Uri` 指定联系人以外的联系人，除非你的确声明了 `READ_CONTACTS` 权限。

6.3 允许其它应用程序启动你的活动

前两节课关注故事的一边：从你的应用程序启动另一个应用程序的活动。但是，如果你的应用程序可以执行一个对另一个应用程序可能是有用的操作，你的应用应该准备响应来自其它应用程序请求的操作。例如，如果你构建一个可以与用户的朋友共享信息或照片的社交应用

程序，你最好能支持 `ACTION_SEND` 意图，这样用户就可以从另一个应用程序发起一个“分享”的操作来启动您的应用程序执行该操作。

要允许其它应用程序来启动你的活动，你需要在 `manifest` 文件中为相应的 `<activity>` 元素添加一个 `<intent-filter>` 元素。

当您的应用程序安装到设备上，系统识别你的意图过滤器，并增加信息到内部的所有已安装的应用程序支持的意图目录。当一个应用程序通过一个隐式意图调用 `startActivity()` 或 `startActivityForResult()`，系统查找哪个活动（或那些活动）能响应这个意图。

添加一个意图过滤器

为了正确地定义你的活动可以处理哪些意图，你添加的每个意图过滤器，在活动接受的操作和数据的类型方面应该尽可能地具体。

该系统可以发送一个给定的意图到活动，如果该活动的意图过滤器的 `Intent` 对象符合以下条件：

Action

一个字符串，命名要执行的动作。通常是一个平台定义好的值，如 `ACTION_SEND` 或 `ACTION_VIEW`。

用 `<action>` 元素在你的意图过滤器中指定它。此元素中指定的值必须是操作的完整字符串名称，而不是 API 常数（见下面的例子）。

Data

说明与意图相关的数据。

用 `<data>` 元素在你的意图过滤器中指定它。使用此元素中的一个或多个属性，你可以只指定 MIME 类型、一个 URI 前缀、一个 URI 方案，或指定这些和其它表明接受的数据类型的组合。

注：如果你不需要声明数据 URI 的具体信息（如您的活动时，处理其它种类的“额外”的数据，而不是一个 URI），你应该指定只有 `Android: MIMETYPE` 的属性声明类型。您的活动的数据处理，如为 `text / plain` 或 `image / jpeg` 文件。

Category

提供了另一种方式描述处理意图的活动的特征，通常涉及到启动它的用户的手势或位置。系统支持几种不同的类，但大多数都很少使用。然而，所有的隐式意图默认用 `CATEGORY_DEFAULT` 定义。

用 `<category>` 元素在你的意图过滤器中指定它。

在您的意图过滤器，你可以声明你的活动接受的条件，通过宣布每个嵌套在的 `<intent-filter>` 元素中对应的 XML 元素。

例如，这里有一个活动的意图过滤器，处理数据类型是文本或图像的 `ACTION_SEND` 意图：

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

每个传入的意图只能指定一个动作和一个数据类型，但可以在每个 `<intent-filter>` 中声明 `<action>`、`<category>` 和 `<data>` 元素的多个实例。

如果任何两个行动和数据对它们的行为是相互排斥的，您应该建立单独的意图过滤器来指定当与数据类型配对时哪些操作是可以接受的。

例如，假设你的活动同时处理 `ACTION_SEND` 和 `ACTION_SENDTO` 意图的文本和图像。在这种情况下，你必须为两个动作定义两个单独的意图过滤器，因为 `ACTION_SENDTO` 意图必须使用数据 `Uri` 通过 `send` 或 `sendto` URI 方案指定收件人的地址使用。例如：

```
<activity android:name="ShareActivity">
    <!-- filter for sending text; accepts SENDTO action with sms URI
    schemes -->
```

```

<intent-filter>

    <action android:name="android.intent.action.SENDTO"/>

    <category android:name="android.intent.category.DEFAULT"/>

    <data android:scheme="sms" />

    <data android:scheme="smsto" />

</intent-filter>

<!-- filter for sending text or images; accepts SEND action and text
or image data -->

<intent-filter>

    <action android:name="android.intent.action.SEND"/>

    <category android:name="android.intent.category.DEFAULT"/>

    <data android:mimeType="image/*"/>

    <data android:mimeType="text/plain"/>

</intent-filter>

</activity>

```

注：为了接收隐式意图，你必须在意图过滤器包括 `CATEGORY_DEFAULT` 类别。

`startActivity()` 和 `startActivityForResult()` 方法针对所有意图，只要它们包含 `CATEGORY_DEFAULT` 类别。如果你没有声明它，没有隐式意图会解决您的活动。

对于发送和接收执行社会分享行为的 `ACTION_SEND` 意图的更多信息，请参阅课程[从其它应用程序接收内容](#)。

在您的活动中处理意图

为了决定在你的活动中要进行什么样的操作，您可以读取启动活动所使用的意图。

您的活动启动后，调用 `getIntent()` 检索启动活动的意图。你可以在活动的生命周期中任意时刻这样做，但通常你应该在早期回调中这样做，如 `onCreate()` 或 `onStart()`。

例如：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    // Get the intent that started this activity
    Intent intent = getIntent();
    Uri data = intent.getData();

    // Figure out what to do based on the intent type
    if (intent.getType().indexOf("image/") != -1) {
        // Handle intents with image data ...
    } else if (intent.getType().equals("text/plain")) {
        // Handle intents with text ...
    }
}
```

返回结果

如果你想返回结果给调用你活动的活动，只需调用 `setResult()` 指定结果代码和结果意图。当你操作完成而用户应返回到原来的活动时，调用 `finish()` 来关闭（和销毁）你的活动。例如：

```
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION",
Uri.parse("content://result_uri");
setResult(Activity.RESULT_OK, result);
finish();
```

您必须始终对结果指定一个结果代码。一般来说，它要么是 `RESULT_OK`，要么是 `RESULT_CANCELED`。然后，必要时您可以给意图提供额外的数据。

注：结果默认设置为 `RESULT_CANCELED`。所以，如果用户在完成操作之前或在你设置结果之前按下返回按钮，原始的活动收到“已取消”的结果。

如果你只需要返回一个整数，表示若干结果选项之一，您可以设置结果代码为任何大于 0 的值。如果您使用的结果代码传递一个整数而你又不需要包括意图，你可以调用 `setResult()`，并只传递一个结果代码。例如：

```
setResult(RESULT_COLOR_RED);  
finish();
```

在这种情况下，有可能只有极少数的可能的结果，这样的结果代码是一个本地定义的整数（大于 0）。当你返回结果到自己的应用程序中的活动时它能运行良好，因为接收结果的活动，可以参照公共常量来确定的结果代码的值。

注：没有必要检查您的活动是否从 `startActivity()` 或 `startActivityForResult()` 启动。只需简单调用 `setResult()`，如果启动您的活动的意图可能想要一个结果。如果起初的活动已经调用过 `startActivityForResult()`，那么系统会传递您提供给 `setResult()` 的结果给它，否则，结果将被忽略。

7 共享内容

Android 应用程序的其中一个伟大之处是它们相互通信与集成的能力。当某个功能并不是您的应用程序的核心，且它已经在其它应用程序中存在时，为什么要重新发明它呢？

本课程涵盖了使用 Intent API 和 `ActionProvider` 对象在应用程序之间发送和接收内容的一些常见的方式。

课程

将内容发送到其它应用程序

了解如何设置你的应用程序能够使用意图发送文本和二进制数据到其它应用程序。

接收来自其它应用程序的内容

了解如何设置您的应用程序能够接收文本和二进制数据从意图。

添加一个简单的分享操作

了解如何添加一个“分享”的操作项到你的操作栏。

将内容发送到其它应用程序

当你构建一个意图时，你必须指定你想意图“触发”的操作。**Android** 定义了一些操作，其中包括 **ACTION_SEND**，你大概可以猜到，它表明意图是把数据从一个活动发送到另一个，甚至跨进程边界。将数据发送到另一个活动，所有你需要做的就是，指定数据及其类型，系统将识别出兼容的接收活动，并把它们显示给用户（如果有多个选项），或立即启动活动（如果只有一个选项）。同样，你可以在你的 **manifest** 文件中指定你的活动从其它应用程序能接收的数据类型。

通过意图在应用程序之间发送和接收数据的最常用于社交分享内容。意图允许用户使用他们最喜爱的应用程序快速简单地分享信息，。

注：添加一个分享操作项目到 **ActionBar** 的最好办法是使用 **ShareActionProvider**，成为可在 API 级别 14。**ShareActionProvider** 在添加一个简单的分享操作的课程中讨论。

发送文本内容

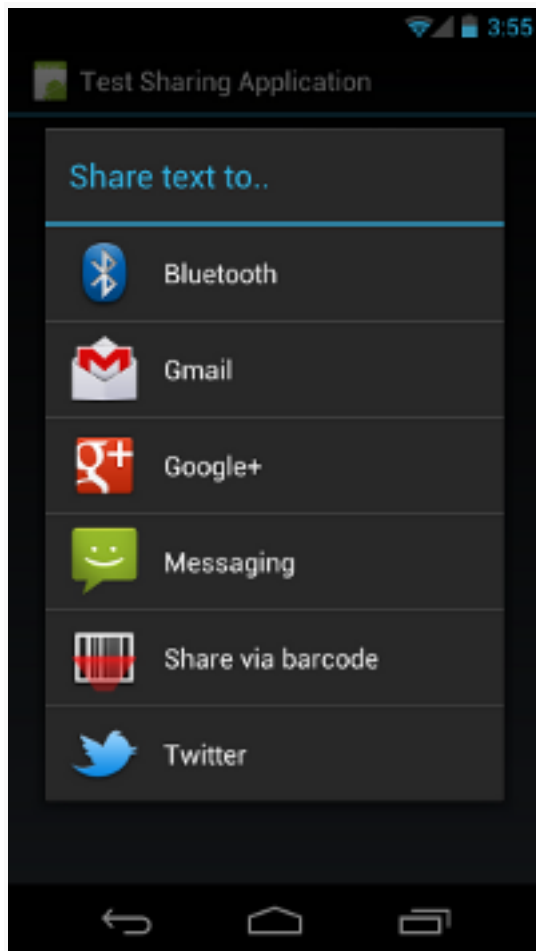


图 1。手机中 `ACTION_SEND` 意图选择器的截图。

`ACTION_SEND` 操作最直接和常见的用途是把文本内容从一个活动发送到另一个。例如，内置的浏览器应用程序可以把当前显示的页面的 `URL` 作为文字用任何应用程序分享。这对于通过电子邮件或社交网络与朋友分享一篇文章或网站是非常有用的。以下是以实现这种类型共享的代码：

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```


如果有一个安装的应用程序用过滤器(filter)匹配 `ACTION_SEND` 和 MIME 类型 `text / plain`, Android 系统将运行它;如果有一个以上的应用程序匹配, 系统会显示一个消歧(disambiguation)对话框(一个选择器), 允许用户选择其中一个应用程序。如果你调用意图的 `Intent.createChooser()`, Android 将**始终**显示选择器。这样做有以下优势:

- 即使用户为意图选择了默认操作, 选择器将仍然可以显示。
- 如果没有应用程序匹配, Android会显示一条系统消息。
- 您可以指定“选择器”对话框的标题。

下面是更新后的代码:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent,
    getResources().getText(R.string.send_to)));
```

结果对话框如图 1 所示。

可选地, 您可以对意图设置一些标准的附加信息(extras): `EXTRA_EMAIL`, `EXTRA_CC`, `EXTRA_BCC`, `EXTRA_SUBJECT`。但是, 如果接收的应用程序并没有设计成使用它们, 则什么都不会发生。您也可以使用自定义的附加信息, 除非接收应用程序能理解, 否则也没有任何效果。通常情况下, 你会使用由接收应用程序本身所定义的自定义额外信息。

注: 某些电子邮件应用程序, 如 Gmail, 期望 `String []`作为额外信息, 比如 `EXTRA_EMAIL` 和 `EXTRA_CC`, 可通过 `putExtra(String, String[])`添加它们到你的意图。

发送二进制内容

要使用 `ACTION_SEND` 操作共享二进制数据, 要设置适当的 MIME 类型, 并把数据的 URI 放在一个名为 `EXTRA_STREAM` 的附加信息中。这通常是用来共享一个图像, 但也可以用来共享任何类型的二进制内容:

```
Intent shareIntent = new Intent();

shareIntent.setAction(Intent.ACTION_SEND);

shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);

shareIntent.setType("image/jpeg");

startActivity(Intent.createChooser(shareIntent,
    getResources().getText(R.string.send_to)));
```

请注意以下几点：

- 您可以使用MIME的类型为“*/*” ，但这只会匹配能够处理一般的数据流的活动，。
- 接收应用程序需要权限访问URI指向的数据。有很多方法来处理这个：
 - 把数据写入到所有的应用程序可以读取的外部/共享存储（如SD卡）的一个文件上。使用`Uri.fromFile()`来创建可以传递到共享意图的`Uri`。但是，请记住，并不是所有的应用程序处理`file://`样式的`Uri`。
 - 在`MODE_WORLD_READABLE`模式下使用`openFileOutput()`把数据写入到你自己的应用程序目录下的一个文件中，然后用`getFileStreamPath()`返回一个`File`。使用之前的选项，`Uri.fromFile()`将为你的共享意图创建一个`file://`样式的`Uri`。
 - 可以使用`scanFile()`扫描图像、视频和音频等媒体文件并添加到系统`MediaStore`。`onScanCompleted()`回调方法返回`content://`样式的`Uri`适用于包含在你的共享意图中。
 - 使用`insertImage()`可以把图像插入到系统`MediaStore`中，它将返回`content://`样式的`Uri`适用于包含在你的共享意图中。
 - 存储数据到你自己的`ContentProvider`中，确保其它应用程序有正确的权限访问您的provider（或使用`per-URI`权限）。

发送多条内容

要分享多条内容，对指向内容的 URI 列表使用 `ACTION_SEND_MULTIPLE` 操作。MIME 类型根据你要分享的混合内容而定。例如，如果你要分享 3 个 JPEG 图像，类型仍是 `"image/jpeg"`。对于混合的图像类型，它应该用 `"image/*"` 来匹配一个处理任何类型图

像的活动。您应该只使用“*/*”，如果你分享了各种各样的类型。如前所述，轮到要接收的应用程序来解析和处理您的数据。下面是一个例子：

```
ArrayList<Uri> imageUris = new ArrayList<Uri>();

imageUris.add(imageUri1); // Add your image URIs here
imageUris.add(imageUri2);

Intent shareIntent = new Intent();

shareIntent.setAction(Intent.ACTION_SEND_MULTIPLE);

shareIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM,
imageUris);

shareIntent.setType("image/*");

startActivity(Intent.createChooser(shareIntent, "Share images to.."));
```

像之前一样，确保所提供的 [URI](#) 指向接收应用程序能够访问的数据。

接收来自其它应用程序的内容

正如你的应用程序可以将数据发送到其它应用程序，所以也可以很容易接收来自应用程序的数据。想想用户如何与你的应用程序交互，以及你要从其它应用程序接收什么样类型的数据。例如，一个社交网络应用程序可能会对从另一个应用程序接收文本内容感兴趣，像一个有趣的网页的 [URL](#)。 [Google+ Android 应用程序](#) 接受文字和单个或多个图像。有了这个程序，用户可以轻松地开始新的 Google+ 发布来自 Android 库应用程序中的照片。

更新你的Manifest

意图过滤器通知系统，应用程序组件愿意接受什么意图。与你在[使用意图发送内容到其它应用程序](#)那课程中通过 `ACTION_SEND` 操作构建意图的方式相似，为了能够接收这个操作的意图您要创建意图过滤器。您使用 `<intent-filter>` 元素在你的 `manifest` 中定义一个意图过滤器。例如，如果您的应用程序处理接收文本内容，任何类型单一图片，或任何类型的多个图片，你的 `manifest` 看起来应该像这样子：

```
<activity android:name=".ui.MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

注：对于意图过滤器和意图解决方案的更多信息，请阅读[意图](#)和[意图过滤器](#)

当另一个应用程序试图通过建设意图，并把它传递给 `startActivity()` 的分享任何这些东西，你的申请将被列为意向选择中的一个选项。如果用户选择应用程序时，将启动相应的活动（在上面的例子。`ui.MyActivity`）。然后，它是由你来处理的内容适当地在自己的代码和 UI。

处理传入的内容

要处理意图分发的内容，通过调用 `getIntent()` 来获得[意向](#)对象开始。一旦你有了这个对象，你可以检查其内容以确定下一步该怎么做。请记住，如果这个活动可以从系统的其它部件开启，如启动器，那么你就需要在检查意图时考虑到这一点。

```

void onCreate (Bundle savedInstanceState) {

    ...

    // Get intent, action and MIME type
    Intent intent = getIntent();
    String action = intent.getAction();
    String type = intent.getType();

    if (Intent.ACTION_SEND.equals(action) && type != null) {
        if ("text/plain".equals(type)) {
            handleSendText(intent); // Handle text being sent
        } else if (type.startsWith("image/")) {
            handleSendImage(intent); // Handle single image being sent
        }
    } else if (Intent.ACTION_SEND_MULTIPLE.equals(action) && type !=
null) {
        if (type.startsWith("image/")) {
            handleSendMultipleImages(intent); // Handle multiple images
being sent
        }
    } else {
        // Handle other intents, such as being started from the home screen
    }
    ...
}

void handleSendText(Intent intent) {
    String sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    if (sharedText != null) {
        // Update UI to reflect text being shared
    }
}
}

```

```
void handleSendImage(Intent intent) {
    Uri imageUri = (Uri)
intent.getParcelableExtra(Intent.EXTRA_STREAM);
    if (imageUri != null) {
        // Update UI to reflect image being shared
    }
}

void handleSendMultipleImages(Intent intent) {
    ArrayList<Uri> imageUris =
intent.getParcelableArrayListExtra(Intent.EXTRA_STREAM);
    if (imageUris != null) {
        // Update UI to reflect multiple images being shared
    }
}
```

注意：检查传入的数据时要格外小心，你永远不知道其它一些应用程序可能会发送给您什么东西。例如，可能设置了错误的 **MIME** 类型，或者发送的图像可能非常大。此外，请记住，在一个单独的线程来处理二进制数据而不是主（“UI”）线程。

更新用户界面，可以像填充 `EditText` 那么简单，也可以像在图像上应用一个有趣的照片滤镜那么复杂。它真的是特定于应用程序的下一步会发生什么。

添加一个简单的分享操作

随着 Android 4.0（API 等级 14）对 `ActionProvider` 的引入，在操作栏上实现有效且用户友好的分享操作，变得更加简单了。一个 `ActionProvider`，一旦附在操作栏的某个菜单项，要同时处理该项的外观和行为。在使用 `ShareActionProvider` 的情况下，你只要提供了一个分享意图，它会完成剩下的工作。

注： `ShareActionProvider` 只在 API 等级 14 和更高的版本中才开始可用。

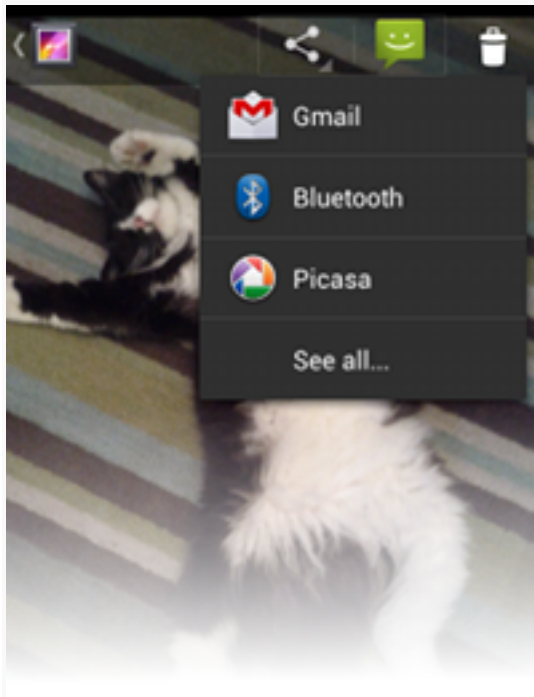


图 1。相册应用程序中的 `ShareActionProvider`。

更新菜单声明

开始使用 `ShareActionProviders` 时，首先在你的菜单资源文件为对应的 `<item>` 定义 `Android: actionProviderClass` 属性：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_share"
        android:showAsAction="ifRoom"
        android:title="Share"
        android:actionProviderClass="android.widget.ShareActionProvid
er" />
    ...
</menu>
```

这代表该项中 `ShareActionProvider` 的外观和功能职责。但是，你需要告诉提供者（provider）你想要分享的东西。

设置分享意图

为了实现 `ShareActionProvider` 的功能，你必须给它提供一个分享意图。这一分享意图应该跟“将内容发送到其它应用程序”那节课所描述的一样，使用 `ACTION_SEND` 操作并通过额外信息像 `EXTRA_TEXT` 和 `EXTRA_STREAM` 设置附加的数据。要指定一个分享意图，当你在您的活动或碎片中填充菜单资源时，首先找到相应的菜单项。接下来，调用 `MenuItem.getActionProvider()` 检索一个 `ShareActionProvider` 实例。使用 `setShareIntent()` 更新与操作项关联的意图。下面是一个例子：

```
private ShareActionProvider mShareActionProvider;

...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate menu resource file.
    getMenuInflater().inflate(R.menu.share_menu, menu);

    // Locate MenuItem with ShareActionProvider
    MenuItem item = menu.findItem(R.id.menu_item_share);

    // Fetch and store ShareActionProvider
    mShareActionProvider = (ShareActionProvider)
item.getActionProvider();

    // Return true to display menu
    return true;
}

// Call to update the share intent
private void setShareIntent(Intent shareIntent) {
    if (mShareActionProvider != null) {
```



```
mShareActionProvider.setShareIntent(shareIntent);  
  
}  
  
}
```

在菜单的创建过程中您可能只需要设置一次共享意图，或者您可能想把它设置好，然后在 UI 变化时再更新它。例如，当您在相册应用程序中全屏浏览照片时，切换照片分享意图也会跟着变化。

有关 `ShareActionProvider` 对象的进一步讨论，请查阅[操作栏指南](#)。