



**Google Android官方培训教程**

**中文版**

# Google Android官方培训课程中文版(v0.1)



Google Android团队在2012年的时候开设了**Android Training**板块 - <http://developer.android.com/training/index.html>，这些课程是学习Android应用开发的绝佳资料。我们通过Github发起开源协作翻译的项目，完成中文版的输出。欢迎大家传阅学习，更欢迎读者加入此协作项目，为完善这份教程贡献你的一点力量！

## Github托管主页

<https://github.com/kesenhoo/android-training-course-in-chinese>

请读者点击Star进行关注并支持！

## 版本信息：

- v0.1 - 2014/08/05

## 离线文档

- PDF: [http://hukai.me/eBooks/Google\\_Android\\_Training\\_Course\\_In\\_Chinese.pdf](http://hukai.me/eBooks/Google_Android_Training_Course_In_Chinese.pdf)
- ePub: [http://hukai.me/eBooks/Google\\_Android\\_Training\\_Course\\_In\\_Chinese.epub](http://hukai.me/eBooks/Google_Android_Training_Course_In_Chinese.epub)
- Mobi: [http://hukai.me/eBooks/Google\\_Android\\_Training\\_Course\\_In\\_Chinese.mobi](http://hukai.me/eBooks/Google_Android_Training_Course_In_Chinese.mobi)

## 参与方式

你可以选择以下的方式帮忙修改纠正这份教程：

1. 通过在线阅读课程的页面，点击目录导航栏左上角的"Edit and Contribute"，会跳转到此面对应的Github源码页面(此操作会自动帮你Fork项目源码到你的账户并更新文件至最新)，然后你可以在自己的仓库下修改，确定提交之后，Github会自动引导你提交Pull Request。
2. 在线阅读的文章底部留言，提出问题与修改意见，我们会及时处理。
3. 写邮件给项目发起人[胡凯](#)，邮箱是kesenhoo at gmail.com，邮件内容注明需要纠正的章节段落位置，并给出纠正的建议。
4. 加入QQ交流群：**363415744**，向群主或者管理员提错误。

## 特别致谢

发起这个项目之后，得到很多人的支持，有经验丰富的Android开发者，也有刚接触Android的爱好者。他们有些已经上班，有些还是学生，有些在国内，还有的在国外！感谢所有参与或者关注这个项目的小伙伴！

下面是参与翻译的小伙伴(Github ID按照课程结构排序)

- [@yuanfentiank789](#)
- [@vincent4j](#)
- [@Lin-H](#)
- [@kesenhoo](#)
- [@fastcome1985](#)
- [@jdneo](#)
- [@XizhiXu](#)
- [@naizhengtan](#)
- [@spencer198711](#)
- [@penzhou](#)
- [@wangyachen](#)
- [@wly2014](#)
- [@fastcome1985](#)
- [@riverfeng](#)
- [@xrayzh](#)
- [@K0ST](#)
- [@Andrwyw](#)
- [@zhaochunqi](#)
- [@lltowq](#)
- [@allenlsy](#)
- [@AllenZheng1991](#)
- [@pedant](#)
- [@craftsmanBai](#)
- [@huanglizhuo](#)

另外感谢[安卓巴士社区](#)，[爱开发社区](#)，[码农周刊](#)对项目的宣传！

## License

本站作品由<https://github.com/kesenhoo/android-training-course-in-chinese>创作，采用[知识共享署名-非商业性使用-相同方式共享 4.0 国际 许可](#)协议进行许可。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/index.html>

# 从这里开始

欢迎来到为Android开发者准备的培训项目。在这里你会找到一系列的课程，这些课程会演示你如何使用可重用的代码来完成特定的任务。所有的课程分为若干不同的小组。你可以通过左边的导航来查看。

第1组：“从这里开始”，教你Android应用开发的最基本的知识。如果你是一个Android应用开发的新手，你应该按照顺序学习完下面的课程：

## [建立你的第一个App\(Building Your First App\)](#)

在你安装Android SDK之后，从这节课开始学习Android应用开发的基础知识。

## [添加ActionBar\(Adding the Action Bar\)](#)

ActionBar是你的Activity中最重要的设计元素之一。尽管ActionBar是从API 11开始被引入的，你仍然可以从Android 2.1开始使用Support Library去实现ActionBar。

## [兼容不同的设备\(Supporting Different Devices\)](#)

学习给应用提供可选择的资源文件来实现如何使用一个APK来使得你的应用能够在不同的设备上获取到最佳的用户体验。

## [管理Activity的生命周期\(Managing the Activity Lifecycle\)](#)

学习Android的Activity的创建与销毁，学习如何通过实现生命周期的回调方法来创建一个无缝的用户体验。

## [使用Fragment建立动态的UI\(Building a Dynamic UI with Fragments\)](#)

学习如何为你的应用建立一套足够灵活的UI，这套UI能够在大屏幕的设备上显示多个UI组件，在小屏幕的设备上呈现紧凑的UI组件。这使得你能够为手机与平板只建立同一个APK。

## [数据保存\(Saving Data\)](#)

学习如何在设备上保存数据。无论这些数据是临时的文件，应用下载的资源，用户的多媒体数据，结构化的数据还是其他。

## [与其他应用的交互\(Interacting with Other Apps\)](#)

学习如何利用其他已经存在应用的既有功能来执行更进一步的用户任务。例如拍照或者在地图上查看某个地址。

编写: [yuanfentiank789](#)

原文: <http://developer.android.com/training/basics/firstapp/index.html>

# 建立第一个App

欢迎开始Android应用开发！本章节教你如何建立你的第一个Android应用程序。您将学到如何创建一个Android项目和运行可调试版本的应用程序。你还将学习一些Android应用程序设计的基础知识，包括如何创建一个简单的用户界面，处理用户输入。

开始本章节学习之前，确保你已经安装了开发环境。你需要：

- 1 下载Android SDK.
- 2 为Eclipse安装ADT插件 (假设你使用Eclipse作为开发工具).
- 3 使用SDK Manager下载最新的SDK tools和platforms。

注意：开始本节学习之前，确保你安装了最新版本的ADT插件和Android SDK。本章节描述的学习过程有可能不适用早期版本。

如果你尚未完成这些任务，开始[Android SDK](#)的下载和安装步骤。安装完成后，你就可以准备开始本章节的学习了。

本章节通过教程的方式逐步建立一个小型的Android应用，来教给你一下Android开发的基本概念，因此对你来说每一步都很重要。

[开始第一节课](#)

编写: [yuanfentiank789](#)

原文: <http://developer.android.com/training/basics/firstapp/creating-project.html>

# 创建Android项目

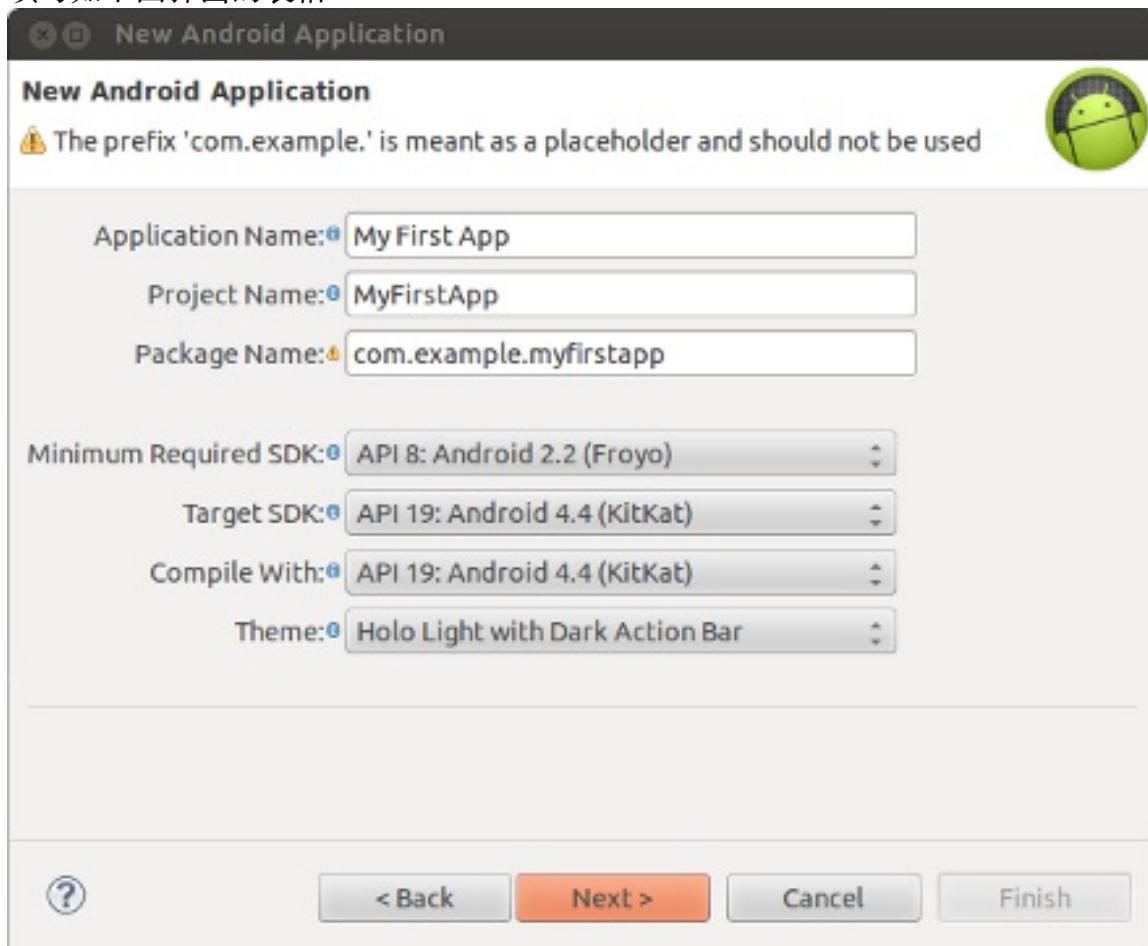
一个Android项目包含了生产Android应用所需要的全部源代码文件，使用Android SDK Tools可以很容易地创建一个新的Android项目，同时创建好项目默认的目录和文件。

本小节介绍如何使用Eclipse（已安装ADT插件）或SDK Tools命令行来创建一个新的项目。

注意：如果你使用Eclipse开发，应该确保已经安装了Android SDK，并且为Eclipse安装了ADT（version 22.6.2或更高版本）插件。否则，请先阅读 [Installing the Android SDK](#)按照向导完成安装。

# 使用Eclipse创建项目

1. 点击Eclipse工具栏的New按钮,如下图所示
2. 在弹出的窗口中打开Android文件夹, 选择Android Application Project, 点击Next.
3. 填写如下图弹出的表格:



- **Application Name:**此处填写想呈现给用户的应用名称, 此处我们使用“My First App”.
- **Project Name:**是项目的文件夹名称和在Eclipse中显示的名称.
- **Package Name:**是应用的包命名空间 (同Java的包的概念) , 该包名在同一Android系统上所有已安装的应用中具有唯一性, 因此, 通常使用你所在公司组织或发布实体的反向域名作为包名的开始是一个很好的选择。此处可以使用“com.example.myfirstapp” , 但是你不能在 Google Play上发布使用 “com.example”作为包名的应用.
- **Minimum Required SDK:**用API level表示你的应用支持的最低Android版本, 为了支持尽可能多的设备, 你应该设置为能支持你应用核心功能的最低API版本。如果某些非核心功能尽在较高版本的API支持, 你可以只在支持这些功能的版本上开启它们(参考[兼容不同的系统版本](#)), 此处采用默认值即可。
- **Target SDK:**表示你测试过你的应用支持的最高Android版本(同样用API level表示).当Android发布最新版本后, 你应该在最新版本的Android测试你的应用同时更新target sdk到Android最新版本, 以便充分利用Android新版本的特性。
- **Compile With:**是你的应用将要编译的目标Android版本, 此处默认为你的SDK已安装的最新Android版本(目前应该是4.1或更高版本, 如果你没有安

装一个可用Android版本，就要先用[SDK Manager](#)来完成安装)，你仍然可以使用较老的版本编译项目，但把该值设为最新版本，你可以使用Android的最新特性同时可以优化应用来提高用户体验，运行在最新的设备上。

- **Theme**:为你的应用指定界面风格，此处采用默认值即可。

点击**Next**

4. 接下来的窗口配置项目，保持默认值即可，点击**Next**。
5. 这一步帮助你给你的应用创建一个启动图标，你也可以自定义应用启动图标，通过用工具为各种屏幕密度的屏幕各创建一个对应图标。但在发布应用之前，应确保你设计的图标符合[Iconography](#)中规定的规范。点击**Next**。
6. 这一步为默认的入口Activity选择一个模板，此处选择**BlankActivity**，然后点击**Next**。
7. 这一步保持Activitiy的默认配置即可。

到此为止，你的Android项目已经是一个基本的“Hello World”程序，包含了一些默认的文件。要运行它，继续[下个小节](#)的学习。

# 使用命令行创建项目

如果你没有使用Eclipse + ADT开发Android项目，也可以在命令行使用SDK提供的tools来创建一个Android项目。

- 打开命令行切换到SDK根目录/tools目录下；
- 执行

```
android list targets
```

会在屏幕上打印出所有你使用Android SDK下载好的可用platforms，找到你想要创建项目的目标platform，记录该platform对应的Id，推荐你使用最新的platform，可以使你的应用支持较老版本的platform，同时允许你为最新的Android设备优化你的应用。如果你没有看到任何可用的platform，你需要使用SDK Manager完成下载安装，参见 [Adding Platforms and Packages](#)。

- 执行

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

替换<target-id>为上一步记录好的Id，替换<path-to-workspace>为你想要保存项目的路径，到此为止，你的Android项目已经是一个基本的“Hello World”程序，包含了一些默认的文件。要运行它，继续[下个小节](#)的学习。

**Note:**把platform-tools/和 tools/添加到环境变量PATH，开发更方便。

编写: [yuanfentiank789](#)

原文: <http://developer.android.com/training/basics/firstapp/running-app.html>

# 执行Android程序

通过[上一节课](#)创建了一个Android项目，项目默认包含一系列源文件，它让您可以立即运行的应用程序。

如何运行Android应用取决于两件事情：你是否有一个Android设备和你是否正在使用Eclipse开发程序。本节课将会教使用Eclipse和命令行两种方式在真实的android设备或者android模拟器上安装并且运行你的应用。

在运行应用之前，你得认识项目里的几个文件和目录：

## AndroidManifest.xml

[manifest file](#) 描述了应用程序的基本特性并且定义了每一个组件。当你学了更多课程，你将会理解这里的各种声明。其中一个很重要的点是：你的manifest应该包括 `<uses-sdk>` 标签。它会利用 `android:minSdkVersion` 和 `android:targetSdkVersion` 两个属性来声明你应用程序对于不同的 android 版本的兼容性。在你的第一个应用里，它看起来应该是这样：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="19">
...
</manifest>
```

你应该总是把 `android:targetSdkVersion` 设置的尽可能的高并且在对应版本的Android系统上测试你的应用。详见[适配不同的系统版本](#)

## src/

这是存放应用的主要源代码的文件夹，默认情况下，里面会包括一个[Activity](#)的类，这个类会在点击应用程序图标启动的时候运行。

## res/

包含一些存放资源文件的目录，例如：

```
drawable-hdpi/
```

存放适用于HDPI屏幕的图片素材。同理其他类似文件夹存放适用于其他屏幕的图片素材。

```
layout/
```

存放定义用户界面的的文件。

```
values/
```

存放其他各种XML文件，也是所有资源的集合，例如字符串和颜色的定义。

当完成该项目的编译和运行工作后，默认的[Activity](#)类启动并加载一个布局文件，界面显示

"Hello World."这本身没有什么值得兴奋的，重要的是你学会了如何运行一个Android应用在你开始进行开发之前。

## 在真实设备上运行

如果你有一个真实的Android设备，以下的步骤可以使你在你的设备上安装和运行你的应用程序：

- 把你的设备用USB线连接到计算机上。如果你是在windows系统上进行开发的，你可能还需要安装你设备对应的USB驱动，详见[OEM USB Drivers](#) 文档。
- 开启设备上的USB调试选项。
  - 在大部分运行Andriod3.2或更老版本系统的设备上，这个选项位于“设置>应用程序>开发选项”里。
  - 在Andriod 4.0或更新版本中，这个选项在“设置>开发人员选项”里。

**注意:** 从Android4.2开始，开发人员选项在默认情况下是隐藏的，想让它可见，可以去“设置>关于手机（或者关于设备）”点击“版本号”七次。再返回就能找到开发人员选项了。

用Eclipse在设备里运行程序：

- 打开项目文件，点击工具栏里的**Run**按钮。



- 在 Run as 弹出窗口中，选择 Android Application 然后点击 OK。

Eclipse 会把应用程序安装到你的设备中并启动应用程序。

或者也可以利用命令行安装运行你的应用程序。

- 命令行切换当前目录到Andriod项目的根目录，执行：

```
ant debug
```

- 确保 Android SDK里的 platform-tools/ 路径已经添加到环境变量的Path中，执行：

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

- 在你的Android设备中找到 MyFirstActivity，点击打开。

以上就是创建并在设备上运行一个应用的全部过程！想要开始开发，点击[next lesson](#)。

## 在模拟器上运行

无论你是用Eclipse还是命令行，在模拟其中运行程序首先要创建一个模拟器，即 Android Virtual Device (AVD)，配置AVD 可以让你模拟在不同版本和尺寸的Android设备运行应用程序。



创建一个 AVD:

- 启动 Android Virtual Device Manager (AVD Manager) 的两种方式：
  - 用Eclipse, 点击工具栏里面Android Virtual Device Manager Android。
  - 在命令行窗口中, 把当前目录切换到<sdk>/tools/ 后执行：



```
android avd
```

- 在 Android Virtual Device Manager 面板中, 点击New.
- 填写AVD的详细信息, 包括名字, 平台版本, SD卡大小以及屏幕大小 (默认是 HVGA)。
- 点击 Create AVD.
- 在Android Virtual Device Manager 选中创建的新AVD, 点击 Start。
- 在模拟器启动完毕后, 解锁模拟器的屏幕。

接下来就可以像前边讲过的一样用Eclipse或命令行来往模拟器发布运行你的应用程序了。

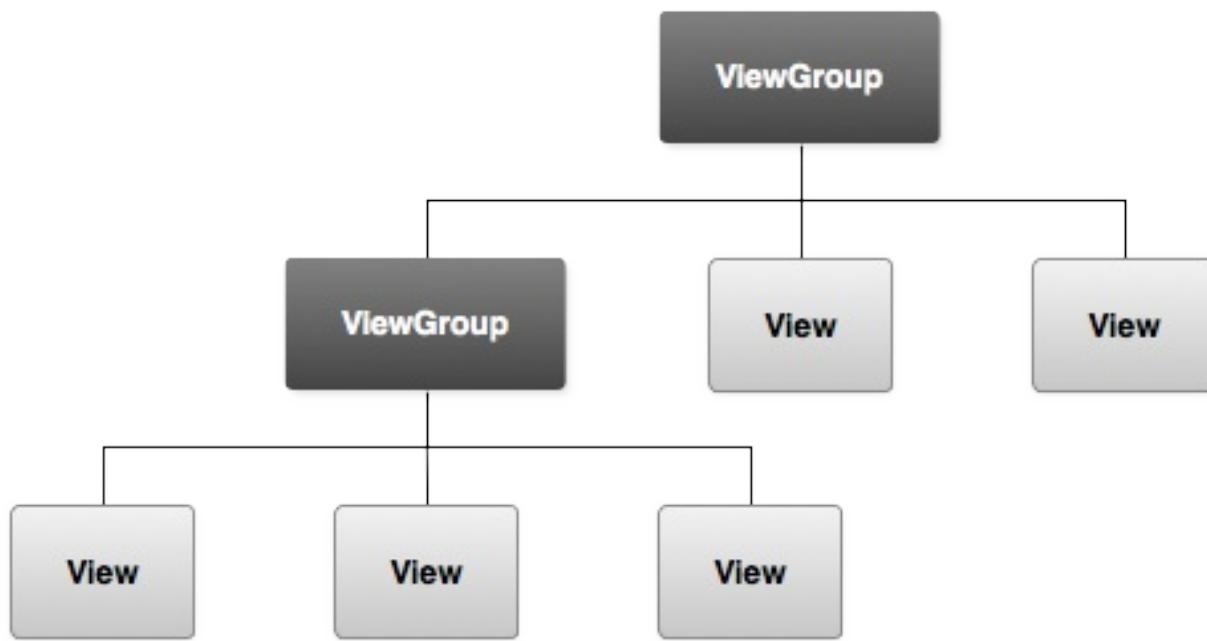
编写: [yuanfentiank789](#)

原文: <http://developer.android.com/training/basics/firstapp/building-ui.html>

# 建立一个简单的用户界面

Android的图形用户界面是由多个View和ViewGroup构建出来的。View是通用的UI窗体小组件，比如按钮或者文本框，而ViewGroup是不可见的用于定义子View布局方式的容器，比如网格部件(grid)和垂直列表部件(list)。

Android提供了一个对应于View和ViewGroup子类的XML标签词汇表，你可以在XML里使用层级视图元素创建自己的UI。



在本小节里，你将学到怎样用XML创建一个带有文本输入框和按钮的界面。在接下来的课里，你将学会对按钮做出响应，当按钮被按下的时候文本框里的内容被发送到另外一个Activity。

可选的布局文件：在XML中定义界面布局而不是在运行时去动态生成布局是有多个原因的，其中最重要的一个原因是这样可以使得你为不同大小的屏幕创建不同的布局文件。例如，你可以创建2个版本的布局文件，告诉系统在小的屏幕上使用其中一个布局文件，在大的屏幕上使用另外一个布局文件。更多信息，请参考[兼容不同的设备](#)

## 创建一个LinearLayout

从目录res/layout里打开fragment\_main.xml文件。

**Note:** 在eclipse中，当你打开布局文件的时候，首先看到的是图形化布局编辑器，这个编辑页是使用所见即所得的工具帮助你创建布局。对于本课来说，你是直接在XML里进行操作，因此点击屏幕下方的main.xml标签进入XML编辑页。

你创建项目时选择的BlankActivity 模板生成的fragment\_main.xml文件包含一个RelativeLayout的根View和一个TextView的子View。

首先，删除TextView标签并修改RelativeLayout为 LinearLayout,然后添加android:orientation 属性并设置该属性为horizontal，修改后结果如下：

```
<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ViewGroup的一个子类，用于放置水平或者垂直放置子视图的部件，由属性android:orientation来设定方向。LinearLayout里的子布局按照XML里设定的布局方向显示在屏幕上。

另外的两个属性android:layout\_width和android:layout\_height，对于所有的Views都需要对这两个属性进行设这。

在这里因为LinearLayout是整个视图的根布局，所以对于宽和高都应该是充满整个屏幕的，通过指定width 和 height属性为"match\_parent"。该值表示子View扩张自己width和height来匹配父控件的width和height。

想要获得更多关于[布局](#)属性的信息，请参照XML布局向导。

# 添加一个文本输入框

在`LinearLayout`里添加一个`EditText`元素就可以创建一个用户可编辑的文本框，和其它View一样，你需要设置XML里的某些属性来指定`EditText`的具体功能，下边是你应该在线性布局里指定的一些属性元素：

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

属性说明：

## android:id

这里定义的是View的唯一标示符，你可以在程序的代码里进行引用，你可以对这个类进行读和修改的操作(在一课里将会用到)

当你想从XML里使用资源类的时候必须使用@符号，紧随@之后的是资源的类型(这里是id)，然后是资源的名字(这里使用的是edit\_message)。(其他的资源可以使用相同的名字只要他们不是相同的资源类型，例如：字符串资源可以使用相同的名字)。

+号只是当你第一次定义一个资源ID的时候需要。这里是告诉SDK此资源ID需要被创建出来。在应用程序被编译之后，SDK就可以直接使用ID值，edit\_message是在项目gen/R.java文件中创建一个新的标示符，这个标示符就和`EditText`关联起来了。一旦资源ID被创建了，其他资源如果引用这个ID就不再需要+号了。这里是唯一一个需要+号的属性。

## android:layout\_width 和 android:layout\_height

对于宽和高不建议指定具体的大小，使用`wrap_content`指定之后，这个视图只是占据内容大小的空间。如果你使用了`fill_parent`，这时`EditText`将会布满整个屏幕，因为它将适应父布局的大小。想要看到更多信息，请参考XML [布局向导](#)。

## android:hint

当文本框为空的时候，会默认显示这个字符串。对于字符串`@string/edit_message`的值所引用的资源应该是定义在单独的文件里，而不是直接使用字符串。因为使用的是值是存在的资源，所以不需要使用+号。然而，由于你还没有定义字符串的值，所以在添加`@string/edit_message`时候会出现编译错误。下边你可以定义字符串资源值来去除这个错误。

**Note:** 该字符串资源与id使用了相同的名称（edit\_message）。然而，对于资源的引用是区分类型的（比如id和字符串），因此，使用相同的名称不会引起冲突。

## 增加字符串资源

当你在用户界面定义一个文本的时候，你应该把每一个文本字符串列入资源文件。对于所有字符串值，字符串资源能够单独的修改，在资源文件里你可以很容易的找到并且做出相应的修改。通过选择定义每个字符串，还允许您对不同语言本地化应用程序。

默认情况下，在`res/values/string.xml`里，你的Android项目包含一个字符串资源文件。打开这个文件，删除已经存在的"hello"字符串，为"edit\_message"增加一个供使用的字符串值。同时在这个文件里，再给button添加一个字符串，命名为"button\_send". 下边就是定义好的string.xml文件内容：

```
<?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="action_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

要想获得跟多的对于不同语言本字符串资源本地化的问题，请参考[兼容不同的设备\(Supporting Different Devices\)](#)。

## 添加一个按钮

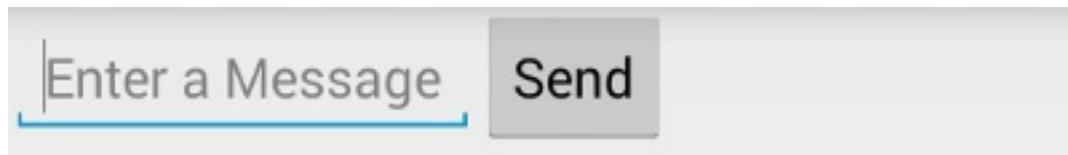
紧跟EditText后边，添加一个Button到布局里。

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

宽和高被设置为"wrap\_content"，这时按钮占据的大小就是按钮里文本的大小。这个按钮不需要指定android:id的属性，因为在Activity代码里不被引用到。

## 让输入框充满整个屏幕的宽度

当前EditText和Button部件只是适应了他们各自内容的大小，如下图所示：



这样设置对按钮来说很合适，但是对于文本框来说就不太好，因为用户可能输入更长的文本内容，需要在左边留有一定的空白空间。因此如果能够沾满整个宽度会更好。

LinearLayout使用权重的属性来达到这个目的，你可以使用`android:layout_weight`属性来设置。

你可以根据每一个部件所占的空间来指定权重值的大小，它的总数是有同级别的部件来决定的。就类似于饮料的成分配方：“两份伏特加酒，一份咖啡利口酒”，意思就是这个酒中伏特加酒占三分之二。例如，你设置一个View的权重是2，另一个View的权重是1，那么总数就是3，这时第一个View占据2/3的空间，第二个占据1/3的空间。如果你再加入第三个View，权重设为1，那么第一个View会占据1/2的空间，剩余的被另外两个View平分。

对于所有的View默认的权重是0，如果你只设置了一个View的权重大于0，那么这个View将占据除去别的View本身占据的空间的所有剩余空间。因此这里设置EditText的权重为1，使其能够占据了按钮之外的所有空间。

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

为了提示布局的效率，在设置权重的时候，你应该把EditText的宽度设置为0dp。如果你设置为"wrap\_content"作为宽度，系统需要自己去计算这个部件所占有的宽度，而此时的因为你设置了权重，所以系统自动回占据剩余空间，EditText的宽度最终成了不起作用的属性。

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

下图展示了设置权重时候的结果



现在看一下完整的布局文件内容：

```
<?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>
```

整个布局默认被[Activity](#)类使用，Activity类是在你创建一个项目的时候SDK工具自动生成的，你可以直接运行app查看运行结果：

在Eclipse里，点击工具栏里的Run 

或者使用命令行，进入你项目的根目录直接执行

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

继续下一小节学习有关怎么对按钮做出相应，同时读取文本里的内容，启动另外一个Activity，以及更多信息。

[下一节：启动另外的Activity](#)

编写: [yuanfentiank789](#)

原文: <http://developer.android.com/training/basics/firstapp/starting-activity.html>

# 启动其他Activity

在完成上一课(建立简单的用户界面)后，你已经拥有了显示一个activity (唯一屏幕) 的app (应用)，并且这个activity包含了一个文本字段和一个按钮。在这节课中，你将会添加一些新的代码到MainActivity中，当用户点击发送(Send)按钮时启动一个新的activity。

## 响应Send(发送)按钮

响应按钮的on-click(点击)事件，打开fragment\_main.xml布局文件然后在[Button\(按钮\)](#)元素中添加android:onClick属性：

```
<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就是当用户点击你屏幕按钮时触发方法的名字。

添加相应的方法在MainActivity类中：

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

请注意，为了让系统能够将这个方法（你刚在MyFirstActivity中添加的sendMessage方法）与在android:onClick属性中提供的方法名字匹配，它们的名字必须一致，特别是，这个方法必须满足以下条件：

- 公共的
- 没有返回值
- 有唯一的视图（View）参数（这个视图就是将被点击的视图）

接下来，你可以在这个方法中编写读取文本内容的代码，并将该内容传到另一个Activity。

# 构建一个Intent

[Intent](#)是在不同组件中提供运行时连接的对象(比如两个Activity)。Intent代表一个应用"想去做什么事"，你可以用它做各种各样的任务，不过大部分的时候他们被用来启动另一个Activity。在sendMessage()方法中创建一个Intent并启动名为DisplayMessageActivity的Activity：

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

**Note:**在Eclipse中，按Ctrl + Shift + O 可以导入缺失的类(在Mac中使用Cmd + Shift + O )

在这个Intent构造函数中有两个参数：第一个参数是Context(之所以可以用this是因为当前Activity(MyFirstActivity)是Context的子类) 系统需要传递Intent的应用组件的class对象 (在这个案例中，这个activity应该被启动)

注意：如果你正在使用的是类似Eclipse的IDE，这里对DisplayMessageActivity的引用会报错，因为这个类还不存在；注意这个错误，你很快就要去创建这个类了。

一个Intent(意图)不仅允许你启动另一个Activity，同时也可以传递一个数据包到另一个Activity，在sendMessage()方法里用[findViewById\(\)](#)方法得到EditText元素，然后将它的文本信息添加到Intent(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);
```

Intent可以携带各种数据类型的集合来作为key-value附加对。putExtra() 方法把键名作为第一个参数，把值作为第二个参数。为了接下来的活动能够查询额外数据，应该用公共常量为意图额外定义键。所以把EXTRA\_MESSAGE定义添加到MainActivity类：

```
public class MainActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE = "com.example.myfirs
    ...
}
```

通常使用应用程序包名作为前缀来定义意图键是很好的做法。如果应用程序与其他应用程序进行交互就可以确保意图键唯一。

## 启动第二个Activity

启动一个Activity，你只需要调用startActivity()方法然后传入你的Intent(意图)系统接收到你的请求后会实例化在Intent中指定的Activity,包含这个方法拥有的，被Send(发送)按钮调用的完整sendMessage()方法现在就像这样：

```
/** 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);
}
```

现在你需要去创建一个DisplayMessageActivity支持程序能够执行起来

# 创建第二个Activity

使用Eclipse创建新的Activity：

1. 在工具栏点击新建。
2. 在弹出窗口打开安卓文件夹，选择安卓活动然后点击下一步。
3. 选择**BlankActivity**，然后点击下一步
4. 填写Activity详细信息：
  - **Project:** MyFirstApp
  - **Activity Name:** DisplayMessageActivity
  - **Layout Name:** activity\_display\_message
  - **Fragment Layout Name:** fragment\_display\_message
  - **Title:** My Message
  - **Hierarchial Parent:** com.example.myfirstapp.MainActivity
  - **Navigation Type:** 无

单击**Finish**。

如果使用的是不同的IDE或者命令行工具，需要在项目的src/目录创建一个名为DisplayMessageActivity.java，与MainActivity.java同目录的文件。

打开DisplayMessageActivity.java文件，如果该文件用Eclipse创建，那么：

- 此类已经包含了所需onCreate()的默认实现，稍后需要更新此实现方法。
- 另外还有一个onCreateOptionsMenu()实现方式，由于应用程序并不需要所以可以直接删除。
- 还有onOptionsItemSelected()实现方式，它可以处理操作栏上拉操作。

还有一个PlaceholderFragment，在本activity中不需要此类。

Fragments把应用程序的功能和用户界面分解成可以复用的模块。想了解更Fragments信息，请参阅[Fragments API Guide](#)，此处暂不使用Fragment。

DisplayMessageActivity类现在应该是这样的：

```
public class DisplayMessageActivity extends ActionBarActivity {

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

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                .add(R.id.container, new PlaceholderFragment()).commit();
        }
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```
// Handle action bar item clicks here. The action bar will
// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml
int id = item.getItemId();
if (id == R.id.action_settings) {
    return true;
}
return super.onOptionsItemSelected(item);
}

/**
 * A placeholder fragment containing a simple view.
 */
public static class PlaceholderFragment extends Fragment {

    public PlaceholderFragment() { }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
                           savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main,
                                         container, false);
        return rootView;
    }
}
```

如果使用IDE而不是Eclipse,参照用上述代码来更新你的 DisplayMessageActivity。

Activity所有子类都必须实现 onCreate()方法。创建活动新实例时系统会调用该方式，此时必须用 setContentView()来定义Activity布局，以对Activity进行初始化。

## 添加标题字符串

如果你使用Eclipse开发，则可以跳过本部分，因为模板提供了新活动的标题字符串。如果你使用的IDE不是Eclipse，需要把新Activity的标题添加到strings.xml文件：

```
<resources>
    ...
    <string name="title_activity_display_message">My Message</string>
</resources>
```

## 将Activity加入manifest(清单)文件

所有Activity必须使用 `activity`元素在AndroidManifest.xml清单文件声明，如果使用 Eclipse创建Activity，则会自动在AndroidManifest.xml配置好对应`activity`元素，其它IDE需要手动配置，最终结果应该看起来这样：

```
<application ... >
    ...
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivit
        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`属性声明了在应用程序中该Activity逻辑层面的父类 Activity的名称。系统使用此值来实现默认导航操作，比如在安卓4.1（API级别16）或者更高版本。使用Support Library，如上所示的`meta-data`元素可以为安卓旧版本提供相同功能。

如果正在使用Eclipse开发，现在可以运行应用程序了。点击发送按钮启动第二个Activity，但它采用的是模板提供的"Hello world"布局，稍后你可以自己更新该布局。因此使用其它IDE也不用担心，因为应用程序尚未编译。

## 获取Intent

每一个被Intent调用的Activity，不管用户将它导航到哪，你都可以在启动的Activity中通过getIntent()方法得到Intent以及Intent包含的数据。在DisplayMessageActivity类的onCreate()方法中，得到intent以及MyFirstActivity提供的附加信息：

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

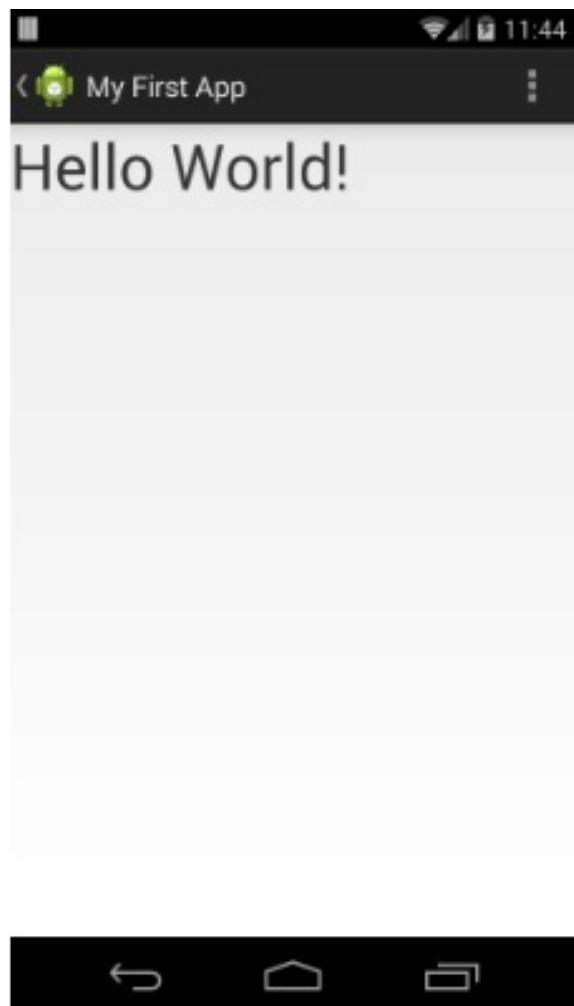
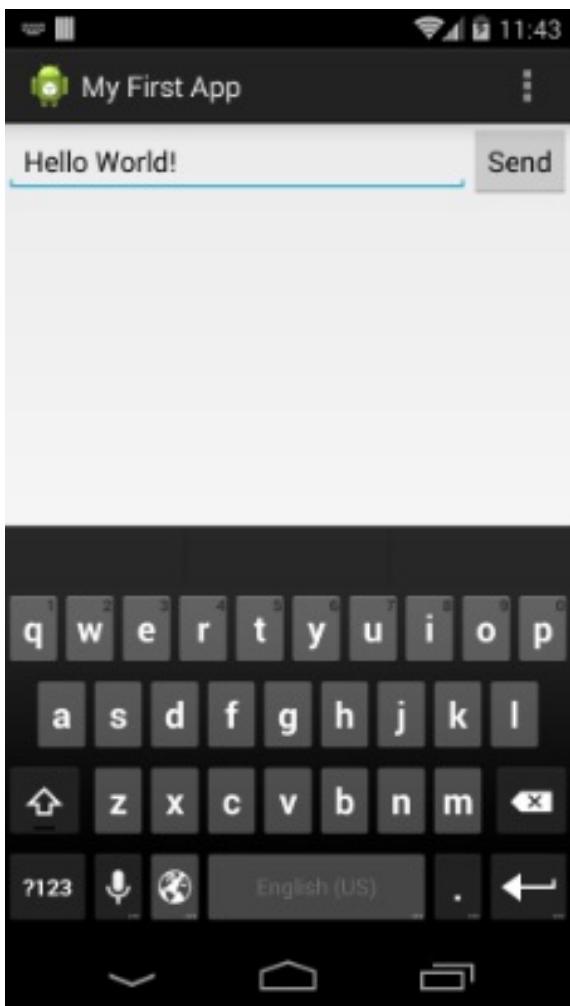
## 显示信息

在屏幕上显示信息，创建一个TextView部件，并且使用setText()设置它的值，然后通过setContentView()方法将TextView作为root(根)视图添加到Activity的布局。

DisplayMessageActivity完整的onCreate()方法现在看起来如下：

```
@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);  
}
```

现在你可以运行app，在文本中输入信息，点击Send(发送)按钮，ok，现在就可以在第二Activity上看到发送过来信息了。如图：



到此为止，你已经创建好你的第一个Android应用了！

编写: [Vincent 4J](#) - 校对:

原文: <http://developer.android.com/training/basics/actionbar/index.html>

# 添加 ActionBar

ActionBar是最重要的设计元素之一，你可以在activity中实现它。它提供了多种UI特性，可以让你的app相比于其他Android app比较一致，为用户所熟悉。核心的功能包括：

- 一个专门空间，给你的app一个个性，并且指示出用户的位置
- 以一种可预见的方式接入重要的操作（比如检索）
- 支持导航和视图切换（通过页签和下拉列表）



本章培训课程对action bar的基本知识提供了一个快速指南。关于action bar的更多特性，请查看[Action Bar](#)指南。

# 课程

- [建立ActionBar](#)

学习如何为你的 activity 添加一个基本的 action bar，不仅支持 Android 3.0 和更高，同时也支持不低于 Android 2.1（通过使用 Andriod Support 库）。

- [添加Action按钮](#)

学习如何在 action bar 中添加和响应用户操作。

- [ActionBar的风格化](#)

学习如何自定义你的 action bar 的外观

- [ActionBar覆盖叠加](#)

学习如何在布局上面叠加 action bar，允许 action bar 隐藏时无缝过渡。

编写: [Vincent 4J](#) - 校对:

原文: <http://developer.android.com/training/basics/actionbar/setting-up.html>

# 建立 Action Bar

Action bar 最基本的形式，就是为 activity 显示标题，并且在标题左边显示一个 app icon。即使在这样简单的形式下，对于所有的 activity 来说，action bar 对告知用户他们当前所处的位置十分有用，并为你的 app 保持了一致性。

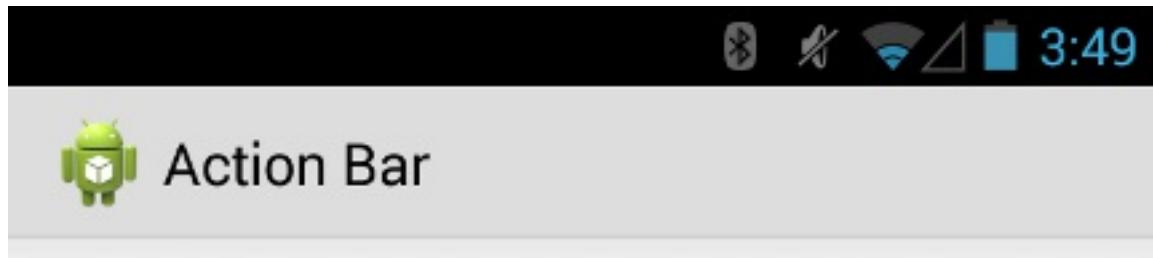


图 1.

一个有 app icon 和 activity 标题的 action bar

设置一个基本的 action bar，需要你的 app 使用一个 action bar 可用的 activity 主题。如何声明这样的主题取决于你的 app 支持的 Android 最低版本。所以本课程根据你的 app 支持的 Android 最低版本分为两部分。

## 仅支持 Android 3.0 及以上版本

从 Android 3.0(API level 11) 开始，所有使用 Theme.Holo 主题（或者它的子类）的所有 activity 都包含 action bar，当 `targetSdkVersion` 或 `minSdkVersion` 属性被设置成“11”或更大时，它是默认主题。

所以，为你的 activity 添加 action bar，只需简单地设置属性为 11 或者更大。例如：

```
<manifest ... >
    <uses-sdk android:minSdkVersion="11" ... />
    ...
</manifest>
```

注释：如果创建一个自定义主题，需确保它使用一个 Theme.Holo 主题作为父辈。  
详情请查看 [Action bar 风格化](#)

到此，你的 app 使用了 Theme.Holo 主题，并且所有的 activity 都显示 action bar。

## 支持 Android 2.1 及以上版本

当 app 运行在 Andriod 3.0 以下版本（不低于 Android 2.1）时，如果要添加 action bar，需要加载 Android Support 库。

通过阅读 安装 Support 库 文档和安装 v7 appcompat 库 来开始（下载完库包之后，按照 添加资源库 的说明来添加）。

一旦 Support 库集成到你的 app 工程之中：

1、更新 activity，以便于它继承于 ActionBarActivity。例如：

```
public class MainActivity extends ActionBarActivity { ... }
```

2、在 manifest 文件中，更新 <application> 元素或者单一的 <activity> 元素来使用一个 Theme.AppCompat 主题。例如：

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

注释：如果创建一个自定义主题，需确保它使用一个 Theme.AppCompat 主题作为父辈。详情请查看 [Action bar 风格化](#)

当 app 运行在 Android 2.1(API level 7) 或者以上时，activity 将包含 action bar。

切记，在 manifest 中正确地设置 app 支持的 API level：

```
<manifest ... >
  <uses-sdk android:minSdkVersion="7" android:targetSdkVersion=
  ...
</manifest>
```

编写: [Vincent 4J](#) - 校对:

原文: <http://developer.android.com/training/basics/actionbar/adding-buttons.html>

# 添加 Action 按钮

Action bar 允许你为当前上下文中最重要的操作添加按钮。那些直接出现在 action bar 中的 icon 和/或文本被称作操作按钮。不匹配的或不足够重要的操作被隐藏在 action overflow 中。



图 1. 一个有检索操作按钮和 action overflow 的 action bar，在 action overflow 里能展现额外的操作

## 在 XML 中指定操作

所有的操作按钮和 action overflow 中其他可用的条目都被定义在 [菜单资源](#) 的 XML 文件中。通过在项目的 res/menu 目录中新增一个 XML 文件来为 action bar 添加操作。

为你想添加到 action bar 中的每个条目添加一个 <item> 元素。例如：

res/menu/main\_activity\_actions.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- Search, should appear as action button -->
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"
          android:showAsAction="ifRoom" />
    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:showAsAction="never" />
</menu>
```

上述声明是这样的，当 action bar 有可用空间时，检索操作将作为一个操作按钮来显示，但设置操作将一直只在 action overflow 中显示。（默认情况下，所有的操作都显示在 action overflow 中，但为每一个操作指明设计意图是很好的做法。）

icon 属性要求每张图片提供一个 resource ID。在 @drawable/ 之后的名字必须是存储在项目目录 res/drawable/ 下图片的名字。例如：ic\_action\_search.png 对应 "@drawable/ic\_action\_search"。同样地，title 属性使用通过 XML 文件定义在项目目录 res/values/ 中的一个 string resource，详情请参见 [创建一个简单的 UI](#)。

注释：当在创建 icon 和其他 bitmap 图片时，你得为优化不同屏幕密度下的显示效果提供多个版本，这一点很重要。在 [支持不同屏幕](#) 课程中将会更详细地讨论。

如果你为了兼容 Android 2.1 以下版本使用了 Support 库，在 android 命名空间下 showAsAction 属性是不可用的。Support 库会提供替代它的属性，你必须声明自己的 XML 命名空间，并且使用该命名空间作为属性前缀。（一个自定义 XML 命名空间需要以你的 app 名称为基础，但是可以取任何你想要的名称，它的作用域仅仅在你声明的文件之内。）例如：

res/menu/main\_activity\_actions.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:yourapp="http://schemas.android.com/apk/res-auto" >
    <!-- Search, should appear as action button -->
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"
          yourapp:showAsAction="ifRoom" />
    ...

```

</menu>

## 为 Action Bar 添加操作

为 action bar 布局菜单条目，是通过在 activity 中实现 [onCreateOptionsMenu\(\)](#) 回调方法来 inflate 菜单资源从而获取 [Menu](#) 对象。例如：

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu items for use in the action bar  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.main_activity_actions, menu);  
    return super.onCreateOptionsMenu(menu);  
}
```

## 为操作按钮添加响应事件

当用户按下某一个操作按钮或者 action overflow 中的其他条目，系统将调用 activity 中 [onOptionsItemSelected\(\)](#) 回调方法。在该方法的实现里面调用 [getItemId\(\)](#) 获取 [MenuItem](#) 来判断哪个条目被按下——返回的 ID 会匹配你声明对应的 <item> 元素中 <android:id> 属性的值。

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle presses on the action bar items  
    switch (item.getItemId()) {  
        case R.id.action_search:  
            openSearch();  
            return true;  
        case R.id.action_settings:  
            openSettings();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

## 为下级 Activity 添加向上按钮

在不是主要入口的其他所有屏中（activity 不位于主屏时），需要在 action bar 中为用户提供一个导航到逻辑父屏的向上按钮。

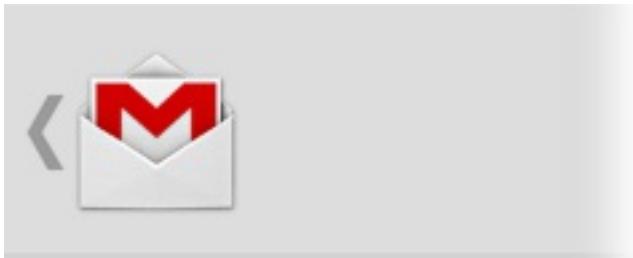


图 2. Gmail 中向上按钮

当运行在 Android 4.1(API level 16) 或更高版本，或者使用 Support 库中的 [ActionBarActivity](#) 时，实现向上导航需要你在 manifest 文件中声明父 activity，同时在 action bar 中设置向上按钮可用。

如何在 manifest 中声明一个 activity 的父辈，例如：

```
<application ... >
    ...
    <!-- The main/home activity (it has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivit"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity"
        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

然后，通过调用 [setDisplayHomeAsUpEnabled\(\)](#) 来把 app icon 设置成可用的向上按钮：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_displaymessage);

    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    // If your minSdkVersion is 11 or higher, instead use:
    // getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

由于系统已经知道 MainActivity 是 DisplayMessageActivity 的父 activity，当用户按下向上按钮时，系统会导航到恰当的父 activity —— 你不需要去处理向上按钮的事件。

更多关于向上导航的信息，请见 [提供向上导航](#)。

编写: [Vincent 4J](#) - 校对:

原文: <http://developer.android.com/training/basics/actionbar/styling.html>

# Action Bar 风格化

Action bar 为用户提供一种熟悉可预测的方式来展示操作和导航，但是这并不意味着你的 app 要看起来和其他 app 一样。如果你想将 action bar 的风格设计的合乎你产品的定位，你只需简单地使用 Android 的 [式样和主题](#) 资源。

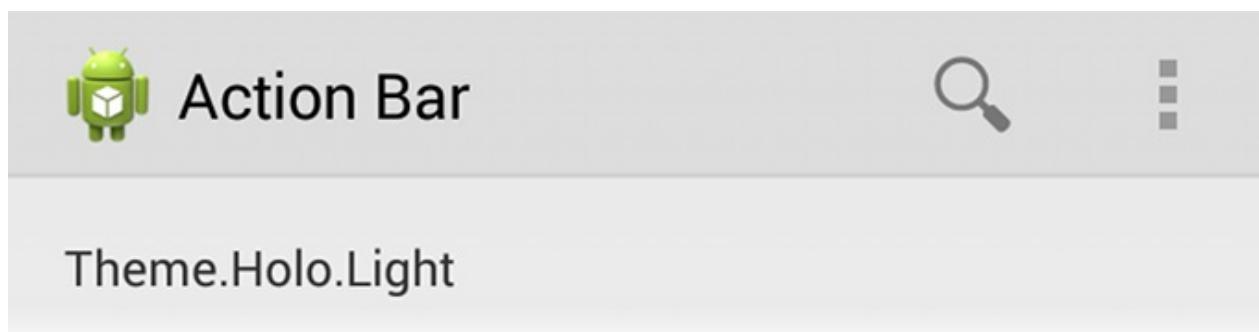
Android 包括一些部分内置的 activity 主题，这些主题中包含“暗”或“淡”的 action bar 式样。你也可以扩展这些主题，以便于更好的为你的 action bar 自定义外观。

注释：如果你为 action bar 使用了 Support 库的 API，那你必须使用（或重写）[Theme.AppCompat](#) 家族式样（甚至[Theme.Holo](#) 家族，在 API level 11 或更高版本中可用）。如此一来，你声明的每一个式样属性都必须被声明两次：一次使用平台的式样属性（`android:` 属性），另一次使用 Support 库中的式样属性（`appcompat.R.attr` 属性——这些属性的上下文其实就是你的 app）。更多细节请查看下面的示例。

## 使用一个 Android 主题

Android 包含两个基本的 activity 主题，这两个主题决定了 action bar 的颜色：

- [Theme.Holo](#)，一个“暗”的主题
- [Theme.Holo.Light](#)，一个“淡”的主题

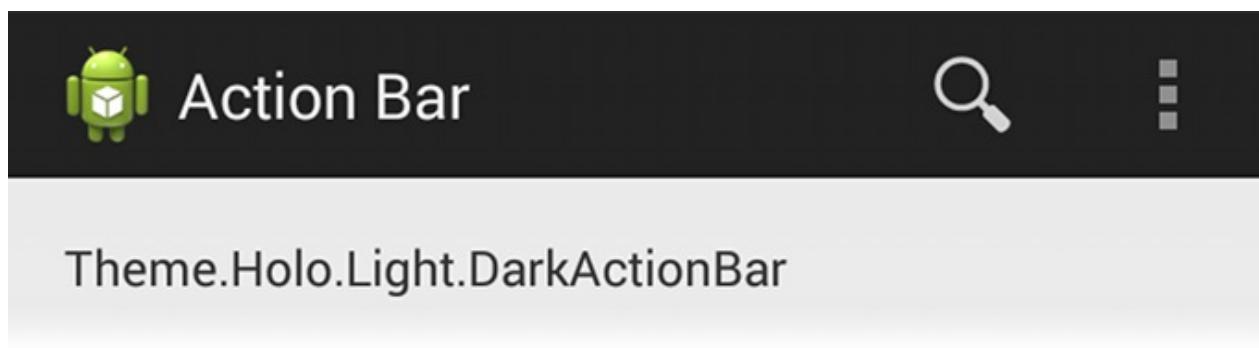


这些主题即可以被应用到 app 全局，又可以为单一的 activity 通过在 manifest 文件中设置 [application](#) 元素或 [activity](#) 元素的 android:theme 属性。

例如：

```
<application android:theme="@android:style/Theme.Holo.Light" ... />
```

你可以通过声明 activity 的主题为 [Theme.Holo.Light.DarkActionBar](#) 来达到如下效果：action bar 为暗色，其他部分为淡色。



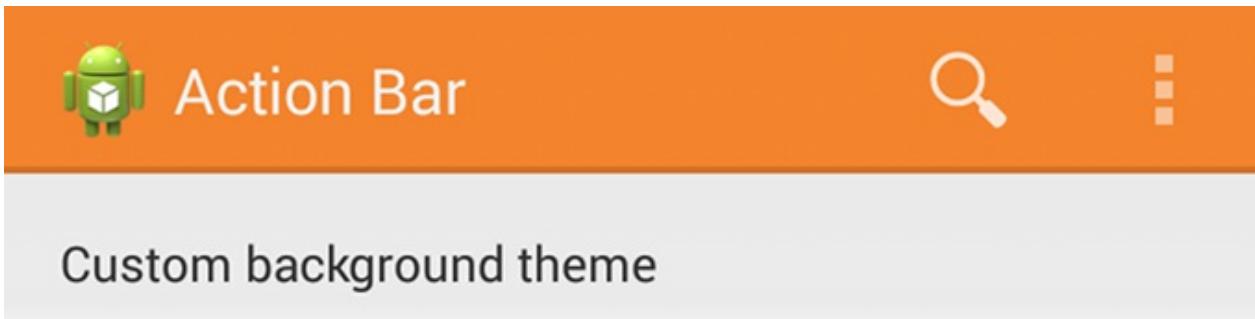
当使用 Support 库时，必须使用 [Theme.AppCompat](#) 主题替代：

- [Theme.AppCompat](#)，一个“暗”的主题
- [Theme.AppCompat.Light](#)，一个“淡”的主题
- [Theme.AppCompat.Light.DarkActionBar](#)，一个带有“暗”action bar 的“淡”主题

一定要确保你使用的 action bar icon 的颜色与 action bar 本身的颜色有差异。为了能帮助到你, [Action Bar Icon Pack](#) 为 Holo “暗”和“淡”的 action bar 提供了标准的 action icon。

# 自定义背景

为 activity 创建一个自定义主题，通过重写 `ActionBarStyle` 属性来改变 action bar 的背景。`ActionBarStyle` 属性指向另一个式样；在该式样里，通过指定一个 drawable 资源来重写 `background` 属性。



如果你的 app 使用了 `navigation tabs` 或 `split action bar`，你也可以通过分别设置 `backgroundStacked` 和 `backgroundSplit` 属性来为这些条指定背景。

注意：声明一个合适的父主题，进而你的自定义主题和式样可以继承父主题的式样，这点很重要。如果没有父式样，你的 action bar 将会失去很多式样属性，除非你自己显式的对他们进行声明。

## 仅支持 Android 3.0 和更高

当仅支持 Android 3.0 和更高版本时，你可以像这样定义 action bar 的背景：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@android:style/Theme.Holo.Light.DarkActionBar">
        <item name="android: actionBarStyle">@style/MyActionBar</item>
    </style>

    <!-- ActionBar styles -->
    <style name="MyActionBar"
        parent="@android:style/Widget.Holo.Light.ActionBar.Solid">
        <item name="android: background">@drawable/actionbar_background</item>
    </style>
</resources>
```

然后，将你的主题应用到你的 app 全局或单个的 activity 之中：

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

## 支持 Android 2.1 和更高

当使用 Support 库时，上面同样的主题必须被替代成如下：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.AppCompat.Light.DarkActionBar">
        <item name="android: actionBarStyle">@style/MyActionBar</it

        <!-- Support library compatibility -->
        <item name="actionBarStyle">@style/MyActionBar</item>
    </style>

    <!-- ActionBar styles -->
    <style name="MyActionBar"
        parent="@style/Widget.AppCompat.Light.ActionBar.Solid.I
        <item name="android: background">@drawable/actionbar_backgr

        <!-- Support library compatibility -->
        <item name="background">@drawable/actionbar_background</it
    </style>
</resources>
```

然后，将你的主题应该到你的 app 全局或单个的 activity 之中：

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

# 自定义文本颜色

修改 action bar 中的文本颜色，你需要分别重写每个元素的属性：

- Action bar 的标题：创建一种自定义式样，并指定 `textColor` 属性；同时，在你的自定义  [actionBarStyle](#) 中为 `titleTextStyle` 属性指定为刚才的自定义式样。

注释：被应用到 `titleTextStyle` 的自定义式样应该使用 [TextAppearance.Holo.Widget.ActionBar.Title](#) 作为父式样。

- Action bar 的页签：在你的 activity 主题中重写  [actionBarTabTextStyle](#)
- Action 按钮：在你的 activity 主题中重写  [actionBarTextColor](#)

## 仅支持 Android 3.0 和更高

当仅支持 Android 3.0 和更高时，你的式样 XML 文件应该是这样的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.Holo">
        <item name="android:actionBarStyle">@style/MyActionBar</item>
        <item name="android:actionBarTabTextStyle">@style/MyActionBarTabText</item>
        <item name="android:actionMenuTextColor">@color/actionbar_text</item>
    </style>

    <!-- ActionBar styles -->
    <style name="MyActionBar"
        parent="@style/Widget.Holo.ActionBar">
        <item name="android:titleTextStyle">@style/MyActionBarTitleText</item>
    </style>

    <!-- ActionBar title text -->
    <style name="MyActionBarTitleText"
        parent="@style/TextAppearance.Holo.Widget.ActionBar.Title">
        <item name="android:textColor">@color/actionbar_text</item>
    </style>

    <!-- ActionBar tabs text styles -->
    <style name="MyActionBarTabText"
        parent="@style/Widget.Holo.ActionBar.TabText">
        <item name="android:textColor">@color/actionbar_text</item>
    </style>
</resources>
```

## 支持 Android 2.1 和更高

当使用 Support 库时，你的式样 XML 文件应该是这样的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.AppCompat">
        <item name="android: actionBarStyle">@style/MyActionBar</item>
        <item name="android: actionBarTabTextStyle">@style/MyAction
        <item name="android: actionMenuTextColor">@color/actionbar_</item>

        <!-- Support library compatibility -->
        <item name="actionBarStyle">@style/MyActionBar</item>
        <item name="actionBarTabTextStyle">@style/MyActionBarTabTe
        <item name="actionMenuTextColor">@color/actionbar_text</it
    </style>

    <!-- ActionBar styles -->
    <style name="MyActionBar"
        parent="@style/Widget.AppCompat.ActionBar">
        <item name="android: titleTextStyle">@style/MyActionBarTitl

        <!-- Support library compatibility -->
        <item name="titleTextStyle">@style/MyActionBarTitleText</i
    </style>

    <!-- ActionBar title text -->
    <style name="MyActionBarTitleText"
        parent="@style/TextAppearance.AppCompat.Widget.ActionBar">
        <item name="android: textColor">@color/actionbar_text</item>
        <!-- The textColor property is backward compatible with th
    </style>

    <!-- ActionBar tabs text -->
    <style name="MyActionBarTabText"
        parent="@style/Widget.AppCompat.ActionBar.TabText">
        <item name="android: textColor">@color/actionbar_text</item>
        <!-- The textColor property is backward compatible with th
    </style>
</resources>
```

# 自定义 Tab Indicator

为 activity 创建一个自定义主题，通过重写 `ActionBarTabStyle` 属性来改变 `navigation tabs` 使用的指示器。`ActionBarTabStyle` 属性指向另一个式样资源；在该式样资源里，通过指定一个状态列表 drawable 来重写 `background` 属性。

注释：一个状态列表 drawable 是重要的，以便通过不同的背景来指出当前选择的 tab 与其他 tab 的区别。更多关于如何创建一个 drawable 资源来处理多个按钮状态，请阅读 [State List](#) 文档。

例如，这是一个状态列表 drawable，为一个 action bar tab 的多种不同状态分别指定背景图片：

res/drawable/actionbar\_tab\_indicator.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- STATES WHEN BUTTON IS NOT PRESSED -->

        <!-- Non focused states -->
        <item android:state_focused="false" android:state_selected="false"
              android:state_pressed="false"
              android:drawable="@drawable/tab_unselected" />
        <item android:state_focused="false" android:state_selected="true"
              android:state_pressed="false"
              android:drawable="@drawable/tab_selected" />

        <!-- Focused states (such as when focused with a d-pad or mouse) -->
        <item android:state_focused="true" android:state_selected="false"
              android:state_pressed="false"
              android:drawable="@drawable/tab_unselected_focused" />
        <item android:state_focused="true" android:state_selected="true"
              android:state_pressed="false"
              android:drawable="@drawable/tab_selected_focused" />

    <!-- STATES WHEN BUTTON IS PRESSED -->

        <!-- Non focused states -->
        <item android:state_focused="false" android:state_selected="false"
              android:state_pressed="true"
              android:drawable="@drawable/tab_unselected_pressed" />
        <item android:state_focused="false" android:state_selected="true"
              android:state_pressed="true"
              android:drawable="@drawable/tab_selected_pressed" />

        <!-- Focused states (such as when focused with a d-pad or mouse) -->
        <item android:state_focused="true" android:state_selected="false"
              android:state_pressed="true"
              android:drawable="@drawable/tab_unselected_pressed" />
```

```
        android:drawable="@drawable/tab_unselected_pressed" />
    <item android:state_focused="true" android:state_selected="true"
          android:state_pressed="true"
          android:drawable="@drawable/tab_selected_pressed" />
</selector>
```

## 仅支持 Android 3.0 和更高

当仅支持 Android 3.0 和更高时，你的式样 XML 文件应该是这样的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
          parent="@style/Theme.Holo">
        <item name="android: actionBarTabStyle">@style/MyActionBarT
    </style>

    <!-- ActionBar tabs styles -->
    <style name="MyActionBarTabs"
          parent="@style/Widget.Holo.ActionBar.TabView">
        <!-- tab indicator -->
        <item name="android:background">@drawable/actionbar_tab_in
    </style>
</resources>
```

## 支持 Android 2.1 和更高

当使用 Support 库时，你的式样 XML 文件应该是这样的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
          parent="@style/Theme.AppCompat">
        <item name="android: actionBarTabStyle">@style/MyActionBarT

        <!-- Support library compatibility -->
        <item name="actionBarTabStyle">@style/MyActionBarTabs</ite
    </style>

    <!-- ActionBar tabs styles -->
    <style name="MyActionBarTabs"
          parent="@style/Widget.AppCompat.ActionBar.TabView">
        <!-- tab indicator -->
        <item name="android:background">@drawable/actionbar_tab_in

        <!-- Support library compatibility -->
```

```
<item name="background">@drawable/ActionBar_tab_indicator</item>
</style>
</resources>
```

## 更多资源

- 关于 action bar 的更多式样属性, 请查看 [Action Bar](#) 指南
- 学习更多式样的工作机制, 请查看 [式样和主题](#) 指南
- 全面的 action bar 式样, 请尝试 [Android Action Bar 式样生成器](#)

编写: [Vincent 4J](#) - 校对:

原文: <http://developer.android.com/training/basics/actionbar/overlaying.html>

# Action Bar 覆盖叠加

默认情况下，action bar 显示在 activity 窗口的顶部，会稍微地减少其他布局的有效空间。如果在用户交互过程中你要隐藏和显示 action bar，可以通过调用 [ActionBar](#) 中的 `hide()` 和 `show()` 来实现。但是，这将会导致 activity 基于新尺寸重现计算和重新绘制布局。

为了避免在 action bar 隐藏和显示过程中调整布局，可以为 action bar 启用叠加模式。在叠加模式下，所有可用的空间都会被用来布局，并且 action bar 会叠加在布局之上。这样布局顶部就会被遮挡，但当 action bar 隐藏或显示时，系统不再需要调整布局而是无缝过渡。

提示：如果你希望 action bar 下面的布局部分可见，可以创建一个背景部分透明的自定义式样的 action bar，如图 1 所示。如何定义 action bar 的背景，请查看 [ActionBar 风格化](#)。

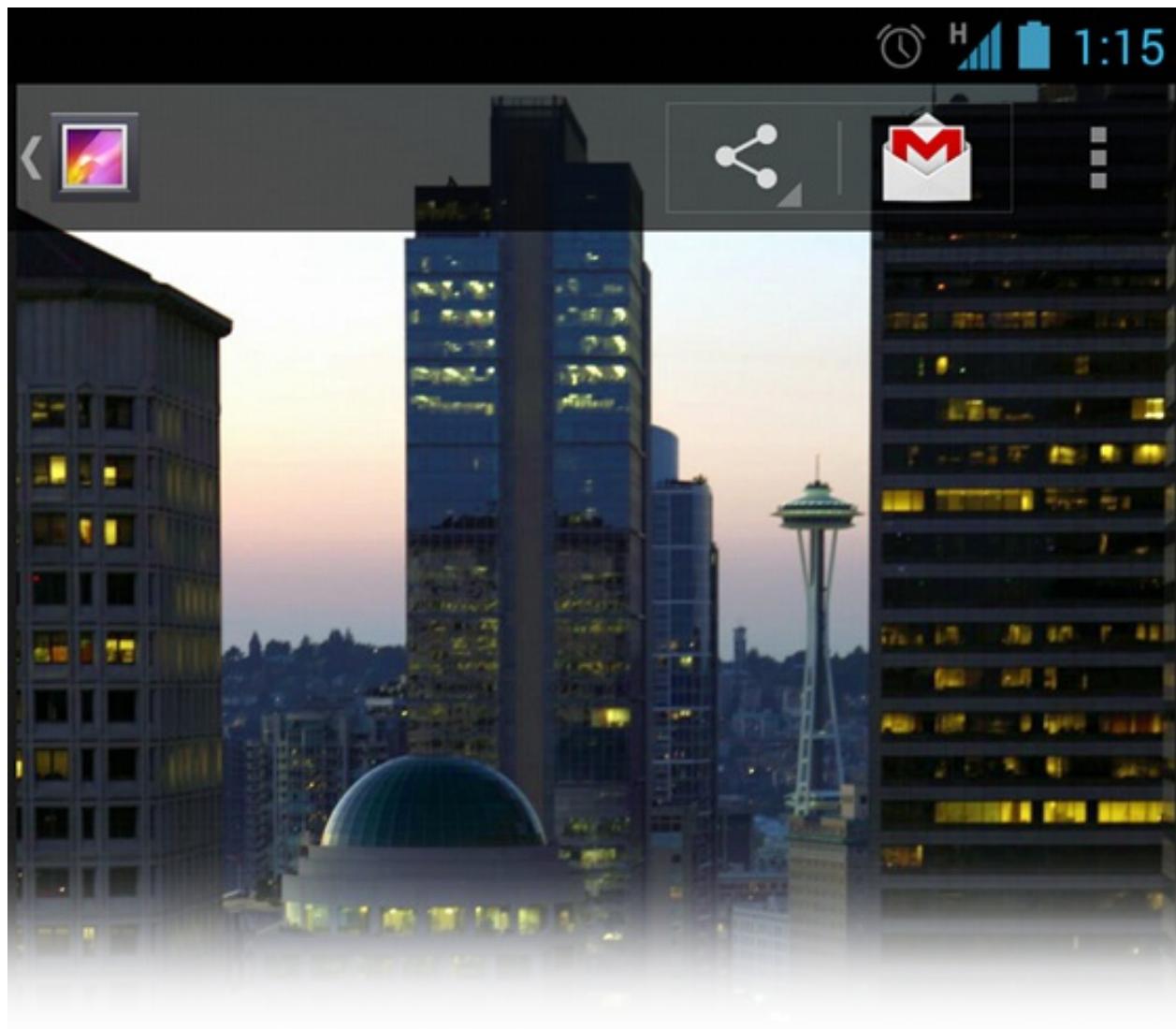


图 1. 叠加模式下的 gallery action bar

## 启用叠加模式

要为 action bar 启用叠加模式，需要自定义一个主题，该主题继承于已经存在的 action bar 主题，并设置 android:windowActionBarOverlay 属性的值为 true。

### 仅支持 Android 3.0 和以上

如果 [minSdkVersion](#) 为 11 或更高，自定义主题必须继承 [Theme.Holo](#) 主题（或者它的子主题）。例如：

```
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@android:style/Theme.Holo">
        <item name="android:windowActionBarOverlay">true</item>
    </style>
</resources>
```

### 支持 Android 2.1 和更高

如果为了兼容运行在 Android 3.0 以下版本的设备而使用了 Support 库，自定义主题必须继承 [Theme.AppCompat](#) 主题（或者它的子主题）。例如：

```
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActionBarTheme"
        parent="@android:style/Theme.AppCompat">
        <item name="android:windowActionBarOverlay">true</item>

        <!-- Support library compatibility -->
        <item name="windowActionBarOverlay">true</item>
    </style>
</resources>
```

请注意，这主题包含两种不同的 windowActionBarOverlay 式样定义：一个带 android: 前缀，另一个不带。带前缀的适用于包含该式样的 Android 版本，不带前缀的适用于通过从 Support 库中读取式样的旧版本。

## 指定布局的顶部边距

当 action bar 启用叠加模式时，它可能会遮挡住本应保持可见状态的布局。为了确保这些布局始终位于 action bar 下部，可以使用  [actionBarSize](#)  属性来指定顶部外边距或顶部内边距的高度来到达。例如：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingTop="?android:attr/actionBarSize">  
    ...  
</RelativeLayout>
```

如果在 action bar 中使用 Support 库，需要移除 android: 前缀。例如：

```
<!-- Support library compatibility -->  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingTop="?attr/actionBarSize">  
    ...  
</RelativeLayout>
```

在这种情况下，不带前缀的 ?attr/actionBarSize 适用于 Android 3.0 和更高的所有版本。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/basics/supporting-devices/index.html>

# 兼容不同的设备

全世界的Android设备有着各种各样的大小和尺寸。而通过各种各样的设备类型，能使你通过你的app接触到广大的用户群体。为了能在各种Android平台上使用，你的app需要兼容各种不同的设备类型。某些重要的变动你需要考虑，比如语言，屏幕尺寸，和Android的版本。

本课程会教你如何使用基础的平台功能，利用替代资源和其他功能，使你的app仅用一个app程序包(APK)，就能向用Android兼容设备的用户提供最优的用户体验。

# 课程

- [适配不同的语言](#)

学习如何使用字符串替代资源实现支持多国语言。

- [适配不同的屏幕](#)

学习如何根据不同尺寸分辨率的屏幕来优化用户体验。

- [适配不同的系统版本](#)

学习如何在使用新的用户编程接口(API)时向下兼容旧版本Android。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/basics/supporting-devices/languages.html>

# 适配不同的语言

把UI中的字符串存储在外部文件，通过代码提取，总是一种很好的做法。Android可以通过工程中的资源目录轻松实现这一功能。

如果你使用Android SDK Tools(详见[Creating an Android Project](#))来创建工程，则在工程的根目录会创建一个res/的目录，目录中包含所有资源类型的子目录。其中包含工程的默认文件比如res/values/strings.xml，用来保存你的字符串值。

## 创建区域设置目录和字符串文件

为了支持多国语言，在res/中创建一个额外的values目录以连字符和ISO国家代码结尾命名，比如values-es/是包含简单的区域资源，语言代码为"es"的区域设置目录。Android会在运行时根据设备的区域设置，加载相应的资源。

若你决定支持某种语言，则需要创建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
```

添加不同区域语言的字符串值到相应的文件。

在运行时，Android系统会根据用户设备当前的区域设置，使用相应的字符串资源。

例如下面列举了几个不同语言对应不同的字符串资源文件。

英语(默认区域语言)，/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>
```

法语，/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>
```

**Note:** 你可以在任何资源类型中使用区域修饰词(或者任何配置修饰符)，比如给

bitmap提供本地化的版本，更多信息见[Localization](#)。

## 使用字符串资源

你可以在你的源代码和其他XML文件中，通过<string>元素的name属性来引用你的字符串资源。

在你的源代码中你可以通过R.string.<string\_name>语法来引用一个字符串资源，很多方法都可以通过这种方式来接受字符串。

例如：

```
// 从你的 app's 资源中获取一个字符串资源
String hello = getResources().getString(R.string.hello_world);

// 或者提供给一个需要字符串作为参数的方法
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

在其他XML文件中，每当XML属性要接受一个字符串值时，你都可以通过@string/<string\_name>语法来引用字符串资源。

例如：

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

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/basics/supporting-devices/screens.html>

# 适配不同的屏幕

Android将设备屏幕归类为两种常规属性：尺寸和分辨率。你应该想到你的app会被安装在各种屏幕尺寸和分辨率的设备中。这样，你的app就应该包含一些可选资源，针对不同的屏幕尺寸和分辨率，来优化你的app外观。

- 有4种普遍尺寸：小(small)，普通(normal)，大(large)，超大(xlarge)
- 4种普遍分辨率：低精度(ldpi)，中精度(mdpi)，高精度(hdpi)，超高精度(xhdpi)

声明针对不同屏幕所用的layout和bitmap，你必须把这些可选资源放置在独立的目录中，与你适配不同语言时的做法类似。

同样要注意屏幕的方向(横向或纵向)也是一种需要考虑的屏幕尺寸变化，所以许多app会修改layout，来针对不同的屏幕方向优化用户体验。

## 创建不同的layout

为了针对不同的屏幕去优化用户体验，你需要对每一种将要支持的屏幕尺寸，创建唯一的XML文件。每一种layout需要保存在相应的资源目录中，目录以-`<screen_size>`为后缀命名。例如，对大尺寸屏幕(large screens)，一个唯一的layout文件应该保存在`res/layout-large/`中。

**Note:**为了匹配合适的屏幕尺寸Android会自动地测量你的layout文件。所以你不需要因不同的屏幕尺寸去担心UI元素的大小，而应该专注于layout结构对用户体验的影响。(比如关键视图相对于同级视图的尺寸或位置)

例如，这个工程包含一个默认layout和一个适配大屏幕的layout：

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

layout文件的名字必须完全一样，为了对相应的屏幕尺寸提供最优的UI，文件的内容不同。

按照惯例在你的app中简单引用：

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

系统会根据你的app所运行的设备屏幕尺寸，在与之对应的layout目录中加载layout。更多关于Android如何选择恰当资源的信息，详见[Providing Resources](#)。

另一个例子，这一个工程中有为适配横向屏幕的layout：

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

默认的，`layout/main.xml`文件用作纵向屏幕layout。

如果你想给横向屏幕提供一个特殊的layout，也适配于大屏幕，那么你需要使用`large`和`land`修饰符。

```
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
```

**Note:**Android 3.2和以上版本支持定义屏幕尺寸的高级方法，它允许你根据屏幕最小长度和宽度，为各种屏幕尺寸指定与密度无关的layout资源。这节课程不会涉及这一新技术，更多信息详见[Designing for Multiple Screens](#)。

## 创建不同的bitmap

你应该为4种普遍分辨率:低, 中, 高, 超高精度, 都提供相适配的bitmap资源。这能帮助你在所有屏幕分辨率中都能有良好的画质和效果。

要生成这些图像, 你应该从原始的矢量图像资源着手, 然后根据下列尺寸比例, 生成各种密度下的图像。

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

这意味着, 如果你针对超高密度的设备生成了一张200x200的图像, 同样的你应该对150x150高密度, 100x100中密度, 和75x75低密度的设备生成同样的资源。

然后, 将这些文件放入相应的drawable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, 当你引用@drawable/awesomeimage时系统会根据屏幕的分辨率选择恰当的bitmap。

**Note:** 低密度(ldpi)资源是非必要的, 当你提供了高精度assets, 系统会把高密度图像按比例缩小一半, 去适配低密度屏幕。

更多关于为app创建图标assets的贴士和指导, 详见[Iconography design](#)。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/basics/supporting-devices/platforms.html>

# 适配不同的系统版本

新的Android版本会为你的app提供更棒的APIs，但你的app仍应该支持旧版本的Android，直到更多的设备升级到新版本为止。这节课向你展示如何在利用新的APIs的同时仍支持旧版本Android。

[Platform Versions](#)的控制面板会定时更新，通过统计访问Google Play Store的设备数量，来显示运行每个版本的安卓设备的分布。一般情况下，在更新你的app至最新Android版本时，最好先保证你的新版app可以支持90%的设备使用。

**Tip:**为了能在几个Android版本中都能提供最好的特性和功能，你应该在你的app中使用[Android Support Library](#)，它能使你的app能在旧平台上使用最近的几个平台的APIs。

## 指定最小和目标API级别

[AndroidManifest.xml](#)文件中描述了你的app的细节，并且标明app支持哪些Android版本。具体来说，[`<uses-sdk>`](#)元素中的`minSdkVersion`和`targetSdkVersion`属性，标明在设计和测试app时，最低兼容API的级别和最高适用的API级别。

例如：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="11">
        ...
    </uses-sdk>
</manifest>
```

随着新版本Android的发布，一些风格和行为可能会改变，为了能使你的app能利用这些变化，而且能适配不同风格的用户的设备，你应该设置`targetSdkVersion`的值去匹配最新的可用Android版本。

## 在运行时检查系统版本

Android在[Build](#)常量类中提供了对每一个版本的唯一代号，在你的app中使用这些代号可以建立条件，保证依赖于高级别的API的代码，只会在这些API在当前系统中可用时，才会执行。

```
private void setUpActionBar() {  
    // 保证我们是运行在Honeycomb或者更高版本时，才使用ActionBar APIs  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {  
        ActionBar actionBar = getActionBar();  
        actionBar.setDisplayHomeAsUpEnabled(true);  
    }  
}
```

**Note:**当解析XML资源时，Android会忽略当前设备不支持的XML属性。所以你可以安全地使用较新版本的XML属性，而不需要担心旧版本Android遇到这些代码时会崩溃。例如如果你设置targetSdkVersion="11"，你的app会在Android 3.0或更高时默认包含[ActionBar](#)。然后添加menu items到action bar时，你需要在你的menu XML资源中设置android:showAsAction="ifRoom"。在跨版本的XML文件中这么做是安全的，因为旧版本的Android会简单地忽略showAsAction属性(就是这样，你并不需要用到res/menu-v11/中单独版本的文件)。

## 使用平台风格和主题

Android提供了用户体验主题，为app提供基础操作系统的外观和体验。这些主题可以在manifest文件中被应用于你的app中。通过使用内置的风格和主题，你的app自然地随着Android新版本的发布，自动适配最新的外观和体验。

使你的activity看起来像对话框：

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

使你的activity有一个透明背景：

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

应用在/res/values/styles.xml中定义的自定义主题：

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

使整个app应用一个主题(全部activities)在元素中添加android:theme属性：

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

更多关于创建和使用主题，详见[Styles and Themes](#)。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/basics/activity-lifecycle/index.html>

# 管理Activity的生命周期(Managing the Activity Lifecycle)

- 当用户进入，退出，回到你的App，在程序中的[Activity](#)实例都经历了生命周期中的不同状态。例如，当你的activity第一次启动的时候，它来到系统的前台，开始接受用户的焦点。在此期间，Android系统调用了一系列的生命周期中的方法。如果用户执行了启动另一个activity或者切换到另一个app的操作，系统又会调用一些生命周期中的方法。
- 在生命周期的回调方法里面，你可以声明当用户离开或者重新进入这个Activity所需要执行的操作。例如，如果你建立了一个streaming video player，在用户切换到另外一个app的时候，你应该暂停video并终止网络连接。当用户返回时，你可以重新建立网络连接并允许用户从同样的位置恢复播放。
- 这一章会介绍一些[Activity](#)中重要的生命周期回调方法，如何使用那些方法使得程序符合用户的期望且在activity不需要的时候不会导致系统资源的浪费。
- 完整的Demo示例：[ActivityLifecycle.zip](#)

# Lessons

这一章我们将学习下面4个课程:

- [启动与销毁Activity](#)

学习关于activity生命周期的基础知识，用户如何启动你的应用以及如何执行activity的创建。

- [暂停与恢复Activity](#)

学习activity暂停发生时，你应该做哪些事情。

- [停止与重启Activity](#)

学习用户离开你的activity与返回activity时会发生的事情。

- [重新创建Activity](#)

学习当你的activity被销毁时发生了什么事情以及在有必要时如何重建你的activity。

编写: [kesenhoo](#) - 校对:

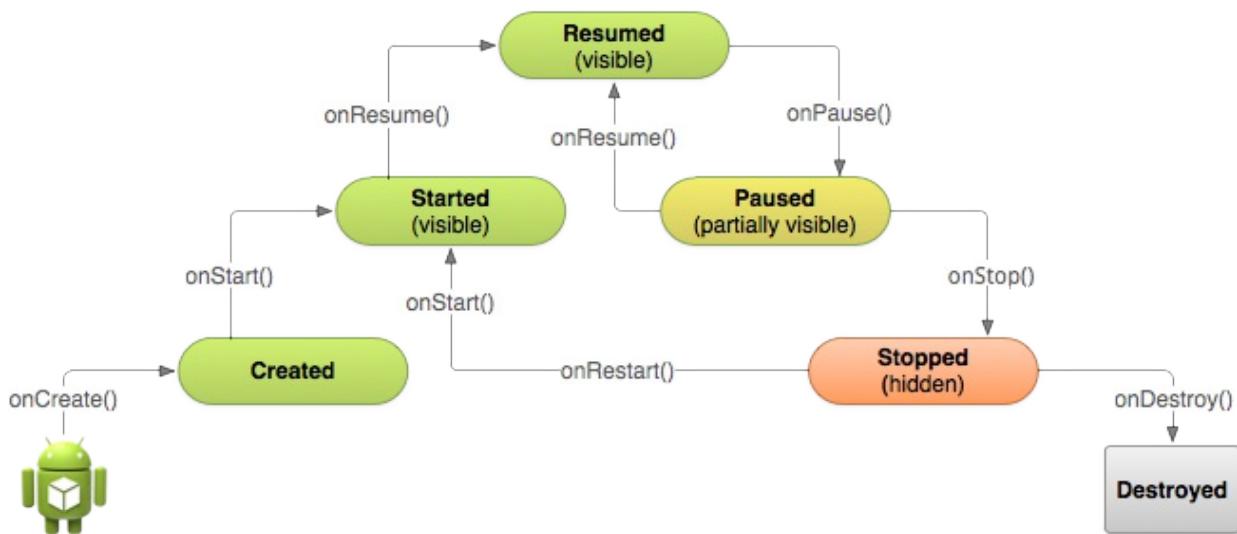
原文: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>

# 启动与销毁Activity

- 不像其他编程范式一样：程序从main()方法开始启动。Android系统根据生命周期的不同阶段唤起对应的回调函数来执行代码。系统存在启动与销毁一个activity的一套有序的回调函数。
- 这一个课会介绍那些生命周期中最重要的回调函数，并演示如何处理启动一个activity所涉及到的回调函数。

# Understand the Lifecycle Callbacks[理解生命周期的回调]

- 在一个activity的生命周期中，系统会像金字塔模型一样去调用一系列的生命周期回调方法。Activity生命周期的每一个阶段就像金字塔中的台阶。当系统创建了一个新的activity实例，每一个回调函数会向上一阶的移动activity状态。金字塔顶端意味着activity是跑在最前端的并且用户可以与它进行交互。
- 当用户开始离开这个activity,为了卸载这个activity，系统会调用其它方法来向下一阶移动activity状态。在某些情况下，activity会隐藏在金字塔下等待(例如当用户切换到其他app),这个时候activity可以重新回到顶端(如果用户回到这个activity)并且恢复用户离开时的状态。
- Figure 1. 下面这张图讲解了activity的生命周期：(显然，这个金字塔模型要比之前Dev Guide里面的生命周期图更加容易理解，更加形象)



- 根据你的activity的复杂度，你也许不需要实现所有的生命周期方法。然而，你需要知道每一个方法的功能并确保你的app能够像用户期望的那样执行。如何实现一个符合用户期待的app，你需要注意下面几点：
  - 当使用你的app的时候，不会因为有来电通话或者切换到其他app而导致程序crash。
  - 当用户没有激活某个组件的时候不要消耗宝贵的系统资源。
  - 当离开你的app并且一段时间后返回，不要丢失用户的使用进度。
  - 当设备发送屏幕旋转的时候，不会crash或者丢失用户的使用进度。
- 在下面的课程中会介绍上图所示的几个生命状态。然而，其中只有三个状态是静态的，这三个状态下activity可以存在一段比较长的时间。(其它几个状态会很快就切换掉，停留的时间比较短暂)
  - **Resumed**: 在这个状态，activity是在最前端的，用户可以与它进行交互。(通常也被理解为"running" 状态)
  - **Paused**: 在这个状态，activity被另外一个activity所遮盖：另外的activity来到最前面，但是半透明的，不会覆盖整个屏幕。被暂停的activity不会再接受用户的输入且不会执行任何代码。
  - **Stopped**: 在这个状态, activity完全被隐藏，不被用户可见。可以认为是在后台。当stopped, activity实例与它的所有状态信息都会被保留，但是activity不能执行任何代码。
- 其它状态 (Created and Started)都是短暂的，系统快速的执行那些回调函数并通过执行下一阶段的回调函数移动到下一个状态。也就是说，在系统调用`onCreate()`, 之后会迅速调用`onStart()`, 之后再迅速执行`onResume()`。上面就是基本的activity生命周期。

## Specify Your App's Launcher Activity[指定你的程序首次启动的Activity]

- 当用户从主界面点击你的程序图标时，系统会调用你的app里面的被声明为"launcher" (or "main") activity中的onCreate()方法。这个Activity被用来当作你的程序的主要进入点。
- 你可以在AndroidManifest.xml中定义哪个activity作为你的主activity。
- 这个main activity必须在manifest使用包括 MAIN action and LAUNCHER category 的标签来声明。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>
```

- 如果你的程序中没有一个activity声明了MAIN action 或者LAUNCHER category，那么在设备的主界面列表里面不会呈现你的app图标。

## Create a New Instance[创建一个新的实例]

- 大多数app都包括许多不同的activities，这样使得用户可以执行不同的动作。不论这个activity是创建的主activity还是为了响应用户行为而新创建的，系统都会调用新的activity实例中的onCreate()方法。
- 你必须实现onCreate()方法来执行程序启动所需要的基本逻辑。
- 例如：下面的onCreate()方法演示了为了建立一个activity所需要的一些基础操作。如，声明UI元素，定义成员变量，配置UI等。*(onCreate里面尽量少做事情，避免程序启动太久都看不到界面)*

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

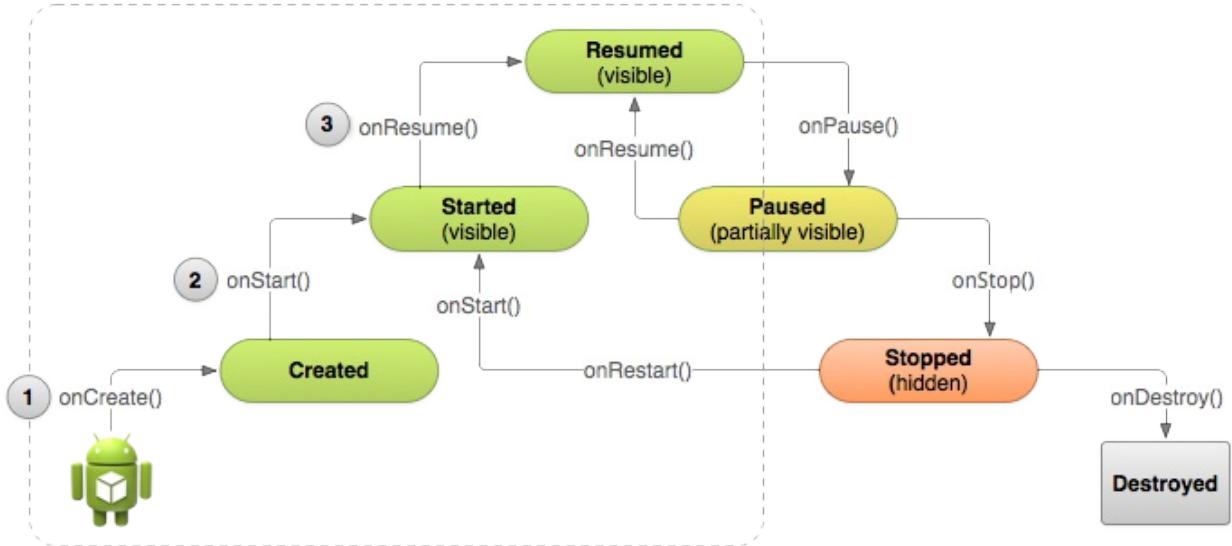
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project res/layout/main_activity.xml
    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 Action Bar
    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);
    }
}
```

- 一旦结束onCreate操作，系统会迅速调用onStart()与onResume()方法。你的activity不会在Created或者Started状态停留。技术上来说，activity在onStart()被调用后会开始被用户可见，但是onResume()会迅速被执行使得activity停留在Resumed状态，直到一些因素发生变化才会改变这个状态。例如接受到一个来电，用户切换到另外一个activity，或者是设备屏幕关闭。
- 在后面的课程中，你将看到其他方法是如何使用的，onStart()与onResume()在用户从Paused or Stopped状态中恢复的时候非常有用。
- **Note:** onCreate()方法包含了一个参数叫做savedInstanceState，这将会在后面的课程：重新创建一个activity的时候涉及到。



- Figure 2. 上图显示了onCreate(), onStart(), and onResume()是如何执行的。当这三个顺序执行的回调函数完成后，activity会到达Resumed状态。

## Destroy the Activity[销毁Activity]

- activity的第一个生命周期回调函数是onCreate(),它最后一个回调是onDestroy().系统会执行这个方法作为你的activity要从系统中完全移除的信号。
- 大多数apps并不需要实现这个方法，因为局部类的references会被destroyed并且你的activity应该在onPause()与onStop()中执行清除的操作。然而，如果你的activity包含了你在onCreate时创建的后台线程，或者是其他有可能导致内存泄漏的资源，你应该在OnDestroy()时杀死他们。

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

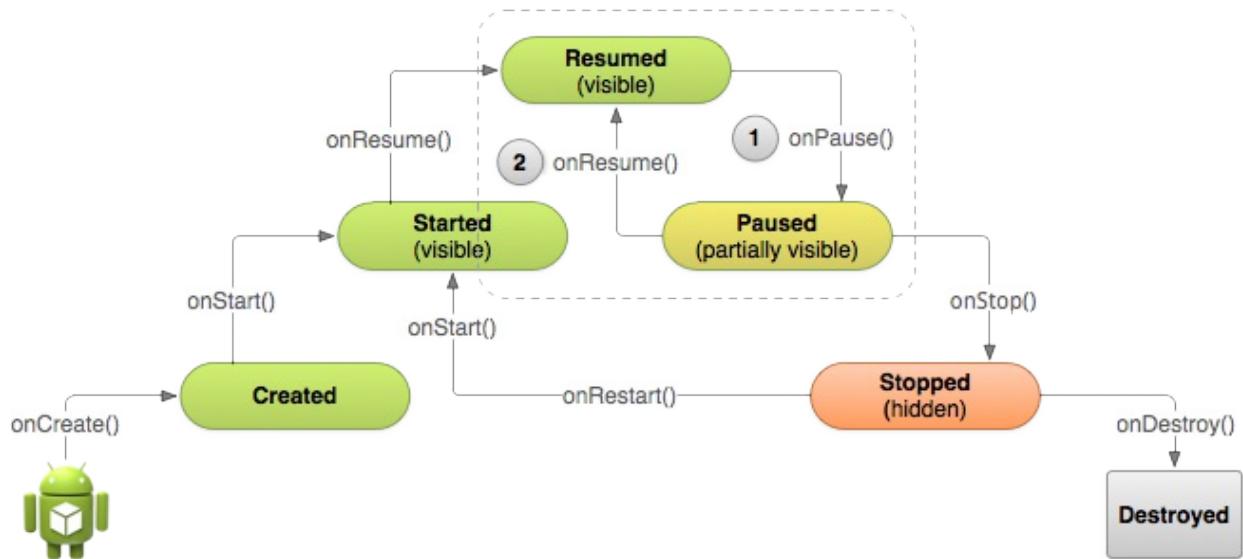
- **Note:** 系统通常是在执行了onPause() and onStop()之后再调用onDestroy()，除非你的程序里面再其他地方调用了finish()方法，这样系统会直接调用onDestory方法，其它生命周期的方法则不会被执行。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/basics/activity-lifecycle/pausing.html>

# 暂停与恢复Activity

- 在使用通常的app时，前端的activity有时候会被其他可见的组件而阻塞(obstructed)，这样会导致当前的activity进入Pause状态。例如，当打开一个半透明的activity时(例如以对话框的形式)，之前的activity会被暂停。只要这个activity仍然被部分可见，之前的activity则一直处于Paused状态。
- 然而，一旦之前的activity被完全阻塞并不可见，它则会进入Stop状态(将在下一小节讨论)。
- 当你的activity进入paused状态，系统会调用你的activity中的onPause()方法，这个方法会停止目前正在运行的操作，比如暂停视频播放。它会保存那些有可能需要长期保存的信息。(请注意是在onPause里面去保存那些信息)。如果用户从暂停状态回到你的activity，系统会恢复那些数据并执行onResume()方法。
- Note:**当你的activity接受到调用onPause()信息，那可能意味着activity将被暂停一段时间，并且用户很可能回到你的activity。然而，那也是用户要离开你的activity的第一个信号。
- Figure 1.**下图显示了，当一个半透明的activity阻塞你的activity时，系统会调用onPause()方法并且这个activity会停留在Paused state (1). 如果用户在这个activity还是在Paused State时回到这个activity，系统则会调用它的onResume() (2).



## Pause Your Activity[暂停你的Activity]

- 当系统调用你的activity中的onPause(),从技术上讲，那意味着你的activity仍然处于部分可见的状态，当时大多数时候，那意味着用户正在离开这个activity并马上会进入Stopped state. 你通常应该在onPause()回调方法里面做下面的事情：
  - 停止动画或者是其他正在运行的操作，那些都会导致CPU的浪费.
  - 提交没有保存的改变，但是仅仅是在用户离开时期待保存的内容(例如邮件草稿).
  - 释放系统资源，例如broadcast receivers, sensors (比如GPS), 或者是其他任何会影响到电量的资源。
  - 例如, 如果你的程序使用Camera, onPause()会是一个比较好的地方去做那些释放资源的操作。

```
@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;  
    }  
}
```

- 通常，你不应该使用onPause()来保存用户改变的数据(例如填入表格中的个人信息)到永久存储上。仅仅当你确认用户期待那些改变能够被自动保存的时候(例如正在撰写邮件草稿)，你可以把那些数据存到永久存储。然而，你应该避免在onPause()时执行CPU-intensive的工作，例如写数据到DB，因为它会导致切换到下一个activity变得缓慢(你应该把那些heavy-load的工作放到onStop()去做)。
- 如果你的activity实际上是要被Stop，那么你应该为了切换的顺畅而减少在OnPause()方法里面的工作量。
- Note:**当你的activity处于暂停状态，[Activity](#)实例是驻留在内存中的，并且在activity恢复的时候重新调用。你不需要在恢复到Resumed状态的一系列回调方法中重新初始化组件。

## Resume Your Activity[恢复你的activity]

- 当用户从Paused状态恢复你的activity时，系统会调用onResume()方法。
- 请注意，系统每次调用这个方法时，activity都处于最前台，包括第一次创建的时候。所以，你应该实现onResume()来初始化那些你在onPause方法里面释放掉的组件，并执行那些activity每次进入Resumed state都需要的初始化动作(例如开始动画与初始化那些只有在获取用户焦点时才需要的组件)
- 下面的onResume()的例子是与上面的onPause()例子相对应的。

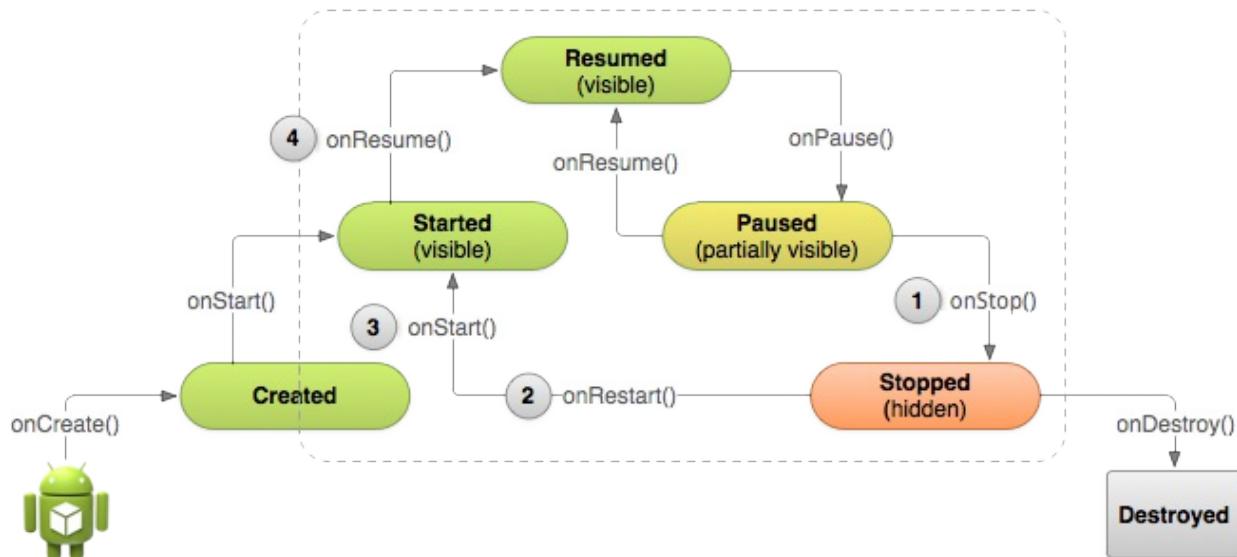
```
@Override  
public void onResume() {  
    super.onResume(); // Always call the superclass method first  
  
    // Get the Camera instance as the activity achieves full user  
    if (mCamera == null) {  
        initializeCamera(); // Local method to handle camera init  
    }  
}
```

编写:[kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/basics/activity-lifecycle/stopping.html>

# 停止与重启Activity

- 恰当的停止与重启你的activity是很重要的，在activity生命周期中，他们能确保用户感知到程序的存在并不会丢失他们的进度。在下面一些关键的场景中会涉及到停止与重启：
  - 用户打开最近使用app的菜单并切换你的app到另外一个app，这个时候你的app是被停止的。如果用户通过统一的办法回到你的app，那么你的activity会重启。
  - 用户在你的app里面执行启动一个新的activity的操作，当前activity会在第二个activity被创建后停止。如果用户点击back按钮，第一个activity会被重启。
  - 用户在使用你的app时接受到一个来电电话。
- Activity类提供了onStop()与onRestart(), 方法来允许你在activity停止与重启时进行调用。不像暂停状态是部分阻塞UI，停止状态是UI不在可见并且用户的焦点转移到另一个activity中。
- Note:**因为系统在activity停止时会在内存中保存了Activity实例。有些时候你不需要实现onStop(),onRestart()甚至是onStart()方法. 因为大多数的activity相对比较简单，activity会自己停止与重启，你只需要使用onPause()来停止正在运行的动作并断开系统资源链接。



- Figure 1.**上图显示：当用户离开你的activity，系统会调用onStop()来停止activity (1). 这个时候如果用户返回，系统会调用onRestart()(2)，之后会迅速调用[onStart\(\)](#) 与 [onResume\(\)](#)(4). 请注意：无论什么原因导致activity停止，系统总是会在onStop()之前调用onPause()方法。

## Stop Your Activity[停止你的activity]

- 当你的activity调用onStop()方法, activity不再可见, 并且应该释放那些不再需要的所有资源。一旦你的activity停止了, 系统会在不再需要这个activity时摧毁它的实例。在极端情况下, 系统会直接杀死你的app进程, 并且不执行activity的onDestroy()回调方法, 因此你需要使用onStop()来释放资源, 从而避免内存泄漏。(这点需要注意)
- 尽管onPause()方法是在onStop()之前调用, 你应该使用onStop()来执行那些CPU intensive的shut-down操作, 例如writing information to a database.
- 例如, 下面是一个在onStop()的方法里面保存笔记草稿到persistent storage的示例:

```
@Override  
protected void onStop() {  
    super.onStop(); // Always call the superclass method first  
  
    // Save the note's current draft, because the activity is stop  
    // 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, getCurrentNoteTitl  
  
    getContentResolver().update(  
        mUri, // The URI for the note to update.  
        values, // The map of column names and new values to  
        null, // No SELECT criteria are used.  
        null // No WHERE columns are used.  
    );  
}
```

- 当你的activity已经停止, [Activity](#)对象会保存在内存中, 并且在activity resume的时候重新被调用到。你不需要在恢复到Resumed state状态前重新初始化那些被保存在内存中的组件。系统同样保存了每一个在布局中的视图的当前状态, 如果用户在EditText组件中输入了text, 它会被保存, 因此不需要保存与恢复它。
- **Note:**即时系统会在activity stop的时候销毁这个activity, 它仍然会保存View objects (such as text in an [EditText](#)) 到一个[Bundle](#)中, 并且在用户返回这个activity时恢复他们(下一个会介绍在activity销毁与重新建立时如何使用[Bundle](#)来保存其他数据的状态).

## Start/Restart Your Activity[启动与重启你的activity]

- 当你的activity从Stopped状态回到前台时，它会调用onRestart().系统再调用onStart()方法，onStart()方法会在每次你的activity可见时都会被调用。onRestart()方法则是只在activity从stopped状态恢复时才会被调用，因此你可以使用它来执行一些特殊的恢复(restoration)工作，请注意之前是被stopped而不是destroy。
- 使用onRestart()来恢复activity状态是不太常见的，因此对于这个方法如何使用没有任何的guidelines。然而，因此你的onStop()方法应该做清除所有activity资源的操作，你将会在重新启动activitiy时re-instantiate那些被清除的资源，同样，你也需要在activity第一次创建时instantiate那些资源。介于上面的原因，你应该使用onStart()作为onStop()所对应方法。因为系统会在创建activity与从停止状态重启activity时都会调用onStart().(这个地方的意思应该是说你在onStop里面做了哪些清除的操作就应该在onStart里面重新把那些清除掉的资源重新创建出来)
- 例如：因为用户很可能在回到这个activity之前需要过一段时间，所以onStart()方法是一个比较好的地方用来验证某些必须的功能是否已经Ready。

```
@Override  
protected void onStart() {  
    super.onStart(); // Always call the superclass method first  
  
    // The activity is either being restarted or started for the f  
    // so this is where we should make sure that GPS is enabled  
    LocationManager locationManager =  
        (LocationManager) getSystemService(Context.LOCATION_SE  
    boolean gpsEnabled = locationManager.isProviderEnabled(Locatio  
  
    if (!gpsEnabled) {  
        // Create a dialog here that requests the user to enable G  
        // with the android.provider.Settings.ACTION_LOCATION_SOUR  
        // to take the user to the Settings screen to enable GPS w  
    }  
}  
  
@Override  
protected void onRestart() {  
    super.onRestart(); // Always call the superclass method first  
  
    // Activity being restarted from stopped state  
}
```

- 当系统Destory你的activity，它会为你的activity调用onDestroy()方法。因为我们在onStop方法里面做释放资源的操作，那么onDestory方法则是你最后去清除那些可能导致内存泄漏的地方。因此你需要确保那些线程都被destroyed并且所有的操作都被停止。

编写:[kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/basics/activity-lifecycle/recreating.html>

# 重新创建Activity

- 有几个场景中,Activity是由于正常的程序行为而被Destory的，例如当用户点击返回按钮或者是你的Activity通过调用finish()来发出停止信号。系统也有可能会在你的Activity处于stop状态且长时间不被使用，或者是在前台activity需要更多系统资源的时候把关闭后台进程，这样来获取更多的内存。
- 当你的Activity是因为用户点击Back按钮或者是activity通过调用finish()结束自己时，系统就丢失了Activity实例这个概念，因为前面的行为意味着不再需要这个activity了。然而，如果因为系统资源紧张而导致Activity的Destory, 系统会在用户回到这个Activity时有这个Activity存在过的记录，系统会使用那些保存的记录数据（描述了当Activity被Destory时的状态）来重新创建一个新的Account实例。那些被系统用来恢复之前状态而保存的数据被叫做 "instance state"，它是一些存放在 Bundle 对象中的key-value pairs.
- **Caution:**你的Activity会在每次旋转屏幕时被destroyed与recreated。当屏幕改变方向时，系统会Destory与Recreate前台的activity，因为屏幕配置被改变，你的Activity可能需要加载一些alternative的资源(例如layout).
- 默认情况下, 系统使用 Bundle 实例来保存每一个视图对象中的信息(例如输入EditText 中的文本内容)。因此，如果你的Activity被destroyed与recreated, 那么layout 的状态信息会自动恢复到之前的状态。然而，你的activity也许存在更多你想要恢复的状态信息，例如记录用户Progress的成员变量(member variables)..
- 请注意为了让你可以保存额外更多的数据到saved instance state。在Activity的声明周期里面存在一个添加的回调函数。这个回调函数并没有在前面课程的图片示例中显示。这个方法是 onSaveInstanceState()，当用户离开你的Activity时，系统会调用它。当系统调用这个函数时，系统会在你的Activity被异常Destory时传递 Bundle 对象，这样你可以增加额外的信息到Bundle中并保存与系统中。然后如果系统在Activity被Destory之后想重新创建这个Activity实例时，之前的那个Bundle对象会(系统)被传递到你的activity的 onRestoreInstanceState() 方法与 onCreate() 方法中。



上图：当系统开始停止你的Activity时，会调用到[onSaveInstanceState\(\)](#)(1)，因此你可以在Activity实例需要重新创建的情况下，指定特定的附加状态数据到Bunde中。如果这个Activity被destroyed而且同样的实例被重新创建，系统会传递在(1)中的状态数据到[onCreate\(\)](#)(2) 与 [onRestoreInstanceState\(\)](#)(3)。

## Save Your Activity State[保存Activity状态]

- 当你的activity开始Stop，系统会调用 onSaveInstanceState()，因此你的Activity可以用键值对的集合来保存状态信息。这个方法会默认保存Activity视图的状态信息，例如在 EditText 组件中的文本或者是 ListView 的滑动位置。
- 为了给Activity保存额外的状态信息，你必须实现onSaveInstanceState() 并增加key-value pairs到 Bundle 对象中，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
    super.onSaveInstanceState(savedInstanceState);
}
```

- Caution: 总是需要调用 onSaveInstanceState() 方法的父类实现，这样默认的父类实现才能保存视图状态的信息。

## Restore Your Activity State [恢复Activity状态]

- 当你的Activity从Destory中重建。你可以从系统传递给你的Activity的Bundle中恢复保存的状态。onCreate()与onRestoreInstanceState()回调方法都接收到了同样的Bundle，里面包含了同样的实例状态信息。
- 因为onCreate()方法会在第一次创建新的Activity实例与重新创建之前被Destory的实例时都被调用，你必须在你尝试读取Bundle对象前Check它是否为null。如果它为Null，系统则是创建一个新的Activity instance，而不是恢复之前被Destory的Activity。
- 下面是一个示例：演示在onCreate方法里面恢复一些数据：

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

- 你也可以选择实现onRestoreInstanceState()，而不是在onCreate方法里面恢复数据。onRestoreInstanceState()方法会在onStart()方法之后执行。系统仅仅会在存在需要恢复的状态信息时才会调用onRestoreInstanceState()，因此你不需要检查Bundle是否为Null：

```
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);  
}
```

- Caution: 与上面保存一样，总是需要调用onRestoreInstanceState()方法的父类实现，这样默认的父类实现才能保存视图状态的信息。如果想了解更多关于运行时状态改变引起的recreate你的activity。请参考[Handling Runtime Changes](#)。

编写:[fastcome1985](#) - 校对:

原文:<http://developer.android.com/training/basics/fragments/index.html>

# 使用Fragment建立动态UI

- 为了在Android上创建动态的、多窗口的用户交互体验，你需要将UI组件封装成模块化进行使用，在activity中你可以对这些模块进行切入切出操作。你可以用[Fragment](#)来创建这些模块，Fragment就像一个嵌套的activity,拥有自己的布局(layout) 以及管理自己的生命周期。
- 如果一个fragment定义了自己的布局，那么在activity中它可以与其他的fragments生成不同的组合，从而为不同的屏幕尺寸生成不同的布局（一个小的屏幕一次只放一个fragment，大的屏幕则可以两个或以上的fragment）。
- 这一章将向你展示如何用fragment来创建动态的用户体验，以及在不同屏幕尺寸的设备上优化你的APP的用户体验。像运行着android1.6这样老版本的设备，也都将继续得到支持。
- 完整的Demo示例：[FragmentBasics.zip](#)

# Lessons

- [创建一个fragment](#)

学习如何创建一个fragment，以及实现它生命周期内的基本功能。

- [构建复杂的UI](#)

学习在APP内，对不同的屏幕尺寸用fragments构建不同的布局。

- [与其他fragments交互](#)

学习fragment与activity以及其他fragments之间交互。

编写:[fastcome1985](#) - 校对:

原文:<http://developer.android.com/training/basics/fragments/creating.html>

# 创建一个Fragment

- 你可以把fragment想象成activity中一个模块化的部分，它拥有自己的生命周期，接收自己的输入事件，可以在activity运行过程中添加或者移除（有点像"子activity"，你可以在不同的activities里面重复使用）。这一课教你继承[Support Library](#) 中的[Fragment](#)，以使你的应用在Android1.6这样的低版本上仍能保持兼容。注意：如果的你的APP的最低API版本是11或以上，你不必使用Support Library，你可以直接使用API框架里面的[Fragment](#)，这节课主要是讲基于Support Library的API，Support Library有一个特殊的包名，有时候与平台版本的API名字有些轻微的不一样。
- 在开始这节课前，你必须先让你的项目引用Support Library。如果你没有使用过Support Library，你可以根据文档 [Support Library Setup](#) 来设置你的项目使用Support Library。当然，你也可以使用包含[action bar](#)的 v7 appcompat library。v7 appcompat library 兼容Android2.1(API level 7),也包含[Fragment](#) APIs。

## 创建一个Fragment类

- 创建一个fragment,首先需要继承[Fragment](#)类, 然后在关键的生命周期方法中插入你APP的逻辑, 就像[activity](#)一样。
- 其中一个区别是当你创建[Fragment](#)的时候, 你必须重写[onCreateView\(\)](#)回调方法来定义你的布局。事实上, 这是使Fragment运行起来, 唯一一个需要你重写的回调方法。比如, 下面是一个自定义布局的示例fragment.

```
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,
    }
}
```

- 就像activity一样, 当fragment从activity添加或者移除、当activity生命周期发生变化时, fragment应该是实现生命周期回调来管理它的状态。例如, 当activity的[onPause\(\)](#)被调用时, 它里面的所有fragment的[onPause\(\)](#)方法也会被触发。

# 用XML将fragment添加到activity

- fragments是可以重用的，模块化的UI组件，每一个Fragment的实例都必须与一个FragmentActivity关联。你可以在activity的XML布局文件中定义每一个fragment来实现这种关联。

注意：FragmentActivity是Support Library提供的一个特殊activity，用来在API11版本以下的系统上处理fragment.如果你APP中的最低版本大于等于11，你可以使用普通的Activity.

- 下面是一个XML布局的例子，当屏幕被认为是large(用目录名称中的large字符来区分)时，它在布局中增加了两个fragment.

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.HeadlineF
        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.ArticleF
        android:id="@+id/article_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

小贴士：更多关于不同屏幕尺寸创建不同布局的信息，请阅读[Supporting Different Screen Sizes](#)

- 然后将这个布局文件用到你的activity中。

```
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);
    }
}
```

- 如果你用的是[v7 appcompat library](#)，你的activity应该改为继承ActionBarActivity，ActionBarActivity是FragmentActivity的一个子类（更多关于这

方面的内容，请阅读[Adding the Action Bar](#)）。

注意：当你用XML布局文件的方式将Fragment添加进activity时，你的Fragment是不能被动态移除的。如果你想要在用户交互的时候把fragment切入与切出，你必须在activity启动后，将fragment添加进activity.这将在下节课讲到。

编写:[fastcome1985](#) - 校对:

原文:<http://developer.android.com/training/basics/fragments/fragment-ui.html>

# 建立灵活动态的UI

- 如果你的APP设计成要支持范围广泛的屏幕尺寸时，在可利用的屏幕空间内，你可以通过在不同的布局配置中重用你的fragment来优化你的用户体验。
- 比如，一个手机设备可能适合一次只有一个fragment的单面板用户交互。相反，在更大屏幕尺寸的平板电脑上，你可能更想要两个fragment并排在一起，用来向用户展示更多信息。

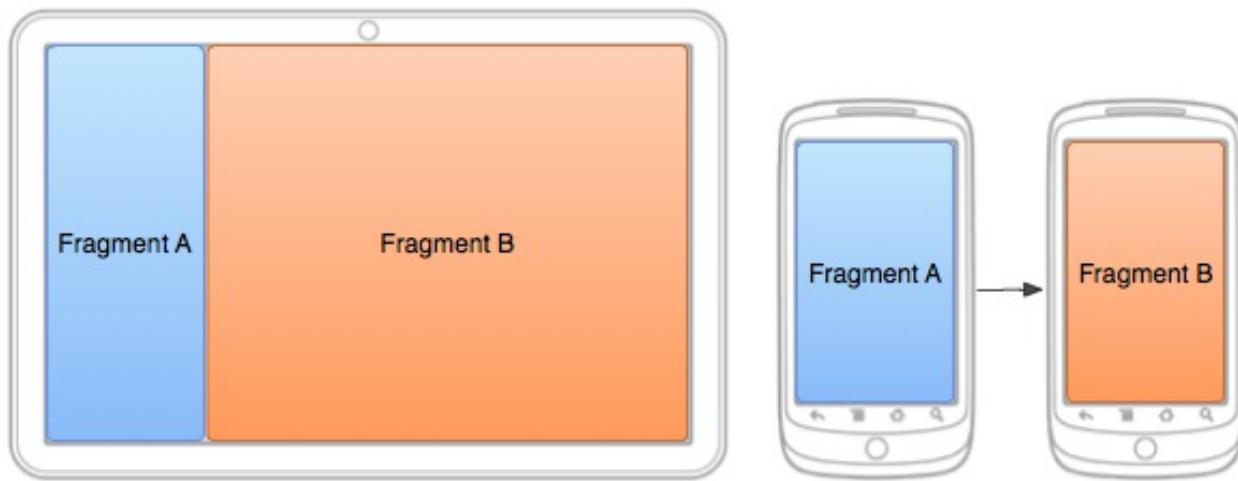


图1：两个fragments，在同一个activity不同屏幕尺寸中用不同的配置来展示。在大屏幕上，两个fragment被并排放置，但是在手机上，一次只放置一个fragment，所以在用户导航中，两个fragment必须进行替换。

- [FragmentManager](#)类提供了方法，让你在activity运行时能够对fragment进行添加，移除，替换，以创建动态的用户体验。

## 在activity运行时添加fragment

- 比起在activity的布局文件中定义fragments,就像[上节课](#)说的用标签,你也可以在activity运行时动态添加fragment,如果你在打算在activity的生命周期内替换fragment,这是必须的。
- 为了执行fragment的增加或者移除操作,你必须用[FragmentManager](#)创建一个[FragmentTransaction](#)对象,FragmentTransaction提供了用来增加、移除、替换以及其它一些操作的APIs。
- 如果你的activity允许fragments移除或者替换,你应该在activity的[onCreate\(\)](#)方法中添加初始化的fragment(s)。
- 运用fragment(除了那些你在运行时添加的)的一个很重要的规则就是在布局中你必须有一个容器[view](#),fragment将会放在这个view里面。
- 下面的这个布局是[上节课](#)的一次只显示一个fragment的布局的替代布局。为了从一个布局替换为另外一个布局,activity的布局包含了一个空的[FrameLayout](#)作为fragment的容器。
- 注意文件名与上节课的布局一样,但是文件目录没有large标识,所以你的布局将会在比large小的屏幕上被使用,因为这个屏幕无法满足同时放置两个fragments

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" />
```

- 在你的activity里面,用Support Library APIs调用[getSupportFragmentManager\(\)](#)方法获取[FragmentManager](#)对象,然后调用[beginTransaction\(\)](#)方法创建一个[FragmentTransaction](#)对象,然后调用[add\(\)](#)方法添加一个fragment.
- 你可以使用同一个[FragmentTransaction](#)进行多次fragment事物。当你完成这些变化操作的时候,必须调用[commit\(\)](#)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
// we could end up with overlapping fragments.
if (savedInstanceState != null) {
    return;
}

// Create a new Fragment to be placed in the activity
HeadlinesFragment firstFragment = new HeadlinesFragmen

// In case this activity was started with special intent
// Intent, pass the Intent's extras to the fragment as
firstFragment.setArguments(getIntent().getExtras());

// Add the fragment to the 'fragment_container' FrameLayout
getSupportFragmentManager().beginTransaction()
    .add(R.id.fragment_container, firstFragment).commit();
}

}

}
```

- 当fragment在activity运行时被添加进来时（不是在XML布局中用定义的），activity可以移除这个fragment或者用另外一个来替换它。

## Fragment替换

- 替换fragment的过程与添加过程类似，只需要将[add\(\)](#)方法替换为[replace\(\)](#)方法。
- 记住当你执行fragment事物的时候，例如移除或者替换，你经常要适当地让用户可以向后导航与"撤销"这次改变。为了让用户向后导航fragment事物，你必须在[FragmentTransaction](#)提交前调用[addToBackStack\(\)](#)方法。

注意：当你移除或者替换一个fragment并把它放入返回栈中时，被移除的fragment的生命周期是stopped(不是destoryed).当用户返回重新恢复这个fragment,它的生命周期是restarts。如果你没把fragment放入返回栈中，那么当他被移除或者替换时，它的生命周期是destoryed。

- 下面是一个fragment替换的例子

```
// Create fragment and give it an argument specifying the 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 to it later.
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

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

- [addToBackStack\(\)](#)方法提供了一个可选的String参数为事物指定了一个唯一的名字。这个名字不是必须的，除非你打算用[FragmentManager.BackStackEntry](#) APIs来进行一些高级的fragments操作。

编写:[fastcome1985](#) - 校对:

原文:<http://developer.android.com/training/basics/fragments/communicating.html>

# Fragments之间的交互

- 为了重用Fragment UI组件，你应该把每一个fragment都构建成完全的自包含的、模块化的组件，定义他们自己的布局与行为。当你定义好这些模块化的Fragments的时候，你就可以让他们关联activity，使他们与Application的逻辑结合起来，实现全局的复合的UI。
- 经常地，你想fragment之间能相互交互，比如基于用户事件改变fragment的内容。所有fragment之间的交互需要通过他们关联的activity，两个fragment之间不应该直接交互。

## 定义一个接口

- 为了让fragment与activity交互，你可以在Fragment类中定义一个接口，并且在activity中实现这个接口。Fragment在他们生命周期的onAttach()方法中捕获接口的实现，然后调用接口的方法来与Activity交互。

下面是一个fragment与activity交互的例子：

```
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Fragment就可以通过调用OnHeadlineSelectedListener接口实例的mCallback中的onArticleSelected()（也可以是其它方法）方法与activity传递消息。
- 举个例子，在fragment中的下面的方法在用户点击列表条目时被调用，fragment用回调接口来传递事件给父Activity。

```
@Override  
public void onListItemClick(ListView l, View v, int position,  
    // Send the event to the host activity  
    mCallback.onArticleSelected(position);  
}
```

## 实现接口

- 为了接收回调事件，宿主activity必须实现在Fragment中定义的接口。
- 举个例子，下面的activity实现了上面例子中的接口。

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

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the H
        // Do something here to display that article
    }
}
```

## 传消息给Fragment

- 宿主activity通过`findFragmentById()`方法来获取`fragment`的实例，然后直接调用`Fragment`的public方法来向`fragment`传递消息。
- 例如，想象一下，上面所示的activity可能包含另外一个`fragment`,这个`fragment`用来展示从上面的回调方法中返回的指定的数据。在这种情况下，activity可以把从回调方法中接收到的信息传递给这个展示数据的`Fragment`.

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

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the H
        // Do something here to display that article

        ArticleFragment articleFrag = (ArticleFragment)
            getSupportFragmentManager().findFragmentById(R.id.

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

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

            // Create fragment and give it an argument for the sel
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);

            FragmentTransaction transaction = getSupportFragmentManagerMa

            // Replace whatever is in the fragment_container view
            // and add the transaction to the back stack so the us
            transaction.replace(R.id.fragment_container, newFragme
            transaction.addToBackStack(null);

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

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/data-storage/index.html>

# 数据保存

虽然可以在onPause()的时候保存一些信息以免用户的使用进度被丢失，但是大多数Android app仍然是需要做保存数据的动作。大多数比较好的apps都需要保存用户的设置信息，而且有一些apps必须维护大量的文件信息与DB信息。这一章节会介绍给你在Android中一些重要的数据存储方法，例如：

- [保存到Preferences](#)

学习使用Shared Preferences文件以Key-Value的方式保存简要的信息。

- [保存到文件](#)

学习保存基本的文件。

- [保存到数据库](#)

学习使用SQLite数据库读写数据。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/data-storage/shared-preference.html>

# 保存到Preference(Saving Key-Value Sets)

如果你有一个相对较小的key-value集合需要保存，你应该使用[SharedPreferences](#) APIs。SharedPreferences 对象指向了一个保存key-value pairs的文件，并且它提供了简单的方法来读写这个文件。每一个 SharedPreferences 文件都是由framework管理的并且可以是私有或者可分享的。这节课会演示如何使用 SharedPreferences APIs 来存储与检索简单的数据。  
**Note:** SharedPreferences APIs 仅仅提供了读写key-value对的功能，请不要与 Preference APIs 相混淆。后者可以帮助你建立一个设置用户配置的页面（尽管它实际上是使用 SharedPreferences 来实现保存用户配置的）。如果想了解更多关于Preference APIs的信息，请参考Settings 指南。

## 获取SharedPreference(Get a Handle to a SharedPreferences)

你可以通过下面两个方法之一来创建或者访问shared preference 文件:

- **getSharedPreferences()** — 如果你需要多个通过名称参数来区分的shared preference 文件, 名称可以通过第一个参数来指定。你可以在你的app里面通过任何一个Context 来执行这个方法。
- **getPreferences()** — 当你的activity仅仅需要一个shared preference文件时。因为这个方法会检索activity下的默认的shared preference文件, 并不需要提供文件名称。

例如: 下面的示例是在 Fragment 中被执行的, 它会访问名为 R.string.preference\_file\_key 的 shared preference文件, 并使用private模式来打开它, 这样的话, 此时文件就仅仅可以被你的app访问了。

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

当命名你的shared preference文件时, 你应该像 "com.example.myapp.PREFERENCE\_FILE\_KEY" 这样来命名。

当然, 如果你的activity仅仅需要一个shared preference文件时, 你可以使用[getPreferences\(\)](#)方法:

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

**Caution:** 如果你创建了一个[MODE\\_WORLD\\_READABLE](#)或者[MODE\\_WORLD\\_WRITEABLE](#) 模式的shared preference文件, 那么任何其他的app只要知道文件名, 则可以访问这个文件。

## 写Shared Preference(Write to Shared Preferences)

为了写shared preferences文件，需要通过执行 `edit()` 来创建一个 `SharedPreferences.Editor`。

通过类似 `.putInt()` 与 `putString()`方法来传递keys与values。然后执行 `commit()` 来提交改变。  
(后来有建议除非是出于线程同步的需要，否则请使用`apply()`方法来替代`commit()`，因为后者有可能会卡到UI Thread.)

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

## 读Shared Preference(Read from Shared Preferences)

为了从shared preference中检索读取数据，可以通过类似 getInt() 与 getString()等方法来读取。在那些方法里面传递你想要获取value对应的key，并且提供一个默认的value。如下：

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

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/data-storage/files.html>

# 保存到文件(Saving to Files)

Android使用与其他平台类似的基于磁盘文件系统(disk-based file systems)。这节课会描述如何在Android文件系统上使用 [File](#) 的读写APIs。File 对象非常适合用来读写那种流式顺序的数据。例如，很适合用来读写图片文件或者是网络中交换的数据。这节课会演示在app中如何执行基本的文件操作任务。假定你已经对linux的文件系统与java.io中标准的I/O APIs有一定认识。

## 存储在Internal还是External(Choose Internal or External Storage)

所有的Android设备都有两个文件存储区域: "internal" 与 "external" 存储。那两个名称来自与早先的Android系统中，那个时候大多数的设备都内置了不可变的内存 (internal storage)，然后再加上一个类似SD card (external storage) 这样可以卸载的存储部件。后来有一些设备把"internal" 与 "external" 的部分都做成不可卸载的内置存储了，虽然如此，但是这一整块还是从逻辑上有被划分为"internal"与"external"的。只是现在不再以是否可以卸载来区分了。下面列出了两者的区别：

- **Internal storage:**

- 总是可用的
- 这里的文件默认是只能被你的app所访问的。
- 当用户卸载你的app的时候，系统会把internal里面的相关文件都清除干净。
- Internal是在你想确保不被用户与其他app所访问的最佳存储区域。

- **External storage:**

- 并不总是可用的，因为用户可以选择把这部分作为USB存储模式，这样就不可以访问了。
- 是大家都可访问的，因此保存到这里的文件是失去访问控制权限的。
- 当用户卸载你的app时，系统仅仅会删除external根目录  
(`getExternalFilesDir()`) 下的相关文件。
- External是在你不需要严格的访问权限并且你希望这些文件能够被其他app 所共享或者是允许用户通过电脑访问时的最佳存储区域。

**Tip:** 尽管app是默认被安装到internal storage的，你还是可以通过在程序的manifest文件中声明`android:installLocation` 属性来指定程序也可以被安装到external storage。当某个程序的安装文件很大，用户会倾向这个程序能够提供安装到external storage的选项。更多安装信息，请参考[App Install Location](#)。

## obtain Permissions for External Storage [获取External存储的权限]

为了写数据到external storage, 你必须在你的manifest文件中申请  
求WRITE\_EXTERNAL\_STORAGE权限:

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

**Caution:**目前，所有的apps都可以在不指定写的权限下做读external storage的动作。但是，这会在以后的版本中被修正。如果你的app需要读的权限(不是写), 那么你需要声明 READ\_EXTERNAL\_STORAGE 权限。为了确保你的app能够在正常工作，你需要现在就声明读权限。但是，如果你的程序有声明写的权限，那么就默认有了读的权限。

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

对于internal storage, 你不需要声明任何权限，因为你的程序默认就有读写程序目录下的文件的权限。

## 保存到Internal Storage(Save a File on Internal Storage)

当保存文件到internal storage时，你可以通过执行下面两个方法之一来获取合适的目录作为File的对象：

- **getFilesDir()** : 返回一个 File，代表了你的app的internal目录。
- **getCacheDir()** : 返回一个 File，代表了你的app的internal缓存目录。请确保这个目录下的文件在一旦不再需要的时候能够马上被删除，还有请给予一个合理的大  
小，例如1MB。如果系统的内存不够，会自行选择删除缓存文件。为了在那些目  
录下创建一个新的文件，你可以使用 File() 构造器，如下：

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

同样，你也可以执行[openFileOutput\(\)](#) 来获取一个 FileOutputStream 用来写文件到internal目录。如下：

```
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();
}
```

如果，你需要缓存一些文件，你可以使用[createTempFile\(\)](#)。例如：下面的方法从URL中抽取了一个文件名，然后再创建了一个以这个文件名命名的文件。

```
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;
}
```

**Note:** 你的app的internal storage 目录是以你的app的包名作为标识存放在Android文件系统的特定目录下[data/data/com.example.xx]。从技术上讲，如果你设置文件为可读的，那么其他app就可以读取你的internal文件。然而，其他app需要知道你的包名与文件名。若是你没有设置为可读或者可写，其他app是没有办法读写的。因此只要你使用[MODE\\_PRIVATE](#)，那么这些文件就不可能被其他app所访问。

## Save a File on External Storage [保存文件到External Storage]

因为external storage可能是不可用的，那么你应该在访问之前去检查是否可用。你可以通过执行 `getExternalStorageState()` 来查询external storage的状态。如果返回的状态是 `MEDIA_MOUNTED`, 那么你可以读写。示例如下：

```
/* 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;
}
```

尽管external storage对与用户与其他app是可修改的，那么你可能会保存下面两种类型的文件。

- **Public files**: 这些文件对与用户与其他app来说是public的，当用户卸载你的app时，这些文件应该保留。例如，那些被你的app拍摄的图片或者下载的文件。
- **Private files**: 这些文件应该是被你的app所拥有的，它们应该在你的app被卸载时删除掉。尽管那些文件从技术上可以被用户与其他app所访问，实际上那些文件对于其他app是没有意义的。所以，当用户卸载你的app时，系统会删除你的app的private目录。例如，那些被你的app下载的缓存文件。

如果你想要保存文件为public形式的，请使用[getExternalStoragePublicDirectory\(\)](#)方法来获取一个 `File` 对象来表示存储在external storage的目录。这个方法会需要你带有一个特定的参数来指定这些public的文件类型，以便于与其他public文件进行分类。参数类型包括[DIRECTORY\\_MUSIC](#) 或者 [DIRECTORY\\_PICTURES](#). 如下：

```
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;
}
```

如果你想要保存文件为私有的方式，你可以通过执行[getExternalFilesDir\(\)](#) 来获取相应的目

录，并且传递一个指示文件类型的参数。每一个以这种方式创建的目录都会被添加到 external storage 封装你的 app 目录下的参数文件夹下（如下则是 albumName）。这下面的文件会在用户卸载你的 app 时被系统删除。如下示例：

```
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;
}
```

如果刚开始的时候，没有预定义的子目录存放你的文件，你可以在 `getExternalFilesDir()` 方法中传递 `null`。它会返回你的 app 在 external storage 下的 private 的根目录。

请记住，`getExternalFilesDir()` 方法会创建的目录会在 app 被卸载时被系统删除。如果你的文件想在 app 被删除时仍然保留，请使用 `getExternalStoragePublicDirectory()`。

不管是使用 `getExternalStoragePublicDirectory()` 来存储可以共享的文件，还是使用 `getExternalFilesDir()` 来储存那些对与你的 app 来说是私有的文件，有一点很重要，那就是你要使用那些类似 `DIRECTORY_PICTURES` 的 API 的常量。那些目录类型参数可以确保那些文件被系统正确的对待。例如，那些以 `DIRECTORY_RINGTONES` 类型保存的文件就会被系统的 media scanner 认为是 ringtone 而不是音乐。

## Query Free Space[查询剩余空间]

如果你事先知道你想要保存的文件大小，你可以通过执行[getFreeSpace\(\)](#) or [getTotalSpace\(\)](#)来判断是否有足够的空间来保存文件，从而避免发生IOException。那些方法提供了当前可用的空间还有存储系统的总容量。

然而，系统并不会授权你写入通过getFreeSpace().查询到的容量文件，如果查询的剩余容易比你的文件大小多几MB，或者说文件系统使用率还不足90%，这样则可以继续进行写的操作，否则你最好不要写进去。

**Note:**你并没有强制要求在写文件之前一定有要去检查剩余容量。你可以尝试先做写的动作，然后通过捕获 IOException 。这种做法仅适合于你并不知道你想要写的文件的确切大小。

## 删除文件(Delete a File)

你应该在不需要使用某些文件的时候，删除它。删除文件最直接的方法是直接执行文件的 delete() 方法。

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

如果文件是保存在internal storage，你可以通过 Context 来访问并通过执行deleteFile()进行删除

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

**Note:** 当用户卸载你的app时， android系统会删除下面的文件：

- 所有保存到internal storage的文件。
- 所有使用getExternalFilesDir()方式保存在external storage的文件 然而，你应该手动删除所有通过 getCacheDir() 方式创建的缓存文件，还有那些通常来说不会再用的文件。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/data-storage/database.html>

# Saving to database 保存到数据库

对于重复或者结构化的数据（如联系人信息）等保存到DB是个不错的主意。这节课假定你已经熟悉SQL数据库的操作。在Android上可能会使用到的APIs，可以从[android.database.sqlite](#)包中找到。

## Define a Schema and Contract[定义Schema与Contract]

SQL中一个重要的概念是schema: 一种DB结构的正式声明。schema是从你创建DB的SQL语句中生成的。你可能会发现创建一个伴随类（companion class）是很有益的，这个类成为合约类（contract class），它用一种系统化并且自动生成文档的方式，显示指定了你的schema样式。

Contract Class是一些常量的容器。它定义了例如URIs, 表名, 列名等。这个contract类允许你在同一个包下与其他类使用同样的常量。它让你只需要在一个地方修改列名，然后这个列名就可以自动传递给你整个code。

一个组织你的contract类的好方法是在你的类的根层级定义一些全局变量，然后为每一个table来创建内部类。

**Note:**通过实现 [BaseColumns](#) 的接口，你的内部类可以继承到一个名为\_ID的主键，这个对于Android里面的一些类似cursor adaptor类是很有必要的。这样能够使得你的DB与Android的framework能够很好的相容。

例如，下面的例子定义了表名与这个表的列名：

```
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";  
    ...  
}
```

为了防止一些人不小心实例化contract类，像下面一样给一个空的构造器。

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

## Create a Database Using a SQL Helper[使用SQL Helper创建DB]

当你定义好了你的DB应该是什么样之后，你应该实现那些创建与维护db与table的方法。下面是一些典型的创建与删除table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
    + FeedReaderContract.FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + C
    ... // Any other options for the CREATE command
    " )";

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

就像保存文件到设备的 internal storage 一样，Android会保存db到你的程序的private的空间上。你的数据是受保护的，因为那些区域默认是私有的，不可被其他程序所访问。

在[SQLiteOpenHelper](#)类中有一些很有用的APIs。当你使用这个类来做一些与你的db有关的操作时，系统会对那些有可能比较耗时的操作（例如创建与更新等）在真正需要的时候才去执行，而不是在app刚启动的时候就去做那些动作。你所需要做的仅仅是执行 `getWritableDatabase()` 或者 `getReadableDatabase()`。

**Note:**因为那些操作可能是很耗时的，请确保你在background thread (AsyncTask or IntentService) 里面去执行 `getWritableDatabase()` 或者 `getReadableDatabase()`。

为了使用 `SQLiteOpenHelper`，你需要创建一个子类并重写 `onCreate()`, `onUpgrade()` 与 `onOpen()`等callback方法。你也许还需要实现 `onDowngrade()`，但是这并不是必需的。

例如，下面是一个实现了`SQLiteOpenHelper`类的例子：

```
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 operation
        // is simplified.
        // To simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
}
```

```
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

为了访问你的db，需要实例化你的 SQLiteOpenHelper的子类：

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

## Put Information into a Database [添加信息到DB]

通过传递一个 ContentValues 对象到 insert() 方法:

```
// 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, conte

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

insert() 方法的第一个参数是table名，第二个参数会使得系统自动对那些 ContentValues 没有提供数据的列填充数据为null，如果第二个参数传递的是null，那么系统则不会对那些没有提供数据的列进行填充。

## Read Information from a Database [从DB中读取信息]

为了从DB中读取数据，需要使用 query() 方法，传递你需要查询的条件。查询后会返回一个 Cursor 对象。

```
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 results  
    null, // Don't filter by rowid  
    sortOrder // The sort order  
) ;
```

下面是演示如何从course对象中读取数据信息：

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

## Delete Information from a Database [删除DB中的信息]

和查询信息一样，删除数据，同样需要提供一些删除标准。DB的API提供了一个防止SQL注入的机制来创建查询与删除标准。**SQL Injection**(随着B/S模式应用开发的发展，使用这种模式编写应用程序的程序员也越来越多。但是由于程序员的水平及经验也参差不齐，相当大一部分程序员在编写代码的时候，没有对用户输入数据的合法性进行判断，使应用程序存在安全隐患。用户可以提交一段数据库查询代码，根据程序返回的结果，获得某些他想得知的数据，这就是所谓的*SQL Injection*，即SQL注入)

这个机制把查询语句划分为选项条款与选项参数两部分。条款部分定义了查询的列是怎么样的，参数部分用来测试是否符合前面的条款。(这里翻译的怪怪的，附上原文，*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.*)因为处理的结果与通常的SQL语句不同，这样可以避免SQL注入问题。

```
// Define 'where' part of query.
String selection = FeedReaderContract.FeedEntry.COLUMN_NAME_ENTRY
// Specify arguments in placeholder order.
String[] selectionArgs = { String.valueOf(rowId) };
// Issue SQL statement.
db.delete(table_name, mySelection, selectionArgs);
```

## Update a Database [更新数据]

当你需要修改DB中的某些数据时，使用 update() 方法。

更新操作结合了插入与删除的语法。

```
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  
String[] selectionArgs = { String.valueOf(rowId) };  
  
int count = db.update(  
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,  
    values,  
    selection,  
    selectionArgs);
```

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/intents/index.html>

# 与其他应用的交互

- 一个Android app通常都会有好几个activities. 每一个activity的界面都可能允许用户执行一些特殊任务（例如查看地图或者是开始拍照等）。为了让用户从一个activity跳到另外一个activity，你的app必须使用Intent来定义你的app想做的事情。当你使用startActivity()的方法，而且参数是intent时，系统会使用这个 Intent 来定义并启动合适的app组件。使用intents还可以让你的app来启动另外一个app里面的activity。
- 一个 Intent 可以显式的指明需要启动的模块，也可以隐式的指明自己可以处理哪种类型的动作。
- 这一章节会演示如何使用Intent 来做一些与其他app之间的简单交互。类似，启动另外一个app,从其他app接受数据，并且使得你的app能够响应从其他发出的intent。

## Lessons

- [Sending the User to Another App:Intent的发送](#)

演示如何创建隐式的Intent来唤起能够接收这个动作的App。

- [Getting a Result from an Activity:接收Activity返回的结果](#)

演示如何启动另外一个Activity并接收返回值。

- [Allowing Other Apps to Start Your Activity:Intent过滤](#)

演示如何通过定义隐式的Intent的过滤器来使得能够被其他应用唤起。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/intents/sending.html>

# Intent的发送

Android中最重要的功能之一就是可以利用一个带有action的intent使得当前app能够跳转到其他的app。例如：如果你的app拥有一个地址想要显示在地图上，你并不需要在你的app里面创建一个activity用来显示地图。你只需要使用Intent来发出查看地址的请求。Android系统则会启动能够显示地图的程序来呈现那个地址。

正如在2.1章节:[Building Your First App:建立你的第一个App](#)中所说的，你必须使用intent来在同一个app的两个activity之间进行切换。在那种情况下通常是定义一个显示（explicit）的intent，它指定了需要叫起组件。然而，当你想要叫起不同的app来执行那个动作，则必须使用隐式（implicit）的intent。

这节课会介绍如何为特殊的动作创建一个implicit intent，并使用它来启动另外一个app去执行intent中的action。

## Build an Implicit Intent[建立一个隐式的Intent]

Implicit intents并不会声明需要启动的组件的类名，它使用的是声明一个需要执行的动作。这个action指定了你想做的事情，例如查看，编辑，发送或者是获取什么。Intents通常会在发送action的同时附带一些数据，例如你想要查看的地址或者是你想要发送的邮件信息。依赖于你想要创建的Intent，这些数据需要是Uri，或者是其他规定的数据类型。如果你的数据是一个Uri，会有一个简单的 Intent() constructor 用来定义action与data。

例如，下面是一个带有指定电话号码的intent。

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

当你的app通过执行startActivity()来启动这个intent时，Phone app会使用之前的电话号码来拨出这个电话。

下面是一些其他intent的例子：

- View a map:

```
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mou
// Or map point based on latitude/longitude
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

- View a web page:

```
Uri webpage = Uri.parse("http://www.android.com");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

另外一些需要"extra"数据的implicit intent。你可以使用 putExtra() 方法来添加那些数据。默认的，系统会根据Uri数据类型来决定需要哪些合适的MIME type。如果你没有在intent中包含一个Uri，则通常需要使用 setType() 方法来指定intent附带的数据类型。设置MIME type 是为了指定哪些activity可以应该接受这个intent。例如：

- Send an email with an attachment:

```
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"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://pat
// You can also attach multiple items by passing an ArrayList of Uri
```

- Create a calendar event:

```
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);
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime);
calendarIntent.putExtra(Events.TITLE, "Ninja class");
calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

**Note:** 这个intent for Calendar的例子只使用于>=API Level 14。

**Note:** 请尽可能的定义你的intent更加确切。例如，如果你想要使用ACTION\_VIEW的intent来显示一张图片，你还应该指定MIME type为image/\*.这样能够阻止其他能够"查看"其他数据类型的app (like a map app) 被这个intent叫起。

## Verify There is an App to Receive the Intent[验证是否有App去接收这个Intent]

尽管Android系统会确保每一个确定的intent会被系统内置的app(such as the Phone, Email, or Calendar app)之一接收，但是你还是应该在触发一个intent之前做验证是否有App接受这个intent的步骤。

**Caution:** 如果你触发了一个intent，而且没有任何一个app会去接收这个intent，那么你的app会crash。

为了验证是否有合适的activity会响应这个intent,需要执行 `queryIntentActivities()` 来获取到能够接收这个intent的所有activity的list。如果返回的List非空，那么你才可以安全的使用这个intent。例如：

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

如果 `isIntentSafe` 是 true, 那么至少有一个app可以响应这个intent。如果是 false则说明没有app可以handle这个intent。

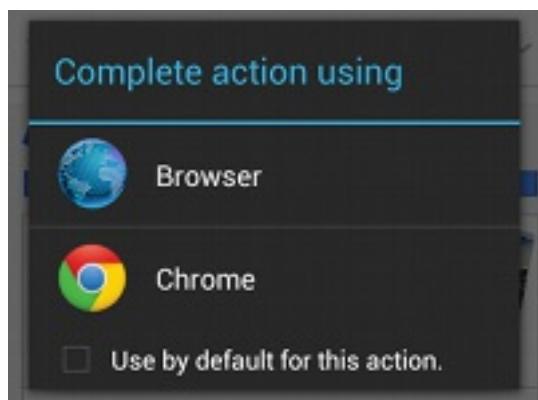
**Note:**你必须在第一次使用之前做这个检查，若是不可行，则应该关闭这个功能。如果你知道某个确切的app能够handle这个intent，你也应该提供给用户去下载这个app的链接。

([see how to link to your product on Google Play](#)).

## Start an Activity with the Intent [使用Intent来启动Activity]

当你创建好了intent并且设置好了extra数据，通过执行startActivity() 来发送到系统。如果系统确定有多个activity可以handle这个intent,它会显示出一个dialog，让用户选择启动哪个app。如果系统发现只有一个app可以handle这个intent，那么就会直接启动这个app。

```
startActivity(intent);
```



下面是一个完整的例子，演示了如何创建一个intent来查看地图，验证有app可以handle这个intent,然后启动它。

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

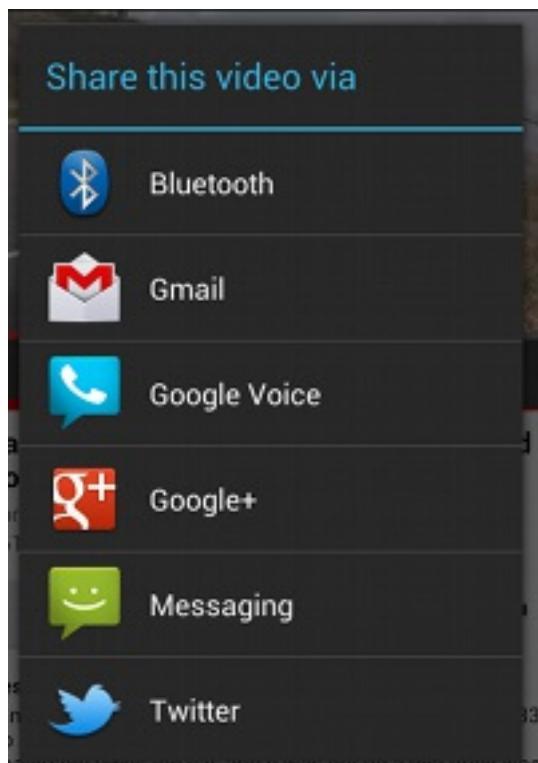
// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

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

## Show an App Chooser[显示一个App选择界面]

请注意，当你发送一个intent，有多个app可以handle的情况，用户可以在弹出dialog的时候，选择默认启动的app（通过勾选dialog下面的选择框，如上图所示）。这个功能对于用户有特殊偏好的时候非常有用（例如用户总是喜欢启动某个app来查看网页，总是喜欢启动某个camera来拍照）。

然而，如果用户希望每次都弹出选择界面，而且每次都不确定会选择哪个app启动，例如分享功能，用户选择分享到哪个app都是不确定的，这个时候，需要强制弹出选择的对话框。（这种情况下用户不能选择默认启动的app）。



为了显示chooser，需要使用createChooser()来创建Intent

```
Intent intent = new Intent(Intent.ACTION_SEND);
...
// Always use string resources for UI text. This says something like "Share via"
String title = getResources().getText(R.string.chooser_title);
// Create and start the chooser
Intent chooser = Intent.createChooser(intent, title);
startActivity(chooser);
```

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/intents/result.html>

# 接收Activity返回的结果

启动另外一个activity并不一定是单向的。你也可以启动另外一个activity然后接受一个result回来。为了接受这个result,你需要使用[startActivityForResult\(\)](#) (而不是[startActivity\(\)](#))。

例如，你的app可以启动一个camera程序并接受拍的照片作为result。或者你可以启动People程序并获取其中联系的人的详情作为result。

当然，被启动的activity需要指定返回的result。它需要把这个result作为另外一个intent对象返回，你的activity需要在[onActivityResult\(\)](#)的回调方法里面去接收result。

**Note:**在执行 `startActivityForResult()`时，你可以使用explicit 或者 implicit 的intent。当你启动另外一个位于你的程序中的activity时，你应该使用explicit intent来确保你可以接收到期待的结果。

## Start the Activity(启动Activity)

对于startActivityForResult() 方法中的intent与之前介绍的并没有什么差异，只不过是需要在这个方法里面多添加一个int类型的参数。这个integer的参数叫做"request code"，它标识了你的请求。当你接收到result Intent时，可以从回调方法里面的参数去判断这个result是否是你想要的。

例如，下面是一个启动activity来选择联系人的例子：

```
static final int PICK_CONTACT_REQUEST = 1; // The request code  
...  
private void pickContact() {  
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, new  
    Uri.parse("content://com.android.contacts/contacts"));  
    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user on  
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST)  
}
```

## Receive the Result(接收Result)

当用户完成了启动之后activity操作之后，系统会调用你的activity的onActivityResult() 回调方法。这个方法有三个参数：

- 你通过startActivityForResult()传递的request code。
- 第二个activity指定的result code。如果操作成功则是RESULT\_OK，如果用户没有操作成功，而是直接点击回退或者其他什么原因，那么则是RESULT\_CANCELED
- 第三个参数则是intent,它包含了返回的result数据部分。

例如，下面是如何处理pick a contact的结果的例子：对应上面的例子

```
@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)  
        }  
    }  
}
```

为了正确的handle这些result，你必须了解那些result intent的格式。对于你自己程序里面的返回result是比较简单的。Apps都会有一些自己的api来指定特定的数据。例如，People app (Contacts app on some older versions) 总是返回一个URI来指定选择的contack，Camera app 则是在data数据区返回一个 Bitmap (see the class about [Capturing Photos](#)).

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/basics/intents/filters.html>

# Intent过滤

前两节课主要讲了从你的app启动另外一个app。但是如果你的app可以响应前面发出的action，那么你的app应该做好响应的准备。例如，如果你创建了一个social app，它可以分享messages 或者 photos 给好友，那么最好你的app能够接收ACTION\_SEND 的intent,这样当用户在其他app触发分享功能的时候，你的app能够出现在待选对话框。

为了使得其他的app能够启动你的activity，你需要在你的manifest文件的 `Activity` 标签下添加 `IntentFilter` 的属性。

当你的app被安装到设备上时，系统可以识别你的intent filter并把这些信息记录下来。当其他app通过执行 `startActivity()` 或者 `startActivityForResult()`方法，并使用implicit intent时，系统可以自动查找出那些可以响应这个intent的activity。

## Add an Intent Filter(添加Intent Filter)

为了尽可能确切的定义你的activity能够handle哪些intent，每一个intent filter都应该尽可能详尽的定义好action与data。

主要有下面三个方面需要定义：

- **Action**:一个想要执行的动作的名称。通常是系统已经定义好的值，例如 ACTION\_SEND 或者 ACTION\_VIEW。
- **Data**:Intent附带数据的描述。可以使用一个或者多个属性，你可以只定义MIME type或者是只指定URI prefix，也可以只定义一个URI scheme，或者是他们综合使用。**Note**: 如果你不想要handle Uri 类型的数据，那么你应该指定 android:mimeType 属性。例如 text/plain or image/jpeg.
- **Category**:提供一个附加的方法来标识这个activity能够handle的intent。通常与用户的手势或者是启动位置有关。系统有支持几种不同的categories,但是大多数都不怎么用的到。而且，所有的implicit intents都默认是 CATEGORY\_DEFAULT 类型的。

```
<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>
```

每一个发送出来的intent只会包含一个action与type，但是handle这个intent的activity的<intent-filter>是可以声明多个<action>,<category>与<data>的。

如果任何的两对action与data是互相矛盾的，你应该创建不同的intent filter来指定特定的action与type。

例如，假设你的activity可以handle 文本与图片，无论是ACTION\_SEND 还是 ACTION\_SENDTO 的intent。在这种情况下，你必须为两个action定义两个不同的intent filter。因为ACTION\_SENDTO intent 必须使用 Uri 类型来指定接收者使用 send 或 sendto 的地址。例如：

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

**Note:**为了接受implicit intents, 你必须在你的intent filter中包含 CATEGORY\_DEFAULT 的 category。 关于更多sending 与 receiving ACTION\_SEND intents来执行social sharing行为的, 请查看上一课: [Getting a Result from an Activity:接收Activity返回的结果](#)

## Handle the Intent in Your Activity [在你的Activity中Handle发送过来的Intent]

为了决定采用哪个action，你可以读取Intent的内容。

你可以执行 `getIntent()` 来获取启动你的activity的那个intent。你可以在activity生命周期的任何时候去执行这个方法，但是你最好是在`onCreate()` 或者 `onStart()` 里面去执行。

```
@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 ...  
    }  
}
```

## Return a Result(返回Result)

如果你想返回一个result给启动你的那个activity，仅仅需要执行 setResult()，通过指定一个result code与result intent。当你的操作成功之后，用户需要返回到原来的activity，通过执行 finish() 来关闭被叫起的activity。

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

你必须总是指定一个result code。通常不是 RESULT\_OK 就是 RESULT\_CANCELED。你可以通过Intent来添加需要返回的数据。

**Note:**默认的result code是RESULT\_CANCELED.因此，如果用户在没有完成操作之前点击了back key，那么之前的activity接受到的result code就是"canceled"。

如果你只是纯粹想要返回一个int来表示某些返回的result数据之一，你可以设置result code为任何大于0的数值。如果你返回的result只是一个int，那么连intent都可以不需要返回了，如下：

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

**Note:**我们没有必要在意你的activity是被用startActivity() 还是 startActivityForResult()方法所叫起的。系统会自动去判断改如何传递result。在不需要的result的case下，result会被自动忽略。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/building-content-sharing.html>

# 分享(Building Apps with Content Sharing)

这一系列课程会教你如何创建可以在不同的应用与设备之间进行分享的应用。

## 分享简单的数据(Sharing Simple Data)

学习如何使得你的应用可以和其他应用进行交互。分享信息，接收信息，为用户数据提供一个简单并且可扩展的方式来执行分享操作。

## 分享文件(Sharing Files)

学习使用一个URI与临时的访问权限来提供安全的文件访问。

## 使用NFC分享文件(Sharing Files with NFC)

学习使用NFC功能实现设备间的文件传递。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/sharing/index.html>

# 分享简单的数据

Android程序中很炫的一个功能是程序之间可以互相通信。为什么要重新发明一个已经存在于另外一个程序中的功能呢，而且这个功能并非自己程序的核心部分。

这一章节会讲述一些通常使用的方法来在不同程序之间通过使用[Intent APIs](#)与[ActionProvider](#)对象来发送与接受content。

## Lessons

- [给其他App发送简单的数据 - Sending Simple Data to Other Apps](#)

学习如何使用intent发送text与binary数据给其他app。

- [接收从其他App返回的数据 - Receiving Simple Data from Other Apps](#)

学习如何通过Intent在你的app中接收来自其他app的text与binary数据。

- [给ActionBar增加分享功能 - Adding an Easy Share Action](#)

学习如何在你的Acitonbar上添加一个分享功能。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/sharing/send.html>

# 给其他App发送简单的数据

当你构建一个intent，你必须指定这个intent需要触发的actions。Android定义了一些actions，包括ACTION\_SEND，这个action表明着这个intent是用来从一个activity发送数据到另外一个activity的，甚至是跨进程之间的。

为了发送数据到另外一个activity，你需要做的是指定数据与数据的类型，系统会识别出能够兼容接受的这些数据的activity并且把这些activity显示给用户进行选择(如果有多个选择)，或者是立即启动Activity(只有一个兼容的选择)。同样的，你可以在manifest文件的Activity描述中添加接受哪些数据类型。

在不同的程序之间使用intent来发送与接受数据是在社交分享内容的时候最常用的方法。Intent使得用户用最常用的程序进行快速简单的分享信息。

**注意:**为ActionBar添加分享功能的最好方法是使用[ShareActionProvider](#)，它能够在API level 14以上进行使用。ShareActionProvider会在第3课中进行详细介绍。

## Send Text Content(分享文本内容)

ACTION\_SEND的最直接与最常用的是从一个Activity发送文本内容到另外一个Activity。例如，Android内置的浏览器可以把当前显示页面的URL作为文本内容分享到其他程序。这是非常有用的，通过邮件或者社交网络来分享文章或者网址给好友。下面是一段Sample Code:

```
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);
```

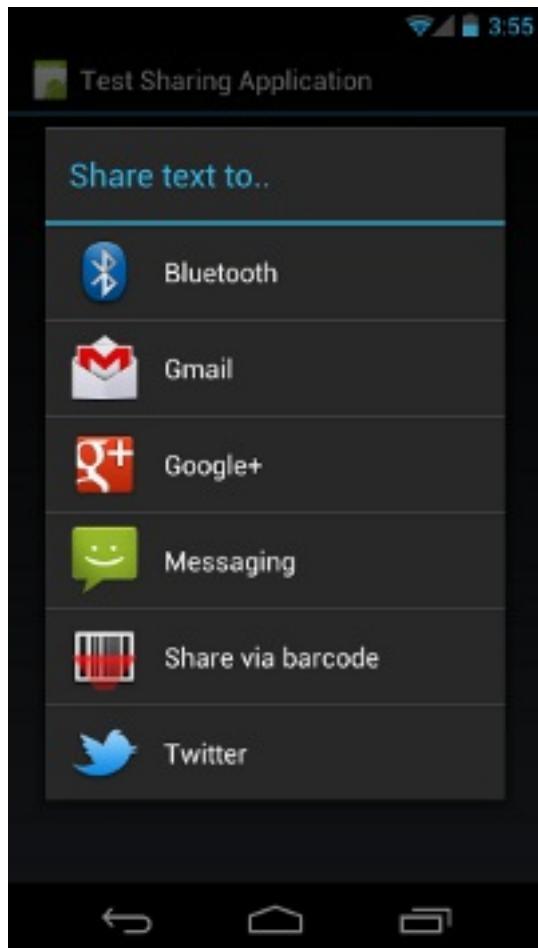
如果设备上有安装某个能够匹配ACTION\_SEND与MIME类型为text/plain程序，那么Android系统会自动把他们都给筛选出来，并呈现Dialog给用户进行选择。如果你为intent调用了Intent.createChooser()，那么Android总是会显示可供选择。这样有一些好处：

- 即使用户之前为这个intent设置了默认的action，选择界面还是会被显示。
- 如果没有匹配的程序，Android会显示系统信息。
- 你可以指定选择界面的标题。

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().getT
```

效果图如下：



Optionally,你可以为intent设置一些标准的附加值，例如：EXTRA\_EMAIL, EXTRA\_CC, EXTRA\_BCC, EXTRA\_SUBJECT.然而，如果接收程序没有针对那些做特殊的处理，则不会有对应的反应。你也可以使用自定义的附加值，但是除非接收的程序能够识别出来，不然没有任何效果。典型的做法是，你使用被接受程序定义的附加值。

注意:一些e-mail程序，例如Gmail,对应接收的是EXTRA\_EMAIL与EXTRA\_CC，他们都是String类型的，可以使用putExtra(string,string[])方法来添加到intent里面。

## Send Binary Content(分享二进制内容)

分享二进制的数据需要结合设置特定的MIME Type，需要在EXTRA\_STREAM里面放置数据的URI,下面有个分享图片的例子，这个例子也可以修改用来分享任何类型的二进制数据：

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);
shareIntent.setType("image/jpeg");
startActivity(Intent.createChooser(shareIntent, getResources().get
```

请注意下面的内容：

- 你可以使用`/*`这样的方式来制定MIME类型，但是这仅仅会match到那些能够处理一般数据类型的Activity(即一般的Activity无法详尽所有的MIME类型)
- 接收的程序需要有访问URI资源的权限。下面有一些方法来处理这个问题：
  - 把文件写到外部存储设备上，类似SDCard，这样所有的app都可以进行读取。使用`Uri.fromFile()`方法来创建可以用在分享时传递到intent里面的`Uri`。然而，请记住，不是所有的程序都遵循`file://`这样格式的`Uri`。
  - 在调用`getFileStreamPath()`返回一个`File`之后，使用带有`MODE_WORLD_READABLE`模式的`openFileOutput()`方法把数据写入到你自己的程序目录下。像上面一样，使用`Uri.fromFile()`创建一个`file://`格式的`Uri`用来添加到`intent`里面进行分享。
  - 媒体文件，例如图片，视频与音频，可以使用`scanFile()`方法进行扫描并存储到`MediaStore`里面。`onScanCompletted()`回调函数会返回一个`content://`格式的`Uri`，这样便于你进行分享的时候把这个uri放到`intent`里面。
  - 图片可以使用`insertImage()`方法直接插入到`MediaStore`系统里面。那个方法会返回一个`content://`格式的`Uri`。
  - 存储数据到你自己的`ContentProvider`里面，确保其他app可以有访问你的`provider`的权限。(或者使用 per-URI permissions)

## Send Multiple Pieces of Content(发送多块内容)

为了同时分享多种不同类型的内容，需要使用ACTION\_SEND\_MULTIPLE与指定到那些数据的URIs列表。MIME类型会根据你分享的混合内容而不同。例如，如果你分享3张JPEG的图片，那么MIME类型仍然是image/jpeg。如果是不同图片格式的话，应该是用image/\*来匹配那些可以接收任何图片类型的activity。如果你需要分享多种不同类型的数据，可以使用\*/\*来表示MIME。像前面描述的那样，这取决于那些接收的程序解析并处理你的数据。下面是一个例子：

```
ArrayList<Uri> imageUrises = new ArrayList<Uri>();
imageUrises.add(imageUri1); // Add your image URIs here
imageUrises.add(imageUri2);

Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND_MULTIPLE);
shareIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, imageUrises);
shareIntent.setType("image/*");
startActivity(Intent.createChooser(shareIntent, "Share images to.."));
```

当然，请确保指定到数据的URIs能够被接收程序所访问(添加访问权限)。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/sharing/receive.html>

# 接收从其他App返回的数据

就像你的程序能够发送数据到其他程序一样，其他程序也能够方便的接收发送过来的数据。需要考虑的是用户与你的程序如何进行交互，你想要从其他程序接收哪些数据类型。例如，一个社交网络程序会希望能够从其他程序接受文本数据，像一个有趣的网址链接。Google+的Android客户端会接受文本数据与单张或者多张图片。用这个app，用户可以简单的从Gallery程序选择一张图片来启动Google+进行发布。

## Update Your Manifest[更新你的manifest文件]

Intent filters通知了Android系统说，一个程序会接受哪些数据。像上一课一样，你可以创建intent filters来表明程序能够接收哪些action。下面是个例子，对三个activit分别指定接受单张图片，文本与多张图片。(这里有不清楚Intent filter的，请参考[Intents and Intent Filters](#))

```
<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>
```

当另外一个程序尝试分享一些东西的时候，你的程序会被呈现在一个列表里面让用户进行选择。如果用户选择了你的程序，相应的activity就应该被调用开启，这个时候就是你如何处理获取到的数据的问题了。

## Handle the Incoming Content[处理接受到的数据]

为了处理从Intent带过来的数据，可以通过调用getIntent()方法来获取到Intent对象。一旦你拿到这个对象，你可以对里面的数据进行判断，从而决定下一步应该做什么。请记住，如果一个activity可以被其他的程序启动，你需要在检查intent的时候考虑这种情况(是被其他程序而调用启动的)。

```
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
        }
    } else if (Intent.ACTION_SEND_MULTIPLE.equals(action) && type
        if (type.startsWith("image/")) {
            handleSendMultipleImages(intent); // Handle multiple i
        }
    } else {
        // Handle other intents, such as being started from the ho
    }
    ...
}

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_ST
    if (imageUri != null) {
        // Update UI to reflect image being shared
    }
}

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

请注意，因为你无法知道其他程序发送过来的数据内容是文本还是其他的数据，因此你需要避免在UI线程里面去处理那些获取到的数据。更新UI可以像更新EditText一样简单，也

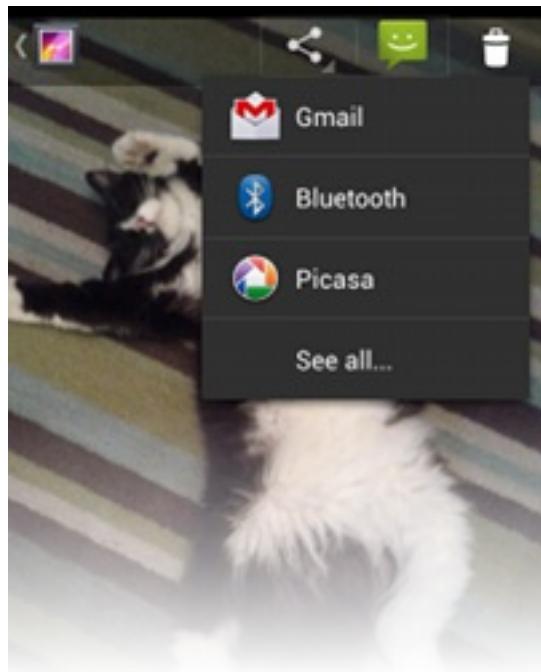
可以是更加复杂一点的操作，例如过滤出感兴趣的图片。It's really specific to your application what happens next.

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/sharing/shareaction.html>

# 添加一个简便的分享动作

这一课会介绍在ActionBar中添加一个高效率且比较友好的Share功能，会使用到ActionProvider(在Android 4.0上才被引进)。它会handle出现share功能的appearance与behavior。在ShareActionProvider的例子里面，你之需要提供一个share intent，剩下的就交给ShareActionProvider来做。



## Update Menu Declarations(更新菜单声明)

使用ShareActionProvider的第一步，在你的menu resources对应item中定义 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.ShareActionPro
...
</menu>
```

这表明了这个item的appearance与function需要与ShareActionProvider匹配。然而，你还是需要告诉provider你想分享的内容。

## Set the Share Intent(设置分享的intent)

为了能够实现ShareActionProvider的功能，你必须提供给它一个intent。这个share intent应该像第一课讲的那样，带有ACTION\_SEND和附加数据(例如EXTRA\_TEXT与EXTRA\_STREAM)的。如何使用ShareActionProvider，请看下面的例子：

```
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.getActionPro  
  
    // Return true to display menu  
    return true;  
}  
  
// Call to update the share intent  
private void setShareIntent(Intent shareIntent) {  
    if (mShareActionProvider != null) {  
        mShareActionProvider.setShareIntent(shareIntent);  
    }  
}
```

你也许在创建菜单的时候仅仅需要设置一次share intent就满足需求了，或者说你可能想先设置share intent，然后根据UI的变化来对intent进行更新。例如，当你在Gallery里面全图查看照片的时候，share intent会在你切换图片的时候进行改变。想要查看更多关于ShareActionProvider的内容，请查看[ActionBar](#)。

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/secure-file-sharing/index.html>

# 分享文件

一个应用经常需要向其他应用发送一个甚至多个文件。例如，一个图库应用可能需要向图片编辑器提供多个文件，或者一个文件管理器可能希望允许用户在外部存储的不同区域之间复制粘贴文件。这里，我们提出一种让应用可以分享文件的方法，即对应用所发出的文件请求进行响应。

在所有情况下，唯一的一个将文件从你的应用发送至另一个应用的安全方法是向接收文件的应用发送这个文件的URI，然后对这个URI授予临时的可访问权限。具有URI临时访问权限的URI是安全的，因为访问权限只授权于接收这个URI的应用，并且它们会自动过期。Android的[FileProvider](#)组件提供了[getUriForFile\(\)](#)方法来创建一个文件的URI。

如果你希望在应用之间共享少量的文本或者数字的数据，你应该发送一个包含该数据的Intent。要学习如何通过Intent发送简单数据，可以阅读：[Sharing Simple Data](#)。

这系列课程将会介绍如何使用Android的[FileProvider](#)组件创建的URI，以及如何向接收URI的应用授予的临时访问权限，来安全地在应用之间共享文件。

# Lessons

- [建立文件分享](#)

学习如何为分享文件初始化你的app。

- [分享文件](#)

学习如何通过生成文件的content URI来传递文件到其他的app。

- [请求分享一个文件](#)

学习如何通过其他app发出文件分享的请求，如何通过URI接收文件以及如何使用URI打开文件。

- [获取文件信息](#)

学习应用如何通过FileProvider提供的content URI获取文件的信息：例如MIME类型，文件大小等。

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/secure-file-sharing/setup-sharing.html>

# 建立文件分享

为了从你的应用安全地将一个文件发送给另一个应用，你需要配置你的应用来提供安全的文件句柄（URI的形式），Android的[FileProvider](#)组件会基于你在XML文件中的具体配置，为文件创建URI。这节课会向你展示如何在你的应用添加[FileProvider](#)的默认的实现，以及如何指定你要共享的文件。

**Note:**[FileProvider](#)是[v4 Support Library](#)中的。关于如何在你的应用中包含此库，可以阅读：[Support Library Setup](#)。

## 指定FileProvider

为你的应用定义一个[FileProvider](#)，需要在你的清单文件中定义一个字段，这个字段指明了需要使用创建URI的权限。除此之外，还需要一个XML文件，它指定了你的应用可以共享的目录路径。

下面的例子展示了如何在清单文件中添加[`<provider>`](#)标签，来指定[FileProvider](#)类，权限和XML文件名：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <application
        ...
        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:grantUriPermissions="true"
            android:exported="false">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/filepaths" />
        </provider>
        ...
    </application>
</manifest>
```

这里，[android:authorities](#)属性字段指定了你希望使用由[FileProvider](#)生成的URI的authority。在这个例子中，这个authority是“com.example.myapp.fileprovider”。对于你自己的应用，定义authority时，是在你的应用包名（[android:package](#)的值）之后追加“fileprovider”。为了学习更多关于authority的知识，可以阅读：[Content URIs](#)，以及[android:authorities](#)。

[`<provider>`](#)下的子标签[`<meta-data>`](#)指定了一个XML文件，它指定了你希望共享的目录路径。“[android:resource](#)”属性字段是这个文件的路径和名字（无“.xml”后缀）。该文件的内容将在下一节讨论。

## 指定可共享目录路径

一旦你在你的清单文件中为你的应用添加了[FileProvider](#)，你需要指定你希望共享文件的目录路径。为了指定这个路径，我们首先在“res/xml/”下创建文件“filepaths.xml”。在这个文件中，为每一个目录添加一个XML标签。下面的例子展示的是一个“res/xml/filepaths.xml”的例子。这个例子也说明了如何在你的内部存储区域共享一个“files/”目录的子目录：

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

在这个例子中，`<files-path>`标签共享的是在你的应用的内部存储中“files/”目录下的目录。“path”属性字段指出了孩子目录为“files/”目录下的子目录“images/”。“name”属性字段告知[FileProvider](#)向在“files/images/”子目录中的文件URI添加一个路径分段（path segment）标记：“myimages”。

`<paths>`标签可以有多个子标签，每一个子标签都指定一个不同的要共享的目录。除了`<files-path>`标签，你可以使用`<external-path>`来分享位于外部存储的文件，而`<cache-path>`标签用来共享在你的内部缓存目录下的目录。学习更多关于指定共享目录的子标签的知识，可以阅读：[FileProvider](#)。

**Note:** XML文件是你定义共享目录的唯一方式，你不可以以代码的形式添加目录。

现在你有一个完整的[FileProvider](#)说明，它在你应用的内部存储中“files/”目录下创建文件的URI，或者是在“files/”中的子目录内的文件创建URI。当你的应用为一个文件创建了URI，它就包含了在`<provider>`标签中指定的Authority（“com.example.myapp.fileprovider”），路径“myimages/”，和文件的名字。

例如，如果你根据这节课的例子定义了一个[FileProvider](#)，然后你需要一个文件“default\_image.jpg”的URI，[FileProvider](#)会返回如下URI：

```
content://com.example.myapp.fileprovider/myimages/default_image.jp
```

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/secure-file-sharing/sharing-file.html>

# 分享文件

一旦你配置了你的应用来使用URI共享文件，你可以响应其他应用关于这些文件的请求。一种响应的方法是在服务端应用端提供一个文件选择接口，它可以由其他应用激活。这种方法可以允许客户端应用端让用户从服务端应用端选择一个文件，然后接收这个文件的URI。

这节课将会向你展示如何在你的应用中创建一个用来选择文件的[Activity](#)，来响应这些索取文件的请求。

## 接收文件请求

为了从客户端应用端接收一个文件索取请求，然后以URI形式进行响应，你的应用应该提供一个选择文件的Activity。客户端应用端通过调用startActivityForResult()来启动这个Activity。该方法包含了一个Intent，它具有ACTION\_PICK的Action。当客户端应用端调用了startActivityForResult()，你的应用可以向客户端应用端返回一个结果，该结果即用户所选文件对应的URI。

学习如何在客户端应用端实现文件索取请求，可以阅读：[请求分享一个文件](#)。

# 创建一个文件选择Activity

为了配置文件选择Activity，我们从在清单文件定义你的Activity开始，在其intent过滤器中，匹配ACTION\_PICK的Action，以及CATEGORY\_DEFAULT和CATEGORY\_OPENABLE的Category。另外，还需要为你的应用设置MIME类型过滤器，来表明你的应用可以向其他应用提供哪种类型的文件。下面的这段代码展示了如何在清单文件中定义新的Activity和intent过滤器：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    ...
    <application>
        ...
        <activity
            android:name=".FileSelectActivity"
            android:label="@+id/File Selector" >
            <intent-filter>
                <action
                    android:name="android.intent.action.PICK" />
                <category
                    android:name="android.intent.category.DEFAULT" />
                <category
                    android:name="android.intent.category.OPENABLE" />
                <data android:mimeType="text/plain" />
                <data android:mimeType="image/*" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## 在代码中定义文件选择Activity

下面，定义一个Activity子类，它用来显示在你内部存储的“files/images/”目录下可以获得的文件，然后允许用户选择期望的文件。下面的代码显示了如何定义这个Activity。并且响应用户的选择：

```
public class MainActivity extends Activity {
    // The path to the root of this app's internal storage
    private File mPrivateRootDir;
    // The path to the "images" subdirectory
    private File mImagesDir;
    // Array of files in the images subdirectory
    File[] mImageFiles;
    // Array of filenames corresponding to mImageFiles
    String[] mImageFilenames;
    // Initialize the Activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Set up an Intent to send back to apps that request a file
        mResultIntent =
            new Intent("com.example.myapp.ACTION_RETURN_FILE");
        // Get the files/ subdirectory of internal storage
        mPrivateRootDir = getFilesDir();
        // Get the files/images subdirectory;
    }
}
```

```
mImagesDir = new File(mPrivateRootDir, "images");
// Get the files in the images subdirectory
mImageFiles = mImagesDir.listFiles();
// Set the Activity's result to null to begin with
setResult(Activity.RESULT_CANCELED, null);
/*
 * Display the file names in the ListView mFileListView.
 * Back the ListView with the array mImageFilenames, which
 * you can create by iterating through mImageFiles and
 * calling File.getAbsolutePath() for each File
 */
...
}

...
}
```

## 响应一个文件选择

一旦一个用户选择了一个共享的文件，你的应用必须明确哪个文件被选择了，然后为这个文件生成一个对应的URI。若[Activity](#)在[ListView](#)中显示了可获得文件的清单，当用户点击了一个文件名时，系统调用了方法[onItemClick\(\)](#)，在该方法中你可以获取被选择的文件。

在[onItemClick\(\)](#)中，为选择的文件文件名获取一个[File](#)对象，然后将它作为参数传递给[getUriForFile\(\)](#)，另外还需传入的参数是你为[FileProvider](#)所指定的[`<provider>`](#)标签值。这个结果URI包含了相应的被访问权限，一个对于文件目录的路径标记（如在XML meta-data中定义的），以及包含扩展名的文件名。有关[FileProvider](#)如何匹配基于XML meta-data的目录路径的信息，可以阅读：[指定可共享目录路径](#)。

下面的例子展示了你如何检测选中的文件并且获得一个URI：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in
    mFileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
    @Override
    /*
     * When a filename in the ListView is clicked, get its
     * content URI and send it to the requesting app
     */
    public void onItemClick(AdapterView<?> adapterView,
                        View view,
                        int position,
                        long rowId) {
    /*
     * Get a File for the selected file name.
     * Assume that the file names are in the
     * mImageFilename array.
     */
    File requestFile = new File(mImageFilename[position]);
    /*
     * Most file-related method calls need to be in
     * try-catch blocks.
     */
    // Use the FileProvider to get a content URI
    try {
        fileUri = FileProvider.getUriForFile(
            MainActivity.this,
            "com.example.myapp.fileprovider",
            requestFile);
    } catch (IllegalArgumentException e) {
        Log.e("File Selector",
              "The selected file can't be shared: " +
              clickedFilename);
    }
    ...
}
});
```

}

记住，你能生成的那些URI所对应的文件，是那些在meta-data文件中包含标签的（即你定义的）目录内的文件，这方面知识在[Specify Sharable Directories](#)中已经讨论过。如果你为一个在你没有指定的目录内的文件调用了[getUriForFile\(\)](#)方法，你会收到一个[IllegalArgumentException](#)。

## 为文件授权

现在你有了你想要共享给其他应用的文件URI，你需要允许客户端应用端访问这个文件。为了允许访问，可以通过将URI添加至一个[Intent](#)，然后为该[Intent](#)设置权限标记。你所授予的权限是临时的，并且当接收应用的任务栈被完成后，会自动过期。

下面的例子展示了如何为文件设置读权限：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks in the ListView
    mFileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
                ...
                if (fileUri != null) {
                    // Grant temporary read permission to the content
                    mResultIntent.addFlags(
                        Intent.FLAG_GRANT_READ_URI_PERMISSION);
                }
                ...
            }
            ...
        });
    ...
}
```

**Caution:** 调用[setFlags\(\)](#)是唯一安全的方法，为你的文件授予临时的被访问权限。避免对文件URI调用[Context.grantUriPermission\(\)](#)，因为通过该方法授予的权限，你只能通过调用[Context.revokeUriPermission\(\)](#)来撤销。

## 与请求应用共享文件

为了与请求文件的应用共享其需要的文件，将包含了URI和响应权限的Intent传递给`setResult()`。当你定义的Activity被结束后，系统会把这个包含了URI的Intent传递给客户端应用。下面的例子展示了你应该如何做：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in
    mListview.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView,
        View view,
        int position,
        long rowId) {
    ...
    if (fileUri != null) {
    ...
        // Put the Uri and MIME type in the result Intent
        mResultIntent.setDataAndType(
            fileUri,
            getContentResolver().getType(fileUri))
        // Set the result
        MainActivity.this.setResult(Activity.RESULT_OK,
            mResultIntent);
    } else {
        mResultIntent.setDataAndType(null, "");
        MainActivity.this.setResult(RESULT_CANCELED,
            mResultIntent);
    }
}
})
});
```

向用户提供一个一旦他们选择了文件就能立即回到客户端应用的方法。一种实现的方法是提供一个勾选框或者一个完成按钮。使用按钮的`android:onClick`属性字段为它关联一个方法。在该方法中，调用`finish()`。例如：

```
public void onDoneClick(View v) {
    // Associate a method with the Done button
    finish();
}
```

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/secure-file-sharing/request-file.html>

# 请求分享一个文件

当一个应用希望访问由其它应用所共享的文件时，请求应用（即客户端）经常会向其它应用（服务端）发送一个文件请求。在大多数情况下，这个请求会在服务端应用启动一个Activity，来显示可以共享的文件。当服务端应用向客户端应用返回了URI后，用户即选择了文件。

这节课将向你展示一个客户端应用如何向服务端应用请求一个文件，接收服务端应用发来的URI，然后使用这个URI打开这个文件。

## 发送一个文件请求

为了向服务端应用发送文件请求，在客户端应用，需要调用[startActivityForResult](#)），同时传递给这个方法一个[Intent](#)，它包含了客户端应用能处理的某个Action，比如[ACTION\\_PICK](#)；以及一个MIME类型。

例如，下面的代码展示了如何向服务端应用发送一个Intent，来启动在[分享文件](#)中提到的[Activity](#)：

```
public class MainActivity extends Activity {
    private Intent mRequestFileIntent;
    private ParcelFileDescriptor mInputPFD;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mRequestFileIntent = new Intent(Intent.ACTION_PICK);
        mRequestFileIntent.setType("image/jpg");
        ...
    }
    ...
    protected void requestFile() {
        /**
         * When the user requests a file, send an Intent to the
         * server app.
         * files.
         */
        startActivityForResult(mRequestFileIntent, 0);
        ...
    }
    ...
}
```

## 访问请求的文件

当服务端应用向客户端应用发回包含URI的Intent时，这个Intent会传递给客户端应用中覆写的onActivityResult()方法当中。一旦客户端应用有了文件的URI，它就可以通过获取其FileDescriptor来访问文件。

文件的安全问题在这一过程中不用过多担心，因为这个客户端应用所收到的所有数据只有URI而已。由于URI不包含目录路径，客户端应用无法查询出或者打开任何服务端应用的其他文件。客户端应用仅仅获取了这个文件的访问渠道和访问的权限。同时访问权限是临时的，一旦这个客户端应用的任务栈被完成了，这个文件将只能被服务端应用访问。

下面的例子展示了客户端应用如何处理发自服务端应用的Intent，以及客户端应用如何使用URI获取FileDescriptor：

```
/*
 * When the Activity of the app that hosts files sets a result
 * finish(), this method is invoked. The returned Intent conta-
 * ins the content URI of a selected file. The result code indicates if
 * the selection worked or not.
 */
@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent returnIntent) {
    // If the selection didn't work
    if (resultCode != RESULT_OK) {
        // Exit without doing anything else
        return;
    } else {
        // Get the file's content URI from the incoming Intent
        Uri returnUri = returnIntent.getData();
        /*
         * Try to open the file for "read" access using the
         * returned URI. If the file isn't found, write to the
         * error log and return.
         */
        try {
            /*
             * Get the content resolver instance for this context
             * to get a ParcelFileDescriptor for the file.
             */
            mInputPFD = getContentResolver().openFileDescriptor(
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Log.e("MainActivity", "File not found.");
            return;
        }
        // Get a regular file descriptor for the file
        FileDescriptor fd = mInputPFD.getFileDescriptor();
        ...
    }
}
```

方法openFileDescriptor()返回一个文件的ParcelFileDescriptor。从这个对象中，客户端应用

可以获取[FileDescriptor](#)对象，然后用户就可以利用这个对象读取这个文件。

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/secure-file-sharing/retrieve-info.html>

# 获取文件信息

当一个客户端应用尝试对一个有URI的文件进行操作时，应用可以向服务端应用索取关于文件的信息，包括文件的数据类型和文件大小。数据类型可以帮助客户端应用确定该文件自己能否处理，文件大小能帮助客户端应用为文件设置合理的缓冲区。

这节课将展示如何通过查询服务端应用的[FileProvider](#)来获取文件的MIME类型和尺寸。

## 获取文件的MIME类型

一个文件的数据类型能够告知客户端应用应该如何处理这个文件的内容。为了得到URI所对应文件的数据类型，客户端应用调用[ContentResolver.getType\(\)](#)。这个方法返回了文件的MIME类型。默认的，一个[FileProvider](#)通过文件的后缀名来确定其MIME类型。

```
...
/*
 * Get the file's content URI from the incoming Intent, then
 * get the file's MIME type
 */
Uri returnUri = returnIntent.getData();
String mimeType = getContentResolver().getType(returnUri);
...
```

## 获取文件名和文件大小

[FileProvider](#)类有一个默认的[query\(\)](#)方法的实现，它返回一个[Cursor](#)，它包含了URI所关联的文件的名字和尺寸。默认的实现返回两列：

### DISPLAY\_NAME

是文件的文件名，一个[String](#)。这个值和[File.getName\(\)](#)所返回的值是一样的。

### SIZE

文件的大小，以字节为单位，一个“long”型。这个值和[File.length\(\)](#)所返回的值是一样的。

客户端应用可以通过将[query\(\)](#)的参数都设置为“null”，只保留URI这一参数，来同时获取文件的[名字](#)和[大小](#)。例如，下面的代码获取一个文件的[名字](#)和[大小](#)，然后在两个[TextView](#)中进行显示：

```
...
/*
 * Get the file's content URI from the incoming Intent,
 * then query the server app to get the file's display name
 * and size.
*/
Uri returnUri = returnIntent.getData();
Cursor returnCursor =
    getContentResolver().query(returnUri, null, null, null
/*
 * Get the column indexes of the data in the Cursor,
 * move to the first row in the Cursor, get the data,
 * and display it.
*/
int nameIndex = returnCursor.getColumnIndex(OpenableColumns.DI
int sizeIndex = returnCursor.getColumnIndex(OpenableColumns.SI
returnCursor.moveToFirst();
TextView nameView = (TextView) findViewById(R.id.filename_text)
TextView sizeView = (TextView) findViewById(R.id.filesize_text)
nameView.setText(returnCursor.getString(nameIndex));
sizeView.setText(Long.toString(returnCursor.getLong(sizeIndex))
...
```

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/beam-files/index.html>

# 使用NFC分享文件

Android允许你通过Android Beam文件传输功能在设备之间传送大文件。这个功能键具有简单的API，同时，它允许用户仅需要点击设备就能启动文件传输的过程。Android Beam会自动地将文件从一台设备拷贝至另外一台，并且在完成时告知用户。

Android Beam文件传输API可以用来处理大量的数据，而在Android4.0（API Level 14）引入的Android BeamNDEF传输API则用来处理少量的数据，比如：URI等一些体积较小的数据。另外，Android Beam是在Android NFC框架中唯一允许你从NFC标签中读取NDEF消息的方法。想要学习更多有关Android Beam的知识，可以阅读：[Beaming NDEF Messages to Other Devices](#)。想要学习更多有关NFC框架的知识，可以阅读：[Near Field Communication](#)。

## Lessons

- [发送文件给其他设备](#)

学习如何发送文件给其他设备。

- [接收其他设备的文件](#)

学习如何接收其他设备发送的文件。

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/beam-files/sending-files.html>

# 发送文件给其他设备

这节课将向你展示如何通过Android Beam文件传输向另一台设备发送大文件。要发送文件，首先需要声明使用NFC和外部存储的权限，你需要测试一下你的设备是否支持NFC，这样，你才能够向Android Beam文件传输提供文件的URI。

使用Android Beam文件传输功能有下列要求：

1. Android Beam文件传输功能只能在Android 4.1（API Level 16）及以上使用。
2. 你希望传送的文件必须放置于外部存储。学习更多关于外部存储的知识，可以阅读：[Using the External Storage](#)。
3. 每个你希望传送的文件必须是全局可读的。你可以通过[File.setReadable\(true, false\)](#)来为文件设置相应的读权限。
4. 你必须提供你要传输文件的URI。Android Beam文件传输无法处理由[FileProvider.getUriForFile](#)生成的URI。

# 在清单文件中声明权限和功能

首先，编辑你的清单文件来声明你的应用所需要声明的权限和功能。

## 声明权限

为了允许你的应用使用Android Beam文件传输控制NFC从外部存储发送文件，你必须在你的应用清单声明下面的权限：

### NFC

允许你的应用通过NFC发送数据。为了声明该权限，添加下面的标签作为一个[<manifest>](#)标签的子标签：

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

### READ\_EXTERNAL\_STORAGE

允许你的应用读取外部存储。为了声明该权限，添加下面的标签作为一个[<manifest>](#)标签的子标签：

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

**Note:** 对于Android 4.2.2 (API Level 17) 及之前的系统版本，这个权限不是必需的。在后续的系统版本中，若应用需要读取外部存储，可能会需要申明该权限。为了保证将来程序稳定性，建议在该权限申明变成必需的之前，就在清单文件中声明好。

## 指定NFC功能

指定你的应用使用NFC，添加[<uses-feature>](#)标签作为一个[<manifest>](#)标签的子标签。设置`android:required`属性字段为`true`，这样可以使得你的应用只有在NFC可以使用时，才能运行。

下面的代码展示了如何指定[<uses-feature>](#)标签：

```
<uses-feature  
        android:name="android.hardware.nfc"  
        android:required="true" />
```

注意，如果你的应用将NFC作为可选的一个功能，但期望在NFC不可使用时程序还能继续执行，你就应该设置`android:required`属性字段为`false`，然后在代码中测试NFC的可用性。

## 指定Android Beam文件传输

由于Android Beam文件传输只能在Android 4.1（API Level 16）及以上的平台使用，如果你的应用将Android Beam文件传输作为一个不可缺少的核心模块，那么你必须指定`<uses-sdk>`标签为：`android:minSdkVersion="16"`。或者，你可以将`android:minSdkVersion`设置为其它值，然后在代码中测试平台版本，这将在下一节中展开。

# 测试设备是否支持Android Beam文件传输

为了在你的应用清单文件中，定义NFC是可选的，你应该使用下面的标签：

```
<uses-feature android:name="android.hardware.nfc" android:required
```

如果你设置了`android:required="false"`，你必须要在代码中测试NFC和Android Beam文件传输是否被支持。

为了在代码中测试Android Beam文件传输，我们先通过[PackageManager.hasSystemFeature\(\)](#)和参数[FEATURE\\_NFC](#)，来测试设备是否支持NFC。下一步，通过[SDK\\_INT](#)的值测试系统版本是否支持Android Beam文件传输。如果Android Beam文件传输是支持的，那么获得一个NFC控制器的实例，它能允许你与NFC硬件进行通信，例如：

```
public class MainActivity extends Activity {  
    ...  
    NfcAdapter mNfcAdapter;  
    // Flag to indicate that Android Beam is available  
    boolean mAndroidBeamAvailable = false;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        ...  
        // NFC isn't available on the device  
        if (!PackageManager.hasSystemFeature(PackageManager.FEATURE_NFC)) {  
            /*  
             * Disable NFC features here.  
             * For example, disable menu items or buttons that act  
             * NFC-related features  
            */  
            ...  
            // Android Beam file transfer isn't supported  
        } else if (Build.VERSION.SDK_INT <  
                  Build.VERSION_CODES.JELLY_BEAN_MR1) {  
            // If Android Beam isn't available, don't continue.  
            mAndroidBeamAvailable = false;  
            /*  
             * Disable Android Beam file transfer features here.  
            */  
            ...  
            // Android Beam file transfer is available, continue  
        } else {  
            mNfcAdapter = NfcAdapter.getDefaultAdapter(this);  
            ...  
        }  
        ...  
    }  
}
```

## 创建一个提供文件的回调函数

一旦你确认了设备支持Android Beam文件传输，那么可以添加一个回调函数，当Android Beam文件传输监测到用户希望与另一个支持NFC的设备发送文件时，系统会调用它。在该回调函数中，返回一个[Uri对象数组](#)，Android Beam文件传输将URI对应的文件拷贝发送给要接收的设备。

要添加这个回调函数，我们需要实现[NfcAdapter.CreateBeamUrisCallback](#)接口，和它的方法：[createBeamUris\(\)](#)，下面是一个例子：

```
public class MainActivity extends Activity {
    ...
    // List of URIs to provide to Android Beam
    private Uri[] mFileUris = new Uri[10];
    ...
    /**
     * Callback that Android Beam file transfer calls to get
     * files to share
     */
    private class FileUriCallback implements
        NfcAdapter.CreateBeamUrisCallback {
        public FileUriCallback() {
        }
        /**
         * Create content URIs as needed to share with another dev
         */
        @Override
        public Uri[] createBeamUris(NfcEvent event) {
            return mFileUris;
        }
    }
    ...
}
```

一旦你实现了这个接口，通过调用[setBeamPushUrisCallback\(\)](#)将回调函数提供给Android Beam文件传输。下面是一个例子：

```
public class MainActivity extends Activity {
    ...
    // Instance that returns available files from this app
    private FileUriCallback mFileUriCallback;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Android Beam file transfer is available, continue
        ...
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        /*
         * Instantiate a new FileUriCallback to handle requests for
         * URIs
         */
    }
}
```

```
mFileUriCallback = new FileUriCallback();
// Set the dynamic callback for URI requests.
mNfcAdapter.setBeamPushUrisCallback(mFileUriCallback, this)
    ...
}
...
}
```

**Note:** 你也可以将[Uri](#)对象数组通过你应用的[NfcAdapter](#)实例，直接提供给NFC框架。如果你能在NFC触碰事件发生之前，定义这些URI，那么你可以选择这个方法。要学习关于这个方法的知识，可以阅读：[NfcAdapter.setBeamPushUris\(\)](#)。

## 定义要发送的文件

为了将一个或更多个文件发送给支持NFC的设备，需要为每一个文件获取一个文件URI（一个具有文件格式（file scheme）的URI），然后将它们添加至一个Uri对象数组中。要传输一个文件，你必须也有读文件的权限。例如，下面的例子展示的是你如何根据文件名获取它的文件URI，然后将URI添加至数组当中：

```
/*
 * Create a list of URIs, get a File,
 * and set its permissions
 */
private Uri[] mFileUrises new Uri[10];
String transferFile = "transferimage.jpg";
File extDir = getExternalFilesDir(null);
File requestFile = new File(extDir, transferFile);
requestFile.setReadable(true, false);
// Get a URI for the File and add it to the list of URIs
fileUri = Uri.fromFile(requestFile);
if (fileUri != null) {
    mFileUrises[0] = fileUri;
} else {
    Log.e("My Activity", "No File URI available for file.");
}
```

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/beam-files/receive-files.html>

# 接收其他设备的文件

Android Beam文件传输将文件拷贝至接收设备上的一个特殊目录。同时使用Android Media Scanner扫描拷贝的文件，并在[MediaStore](#) provider中为媒体文件添加对应的条目记录。这节课将向你展示当文件拷贝完成时要如何响应，以及在接收设备上应该如何放置拷贝的文件。

## 响应请求并显示数据

当Android Beam文件传输将文件拷贝至接收设备后，它会发布一个通知，包含了一个[Intent](#)，它有一个[ACTION\\_VIEW](#)的Action，第一个传输文件的MIME类型，和一个指向第一个文件的URI。当用户点击了这个通知后，intent会被发送至系统。为了让你的应用能够响应这个intent，我们需要为响应的[Activity](#)所对应的[`<activity>`](#)标签添加[`<intent-filter>`](#)标签，在[`<intent-filter>`](#)标签中，添加下面的子标签：

```
<action android:name="android.intent.action.VIEW" />
```

用来匹配通知中的intent。

```
<category android:name="android.intent.category.CATEGORY_DEFAULT" />
```

匹配隐式的[Intent](#)。

```
<data android:mimeType="mime-type" />
```

匹配一个MIME类型。要指定那些你的应用能够处理的类型。

例如，下面的例子展示了如何添加一个intent filter来激活你的activity：

```
<activity
    android:name="com.example.android.nfctransfer.ViewActivity"
    android:label="Android Beam Viewer" >
    ...
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT" />
        ...
    </intent-filter>
</activity>
```

**Note:** 不仅仅只有Android Beam文件传输会发送含有[ACTION\\_VIEW](#)的intent。在接收设备上的其它应用也有可能会发送含有该行为的intent。我们马上会进一步讨论这一问题。

## 请求文件读权限

如果要读取Android Beam文件传输所拷贝到设备上的文件，需要READ\_EXTERNAL\_STORAGE权限。例如：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_ST
```

如果你希望将文件拷贝至你自己应用的存储区，那么需要的权限改为WRITE\_EXTERNAL\_STORAGE，另外WRITE\_EXTERNAL\_STORAGE权限包含了READ\_EXTERNAL\_STORAGE权限。

**Note:** 对于Android 4.2.2 (API Level 17)，READ\_EXTERNAL\_STORAGE权限仅在用户选择要读文件时才是强制需要的。而在今后的版本中会在所有情况下都需要该权限。为了保证应用在未来的兼容性，建议在清单文件中声明该权限。

由于你的应用对于其内部存储区域具有控制权，所以若要将文件拷贝至你应用的内部存储区域，写权限是不需要声明的。

## 获取拷贝文件的目录

Android Beam文件传输一次性将所有文件拷贝到目标设备的一个目录内，Android Beam文件传输通知所发出的[Intent](#)中包含有URI，他指向了第一个传输的文件。然而，你的应用也有可能接收到除了Android Beam文件传输之外的某个来源所发出的含有[ACTION\\_VIEW](#)行为的Intent。为了明确你应该如何处理接收的Intent，你需要检查它的scheme和authority。

为了获得URI的scheme，调用[Uri.getScheme\(\)](#)，下面的代码展示了如何明确架构并处理URI：

```
public class MainActivity extends Activity {  
    ...  
    // A File object containing the path to the transferred files  
    private File mParentPath;  
    // Incoming Intent  
    private Intent mIntent;  
    ...  
    /*  
     * Called from onNewIntent() for a SINGLE_TOP Activity  
     * or onCreate() for a new Activity. For onNewIntent(),  
     * remember to call setIntent() to store the most  
     * current Intent  
     */  
    private void handleViewIntent() {  
        ...  
        // Get the Intent action  
        mIntent = getIntent();  
        String action = mIntent.getAction();  
        /*  
         * For ACTION_VIEW, the Activity is being asked to display  
         * Get the URI.  
         */  
        if (TextUtils.equals(action, Intent.ACTION_VIEW)) {  
            // Get the URI from the Intent  
            Uri beamUri = mIntent.getData();  
            /*  
             * Test for the type of URI, by getting its scheme value  
             */  
            if (TextUtils.equals(beamUri.getScheme(), "file")) {  
                mParentPath = handleFileUri(beamUri);  
            } else if (TextUtils.equals(  
                beamUri.getScheme(), "content")) {  
                mParentPath = handleContentUri(beamUri);  
            }  
        }  
        ...  
    }  
    ...  
}
```

## 从文件URI中获取目录

如果接收的Intent包含一个文件URI，则该URI包含了一个文件的绝对文件名，包括了完整的路径和文件名。对于Android Beam文件传输来说，目录路径指向了其它传输文件的位置（如果有其它传输文件的话），要获得这个目录路径，要取得URI的路径部分（URI中除去“file:”前缀的部分），根据路径创建一个File对象，然后获取这个File的父目录：

```
...
public String handleFileUri(Uri beamUri) {
    // Get the path part of the URI
    String fileName = beamUri.getPath();
    // Create a File object for this filename
    File copiedFile = new File(fileName);
    // Get a string containing the file's parent directory
    return copiedFile.getParent();
}
...
```

## 从内容URI获取目录

如果接收的Intent包含一个内容URI，这个URI可能指向的是一个存储于MediaStore Content Provider的目录和文件名。你可以通过检测URI的authority值来判断是否是MediaStore的内容URI。一个MediaStore的内容URI可能来自Android Beam文件传输也可能来自其它应用，但不管怎么样，你都能根据该内容URI获得一个目录和文件名。

你也可以接收一个ACTION\_VIEW的Intent，它包含有一个content provider的URI，而不是MediaStore，在这种情况下，这个内容URI不包含MediaStore的authority，且这个URI一般不指向一个目录。

Note：对于Android Beam文件传输，如果第一个接收的文件，其MIME类型为“audio/”，“image/”或者“video/\*”，那么你会接收这个在ACTION\_VIEW的Intent中的内容URI。Android Beam文件传输会在它存储传输文件的目录内运行Media Scanner，以此为媒体文件添加索引。同时Media Scanner将结果写入MediaStore的content provider，之后它将第一个文件的内容URI回递给Android Beam文件传输。这个内容URI就是你在通知Intent中所接收到的。要获得第一个文件的目录，你需要使用该内容URI从MediaStore中获取它。

## 指明Content Provider

为了明确你能从内容URI中获取文件目录，你可以通过调用Uri.getAuthority()获取URI的Authority，以此确定与该URI相关联的Content Provider。其结果有两个可能的值：

### MediaStore.AUTHORITY

表明这个URI关联了被MediaStore追踪的一个文件或者多个文件。可以从MediaStore中获取文件的全名，目录名就自然可以从文件全名中获取。

### 其他值

来自其他Content Provider的内容URI。可以显示与该内容URI相关联的数据，但是不要尝试去获取文件目录。

为了`MediaStore`的内容URI中获取目录，执行一个查询操作，它将`Uri`参数指定为收到的内容URI，列名为`MediaColumns.DATA`。返回的`Cursor`包含了完整路径和URI所代表的文件名。该目录路径下还包含了由Android Beam文件传输传送到该设备上的其它文件。

下面的代码展示了你要如何测试内容URI的Authority，并获取传输文件的路径和文件名：

```
...
    public String handleContentUri(Uri beamUri) {
        // Position of the filename in the query Cursor
        int filenameIndex;
        // File object for the filename
        File copiedFile;
        // The filename stored in MediaStore
        String fileName;
        // Test the authority of the URI
        if (!TextUtils.equals(beamUri.getAuthority(), MediaStore.A
            /*
             * Handle content URIs for other content providers
             */
            // For a MediaStore content URI
        } else {
            // Get the column that contains the file name
            String[] projection = { MediaStore.MediaColumns.DATA };
            Cursor pathCursor =
                getContentResolver().query(beamUri, projection,
                null, null, null);
            // Check for a valid cursor
            if (pathCursor != null &&
                pathCursor.moveToFirst()) {
                // Get the column index in the Cursor
                filenameIndex = pathCursor.getColumnIndex(
                    MediaStore.MediaColumns.DATA);
                // Get the full file name including path
                fileName = pathCursor.getString(filenameIndex);
                // Create a File object for the filename
                copiedFile = new File(fileName);
                // Return the parent directory of the file
                return new File(copiedFile.getParent());
            } else {
                // The query didn't work; return null
                return null;
            }
        }
    }
...
}
```

要学习更多关于从Content Provider获取数据的知识，可以阅读：[Retrieving Data from the Provider](#)。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/building-multimedia.html>

# 多媒体

下面的这些课程会教你如何创建更加符合用户期待的富媒体的应用。

## [管理音频播放\(Managing Audio Playback\)](#)

如何响应音频硬件按钮的点击事件，在播放音频的时候请求audio focus，以及如何正确的响应audio focus的改变。

## [拍照\(Capturing Photos\)](#)

如何利用以及存在的相机应用进行拍照，如何直接控制相机硬件实现你自己的相机应用。

## [打印\(Printing Content\)](#)

如何打印照片，HTML文档，自定义的文档。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/managing-audio/index.html>

# 管理音频播放

如果你的App在播放音频，显然用户能够以预期的方式来控制音频是很重要的。为了保证好的用户体验，同样App能够获取音频焦点是很重要的，这样才能确保不会在同一时刻出现多个App的声音。

在学习这个课程后，你将能够创建对硬件音量按钮进行响应的App，当按下音量按钮的时候需要获取到当前音频的焦点，然后以适当的方式改变音量从而进行响应用户的行为。

# Lessons

- [控制音量与音频播放 - Controlling Your App's Volume and Playback](#)

Learn how to ensure your users can control the volume of your app using the hardware or software volume controls and where available the play, stop, pause, skip, and previous media playback keys.

- [管理音频焦点 - Managing Audio Focus](#)

With multiple apps potentially playing audio it's important to think about how they should interact. To avoid every music app playing at the same time, Android uses audio focus to moderate audio playback. Learn how to request the audio focus, listen for a loss of audio focus, and how to respond when that happens.

- [兼容音频输出设备 - Dealing with Audio Output Hardware](#)

Audio can be played from a number of sources. Learn how to find out where the audio is being played and how to handle a headset being disconnected during playback.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/managing-audio/volume-playback.html>

# 控制音量与音频播放

一个好的用户体验是可预期可控的。如果App是在播放音频，那么显然我们需要做到能够通过硬件按钮，软件按钮，蓝牙耳麦等来控制音量。同样的，我们需要能够进行play, stop, pause, skip, and previous等动作，并且进行对应的响应。

## 鉴别使用的是哪个音频流(Identify Which Audio Stream to Use)

首先需要知道的是我们的App会使用到哪些音频流。Android为播放音乐，闹铃，通知铃，来电声音，系统声音，打电话声音与DTMF频道分别维护了一个隔离的音频流。这是我们能够控制不同音频的前提。其中大多数都是被系统限制的，不能胡乱使用。除了你的App是需要做替换闹钟的铃声的操作，那么几乎其他的播放音频操作都是使用"STREAM\_MUSIC"音频流。

## 使用硬件音量键来控制App的音量(Use Hardware Volume Keys to Control Your App's Audio Volume)

默认情况下，按下音量控制键会调节当前被激活的音频流，如果你的App没有在播放任何声音，则会调节响铃的声音。如果是一个游戏或者音乐程序，需要在不管是否目前正在播放歌曲或者游戏目前是否发出声音的时候，按硬件的音量键都会有对应的音量调节。我们需要监听音量键是否被按下，Android提供了setVolumeControlStream()的方法来直接控制指定的音频流。在鉴别出App会使用哪个音频流之后，需要在Activity或者Fragment创建的时候就设置音量控制，这样能确保不管App是否可见，音频控制功能都正常工作。

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

## 使用硬件的播放控制按键来控制App的音频播放(Use Hardware Playback Control Keys to Control Your App's Audio Playback)

媒体播放按钮，例如play, pause, stop, skip, and previous的功能同样可以在一些线控，耳麦或者其他无线控制设备上实现。无论用户按下上面任何设备上的控制按钮，系统都会广播一个带有ACTION\_MEDIA\_BUTTON的Intent。为了响应那些操作，需要像下面一样注册一个BroadcastReceiver在Manifest文件中。

```
<receiver android:name=".RemoteControlReceiver">
    <intent-filter>
        <action android:name="android.intent.action.MEDIA_BUTTON">
    </intent-filter>
</receiver>
```

Receiver需要判断这个广播是来自哪个按钮的操作，Intent在EXTRA\_KEY\_EVENT中包含了KEY的信息，同样KeyEvent类包含了一列KEYCODE\_MEDIA\_的静态变量来表示不同的媒体按钮，例如KEYCODE\_MEDIA\_PLAY\_PAUSE与KEYCODE\_MEDIA\_NEXT.

```
public class RemoteControlReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction()))
            KeyEvent event = (KeyEvent) intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
        if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode())
            // Handle key press.
    }
}
```

因为可能有多个程序都同样监听了哪些控制按钮，那么必须在代码中特意控制当前哪个Receiver会进行响应。下面的例子显示了如何使用AudioManager来注册监听与取消监听，当Receiver被注册上时，它将是唯一响应Broadcast的Receiver。

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE)
...
// Start listening for button presses
am.registerMediaButtonEventReceiver(RemoteControlReceiver);
...
// Stop listening for button presses
am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
```

通常，App需要在Receiver没有激活或者不可见的时候（比如在onStop的方法里面）取消注册监听。但是在媒体播放的时候并没有那么简单，实际上，我们需要在后台播放歌曲的时候同样需要进行响应。一个比较好的注册与取消监听的方法是当程序获取与失去音频焦点的时候进行操作。这个内容会在后面的课程中详细讲解。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/managing-audio/audio-focus.html>

# 管理音频焦点

很多App都可以播放音频，因此在播放前如何获取到音频焦点就显得很重要了，这样可以避免同时出现多个声音，Android使用audio focus来节制音频的播放，仅仅是获取到audio focus的App才能够播放音频。

在App开始播放音频之前，它需要经过发出请求[request]->接受请求[receive]->音频焦点锁定[Audio Focus]的过程。同样，它需要知道如何监听失去音频焦点[lose of audio focus]的事件并进行合适的响应。

**请求获取音频焦点(Request the Audio Focus)**

通过call [requestAudioFocus\(\)](#) 方法来获取你想要获取到的音频流焦点。如果请求成功这个方法会返回 AUDIOFOCUS\_REQUEST\_GRANTED 。

我们必须指定正在使用哪个音频流，而且需要确定请求的是短暂的还是永久的audio focus。

- 短暂的焦点锁定：当期待播放一个短暂的音频的时候（比如播放导航指示）
  - 永久的焦点锁定：当计划播放可预期到的较长的音频的时候（比如播放音乐）

下面是一个在播放音乐的时候请求永久的音频焦点的例子，我们必须在开始播放之前立即请求音频焦点，比如在用户点击播放或者游戏中下一关开始的片头音乐。

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE)
...
// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                                  AudioManager.STREAM_MUSIC,
                                  // Request permanent focus.
                                  AudioManager.AUDIOFOCUS_GAIN);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
    // Start playback.
}
```

一旦结束了播放，需要确保call `abandonAudioFocus()`方法。这样会通知系统说你不再需要获取焦点并且取消注册`AudioManager.OnAudioFocusChangeListener`的监听。在这样释放短暂音频焦点的case下，可以允许任何打断的App继续播放。

```
// Abandon audio focus when playback complete  
am.abandonAudioFocus(afChangeListener);
```

当请求短暂音频焦点的时候，我们可以选择是否开启“ducking”。Ducking是一个特殊的机制使得允许音频间歇性的短暂播放。通常情况下，一个好的App在失去音频焦点的时候它会立即保持安静。如果我们选择在请求短暂音频焦点的时候开启了ducking，那意味着其它App可以继续播放，仅仅是在这一刻降低自己的音量，在短暂重新获取到音频焦点后恢复正常音量(也就是说：不用理会这个请求短暂焦点的请求，这并不会导致目前在播放的音频受到牵制，比如在播放音乐的时候突然出现一个短暂的短信提示声音，这个时候仅仅是把播放歌曲的音量暂时调低，好让短信声能够让用户听到，之后立马恢复正常播放)。

```
if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {  
    // Start playback.  
}
```

## 处理失去音频焦点Handle the Loss of Audio Focus

如果A程序可以请求获取音频焦点，那么在B程序请求获取的时候，A获取到的焦点就会失去。显然我们需要处理失去焦点的事件。

在音频焦点的监听器里面，当接受到描述焦点改变的事件时会触发[onAudioFocusChange\(\)](#)回调方法。对应于获取焦点的三种类型，我们同样会有三种失去焦点的类型。

失去短暂焦点：通常在失去这种焦点的情况下，我们会暂停当前音频的播放或者降低音量，同时需要准备恢复播放在重新获取到焦点之后。

失去永久焦点：假设另外一个程序开始播放音乐等，那么我们的程序就应该有效的结束自己。实用的做法是停止播放，移除button监听，允许新的音频播放器独占监听那些按钮事件，并且放弃自己的音频焦点。

在重新播放器自己的音频之前，我们需要确保用户重新点击自己App的播放按钮等。

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AudioManager.AUDIOFOCUS_LOSS_TRANSIENT
            // Pause playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Resume playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_LOSS) {
            am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
            am.abandonAudioFocus(afChangeListener);
            // Stop playback
        }
    }
};
```

在上面失去短暂焦点的例子中，如果允许ducking，那么我们可以选择“duck”的行为而不是暂停当前的播放。

## Duck! [闪避]

Ducking是一个特殊的机制使得允许音频间歇性的短暂播放。在Ducking的情况下，正常播放的歌曲会降低音量来凸显这个短暂的音频声音，这样既让这个短暂的声音比较突出，又不至于打断正常的声音。

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK) {
            // Lower the volume
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Raise it back to normal
        }
    }
};
```

监听失去音频焦点是最重要的广播之一，但不是唯一的方法。系统广播了一系列的intent来警示你去改变用户的音频使用体验。下节课会演示如何监视那些广播来提升用户的整体体验。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/managing-audio/audio-output.html>

# 兼容音频输出设备

用户在播放音乐的时候有多个选择，可以使用内置的扬声器，有线耳机或者是支持A2DP的蓝牙耳机。【补充：A2DP全名是Advanced Audio Distribution Profile 蓝牙音频传输模型协定！A2DP是能够采用耳机内的芯片来堆栈数据，达到声音的高清晰度。有A2DP的耳机就是蓝牙立体声耳机。声音能达到44.1kHz，一般的耳机只能达到8kHz。如果手机支持蓝牙，只要装载A2DP协议，就能使用A2DP耳机了。还有消费者看到技术参数提到蓝牙V1.0 V1.1 V1.2 V2.0——这些是指蓝牙的技术版本，是指通过蓝牙传输的速度，他们是否支持A2DP具体要看蓝牙产品制造商是否使用这个技术。来自[百度百科](#)】

## Check What Hardware is Being Used(检测目前正在使用的硬件设备)

选择的播放设备会影响App的行为。可以使用AudioManager来查询某个音频输出到扬声器，有线耳机还是蓝牙上。

```
if (isBluetoothA2dpOn()) {  
    // Adjust output for Bluetooth.  
} else if (isSpeakerphoneOn()) {  
    // Adjust output for Speakerphone.  
} else if (isWiredHeadsetOn()) {  
    // Adjust output for headsets  
} else {  
    // If audio plays and noone can hear it, is it still playing?  
}
```

## Handle Changes in the Audio Output Hardware(处理音频输出设备的改变)

当有线耳机被拔出或者蓝牙设备断开连接的时候，音频流会自动输出到内置的扬声器上。假设之前播放声音很大，这个时候突然转到扬声器播放会显得非常嘈杂。

幸运的是，系统会在那种事件发生时会广播带有ACTION\_AUDIO\_BECOMING\_NOISY的intent。无论何时播放音频去注册一个BroadcastReceiver来监听这个intent会是比较好的做法。

在音乐播放器下，用户通常希望发生那样事情的时候能够暂停当前歌曲的播放。在游戏里，通常会选择减低音量。

```
private class NoisyAudioStreamReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intent
                // Pause the playback
        )
    }
}

private IntentFilter intentFilter = new IntentFilter(AudioManager.

private void startPlayback() {
    registerReceiver(myNoisyAudioStreamReceiver(), intentFilter);
}

private void stopPlayback() {
    unregisterReceiver(myNoisyAudioStreamReceiver());
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/camera/index.html>

# 拍照

在多媒体流行之前，世界是沉闷(dismal)并且特色稀少(featureless)的。还记得Gopher? (*Gopher*是计算机上的一个工具软件，是*Internet*提供的一种由菜单式驱动的信息查询工具，采用客户机/服务器模式)。因为你的app将要成为你的用户的生活的一部分，请赋予你的app能够把用户生活装进去的功能。使用内置的Camera，你的程序可以使得用户扩展

(augment) 他们所看的事物，生成唯一的头像，查找角落的人偶(zombies)，或者仅仅是分享他们的经验。

这一章节，会教你如何简单的使用已经存在的Camera程序。在后面的课程中，你会更加深入的 (dive deeper) 学习如何直接控制Camera的硬件。

试试下面的例子程序

[PhotoIntentActivity.zip](#)

# Lessons

- [简单的拍照操作 - Taking Photos Simply](#)

Leverage other applications and capture photos with just a few lines of code.

- [简单的录像操作 - Recording Videos Simply](#)

Leverage other applications and record videos with just a few lines of code.

- [控制相机硬件 - Controlling the Camera](#)

Control the camera hardware directly and implement your own camera application.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/camera/photobasics.html>

# 简单的拍照

假设你想通过你的客户端程序实现一个聚合全球天气的地图，上面会有各地的当前天气图片。那么集合图片只是你程序的一部分。你想要最简单的动作来获取图片，而不是重新发明（reinvent）一个camera。幸运的是，大多数Android设备都已经至少安装了一款相机程序。在这节课中，你会学习，如何拍照

## Request Camera Permission(请求使用相机权限)

在写程序之前，需要在你的程序的manifest文件中添加下面的权限：

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera" />
    ...
</manifest ... >
```

如果你的程序并不需要一定有Camera，可以添加`android:required="false"` 的tag属性。这样的话，Google Play 也会允许没有camera的设备下载这个程序。当然你有必要在使用Camera之前通过`hasSystemFeature(PackageManager.FEATURE_CAMERA)`方法来检查设备上是否有Camera。如果没有，你应该关闭你的Camera相关的功能！

## Take a Photo with the Camera App(使用相机应用程序进行拍照)

Android中的方法是：启动一个Intent来完成你想要的动作。这个步骤包含三部分： Intent本身，启动的外部 Activity, 与一些处理返回照片的代码。如：

```
private void dispatchTakePictureIntent(int requestCode) {  
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
    startActivityForResult(takePictureIntent, requestCode);  
}
```

当然在发出Intent之前，你需要检查是否有app会来handle这个intent，否则会引起启动失败：

```
public static boolean isIntentAvailable(Context context, String action) {  
    final PackageManager packageManager = context.getPackageManager();  
    final Intent intent = new Intent(action);  
    List<ResolveInfo> list =  
        packageManager.queryIntentActivities(intent, PackageManager.MATCH_DEFAULT_ONLY);  
    return list.size() > 0;  
}
```

## View the Photo(查看照片)

Android的Camera程序会把拍好的照片编码为bitmap，使用extra value的方式添加到返回的Intent当中，对应的key为data。

```
private void handleSmallCameraPhoto(Intent intent) {  
    Bundle extras = intent.getExtras();  
    mImageBitmap = (Bitmap) extras.get("data");  
    mImageView.setImageBitmap(mImageBitmap);  
}
```

**Note:** 这仅仅是处理一张很少的缩略图而已，如果是大的全图，需要做更多的事情来避免ANR。

## Save the Photo(保存照片)

如果你提供一个file对象给Android的Camera程序，它会保存这张全图到给定的路径下。你必须提供存储的卷名，文件夹名与文件名。对于2.2以上的系统，如下操作即可：

```
storageDir = new File(  
    Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES  
)  
,  
    getAlbumName()  
)  
;
```

## Set the file name(设置文件名)

正如上面描述的那样，文件的路径会有设备的系统环境决定。你自己需要做的只是定义个不会引起文件名冲突的命名scheme。下面会演示一种解决方案：

```
private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp =
        new SimpleDateFormat("yyyyMMdd_HHmss").format(new Date())
    String imageName = JPEG_FILE_PREFIX + timeStamp + "_";
    File image = File.createTempFile(
        imageName,
        JPEG_FILE_SUFFIX,
        getAlbumDir()
    );
    mCurrentPhotoPath = image.getAbsolutePath();
    return image;
}
```

## **Append the file name onto the Intent(把文件名添加到网络上)**

Once you have a place to save your image, pass that location to the camera application via the Intent.

```
File f = createImageFile();
takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(f))
```

## Add the Photo to a Gallery(添加照片到相册)

对于大多数人来说，最简单查看你的照片的方式是通过系统的Media Provider。下面会演示如何触发系统的Media Scanner来添加你的照片到Media Provider的DB中，这样使得相册程序与其他程序能够读取到那些图片。

```
private void galleryAddPic() {  
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);  
    File f = new File(mCurrentPhotoPath);  
    Uri contentUri = Uri.fromFile(f);  
    mediaScanIntent.setData(contentUri);  
    this.sendBroadcast(mediaScanIntent);  
}
```

## Decode a Scaled Image(解码缩放图片)

在有限的内存下，管理全尺寸的图片会很麻烦。下面会介绍如何缩放图片来适应程序的显示：

```
private void setPic() {  
    // Get the dimensions of the View  
    int targetW = mImageView.getWidth();  
    int targetH = mImageView.getHeight();  
  
    // Get the dimensions of the bitmap  
    BitmapFactory.Options bmOptions = new BitmapFactory.Options();  
    bmOptions.inJustDecodeBounds = true;  
    BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);  
    int photoW = bmOptions.outWidth;  
    int photoH = bmOptions.outHeight;  
  
    // Determine how much to scale down the image  
    int scaleFactor = Math.min(photoW/targetW, photoH/targetH);  
  
    // Decode the image file into a Bitmap sized to fill the View  
    bmOptions.inJustDecodeBounds = false;  
    bmOptions.inSampleSize = scaleFactor;  
    bmOptions.inPurgeable = true;  
  
    Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);  
    mImageView.setImageBitmap(bitmap);  
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/camera/videobasics.html>

# 简单的录像

这节课会介绍如何使用现有的Camera程序来录制一个视频。和拍照一样，我们没有必要去重新发明录像程序。大多数的Android程序都有自带Camera来进行录像。(这一课的内容大多数与前面一课类似，简要带过，一些细节不赘述了)

## Request Camera Permission [请求权限]

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera" />
    ...
</manifest ... >
```

与上一课的拍照一样，你可以在启动Camera之前，使用  
hasSystemFeature(PackageManager.FEATURE\_CAMERA).来检查是否存在Camera。

## Record a Video with a Camera App(使用相机程序来录制视频)

```
private void dispatchTakeVideoIntent() {
    Intent takeVideoIntent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    startActivityForResult(takeVideoIntent, ACTION_TAKE_VIDEO);
}

public static boolean isIntentAvailable(Context context, String action) {
    final PackageManager packageManager = context.getPackageManager();
    final Intent intent = new Intent(action);
    List<ResolveInfo> list =
        packageManager.queryIntentActivities(intent,
            PackageManager.MATCH_DEFAULT_ONLY);
    return list.size() > 0;
}
```

## View the Video(查看视频)

Android的Camera程序会把拍好的视频地址返回。下面的代码演示了，如何查询到这个视频并显示到VideoView.

```
private void handleCameraVideo(Intent intent) {  
    mVideoUri = intent.getData();  
    mVideoView.setVideoURI(mVideoUri);  
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/camera/cameradirect.html>

# 控制相机硬件

在这一节课，我们会讨论如何通过使用framework的APIs来直接控制相机的硬件。直接控制设备的相机，相比起拍照与录像来说，要复杂一些。然而，如果你想要创建一个专业的特殊的相机程序，这节课会演示这部分内容。

## Open the Camera Object(打开相机对象)

获取到 Camera 对象是直接控制Camera的第一步。正如Android自带的相机程序一样，推荐访问Camera的方式是在onCreate方法里面另起一个Thread来打开Camera。这个方法可以避免因为打开工作比较费时而引起ANR。在一个更加基础的实现方法里面，打开Camera的动作被延迟到onResume()方法里面去执行，这样使得代码能够更好的重用，并且保持控制流程不会复杂化。(原文是：In a more basic implementation, opening the camera can be deferred to the onResume() method to facilitate code reuse and keep the flow of control simple.)

在camera正在被另外一个程序使用的时候去执行 Camera.open() 会抛出一个exception，所以需要捕获起来。

```
private boolean safeCameraOpen(int id) {  
    boolean qOpened = false;  
  
    try {  
        releaseCameraAndPreview();  
        mCamera = Camera.open(id);  
        qOpened = (mCamera != null);  
    } catch (Exception e) {  
        Log.e(getString(R.string.app_name), "failed to open Camera"  
              e.printStackTrace();  
    }  
  
    return qOpened;  
}  
  
private void releaseCameraAndPreview() {  
    mPreview.setCamera(null);  
    if (mCamera != null) {  
        mCamera.release();  
        mCamera = null;  
    }  
}
```

自从API level 9开始， camera的framework可以支持多个cameras。如果你使用 open()，你会获取到最后的一个camera。

# Create the Camera Preview(创建相机预览界面)

拍照通常需要提供一个预览界面来显示待拍的事物。和拍照类似，你需要使用一个 SurfaceView 来展现录制的画面。

## Preview Class

为了显示一个预览界面，你需要一个Preview类。这个类需要实现 android.view.SurfaceHolder.Callback 接口，这个接口用来传递从camera硬件获取的数据到程序。

```
class Preview extends ViewGroup implements SurfaceHolder.Callback

    SurfaceView mSurfaceView;
    SurfaceHolder mHolder;

    Preview(Context context) {
        super(context);

        mSurfaceView = new SurfaceView(context);
        addView(mSurfaceView);

        // Install a SurfaceHolder.Callback so we get notified when
        // underlying surface is created and destroyed.
        mHolder = mSurfaceView.getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    ...
}
```

这个Preview类必须在查看图片之前传递给 Camera 对象。正如下面描述的：

## Set and Start the Preview

一个Camera实例与它相关的preview必须以一种指定的顺序来创建，首先是创建Camera对象。在下面的示例中，初始化camera的动作被封装起来，这样，无论用户想对Camera做任何的改变，都通过执行setCamera() 来呼叫[Camera.startPreview\(\)](#)。Preview对象必须在surfaceChanged() 的回调方法里面去做重新创建的动作。

```
public void setCamera(Camera camera) {
    if (mCamera == camera) { return; }

    stopPreviewAndFreeCamera();

    mCamera = camera;

    if (mCamera != null) {
        List<Size> localSizes = mCamera.getParameters().getSupportedPreviewSizes();
        requestLayout();
    }
}
```

```
try {
    mCamera.setPreviewDisplay(mHolder);
} catch (IOException e) {
    e.printStackTrace();
}

/*
Important: Call startPreview() to start updating the preview.
be started before you can take a picture.
*/
mCamera.startPreview();
}

}
```

## Modify Camera Settings(修改相机设置)

相机设置可以改变拍照的方式，从缩放级别到曝光补偿(exposure compensation)。下面的例子仅仅演示了改变预览大小的设置，更多设置请参考Camera的源代码。

```
public void surfaceChanged(SurfaceHolder holder, int format, int w
    // Now that the size is known, set up the camera parameters an
    // the preview.
    Camera.Parameters parameters = mCamera.getParameters();
    parameters.setPreviewSize(mPreviewSize.width, mPreviewSize.hei
    requestLayout();
    mCamera.setParameters(parameters);

    /*
        Important: Call startPreview() to start updating the preview
        started before you can take a picture.
    */
    mCamera.startPreview();
}
```

## Set the Preview Orientation(设置预览方向)

大多数相机程序会锁定预览为横屏的，因为那是人拍照的自然方式。设置里面并没有阻止你去拍竖屏的照片，这些信息会被记录在EXIF里面。[setCameraDisplayOrientation\(\)](#) 方法可以使得你改变预览的方向，并且不会影响到图片被记录的效果。然而，在Android API level 14之前，你必须在改变方向之前，先停止你的预览，然后才能去重启它。

## Take a Picture(拍一张图片)

只要预览开始之后，可以使用[Camera.takePicture\(\)](#) 方法来拍下一张图片。你可以创建 Camera.PictureCallback 与 Camera.ShutterCallback 对象并传递他们到Camera.takePicture()中。

如果你想要做连拍的动作，你可以创建一个Camera.PreviewCallback 并实现 onPreviewFrame().你还可以选择几个预览帧来进行拍照，或是建立一个延迟拍照的动作。

## Restart the Preview(重启预览)

在图片被获取后，你必须在用户拍下一张图片之前重启预览。在下面的示例中，通过重载 shutter button 来实现重启。

```
@Override  
public void onClick(View v) {  
    switch (mPreviewState) {  
        case K_STATE_FROZEN:  
            mCamera.startPreview();  
            mPreviewState = K_STATE_PREVIEW;  
            break;  
  
        default:  
            mCamera.takePicture( null, rawCallback, null );  
            mPreviewState = K_STATE_BUSY;  
    } // switch  
    shutterBtnConfig();  
}
```

## Stop the Preview and Release the Camera(停止预览并释放相机)

当你的程序在使用Camera之后，有必要做清理的动作。特别是，你必须释放 Camera 对象，不然会引起其他app crash。

那么何时应该停止预览并释放相机呢？在预览的surface被摧毁之后，可以做停止预览与释放相机的动作。如下所示：

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the preview if we
    // have a camera.
    if (mCamera != null) {
        /*
         * Call stopPreview() to stop updating the preview surface.
         */
        mCamera.stopPreview();
    }
}

/**
 * When this function returns, mCamera will be null.
 */
private void stopPreviewAndFreeCamera() {
    if (mCamera != null) {
        /*
         * Call stopPreview() to stop updating the preview surface.
         */
        mCamera.stopPreview();

        /*
         * Important: Call release() to release the camera for use by
         * other applications. Applications should release the camera
         * immediately in onSurfaceDestroyed().
         */
        mCamera.release();

        mCamera = null;
    }
}
```

在这节课的前面，这一些系列的动作也是setCamera()方法的一部分，因此初始化一个camera的动作，总是从停止预览开始的。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/printing/index.html>

# 打印

Android用户经常需要在设备上单独地阅览信息，但也有时候为了分享信息而不得不给其他人看自己的设备屏幕，这显然不是分享信息的好方法。若能够从你的Android应用把希望分享的信息打印出来，这将给用户提供一种从你应用获取更多信息的好办法，这么做还能将信息分享给不使用你的应用的其他人。打印的同时还能允许他们创建信息的快照，而这一切不需要无线网络连接，也不会消耗过多电量。

在Android 4.4（API Level 19）及更高的系统版本中，框架提供了直接从Android应用打印图片和文字的服务。这系列课程将展示如何在你的应用中打印：包括打印图片，HTML页面以及创建自定义的打印文档。

# Lessons

- [打印照片](#)

这节课将向你展示如何打印一个图像。

- [打印HTML文档](#)

这节课将向你展示如何打印一个图像HTML文档。

- [打印自定义文档](#)

这节课将向你展示如何连接到Android打印管理器，创建一个打印适配器并建立要打印的内容。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/printing/photos.html>

# 打印照片

拍摄并分享照片是移动设备最流行的用法之一。如果你的应用拍摄了照片，想要展示他们，或者允许用户共享照片，你就应该考虑在你的应用中可以打印他们。[Android Support Library](#)提供了一个方便的函数，它可以仅仅使用很少量的代码和一些简单的打印布局配置集，就能打印出照片来。

这堂课将向你展示如何使用v4 support library中的[PrintHelper](#)类来打印一幅图片。

## 打印一幅图片

Android Support Library中的[PrintHelper](#)类提供了一个打印图片的简单方法。这个类有一个简单的布局选项：[setScaleMode\(\)](#)，它能允许你使用下面的两个选项之一：

- [SCALE\\_MODE\\_FIT](#): 这个选项会调整图像大小，这样整个图像就会在打印有效区域内全部显示出来（缩放至长和宽都包含在纸张页面内）。
- [SCALE\\_MODE\\_FILL](#): 这个选项同样会调整图像大小使图像充满整个打印有效区域，即让图像充满这个纸张页面。这就意味着如果选择这个选项，那么图片的一部分（顶部和底部，或者左侧和右侧）将无法打印出来。如果你不设置图像布局的选项，该模式将是默认的图像拉伸方式。

这两个[setScaleMode\(\)](#)的图像缩放选项都会保持图像原有的长宽比。下面的代码展示了如何创建一个[PrintHelper](#)类的实例，设置缩放选项，并开始打印进程：

```
private void doPhotoPrint() {  
    PrintHelper photoPrinter = new PrintHelper(getActivity());  
    photoPrinter.setScaleMode(PrintHelper.SCALE_MODE_FIT);  
    Bitmap bitmap = BitmapFactory.decodeResource(getResources(),  
        R.drawable.droids);  
    photoPrinter.printBitmap("droids.jpg - test print", bitmap);  
}
```

这个方法可以作为一个菜单项的行为来被调用。注意对于那些不一定都能支持的菜单项（比如打印），应该放置在“更多菜单（overflow menu）”中。要获取更多知识，可以阅读：[Action Bar](#)。

在[printBitmap\(\)](#)被调用之后，你的应用不再需要其他的操作了。之后Android打印界面就会出现，允许用户选择一个打印机和它的打印选项。之后用户就可以打印图像或者取消这一次操作。如果用户选择了打印图像，那么一个打印的任务就被创建了，并且一个打印的提醒通知会显示在系统的通知栏中。

如果你希望在你的打印输出中包含更多的内容，而不仅仅是一张图片，你就必须构造一个打印文档。这方面知识将会在后面的两节课程中展开。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/printing/html-docs.html>

# 打印HTML文档

在Android上要打印比一副照片更丰富的内容，换句话说，若需要将文本和图片组合在一个打印的文档中。Android框架提供了一种使用HTML语言来组织一个文档并打印的方法，它使用的代码数量是很小的。

[WebView](#)类在Android 4.4（API Level 19）中得到了更新，使得它可以打印HTML内容。这个类允许你加载一个本地的HTML资源或者从一个网页下载一个页面，创建一个打印任务，并把它交给Android打印服务。

这节课将向您展示如何快速地构建一个HTML文档，它包含文本和图片，并使用[WebView](#)来打印它。

## 加载一个HTML文档

用[WebView](#)打印一个HTML文档，会涉及到加载一个HTML资源，或者以String的形式构建一个HTML文档。这一节将描述如何构建一个HTML的字符串并将它加载到[WebView](#)中，以备打印。

这个View对象一般被用来作为一个activity布局的一部分。然而，如果你的应用不使用[WebView](#)，你可以创建一个该类的实例，以进行打印。创建该自定义打印界面的主要步骤是：

1. 在HTML资源加载完毕后，创建一个[WebViewClient](#)用来启动一个打印任务。
2. 加载HTML资源至[WebView](#)对象。

下面的代码展示了如何创建一个简单的[WebViewClient](#)并且加载一个动态创建的HTML文档：

```
private WebView mWebView;

private void doWebViewPrint() {
    // Create a WebView object specifically for printing
    WebView webView = new WebView(getActivity());
    webView.setWebViewClient(new WebViewClient() {

        public boolean shouldOverrideUrlLoading(WebView view,
            return false;
    }

    @Override
    public void onPageFinished(WebView view, String url) {
        Log.i(TAG, "page finished loading " + url);
        createWebPrintJob(view);
        mWebView = null;
    }
});

// Generate an HTML document on the fly:
String htmlDocument = "<html><body><h1>Test Content</h1><p>Tes
    "testing, testing...</p></body></html>";
webView.loadDataWithBaseURL(null, htmlDocument, "text/HTML", "
    // Keep a reference to WebView object until you pass the Print
    // to the PrintManager
    mWebView = webView;
}
```

**Note:** 确保你所调用的生成打印的任务发生在之前那一节所创建的[WebViewClient](#)中的[onPageFinished\(\)](#)方法内。如果你不等待页面加载完毕后再打印，打印的输出可能会不完整或空白，甚至可能会失败。

**Note:** 上面的样例代码维护了一个[WebView](#)对象实例，这样就保证了它不会在打印任务创建之前就被垃圾回收器所回收。请确保你在你的实现中也同样这么做，

否则打印的进程可能会无法继续执行。

如果你希望页面中包含图像，将这个图像文件放置在你的工程的“assets/”目录，并指定一个基URL，作为 [loadDataWithBaseUrl\(\)](#)方法的第一个参数，就像下面所显示的一样：

```
webView.loadDataWithBaseUrl("file:///android_asset/images/", htmlB  
"text/HTML", "UTF-8", null);
```

你也可以加载一个网页来打印，方法是将 [loadDataWithBaseUrl\(\)](#)方法替换为[loadUrl\(\)](#)，如下所示：

```
// Print an existing web page (remember to request INTERNET permis  
webView.loadUrl("http://developer.android.com/about/index.html");
```

当使用[WebView](#)来创建一个打印文档时，你要注意下面的一些限制：

- 你不能为文档添加页眉和页脚，包括页号。
- HTML文档的打印选项不包含选择打印的页数范围，例如：对于一个10页的HTML文档，只打印2到4页是不可以的。
- 一个[WebView](#)的实例只能在同一时间处理一个打印任务。
- 若一个HTML文档包含CSS打印属性，比如一个landscape属性，是不支持的。
- 你不能通过一个HTML文档中的JavaScript脚本来激活打印。

**Note:** 一旦在布局中包含的[WebView](#)对象加载好了文档，就可以打印[WebView](#)对象的内容。

如果你希望创建一个更加自定义化的打印输出并希望可以完全控制打印页面上绘制的内容，可以学习下一节课程：[打印自定义文档](#)

## 创建一个打印任务

在创建了[WebView](#)并加载了你的HTML内容之后，你的应用就基本完成了打印进程中，属于它的任务。下一步是访问[PrintManager](#)，创建一个打印适配器，并在最后，创建一个打印任务。下面的代码展示了如何执行这些步骤：

```
private void createWebPrintJob(WebView webView) {  
  
    // Get a PrintManager instance  
    PrintManager printManager = (PrintManager) getActivity()  
        .getSystemService(Context.PRINT_SERVICE);  
  
    // Get a print adapter instance  
    PrintDocumentAdapter printAdapter = webView.createPrintDocumentAdapter();  
  
    // Create a print job with name and adapter instance  
    String jobName = getString(R.string.app_name) + " Document";  
    PrintJob printJob = printManager.print(jobName, printAdapter,  
        new PrintAttributes.Builder().build());  
  
    // Save the job object for later status checking  
    mPrintJobs.add(printJob);  
}
```

这个例子保存了应用使用的[PrintJob](#)对象的实例，这是不必要的。你的应用可以使用这个对象来跟踪打印任务执行时的进度。当你希望监控你应用中的打印任务是否完成，是否失败或者是否被用户取消，这个方法非常有用。另外，不需要创建一个应用内置的通知，因为打印框架会自动的创建一个该打印任务的系统通知。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/printing/custom-docs.html>

# 打印自定义文档

对一些应用，比如绘图应用，页面布局应用和其它一些聚焦于图像输出的应用，创建美丽的打印页面是它的核心功能。在这种情况下，仅仅打印一副图片或一个HTML文档就够了。这种类型应用的打印输出需要精确地控制每个进入页面的东西，包括字体，文本流，分页符，页眉，页脚和一些图像元素。

创建完全由你自定义的打印输出需要投入比之前讨论的方法更多的编程精力。你必须构建可以和打印架构相互通信的组件，调整打印选项，绘制页面元素并管理多个页面的打印。

这节课将向你展示如何连接打印管理器，创建一个打印适配器并构建要打印的内容。

## 连接打印管理器

当你的应用直接管理打印进程，在收到来自用户的打印请求后，第一步要做的是连接Android打印框架并获取一个[PrintManager](#)类的实例。这个类允许你初始化一个打印任务并开始打印生命周期。下面的代码展示了如何获得打印管理器并开始打印进程。

```
private void doPrint() {  
    // Get a PrintManager instance  
    PrintManager printManager = (PrintManager) getActivity()  
        .getSystemService(Context.PRINT_SERVICE);  
  
    // Set job name, which will be displayed in the print queue  
    String jobName = getActivity().getString(R.string.app_name) +  
        " Print Job";  
  
    // Start a print job, passing in a PrintDocumentAdapter implemen-  
    // to handle the generation of a print document  
    printManager.print(jobName, new MyPrintDocumentAdapter(getActivity(),  
        null); //  
}
```

上面的代码展示了如何命名一个打印任务并且设置一个[PrintDocumentAdapter](#)类的实例，它处理打印生命周期的每一步。打印适配器的实现会在下一节中进行讨论。

**Note:** `print()`方法的最后一个参数接收一个[PrintAttributes](#)对象。你可以使用这个参数向打印框架进行一些打印设置，以及基于前一个打印周期的预设，从而改善用户体验。你也可以使用这个参数对被打印对象进行设置一些更符合实际情况的设定，比如当打印一副照片时，设置打印的方向与照片方向一致。

## 创建一个打印适配器

打印适配器和Android打印框架交互并处理打印过程的每一步。这个过程需要用户在创建打印文档前选择打印机和打印选项。这些选项可以影响最终的输出，因为用户选择的打印机可能会有不同的打印的能力，不同的页面尺寸或不同的页面方向。当这些选项配置好之后，这个打印框架会询问你的适配器进行布局和生成一个打印文档，作为最终打印的前期准备。一旦用户点击了打印按钮，框架会接收最终的打印文档，并将它传递给一个打印提供程序（Print Provider）来打印输出。在打印过程中，用户可以选择取消打印，所以你的打印适配器必须监听并响应一个取消请求。

[PrintDocumentAdapter](#)抽象类被设计用来处理打印的生命周期，它有四个主要的回调函数。你必须在你的打印适配器中实现这些方法，以此来恰当地和打印框架交互：

- [onStart\(\)](#): 一旦打印的进程开始了就被调用。如果你的应用有任何一次性的准备任务要执行，比如获取一个要打印数据的快照，那么将它们在此处执行。在你的适配器中，这个回调函数不是必须实现的。
- [onLayout\(\)](#): 每次一个用户改变了一个打印设置并影响了打印的输出时调用，比如改变了页面的尺寸，或者页面的方向，给你的应用一个机会去重新计算要打印页面的布局。这个方法必须返回打印文档包含多少页面。
- [onWrite\(\)](#): 调用它以此将打印页面交付给一个要打印的文件。这个方法可以在被[onLayout\(\)](#)调用后调用一次或多次。
- [onFinish\(\)](#): 一旦打印进程结束后被调用。如果你的应用有任何一次性销毁任务要执行，在这里执行。这个回调函数不是必须实现的。

下面的部分将介绍如何实现布局和写方法，这两个方法是一个打印适配器的核心功能。

**Note:** 这些适配器的回调函数会在你的主线程上被调用。如果你的这些方法的实现需要花费大量的时间，那么应该在一个另外的线程里执行。例如：你可以将布局或者写入打印文档的操作封装在一个[AsyncTask](#)对象中。

### 计算打印文档信息

在一个[PrintDocumentAdapter](#)类的实现中，你的应用必须指定所创建文档的类型并计算所有打印任务所需要的页数，提供被打印页面的尺寸信息。在适配器中[onLayout\(\)](#)方法的实现中会执行这些计算，并提供打印任务输出的信息，这些信息在一个[PrintDocumentInfo](#)类中，包括页数和内容类型。下面的例子展示了[PrintDocumentAdapter](#)中[onLayout\(\)](#)方法的基本实现：

```
@Override
public void onLayout(PrintAttributes oldAttributes,
                     PrintAttributes newAttributes,
                     CancellationSignal cancellationSignal,
                     LayoutResultCallback callback,
                     Bundle metadata) {
    // Create a new PdfDocument with the requested page attributes
    mPdfDocument = new PrintedPdfDocument(getActivity(), newAttrib
    // Respond to cancellation request
    if (cancellationSignal.isCancelled()) {
```

```

        callback.onLayoutCancelled();
        return;
    }

    // Compute the expected number of printed pages
    int pages = computePageCount(newAttributes);

    if (pages > 0) {
        // Return print information to print framework
        PrintDocumentInfo info = new PrintDocumentInfo
            .Builder("print_output.pdf")
            .setContentType(PrintDocumentInfo.CONTENT_TYPE_DOC)
            .setPageCount(pages);
            .build();

        // Content layout reflow is complete
        callback.onLayoutFinished(info, true);
    } else {
        // Otherwise report an error to the print framework
        callback.onLayoutFailed("Page count calculation failed.");
    }
}

```

[onLayout\(\)](#)方法的执行结果有三种：完成，取消或失败（计算布局无法顺利完成时会失败）。你必须通过调用[PrintDocumentAdapter.LayoutResultCallback](#)对象中的适当方法来指明这些结果中的一个。

**Note:** [onLayoutFinished\(\)](#)方法的布尔参数明确了这个布局内容是否和上一次请求相比改变了。恰当地设定了这个参数将避免打印框架不必要的调用[onWrite\(\)](#)方法，缓存之前的打印文档，并提升性能。

[onLayout\(\)](#)的主要工作是计算打印文档的页数，作为交给打印机的参数。如何计算页数则高度依赖于你的应用时如何布局打印页面的。下面的代码展示了页数是如何根据打印方向确定的：

```

private int computePageCount(PrintAttributes printAttributes) {
    int itemsPerPage = 4; // default item count for portrait mode

    MediaSize pageSize = printAttributes.getMediaSize();
    if (!pageSize.isPortrait()) {
        // Six items per page in landscape orientation
        itemsPerPage = 6;
    }

    // Determine number of print items
    int printItemCount = getPrintItemCount();

    return (int) Math.ceil(printItemCount / itemsPerPage);
}

```

## 将打印文档写入文件

当需要将打印输出写入一个文件时，Android打印框架会调用你应用的PrintDocumentAdapter类的onWrite()方法。这个方法的参数指定了哪一页要被打印以及要使用的输出文件。该方法的实现必须将每一个请求页的内容交付给一个多页PDF文档文件。当这个过程结束以后，你需要调用对象的onWriteFinished()回调方法。

**Note:** Android打印框架可能会在每次调用onLayout()后，调用onWrite()方法一次甚至更多次。在这节课当中，有一件非常重要的事情是当打印内容的布局没有变化时，需要将onLayoutFinished()方法的布尔参数设置为“false”，以此避免不必要的重写打印文档的操作。

**Note:** onLayoutFinished()方法的布尔参数明确了这个布局内容是否和上一次请求相比改变了。恰当地设定了这个参数将避免打印框架不必要的调用onLayout()方法，缓存之前的打印文档，并提升性能。

下面的代码展示了使用PrintedPdfDocument类的打印过程基本原理，并创建了一个PDF文件：

```
@Override
public void onWrite(final PageRange[] pageRanges,
                    final ParcelFileDescriptor destination,
                    final CancellationSignal cancellationSignal,
                    final WriteResultCallback callback) {
    // Iterate over each page of the document,
    // check if it's in the output range.
    for (int i = 0; i < totalPages; i++) {
        // Check to see if this page is in the output range.
        if (containsPage(pageRanges, i)) {
            // If so, add it to writtenPagesArray. writtenPagesArr
            // is used to compute the next output page index.
            writtenPagesArray.append(writtenPagesArray.size(), i);
            PdfDocument.Page page = mPdfDocument.startPage(i);

            // check for cancellation
            if (cancellationSignal.isCancelled()) {
                callback.onWriteCancelled();
                mPdfDocument.close();
                mPdfDocument = null;
                return;
            }

            // Draw page content for printing
            drawPage(page);

            // Rendering is complete, so page can be finalized.
            mPdfDocument.finishPage(page);
        }
    }

    // Write PDF document to file
    try {
```

```
mPdfDocument.writeTo(new FileOutputStream(
    destination.getFileDescriptor()));
} catch (IOException e) {
    callback.onWriteFailed(e.toString());
    return;
} finally {
    mPdfDocument.close();
    mPdfDocument = null;
}
PageRange[] writtenPages = computeWrittenPages();
// Signal the print framework the document is complete
callback.onWriteFinished(writtenPages);

...
}
```

这个代码中将PDF页面递交给了drawPage()方法，这个方法会在下一部分介绍。

就布局而言，[onWrite\(\)](#)方法的执行可以有三种结果：完成，取消或者失败（内容无法被写入）。你必须通过调用[PrintDocumentAdapter.WriteResultCallback](#)对象中的适当方法来指明这些结果中的一个。

**Note:** 递交一个打印的文档可以是一个和大量资源相关的操作。为了避免阻塞应用的主UI线程，你应该考虑将页面的递交和写操作在另一个线程中执行，比如在[AsyncTask](#)中执行。关于更多异步任务线程的知识，可以阅读：[Processes and Threads](#)。

## 绘制PDF页面内容

当你的应用打印时，你的应用必须生成一个PDF文档并将它传递给Android打印框架来打印。你可以使用任何PDF生成库来协助完成这个操作。本节将展示如何使用[PrintedPdfDocument](#)类从你的内容生成PDF页面。

[PrintedPdfDocument](#)类使用一个[Canvas](#)对象来在PDF页面上绘制元素，和在activity布局上进行绘制很类似。你可以在打印页面上使用[Canvas](#)的绘图方法绘制元素。下面的代码展示了如何使用相关的函数在PDF文档页面上绘制简单元素：

```
private void drawPage(PdfDocument.Page page) {
    Canvas canvas = page.getCanvas();

    // units are in points (1/72 of an inch)
    int titleBaseLine = 72;
    int leftMargin = 54;

    Paint paint = new Paint();
    paint.setColor(Color.BLACK);
    paint.setTextSize(36);
    canvas.drawText("Test Title", leftMargin, titleBaseLine, paint);

    paint.setTextSize(11);
    canvas.drawText("Test paragraph", leftMargin, titleBaseLine + 18, paint);

    paint.setColor(Color.BLUE);
    canvas.drawRect(100, 100, 172, 172, paint);
}
```

当使用[Canvas](#)在一个PDF页面上绘图时，元素通过单位“点（point）”来指定大小，它是七十二分之一英寸大小。确保你使用这个测量单位来指定页面上的元素大小。在定位绘制的元素时，坐标系的原点（即（0,0））在页面的最左上角。

**Tip:** 虽然[Canvas](#)对象允许你将打印元素放置在一个PDF文档的边缘，但许多打印机并不能在纸张边缘打印。所以当你使用这个类构建一个打印文档时，确保你考虑了那些无法打印的边缘区域。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/building-graphics.html>

# 图像

These classes teach you how to accomplish tasks with graphics that can give your app an edge on the competition. If you want to go beyond the basic user interface to create a beautiful visual experience, these classes will help you get there.

## [高效显示Bitmap - Displaying Bitmaps Efficiently](#)

How to load and process bitmaps while keeping your user interface responsive and avoid exceeding memory limits.

## [使用OpenGL ES显示图像 - Displaying Graphics with OpenGL ES](#)

How to create OpenGL graphics within the Android app framework and respond to touch input.

## [添加动画 - Adding Animations](#)

How to add transitional animations to your user interface.

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/index.html>

# 高效显示Bitmap

这一章节会介绍一些通用的用来处理与加载Bitmap对象的方法，这些技术能够使得不会卡到程序的UI并且避免程序消耗过度内存.如果你不注意这些，Bitmaps会迅速的消耗你可用的内存而导致程序crash,出现下面的异常:`java.lang.OutOfMemoryError: bitmap size exceeds VM budget.`

有许多原因说明在你的Android程序中加载Bitmaps是非常棘手的，需要你特别注意:

- 移动设备的系统资源有限。Android设备对于单个程序至少需要16MB的内存。[Android Compatibility Definition Document \(CDD\)](#), Section 3.7. Virtual Machine Compatibility 给出了对于不同大小与密度的屏幕的最低内存需求. 程序应该在这个最低内存限制下最优化程序的效率。当然，大多数设备的都有更高的限制需求.
- Bitmap会消耗很多内存，特别是对于类似照片等更加丰富的图片. 例如，[Galaxy Nexus](#)的照相功能能够拍摄2592x1936 pixels (5 MB)的图片. 如果bitmap的配置是使用[ARGB\\_8888](#) (the default from the Android 2.3 onward)，那么加载这张照片到内存会大概需要19MB( $2592 \times 1936 \times 4$  bytes) 的内存, 这样的话会迅速消耗掉设备的整个内存.
- Android app的UI通常会在一次操作中立即加载许多张bitmaps. 例如在[ListView](#), [GridView](#) 与 [ViewPager](#) 等组件中通常会需要一次加载许多张bitmaps，而且需要多加载一些内容为了用户可能的滑动操作。

# Lessons

- [Loading Large Bitmaps Efficiently:高效的加载大图](#)

这节课会带领你学习如何解析很大的Bitmaps并且避免超出程序的内存限制。

- [Processing Bitmaps Off the UI Thread:非UI线程处理Bitmaps](#)

处理Bitmap(裁剪,下载等操作)不能执行在主线程。这节课会带领你学习如何使用AsyncTask在后台线程对Bitmap进行处理，并解释如何处理并发带来的问题。

- [Caching Bitmaps:缓存Bitmap](#)

这节课会带领你学习如何使用内存与磁盘缓存来提升加载多张Bitmaps时的响应速度与流畅度。

- [Managing Bitmap Memory:管理Bitmap占用的内存](#)

这节课会介绍为了最大化程序的性能如何管理Bitmap的内存占用。

- [Displaying Bitmaps in Your UI](#)

这节课会把前面介绍的内容综合起来，演示如何在类似[ViewPager](#)与[GridView](#)的控件中使用后台线程与缓存进行加载多张Bitmaps。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/load-bitmap.html>

# Loading Large Bitmaps Efficiently(有效地加载大尺寸位图)

图片有不同的形状与大小。在大多数情况下它们的实际大小都比需要呈现出来的要大很多。例如，系统的Gallery程序会显示那些你使用设备camera拍摄的图片，但是那些图片的分辨率通常都比你的设备屏幕分辨率要高很多。

考虑到程序是在有限的内存下工作，理想情况是你只需要在内存中加载一个低分辨率的版本即可。这个低分辨率的版本应该是与你的UI大小所匹配的，这样才便于显示。一个高分辨率的图片不会提供任何可见的好处，却会占用宝贵的(precious)的内存资源，并且会在快速滑动图片时导致(incurs)附加的效率问题。

这一课会介绍如何通过加载一个低版本的图片到内存中去decoding大的bitmaps，从而避免超出程序的内存限制。

## Read Bitmap Dimensions and Type(读取位图的尺寸与类型)

BitmapFactory 类提供了一些decode的方法 ([decodeByteArray\(\)](#), [decodeFile\(\)](#), [decodeResource\(\)](#), etc.) 用来从不同的资源中创建一个Bitmap. 根据你的图片数据源来选择合适的decode方法. 那些方法在构造位图的时候会尝试分配内存, 因此会容易导致OutOfMemory的异常。每一种decode方法都提供了通过[BitmapFactory.Options](#) 来设置一些附加的标记来指定decode的选项。设置[inJustDecodeBounds](#) 属性为true可以在decoding的时候避免内存的分配, 它会返回一个null的bitmap, 但是 outWidth, outHeight 与 outMimeType 还是可以获取。这个技术可以允许你在构造bitmap之前优先读图片的尺寸与类型。

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), R.id.myimage, options)
int imageHeight = options.outHeight;
int imageWidth = options.outWidth;
String imageType = options.outMimeType;
```

为了避免java.lang.OutOfMemory 的异常, 我们在真正decode图片之前检查它的尺寸, 除非你确定这个数据源提供了准确无误的图片且不会导致占用过多的内存。

## Load a Scaled Down Version into Memory(加载一个按比例缩小的版本到内存中)

通过上面的步骤我们已经知道了图片的尺寸，那些数据可以用来决定是应该加载整个图片到内存中还是加一个缩小的版本。下面有一些因素需要考虑：

- 评估加载完整图片所需要耗费的内存。
- 程序在加载这张图片时会涉及到其他内存需求。
- 呈现这张图片的组件的尺寸大小。
- 屏幕大小与当前设备的屏幕密度。

例如，如果把一个原图是1024\*768 pixel的图片显示到ImageView为128\*96 pixel的缩略图就没有必要把整张图片都加载到内存中。

为了告诉decoder去加载一个低版本的图片到内存，需要在你的[BitmapFactory.Options](#) 中设置 inSampleSize 为 true 。For example, 一个分辨率为2048x1536 的图片，如果设置 inSampleSize 为4，那么会产出一个大概为512x384的bitmap。加载这张小的图片仅仅使用大概0.75MB，如果是加载全图那么大概要花费12MB(前提都是bitmap的配置是 ARGB\_8888). 下面有一段根据目标图片大小来计算Sample图片大小的Sample Code:

```
public static int calculateInSampleSize(
    BitmapFactory.Options options, int reqWidth, int reqHe
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {
        if (width > height) {
            inSampleSize = Math.round((float)height / (float)reqHe
        } else {
            inSampleSize = Math.round((float)width / (float)reqWid
        }
    }
    return inSampleSize;
}
```

**Note:** 设置[inSampleSize](#)为2的幂对于decoder会更加的有效率，然而，如果你打算把调整过大小的图片Cache到磁盘上，设置为更加接近的合适大小则能够更加有效的节省缓存的空间。

为了使用这个方法，首先需要设置 [inJustDecodeBounds](#) 为 true, 把options的值传递过来，然后使用 [inSampleSize](#) 的值并设置 inJustDecodeBounds 为 false 来重新Decode一遍。

```
public static Bitmap decodeSampledBitmapFromResource(Resources res
    int reqWidth, int reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensio
```

```
final BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(res, resId, options);

// Calculate inSampleSize
options.inSampleSize = calculateInSampleSize(options, reqWidth);

// Decode bitmap with inSampleSize set
options.inJustDecodeBounds = false;
return BitmapFactory.decodeResource(res, resId, options);
}
```

使用上面这个方法可以简单的加载一个任意大小的图片并显示为100\*100 pixel的缩略图形式。像下面演示的一样：

```
mImageView.setImageBitmap(
    decodeSampledBitmapFromResource(getResources(), R.id.myimage,
```

你可以通过替换合适的[BitmapFactory.decode\\*](#) 方法来写一个类似的方法从其他的数据源进行decode bitmap。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/process-bitmap.html>

# 非UI线程处理Bitmap

在上一课中有介绍一系列的[BitmapFactory.decode\\*](#))方法，当数据源是网络或者是磁盘时(或者是任何实际源不在内存的)，这些方法都不应该在main UI 线程中执行。那些情况下加载数据是不可以预知的，它依赖于许多因素(从网络或者硬盘读取数据的速度, 图片的大小, CPU的速度, etc.)。如果其中任何一个任务卡住了UI thread, 系统会出现ANR的错误。

这一节课会介绍如何使用 AsyncTask 在后台线程中处理bitmap并且演示了如何处理并发(concurrency)的问题。

## Use an AsyncTask(使用AsyncTask)

[AsyncTask](#) 类提供了一个简单的方法来在后台线程执行一些操作，并且可以把后台的结果呈现到UI线程。下面是一个加载大图的示例：

```
class BitmapWorkerTask extends AsyncTask {
    private final WeakReference imageViewReference;
    private int data = 0;

    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage
        imageViewReference = new WeakReference(imageView);
    }

    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        data = params[0];
        return decodeSampledBitmapFromResource(getResources(), data);
    }

    // Once complete, see if ImageView is still around and set bitmap
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
```

为[ImageView](#)使用[WeakReference](#) 确保了 AsyncTask 所引用的资源可以被GC(garbage collected)。因为当任务结束时不能确保 [ImageView](#) 仍然存在，因此你必须在 [onPostExecute\(\)](#) 里面去检查引用。这个ImageView 也许已经不存在了，例如，在任务结束时用户已经不在那个Activity或者是设备已经发生配置改变(旋转屏幕等)。

开始异步加载位图，只需要创建一个新的任务并执行它即可：

```
public void loadBitmap(int resId, ImageView imageView) {
    BitmapWorkerTask task = new BitmapWorkerTask(imageView);
    task.execute(resId);
}
```

## Handle Concurrency(处理并发问题)

通常类似 ListView 与 GridView 等视图组件在使用上面演示的 AsyncTask 方法时会同时带来另外一个问题。为了更有效的处理内存，那些视图的子组件会在用户滑动屏幕时被循环使用。如果每一个子视图都触发一个 AsyncTask，那么就无法确保当前视图在结束task 时，分配的视图已经进入循环队列中给另外一个子视图进行重用。而且，无法确保所有的异步任务能够按顺序执行完毕。

[Multithreading for Performance](#) 这篇博文更进一步的讨论了如何处理并发并且提供了一种解决方法，当任务结束时 ImageView 保存一个最近常使用的 AsyncTask 引用。使用类似的方法，AsyncTask 可以扩展出一个类似的模型。创建一个专用的 Drawable 子类来保存一个可以回到当前工作任务的引用。在这种情况下，BitmapDrawable 被用来作为占位图片，它可以在任务结束时显示到 ImageView 中。

创建一个专用的 [Drawable](#) 的子类来储存返回工作任务的引用。在这种情况下，当任务完成时 [BitmapDrawable](#) 会被使用，placeholder image 才会在 [ImageView](#) 中被显示：

```
static class AsyncDrawable extends BitmapDrawable {
    private final WeakReference<BitmapWorkerTask> bitmapWorkerTaskReference;

    public AsyncDrawable(Resources res, Bitmap bitmap,
        BitmapWorkerTask bitmapWorkerTask) {
        super(res, bitmap);
        bitmapWorkerTaskReference =
            new WeakReference<BitmapWorkerTask>(bitmapWorkerTask);
    }

    public BitmapWorkerTask getBitmapWorkerTask() {
        return bitmapWorkerTaskReference.get();
    }
}
```

在执行 [BitmapWorkerTask](#) 之前，你需要创建一个 AsyncDrawable 并且绑定它到目标组件 ImageView 中：

```
public void loadBitmap(int resId, ImageView imageView) {
    if (cancelPotentialWork(resId, imageView)) {
        final BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        final AsyncDrawable asyncDrawable =
            new AsyncDrawable(getResources(), mPlaceHolderBitmap);
        imageView.setImageDrawable(asyncDrawable);
        task.execute(resId);
    }
}
```

在上面的代码示例中，cancelPotentialWork 方法检查确保了另外一个在 ImageView 中运行的任务得以取消。如果是这样，它通过执行 cancel() 方法来取消之前的一个任务。在小部分情况下，New 出来的任务有可能已经存在，这样就不需要执行这个任务了。下面演示了如何实现一个 cancelPotentialWork。

```
public static boolean cancelPotentialWork(int data, ImageView image
```

```

final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask();

if (bitmapWorkerTask != null) {
    final int bitmapData = bitmapWorkerTask.data;
    if (bitmapData != data) {
        // Cancel previous task
        bitmapWorkerTask.cancel(true);
    } else {
        // The same work is already in progress
        return false;
    }
}
// No task associated with the ImageView, or an existing task
return true;
}

```

在上面有一个帮助方法，`getBitmapWorkerTask()`，被用作检索任务是否已经被分配到指定的 `ImageView`：

```

private static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
    if (imageView != null) {
        final Drawable drawable = imageView.getDrawable();
        if (drawable instanceof AsyncDrawable) {
            final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
            return asyncDrawable.getBitmapWorkerTask();
        }
    }
    return null;
}

```

最后一步是在 `BitmapWorkerTask` 的 `onPostExecute()` 方法里面做更新操作：

```

class BitmapWorkerTask extends AsyncTask {
    ...

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (isCancelled()) {
            bitmap = null;
        }

        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            final BitmapWorkerTask bitmapWorkerTask =
                getBitmapWorkerTask(imageView);
            if (this == bitmapWorkerTask && imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}

```

这个方法不仅仅适用于 ListView 与 GridView 组件，在那些需要循环利用子视图的组件中同样适用。只需要在设置图片到 ImageView 的地方调用 loadBitmap 方法。例如，在 GridView 中实现这个方法会是在 getView() 方法里面调用。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html>

# Cached Bitmap

加载单个Bitmap到UI是简单直接的，但是如果你需要一次加载大量的图片，事情则会变得复杂起来。在大多数情况下(例如在ListView, GridView or ViewPager)，显示图片的数量通常是没有限制的。

通过循环利用子视图可以抑制内存的使用，GC(garbage collector)也会释放那些不再需要使用的bitmap。这些机制都非常好，但是为了保持一个流畅的用户体验，你想要在屏幕滑回来时避免每次重复处理那些图片。内存与磁盘缓存通常可以起到帮助的作用，允许组件快速的重新加载那些处理过的图片。

这一课会介绍在加载多张位图时使用内存Cache与磁盘Cache来提高反应速度与UI的流畅度。

# Use a Memory Cache(使用内存缓存)

内存缓存以花费宝贵的程序内存为前提来快速访问位图。[LruCache](#) 类(在Support Library 中也可以找到) 特别合适用来caching bitmaps，用一个strong referenced的[LinkedHashMap](#) 来保存最近引用的对象，并且在Cache超出设置大小的时候踢出(evict)最近最少使用到的对象。

**Note:** 在过去, 一个比较流行的内存缓存实现方法是使用 SoftReference or WeakReference , 然而这是不推荐的。从Android 2.3 (API Level 9) 开始, GC变得更加频繁的去释放soft/weak references, 这使得他们就显得效率低下. 而且在Android 3.0 (API Level 11)之前, 备份的bitmap是存放在native memory 中, 它不是以可预知的方式被释放, 这样可能导致程序超出它的内存限制而崩溃。

为了给LruCache选择一个合适的大小, 有下面一些因素需要考虑到:

- 你的程序剩下了多少可用的内存?
- 多少图片会被一次呈现到屏幕上? 有多少图片需要准备好以便马上显示到屏幕?
- 设备的屏幕大小与密度是多少? 一个具有特别高密度屏幕(xhdpi)的设备, 像 Galaxy Nexus 会比 Nexus S (hdpi)需要一个更大的Cache来hold住同样数量的图片.
- 位图的尺寸与配置是多少, 会花费多少内存?
- 图片被访问的频率如何? 是其中一些比另外的访问更加频繁吗? 如果是, 也许你想要保存那些最常访问的到内存中, 或者为不同组别的位图(按访问频率分组)设置多个LruCache 对象。
- 你可以平衡质量与数量吗? 某些时候保存大量低质量的位图会非常有用, 在加载更高质量图片的任务则交给另外一个后台线程。

没有指定的大小与公式能够适用与所有的程序, 那取决于分析你的使用情况后提出一个合适的解决方案。一个太小的Cache会导致额外的花销却没有明显的好处, 一个太大的Cache同样会导致java.lang.OutOfMemory的异常(Cache占用太多内存, 其他活动则会因为内存不够而异常), 并且使得你的程序只留下小部分的内存用来工作。

下面是一个为bitmap建立LruCache 的示例:

```
private LruCache mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Get memory class of this device, exceeding this amount will
    // OutOfMemory exception.
    final int memClass = ((ActivityManager) context.getSystemService(
        Context.ACTIVITY_SERVICE)).getMemoryClass();

    // Use 1/8th of the available memory for this memory cache.
    final int cacheSize = 1024 * 1024 * memClass / 8;

    mMemoryCache = new LruCache(cacheSize) {
        @Override
        protected int sizeOf(String key, Bitmap bitmap) {
            // The cache size will be measured in bytes rather than
            return bitmap.getByteCount();
        }
    };
}
```

```

        }
    };
    ...
}

public void addBitmapToMemoryCache(String key, Bitmap bitmap) {
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }
}

public Bitmap getBitmapFromMemCache(String key) {
    return mMemoryCache.get(key);
}

```

**Note:**在上面的例子中,有1/8的程序内存被作为Cache.在一个常见的设备上(hdpi),最小大概有4MB (32/8).如果一个填满图片的GridView组件放置在800x480像素的手机屏幕上,大概会花费1.5MB (800x480x4 bytes),因此缓存的容量大概可以缓存2.5页的图片内容.

当加载位图到 ImageView 时, LruCache 会先被检查是否存在这张图片。如果找到有, 它会被用来立即更新 ImageView 组件, 否则一个后台线程则被触发去处理这张图片。

```

public void loadBitmap(int resId, ImageView imageView) {
    final String imageKey = String.valueOf(resId);

    final Bitmap bitmap = getBitmapFromMemCache(imageKey);
    if (bitmap != null) {
        mImageView.setImageBitmap(bitmap);
    } else {
        mImageView.setImageResource(R.drawable.image_placeholder);
        BitmapWorkerTask task = new BitmapWorkerTask(mImageView);
        task.execute(resId);
    }
}

```

上面的程序中 [BitmapWorkerTask](#) 也需要做添加到内存Cache中的动作:

```

class BitmapWorkerTask extends AsyncTask {
    ...
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        final Bitmap bitmap = decodeSampledBitmapFromResource(
            getResources(), params[0], 100, 100));
        addBitmapToMemoryCache(String.valueOf(params[0]), bitmap);
        return bitmap;
    }
    ...
}

```

}

## Use a Disk Cache(使用磁盘缓存)

内存缓存能够提高访问最近查看过的位图，但是你不能保证这个图片会在Cache中。像类似 GridView 等带有大量数据的组件很容易就填满内存Cache。你的程序可能会被类似 Phone call 等任务而中断，这样后台程序可能会被杀死，那么内存缓存就会被销毁。一旦用户恢复前面的状态，你的程序就又需要为每个图片重新处理。

磁盘缓存磁盘缓存可以用来保存那些已经处理好的位图，并且在那些图片在内存缓存中不可用时减少加载的次数。当然从磁盘读取图片会比从内存要慢，而且读取操作需要在后台线程中处理，因为磁盘读取操作是不可预期的。

**Note:**如果图片被更频繁的访问到，也许使用 [ContentProvider](#) 会更加的合适，比如在Gallery程序中。

在下面的sample code中实现了一个基本的 DiskLruCache。然而，Android 4.0 的源代码提供了一个更加robust并且推荐使用的DiskLruCache 方案。

(libcore/luni/src/main/java/libcore/io/DiskLruCache.java). 因为向后兼容，所以在前面发布的Android版本中也可以直接使用。(quick search 提供了一个实现这个解决方案的示例)。

```
private DiskLruCache mDiskCache;
private static final int DISK_CACHE_SIZE = 1024 * 1024 * 10; // 10
private static final String DISK_CACHE_SUBDIR = "thumbnails";

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Initialize memory cache
    ...
    File cacheDir = getCacheDir(this, DISK_CACHE_SUBDIR);
    mDiskCache = DiskLruCache.openCache(this, cacheDir, DISK_CACHE_SIZE);
    ...
}

class BitmapWorkerTask extends AsyncTask {
    ...
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        final String imageKey = String.valueOf(params[0]);

        // Check disk cache in background thread
        Bitmap bitmap = getBitmapFromDiskCache(imageKey);

        if (bitmap == null) { // Not found in disk cache
            // Process as normal
            final Bitmap bitmap = decodeSampledBitmapFromResource(
                getResources(), params[0], 100, 100));
        }
    }

    // Add final bitmap to caches
}
```

```

        addBitmapToCache(String.valueOf(imageKey, bitmap);

        return bitmap;
    }

    ...
}

public void addBitmapToCache(String key, Bitmap bitmap) {
    // Add to memory cache as before
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }

    // Also add to disk cache
    if (!mDiskCache.containsKey(key)) {
        mDiskCache.put(key, bitmap);
    }
}

public Bitmap getBitmapFromDiskCache(String key) {
    return mDiskCache.get(key);
}

// Creates a unique subdirectory of the designated app cache direc
// but if not mounted, falls back on internal storage.
public static File getCacheDir(Context context, String uniqueName)
    // Check if media is mounted or storage is built-in, if so, tr
    // otherwise use internal cache dir
    final String cachePath = Environment.getExternalStorageState()
        || !Environment.isExternalStorageRemovable() ?
            context.getExternalCacheDir().getPath() : cont

    return new File(cachePath + File.separator + uniqueName);
}

```

**Note:**即使是初始化磁盘缓存，也需要进行磁盘操作，所以不应该在主线程中进行。但是这也意味着在初始化之前缓存可以被访问。为了解决这种操作，在上面的实现中，lock object用来确保在磁盘缓存完成初始化之前，app无法对它进行读取。

内存缓存的检查是可以在UI线程中进行的，磁盘缓存的检查需要在后台线程中处理。磁盘操作永远都不应该在UI线程中发生。当图片处理完成后，最后的位图需要添加到内存缓存与磁盘缓存中，方便之后的使用。

# Handle Configuration Changes(处理配置改变)

运行时配置改变，例如屏幕方向的改变会导致Android去destory并restart当前运行的Activity。(关于这一行为的更多信息，请参考[Handling Runtime Changes](#)). 你想要在配置改变时避免重新处理所有的图片，这样才能提供给用户一个良好的平滑过度的体验。

幸运的是，在前面介绍Use a Memory Cache的部分，你已经知道如何建立一个内存缓存。这个缓存可以通过使用一个Fragment去调用[setRetainInstance\(true\)](#) 传递到新的Activity中。在这个activity被recreate之后，这个保留的Fragment会被重新附着上。这样你就可以访问Cache对象，从中获取到图片信息并快速的重新添加到ImageView对象中。

下面配置改变时使用Fragment来重新获取LruCache 的示例：

```
private LruCache mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    RetainFragment mRetainFragment =
        RetainFragment.findOrCreateRetainFragment(getFragmentManager());
    mMemoryCache = mRetainFragment.mRetainedCache;
    if (mMemoryCache == null) {
        mMemoryCache = new LruCache(cacheSize) {
            ... // Initialize cache here as usual
        };
        mRetainFragment.mRetainedCache = mMemoryCache;
    }
    ...
}

class RetainFragment extends Fragment {
    private static final String TAG = "RetainFragment";
    public LruCache mRetainedCache;

    public RetainFragment() {}

    public static RetainFragment findOrCreateRetainFragment(FragmentManager fm) {
        RetainFragment fragment = (RetainFragment) fm.findFragmentByTag(TAG);
        if (fragment == null) {
            fragment = new RetainFragment();
        }
        return fragment;
    }
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
}
}
```

为了测试上面的效果，尝试对比retaining这个Fragment与没有这样做的时候去旋转屏幕。你会发现从内存缓存中重新绘制几乎没有卡的现象，而从磁盘缓存则显得稍慢，如果两个

缓存中都没有，则处理速度像平时一样。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/manage-bitmap-memory.html>

# 管理Bitmap的内存占用

作为缓存Bitmaps的进一步延伸, 为了促进GC与bitmap的重用, 你还有一些特定的事情可以做. 推荐的策略会根据Android的版本不同而有所差异. [BitmapFun](#)的示例程序会演示如何设计你的程序使得能够在不同的Android平台上高效的运行.

我们首先要知道Android管理bitmap memory的演变进程:

- 在Android 2.2 (API level 8)以及之前, 当GC发生时, 你的应用的线程是会stopped的. 这导致了一个滞后, 它会降低效率. 在**Android 2.3**上, 添加了并发GC的机制, 这意味着在一个**bitmap**不再被引用到之后, 内存会被立即**reclaimed**.
- 在Android 2.3.3 (API level 10)已经之后, 一个bitmap的像素级数据是存放在native内存中的. 这些数据与bitmap本身是隔离的, bitmap本身是被存放在Dalvik heap中. 在native内存中的pixel数据不是以可以预测的方式去释放的, 这意味着有可能导致一个程序容易超过它的内存限制并Crash. 在**Android 3.0 (API Level 11)**, pixel数据则是与**bitmap**本身一起存放在**dalvik heap**中.

下面会介绍如何在不同的Android版本上优化bitmap内存使用.

## Manage Memory on Android 2.3.3 and Lower

在Android 2.3.3 (API level 10) 以及更低版本上, 推荐使用[recycle\(\)](#). 如果在你的程序中显示了大量的bitmap数据, 你很可能会遇到[OutOfMemoryError](#)错误. [recycle\(\)](#)方法可以使得程序尽快的reclaim memory.

**Caution:**只有你确保这个bitmap不再需要用到的时候才应该使用recycle(). 如果你执行recycle(), 然后尝试绘画这个bitmap, 你将得到错误:"Canvas: trying to use a recycled bitmap".

下面的例子演示了使用recycle()的例子. 它使用了引用计数的方法(mDisplayRefCount 与 mCacheRefCount)来追踪一个bitmap目前是否有被显示或者是在缓存中. 当下面条件满足时回收bitmap:

- mDisplayRefCount 与 mCacheRefCount 的引用计数均为 0.
- bitmap不为null, 并且它还没有被回收.

```
private int mCacheRefCount = 0;
private int mDisplayRefCount = 0;
...
// Notify the drawable that the displayed state has changed.
// Keep a count to determine when the drawable is no longer displayed.
public void setIsDisplayed(boolean isDisplayed) {
    synchronized (this) {
        if (isDisplayed) {
            mDisplayRefCount++;
            mHasBeenDisplayed = true;
        } else {
            mDisplayRefCount--;
        }
    }
    // Check to see if recycle() can be called.
    checkState();
}

// Notify the drawable that the cache state has changed.
// Keep a count to determine when the drawable is no longer being cached.
public void setIsCached(boolean isCached) {
    synchronized (this) {
        if (isCached) {
            mCacheRefCount++;
        } else {
            mCacheRefCount--;
        }
    }
    // Check to see if recycle() can be called.
    checkState();
}

private synchronized void checkState() {
```

```
// If the drawable cache and display ref counts = 0, and this
// has been displayed, then recycle.
if (mCacheRefCount <= 0 && mDisplayRefCount <= 0 && mHasBeenDi
    && hasValidBitmap()) {
    getBitmap().recycle();
}
}

private synchronized boolean hasValidBitmap() {
    Bitmap bitmap = getBitmap();
    return bitmap != null && !bitmap.isRecycled();
}
```

# Manage Memory on Android 3.0 and Higher

在Android 3.0 (API Level 11) 介绍了[BitmapFactory.Options.inBitmap](#). 如果这个值被设置了, decode方法会在加载内容的时候去reuse已经存在的bitmap. 这意味着bitmap的内存是被reused的, 这样可以提升性能, 并且减少了内存的allocation与de-allocation. 在使用inBitmap时有几个注意点(caveats):

- reused的bitmap必须和原数据内容大小一致, 并且是JPEG 或者 PNG 的格式(或者是某个resource 与 stream).
- reused的bitmap的[configuration](#)值如果有设置, 则会覆盖掉[inPreferredConfig](#)值.
- 你应该总是使用decode方法返回的bitmap, 因为你不可以假设reusing的bitmap是可用的(例如, 大小不对).

## Save a bitmap for later use

下面演示了一个已经存在的bitmap是如何被存放起来以便后续使用的. 当一个应用运行在Android 3.0或者更高的平台上并且bitmap被从LruCache中移除时, bitmap的一个soft reference会被存放在[HashSet](#)中, 这样便于之后有可能被[inBitmap](#)进行reuse:

```
HashSet<SoftReference<Bitmap>> mReusableBitmaps;
private LruCache<String, BitmapDrawable> mMemoryCache;

// If you're running on Honeycomb or newer, create
// a HashSet of references to reusable bitmaps.
if (Utils.hasHoneycomb()) {
    mReusableBitmaps = new HashSet<SoftReference<Bitmap>>();
}

mMemoryCache = new LruCache<String, BitmapDrawable>(mCacheParams.m
    // Notify the removed entry that is no longer being cached.
    @Override
    protected void entryRemoved(boolean evicted, String key,
        BitmapDrawable oldValue, BitmapDrawable newValue) {
        if (RecyclingBitmapDrawable.class.isInstance(oldValue)) {
            // The removed entry is a recycling drawable, so notif
            // that it has been removed from the memory cache.
            ((RecyclingBitmapDrawable) oldValue).setIsCached(false)
        } else {
            // The removed entry is a standard BitmapDrawable.
            if (Utils.hasHoneycomb()) {
                // We're running on Honeycomb or later, so add the
                // to a SoftReference set for possible use with in
                mReusableBitmaps.add
                    (new SoftReference<Bitmap>(oldValue.getBit
                )
            }
        }
    }
}
....
```

## Use an existing bitmap

在运行的程序中，decoder方法会去做检查看是否有可用的bitmap. 例如：

```
public static Bitmap decodeSampledBitmapFromFile(String filename,
    int reqWidth, int reqHeight, ImageCache cache) {

    final BitmapFactory.Options options = new BitmapFactory.Option
    ...
    BitmapFactory.decodeFile(filename, options);
    ...

    // If we're running on Honeycomb or newer, try to use inBitmap
    if (Utils.hasHoneycomb()) {
        addInBitmapOptions(options, cache);
    }
    ...
    return BitmapFactory.decodeFile(filename, options);
}
```

下面的代码演示了上面被执行的addInBitmapOptions()方法. 它会为inBitmap查找一个已经存在的bitmap设置为value. 注意这个方法只是去为inBitmap尝试寻找合适的值，但是并不一定能够找到：

```
private static void addInBitmapOptions(BitmapFactory.Options options
    ImageCache cache) {
    // inBitmap only works with mutable bitmaps, so force the deco
    // return mutable bitmaps.
    options.inMutable = true;

    if (cache != null) {
        // Try to find a bitmap to use for inBitmap.
        Bitmap inBitmap = cache.getBitmapFromReusableSet(options);

        if (inBitmap != null) {
            // If a suitable bitmap has been found, set it as the
            // inBitmap.
            options.inBitmap = inBitmap;
        }
    }
}

// This method iterates through the reusable bitmaps, looking for
// to use for inBitmap:
protected Bitmap getBitmapFromReusableSet(BitmapFactory.Options op
    Bitmap bitmap = null;

    if (mReusableBitmaps != null && !mReusableBitmaps.isEmpty()) {
        final Iterator<SoftReference<Bitmap>> iterator
            = mReusableBitmaps.iterator();
        Bitmap item;

        while (iterator.hasNext()) {
            item = iterator.next().get();

```

```
        if (null != item && item.isMutable()) {
            // Check to see if the item can be used for inBitmap
            if (canUseForInBitmap(item, options)) {
                bitmap = item;

                // Remove from reusable set so it can't be used again
                iterator.remove();
                break;
            }
        } else {
            // Remove from the set if the reference has been cleared
            iterator.remove();
        }
    }

    return bitmap;
}
```

最后，下面这个方法去判断候选bitmap是否满足inBitmap的大小条件：

```
private static boolean canUseForInBitmap(
    Bitmap candidate, BitmapFactory.Options targetOptions) {
    int width = targetOptions.outWidth / targetOptions.inSampleSize;
    int height = targetOptions.outHeight / targetOptions.inSampleSize;

    // Returns true if "candidate" can be used for inBitmap re-use
    // "targetOptions".
    return candidate.getWidth() == width && candidate.getHeight() == height;
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/displaying-bitmaps/display-bitmap.html>

# 在UI上显示Bitmap

这一课会演示如何运用前面几节课的内容，使用后台线程与Cache机制来加载图片到ViewPager与GridView组件，并且学习处理并发与配置改变问题。

# Load Bitmaps into a ViewPager Implementation(实现加载图片到ViewPager)

[swipe view pattern](#)是一个用来切换显示不同详情界面的很好的方法。(关于这种效果请先参看[Android Design: Swipe Views](#)).

你可以通过[PagerAdapter](#)与[ViewPager](#)组件来实现这个效果. 然而, 一个更加合适的Adapter是PagerAdapter 的子类[FragmentStatePagerAdapter](#):它可以在某个ViewPager中的子视图切换出屏幕时自动销毁与保存 Fragments 的状态。这样能够保持消耗更少的内存。

**Note:** 如果你只有为数不多的图片并且确保不会超出程序内存限制, 那么使用PagerAdapter 或 FragmentPagerAdapter 会更加合适。

下面是一个使用ViewPager与ImageView作为子视图的示例。主Activity包含有ViewPager 和 adapter。

```
public class ImageDetailActivity extends FragmentActivity {
    public static final String EXTRA_IMAGE = "extra_image";

    private ImagePagerAdapter mAdapter;
    private ViewPager mPager;

    // A static dataset to back the ViewPager adapter
    public final static Integer[] imageResIds = new Integer[] {
        R.drawable.sample_image_1, R.drawable.sample_image_2,
        R.drawable.sample_image_4, R.drawable.sample_image_5,
        R.drawable.sample_image_7, R.drawable.sample_image_8,
        R.drawable.sample_image_3, R.drawable.sample_image_6
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.image_detail_pager); // Contains j

        mAdapter = new ImagePagerAdapter(getSupportFragmentManager());
        mPager = (ViewPager) findViewById(R.id.pager);
        mPager.setAdapter(mAdapter);
    }

    public static class ImagePagerAdapter extends FragmentStatePage
        private final int mSize;

        public ImagePagerAdapter(FragmentManager fm, int size) {
            super(fm);
            mSize = size;
        }

        @Override
        public int getCount() {
            return mSize;
        }
    }
}
```

```

    @Override
    public Fragment getItem(int position) {
        return ImageDetailFragment.newInstance(position);
    }
}

```

Fragment 里面包含了ImageView 的子组件:

```

public class ImageDetailFragment extends Fragment {
    private static final String IMAGE_DATA_EXTRA = "resId";
    private int mImageNum;
    private ImageView mImageView;

    static ImageDetailFragment newInstance(int imageNum) {
        final ImageDetailFragment f = new ImageDetailFragment();
        final Bundle args = new Bundle();
        args.putInt(IMAGE_DATA_EXTRA, imageNum);
        f.setArguments(args);
        return f;
    }

    // Empty constructor, required as per Fragment docs
    public ImageDetailFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mImageNum = getArguments() != null ? getArguments().getInt("resId");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // image_detail_fragment.xml contains just an ImageView
        final View v = inflater.inflate(R.layout.image_detail_fragment, container, false);
        mImageView = (ImageView) v.findViewById(R.id.imageView);
        return v;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        final int resId = ImageDetailActivity.imageResIds[mImageNum];
        mImageView.setImageResource(resId); // Load image into ImageView
    }
}

```

希望你有发现上面示例存在的问题：在UI Thread中读取图片可能会导致程序ANR。使用在Lesson 2中学习的 AsyncTask 会比较好。

```

public class ImageDetailActivity extends FragmentActivity {
    ...

    public void loadBitmap(int resId, ImageView imageView) {
        mImageView.setImageResource(R.drawable.image_placeholder);
        BitmapWorkerTask task = new BitmapWorkerTask(mImageView);
        task.execute(resId);
    }

    ... // include BitmapWorkerTask class
}

public class ImageDetailFragment extends Fragment {
    ...

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (ImageDetailActivity.class.isInstance(getActivity())) {
            final int resId = ImageDetailActivity.imageResIds[mImageIndex];
            // Call out to ImageDetailActivity to load the bitmap
            ((ImageDetailActivity) getActivity()).loadBitmap(resId);
        }
    }
}

```

在 BitmapWorkerTask 中做一些例如resizing or fetching images from the network，不会卡到UI Thread。如果后台线程不仅仅是做个简单的直接加载动作，增加一个内存Cache或者磁盘 Cache会比较好[参考Lesson 3]，下面是一些为了内存Cache而附加的内容：

```

public class ImageDetailActivity extends FragmentActivity {
    ...

    private LruCache mMemoryCache;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // initialize LruCache as per Use a Memory Cache section
    }

    public void loadBitmap(int resId, ImageView imageView) {
        final String imageKey = String.valueOf(resId);

        final Bitmap bitmap = mMemoryCache.get(imageKey);
        if (bitmap != null) {
            mImageView.setImageBitmap(bitmap);
        } else {
            mImageView.setImageResource(R.drawable.image_placeholder);
            BitmapWorkerTask task = new BitmapWorkerTask(mImageView);
            task.execute(resId);
        }
    }
}

```

```
    ... // include updated BitmapWorkerTask from Use a Memory Cache  
}
```

# Load Bitmaps into a GridView Implementation(实现加载图片到GridView)

[Grid list building block](#) 是一种有效显示大量图片的方式。这样能够一次显示许多图片，而且那些即将被显示的图片也处于准备显示状态。如果你想要实现这种效果，你必须确保UI是流畅的，能够控制内存使用，并且正确的处理并发问题（因为 GridView 会循环使用子视图）。

下面是一个在Fragment里面内置了ImageView作为GridView子视图的示例：

```
public class ImageGridFragment extends Fragment implements Adapter
    private ImageAdapter mAdapter;

    // A static dataset to back the GridView adapter
    public final static Integer[] imageResIds = new Integer[] {
        R.drawable.sample_image_1, R.drawable.sample_image_2,
        R.drawable.sample_image_4, R.drawable.sample_image_5,
        R.drawable.sample_image_7, R.drawable.sample_image_8,

    // Empty constructor as per Fragment docs
    public ImageGridFragment() { }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mAdapter = new ImageAdapter(getActivity());
    }

    @Override
    public View onCreateView(
        LayoutInflater inflater, ViewGroup container, Bundle s
        final View v = inflater.inflate(R.layout.image_grid_fragme
        final GridView mGridView = (GridView) v.findViewById(R.id.
        mGridView.setAdapter(mAdapter);
        mGridView.setOnItemClickListener(this);
        return v;
    }

    @Override
    public void onItemClick(AdapterView parent, View v, int positi
        final Intent i = new Intent(getActivity(), ImageDetailActi
        i.putExtra(ImageDetailActivity.EXTRA_IMAGE, position);
        startActivity(i);
    }

    private class ImageAdapter extends BaseAdapter {
        private final Context mContext;

        public ImageAdapter(Context context) {
            super();
            mContext = context;
        }
    }
}
```

```

@Override
public int getCount() {
    return imageResIds.length;
}

@Override
public Object getItem(int position) {
    return imageResIds[position];
}

@Override
public long getItemId(int position) {
    return position;
}

@Override
public View getView(int position, View convertView, ViewGroup
    ImageView imageView;
    if (convertView == null) { // if it's not recycled, initialize some
        imageView = new ImageView(mContext);
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setLayoutParams(new GridView.LayoutParams(
            LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
    } else {
        imageView = (ImageView) convertView;
    }
    //请注意下面的代码
    imageView.setImageResource(imageResIds[position]); // Load
    return imageView;
}
}

```

又一次，这一个实现的问题是图片是在UI线程中被设置。当处理小的图片时可以，但其他需要额外操作的处理，都会使你的UI慢下来。

与前面加载到图片到ViewPager一样，如果setImageResource的操作会比较耗时，有可能会卡到UI Thread。可以使用类似前面异步处理图片与增加缓存的方法来解决那个问题。然而，我们还需要考虑GridView的循环机制所带来的并发问题。为了处理这个问题，请参考前面的课程。下面是一个更新的解决方案：

```

public class ImageGridFragment extends Fragment implements Adapter
{
    ...

    private class ImageAdapter extends BaseAdapter {
        ...

        @Override
        public View getView(int position, View convertView, ViewGroup
            ...
            loadBitmap(imageResIds[position], imageView)
            return imageView;
        }
    }
}

```

```
public void loadBitmap(int resId, ImageView imageView) {
    if (cancelPotentialWork(resId, imageView)) {
        final BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        final AsyncDrawable asyncDrawable =
            new AsyncDrawable(getResources(), mPlaceHolderImage);
        imageView.setImageDrawable(asyncDrawable);
        task.execute(resId);
    }
}

static class AsyncDrawable extends BitmapDrawable {
    private final WeakReference<BitmapWorkerTask> bitmapWorkerTaskReference;

    public AsyncDrawable(Resources res, Bitmap bitmap,
        BitmapWorkerTask bitmapWorkerTask) {
        super(res, bitmap);
        bitmapWorkerTaskReference =
            new WeakReference<BitmapWorkerTask>(bitmapWorkerTask);
    }

    public BitmapWorkerTask getBitmapWorkerTask() {
        return bitmapWorkerTaskReference.get();
    }
}

public static boolean cancelPotentialWork(int data, ImageView imageView) {
    final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask(imageView);

    if (bitmapWorkerTask != null) {
        final int bitmapData = bitmapWorkerTask.data;
        if (bitmapData != data) {
            // Cancel previous task
            bitmapWorkerTask.cancel(true);
        } else {
            // The same work is already in progress
            return false;
        }
    }
    // No task associated with the ImageView, or an existing task
    // has been cancelled
    return true;
}

private static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
    if (imageView != null) {
        final Drawable drawable = imageView.getDrawable();
        if (drawable instanceof AsyncDrawable) {
            final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
            return asyncDrawable.getBitmapWorkerTask();
        }
    }
    return null;
}

... // include updated BitmapWorkerTask class
```

**Note:**对于 ListView 同样可以套用上面的方法。

上面的方法提供了足够的弹性，使得你可以做从网络加载与 Resize 大的数码照片等操作而不至于卡到 UI Thread。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/index.html>

# 使用OpenGL ES显示图像

Android框架提供了大量的标准工具，用来创建吸引人的，功能化的用户接口。然而，如果你希望对你的应用在屏幕上的绘图行为进行更多的控制，或者你在尝试建立三维图像，那么你就需要一个不同的工具了。由Android框架提供的OpenGL ES接口提供了显示高级动画图形的工具，它的功能仅仅受限于你自身的想象力，并且在许多Android设备上搭载的图形处理单元（GPU）都能为其提供GPU加速等性能优化。

这系列课程将教会你使用OpenGL搭建基本的应用，包括配置，绘制对象，移动图形单元及响应点击事件。

这系列课程所使用的样例代码使用的是OpenGL ES 2.0接口，这是当前Android设备所推荐的接口版本。关于跟多OpenGL ES的版本信息，可以阅读：[OpenGL开发手册](#)。

**Note:** 注意不要把OpenGL ES 1.x版本的接口和OpenGL ES 2.0的接口混合调用。  
这两种版本的接口不是通用的。如果尝试混用它们，其输出结果可能会让你感到无奈和沮丧。

## **Sample Code**

[OpenGLES.zip](#)

# Lessons

- [建立OpenGL ES的环境](#)

这节课将向你展示如何配置一个可以画OpenGL图形的应用。

- [定义Shapes](#)

这节课将向你展示如何定义形状，以及你需要了解面和弯曲（faces and winding）这两个概念的原因。

- [绘制Shapes](#)

学习如何在你的应用中绘制OpenGL图形。

- [运用投影与相机视图](#)

学习如何使用投影和相机视图，来获得你所绘制对象的一个新透视效果。

- [添加移动](#)

学习如何对一个OpenGL对象添加基本的运动效果。

- [响应触摸事件](#)

学习如何对OpenGL图形进行基本的交互。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/environment.html>

# 建立OpenGL ES的环境

要在你的应用中使用OpenGL ES绘制图像，你必须为它们创建一个View容器。一个比较直接的方法是同时实现一个[GLSurfaceView](#)和一个[GLSurfaceView.Renderer](#)。[GLSurfaceView](#)是那些用OpenGL所绘制的图形的View容器，而[GLSurfaceView.Renderer](#)则用来控制在该View中绘制的内容。关于这两个类的更多信息，你可以阅读：[OpenGL ES开发手册](#)。

使用[GLSurfaceView](#)只是一种将你的应用与OpenGL ES合并起来的方法。对于一个全屏的或者接近全屏的图形View，使用它是一个理想的选择。开发者如果希望把OpenGL ES的图形融合在布局的一小部分里面，那么可以考虑使用[TextureView](#)。对于自己动手开发的开发者来说（DIY），还可以通过使用[SurfaceView](#)来搭建一个OpenGL ES View，但这将需要编写更多的代码。

在这节课中，我们将解释如何在一个简单地应用activity中完成[GLSurfaceView](#)和[GLSurfaceView.Renderer](#)的最小实现。

## 在配置文件中声明使用OpenGL ES

为了让你的应用能够使用OpenGL ES 2.0接口，你必须将下列声明添加到配置文件当中：

```
<uses-feature android:glEsVersion="0x00020000" android:required="t
```

如果你的应用使用纹理压缩（texture compression），那么你必须对你支持的压缩格式也进行声明，这样的话那些不支持这些格式的设备就不会尝试运行你的应用：

```
<supports-gl-texture android:name="GL_OES_compressed_ETC1_RGB8_tex<supports-gl-texture android:name="GL_OES_compressed_paletted_text
```

更多关于纹理压缩的内容，可以阅读：[OpenGL开发手册](#)。

## 为OpenGL ES图形创建一个activity

使用OpenGL ES的安卓应用就像其它类型的应用有自己的用户接口一样，也拥有多个activity。主要的区别就在于activity布局上的不同。在许多应用中你可能会使用[TextView](#), [Button](#)和[ListView](#)，在使用OpenGL ES的应用中，你需要添加一个[GLSurfaceView](#)。

下面的代码展示了使用一个[GLSurfaceView](#)的最小化实现。它作为主View：

```
public class OpenGLES20Activity extends Activity {  
  
    private GLSurfaceView mGLView;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Create a GLSurfaceView instance and set it  
        // as the ContentView for this Activity.  
        mGLView = new MyGLSurfaceView(this);  
        setContentView(mGLView);  
    }  
}
```

**Note:** OpenGL ES 2.0需要Android 2.2（API Level 8）或更高版本的系统，所以确保你的Android项目的API版本满足该要求。

## 构建一个GLSurfaceView对象

一个[GLSurfaceView](#)是一个特定的View，在View中你可以绘制OpenGL ES图形。不过它自己所做的事情并不多。对于绘制对象的控制实际上是由你在该View中配置的[GLSurfaceView.Renderer](#)所负责的。事实上，这个对象的代码非常简短，你可能会希望跳过继承它，并且只创建一个未经修改的GLSurfaceView实例，不过请不要这么做。你需要继承该类来捕捉触控事件，这方面知识在[响应触摸事件](#)（该系列课程的最后一节课）中会做进一步的介绍。

[GLSurfaceView](#)的核心代码是很小的，所以对于一个快速地实现，通常可以在activity中创建一个内部类并使用它：

```
class MyGLSurfaceView extends GLSurfaceView {  
  
    public MyGLSurfaceView(Context context) {  
        super(context);  
  
        // Set the Renderer for drawing on the GLSurfaceView  
        setRenderer(new MyRenderer());  
    }  
}
```

当使用OpenGL ES 2.0时，你必须对你的[GLSurfaceView](#)构造函数添加另一个调用，以此明确你希望使用的是2.0版本的接口：

```
// Create an OpenGL ES 2.0 context  
setEGLContextClientVersion(2);
```

**Note:** 如果你在使用OpenGL ES 2.0版本的接口，确保在你的应用配置文件中也进行了相关声明。这在之前的章节中已经讨论过了。

另一个对于[GLSurfaceView](#)实现的可选选项，是将渲染模式设置为：[GLSurfaceView.RENDERMODE\\_WHEN\\_DIRTY](#)，其含义是：仅在你的绘画数据发生变化时才在视图中进行绘画操作：

```
// Render the view only when there is a change in the drawing data  
setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
```

这一配置将防止[GLSurfaceView](#)框架被重新绘制，直到你调用了[requestRender\(\)](#)，这将让应用的性能及效率得到提高。

## 构建一个渲染类

[GLSurfaceView.Renderer](#)类的实现，或者说在一个应用中使用OpenGL ES来进行渲染，正是事情变得有趣的地方。该类会控制和其相关联的[GLSurfaceView](#)，决定在上面画什么。一共有三个渲染器的方法被Android系统调用，以此来明确要在[GLSurfaceView](#)上画什么以及如何画：

- [onSurfaceCreated\(\)](#): 调用一次，用来配置视图的OpenGL ES环境。
- [onDrawFrame\(\)](#): 每次重画视图时被调用。
- [onSurfaceChanged\(\)](#): 如果视图的几何形态发生变化时会被调用，例如当设备的屏幕方向发生改变时。

下面是一个非常基本的OpenGL ES渲染器的实现，作用仅仅是在[GLSurfaceView](#)中画一个灰色的背景：

```
public class MyGLRenderer implements GLSurfaceView.Renderer {  
  
    public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
        // Set the background frame color  
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  
    }  
  
    public void onDrawFrame(GL10 unused) {  
        // Redraw background color  
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);  
    }  
  
    public void onSurfaceChanged(GL10 unused, int width, int height)  
    {  
        GLES20.glViewport(0, 0, width, height);  
    }  
}
```

就仅仅是这样！上面的代码创建了一个简单地应用程序，它使用OpenGL显示一个灰色的屏幕。虽然它的代码做的事情并不怎么有趣，但是通过创建这些类，你已经为使用OpenGL绘制图形有了基本的认识和铺垫。

**Note:** 你可能想知道，在你明明使用的是OpenGL ES 2.0的接口的时候，为什么这些方法有一个[GL10](#)的参数。这是因为这些方法在2.0接口中被简单地重用了，以此来保持Android框架尽量简单。

如果你对OpenGL ES接口很熟悉，那么你现在就可以在你的应用中部署一个OpenGL ES的环境并绘制图形。然而，如果你希望获取更多的帮助来学会使用OpenGL，那么请继续学习下一节课程获取更多的知识。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/shapes.html>

# 定义Shapes

在一个OpenGL ES视图的上下文中定义形状，是创建你的杰作所需的第一步。在不知道关于OpenGL ES如何期望你来定义图形对象的基本知识的时候，通过OpenGL ES 绘图可能会有些困难。

这节课将解释OpenGL ES相对于Android设备屏幕的坐标系，定义形状和形状表面的基本知识，如定义一个三角形和一个矩形。

## 定义一个三角形

OpenGL ES允许你使用三维空间的坐标来定义绘画对象。所以在你能画三角形之前，你必须先定义它的坐标。在OpenGL中，典型的方法是以浮点数的形式为坐标定义一个顶点数组。为了让效率最大化，你可以将坐标写入一个[ByteBuffer](#)，它将会传入OpenGL ES的图形处理流程中：

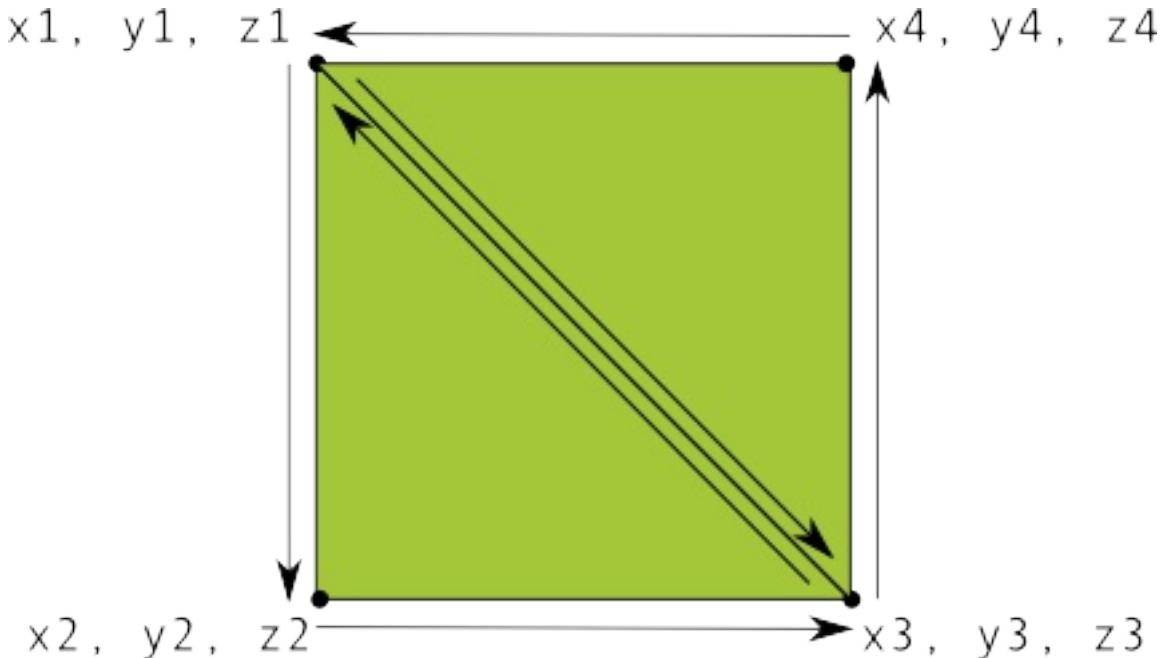
```
public class Triangle {  
  
    private FloatBuffer vertexBuffer;  
  
    // number of coordinates per vertex in this array  
    static final int COORDS_PER_VERTEX = 3;  
    static float triangleCoords[] = { // in counterclockwise order  
        0.0f, 0.622008459f, 0.0f, // top  
        -0.5f, -0.311004243f, 0.0f, // bottom left  
        0.5f, -0.311004243f, 0.0f // bottom right  
    };  
  
    // Set color with red, green, blue and alpha (opacity) values  
    float color[] = { 0.63671875f, 0.76953125f, 0.22265625f, 1.0f };  
  
    public Triangle() {  
        // initialize vertex byte buffer for shape coordinates  
        ByteBuffer bb = ByteBuffer.allocateDirect(  
            // (number of coordinate values * 4 bytes per float)  
            triangleCoords.length * 4);  
        // use the device hardware's native byte order  
        bb.order(ByteOrder.nativeOrder());  
  
        // create a floating point buffer from the ByteBuffer  
        vertexBuffer = bb.asFloatBuffer();  
        // add the coordinates to the FloatBuffer  
        vertexBuffer.put(triangleCoords);  
        // set the buffer to read the first coordinate  
        vertexBuffer.position(0);  
    }  
}
```

默认情况下，OpenGL ES会假定一个坐标系，在这个坐标系中，[0, 0, 0]（分别对应X轴坐标，Y轴坐标，Z轴坐标）对应的是GLSurfaceView框架的中心。[1, 1, 0]对应的是框架的右上角，[-1, -1, 0]对应的则是左下角。如果想要看此坐标系的插图说明，可以阅读[OpenGL ES开发手册](#)。

注意到这个形状的坐标是以逆时针顺序定义的。绘制的顺序非常关键，因为它定义了哪一面是形状的正面（你希望绘制的一面），以及背面（你可以使用OpenGL ES的cull face特性来让它不要绘制）。更多关于该方面的信息，可以阅读[OpenGL ES开发手册](#)。

## 定义一个矩形

在OpenGL中定义三角形非常简单，那么你是否想要增加一些复杂性呢？比如，定义一个矩形？有很多方法可以用来定义矩形，不过在OpenGL ES中最典型的方法是使用两个三角形拼接在一起：



再一次地，你需要以逆时针的形式为三角形顶点定义坐标来表现这个图形，并将值放入一个[ByteBuffer](#)中。为了避免由两个三角形共享的顶点被重复定义，可以使用一个绘制列表来告诉OpenGL ES图形处理流程应该如何画这些顶点。下面是代码样例：

```
public class Square {  
  
    private ByteBuffer vertexBuffer;  
    private ShortBuffer drawListBuffer;  
  
    // number of coordinates per vertex in this array  
    static final int COORDS_PER_VERTEX = 3;  
    static float squareCoords[] = {  
        -0.5f,  0.5f,  0.0f,      // top left  
        -0.5f, -0.5f,  0.0f,      // bottom left  
        0.5f, -0.5f,  0.0f,      // bottom right  
        0.5f,  0.5f,  0.0f };    // top right  
  
    private short drawOrder[] = { 0, 1, 2, 0, 2, 3 }; // order to  
  
    public Square() {  
        // initialize vertex byte buffer for shape coordinates  
        ByteBuffer bb = ByteBuffer.allocateDirect(  
            // (# of coordinate values * 4 bytes per float)  
            squareCoords.length * 4);  
        bb.order(ByteOrder.nativeOrder());  
        vertexBuffer = bb.asFloatBuffer();  
        vertexBuffer.put(squareCoords);  
        vertexBuffer.position(0);  
    }  
}
```

```
// initialize byte buffer for the draw list
ByteBuffer dlb = ByteBuffer.allocateDirect(
    // (# of coordinate values * 2 bytes per short)
    drawOrder.length * 2);
dlb.order(ByteOrder.nativeOrder());
drawListBuffer = dlb.asShortBuffer();
drawListBuffer.put(drawOrder);
drawListBuffer.position(0);
}
}
```

该样例给了你一个如何使用OpenGL创建复杂图形的启发，通常来说，你需要使用三角形的集合来绘制对象。在下一节课中，你将学习如何在屏幕上画这些形状。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/draw.html>

# 绘制Shapes

在你定义了需要OpenGL绘制的形状之后，你可能希望绘制它们。使用OpenGL ES 2.0绘制图形可能会比你想象当中需要更多的代码，因为API中提供了大量对于图形处理流程的控制。

这节课将解释如何使用OpenGL ES 2.0接口画出在上一节课中定义的图形。

## 初始化形状

在你开始绘画之前，你需要初始化并加载你期望绘制的图形。除非你所使用的形状结构（原始坐标）在执行过程中发生了变化，不然的话你应该在渲染器的[onSurfaceCreated\(\)](#)方法中初始化它们，这样做是处于内存和执行效率的考量。

```
public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
    ...  
  
    // initialize a triangle  
    mTriangle = new Triangle();  
    // initialize a square  
    mSquare = new Square();  
}
```

## 画一个形状

使用OpenGL ES 2.0画一个定义的形状需要较多代码，因为你需要提供很多图形处理流程的细节。具体而言，你必须定义如下几项：

- 顶点着色器（Vertex Shader）：OpenGL ES代码用来渲染形状的顶点。
- 片段着色器（Fragment Shader）：OpenGL ES代码用来渲染形状的表面，使用颜色或纹理。
- 程式（Program）：一个OpenGL ES对象，包含了你希望用来绘制一个或更多图形所要用到的着色器。

你需要至少一个顶点着色器来绘制一个形状，以及一个片段着色器为该形状上色。这些着色器必须被编译然后添加至一个OpenGL ES程式当中，它用来绘制形状。下面的代码展示了你可以用来画一个图形的基本着色器：

```
private final String vertexShaderCode =  
    "attribute vec4 vPosition;" +  
    "void main() {" +  
    "    gl_Position = vPosition;" +  
    "}" ;  
  
private final String fragmentShaderCode =  
    "precision mediump float;" +  
    "uniform vec4 vColor;" +  
    "void main() {" +  
    "    gl_FragColor = vColor;" +  
    "}" ;
```

着色器包含了OpenGL Shading Language (GLSL) 代码，它必须先被编译然后才能在OpenGL环境中使用。要编译这个代码，在你的渲染器类中创建一个辅助方法：

```
public static int loadShader(int type, String shaderCode) {  
  
    // create a vertex shader type (GLES20.GL_VERTEX_SHADER)  
    // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)  
    int shader = GLES20.glCreateShader(type);
```

```
// add the source code to the shader and compile it
GLES20.glShaderSource(shader, shaderCode);
GLES20.glCompileShader(shader);

return shader;
}
```

为了绘制你的图形，你必须编译着色器代码，将它们添加至一个OpenGL ES程式对象中，然后执行连接。在你的绘制对象的构造函数里做这些事情，这样上述步骤就只用执行一次。

**Note:** 编译OpenGL ES着色器及连接操作对于CPU周期和处理时间而言，消耗是巨大的，所以你应该避免重复执行这些事情。如果你在执行期间不知道你的着色器内容，那么你应该在构建你的应用时，确保它们值创建了一次，并且缓存以备后续使用。

```
public class Triangle() {  
    ...  
  
    int vertexShader = loadShader(GLES20.GL_VERTEX_SHADER, vertexS  
    int fragmentShader = loadShader(GLES20.GL_FRAGMENT_SHADER, fra  
  
    mProgram = GLES20.glCreateProgram(); // create empty program  
    GLES20.glAttachShader(mProgram, vertexShader); // add the ve  
    GLES20.glAttachShader(mProgram, fragmentShader); // add the fr  
    GLES20.glLinkProgram(mProgram); // creates OpenGL program object  
}
```

至此，你已经完全准备好添加实际的调用来绘制你的图形了。使用OpenGL ES绘制图形需要你定义一些变量来告诉渲染流程你需要画什么以及如何去画。既然绘制属性会根据形状的不同而发生变化，把绘制逻辑包含在形状类里面将是一个不错的主意。

为图形创建一个“draw()”方法。这个代码为形状的顶点着色器和形状着色器设置了位置和颜色值，进而执行绘图功能：

```
    vertexStride, vertexBuffer);

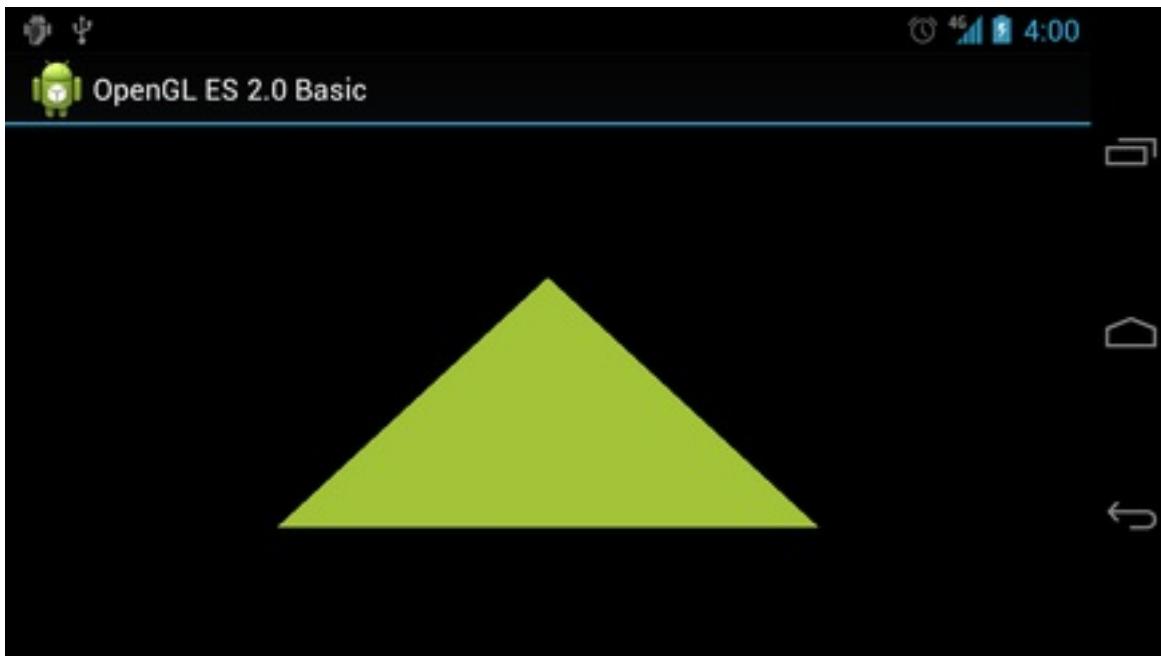
    // get handle to fragment shader's vColor member
    mColorHandle = GLES20.glGetUniformLocation(mProgram, "vColor");

    // Set color for drawing the triangle
    GLES20 glUniform4fv(mColorHandle, 1, color, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
```

一旦你完成了上述所有代码，画这个对象就仅需要在你渲染器的[onDrawFrame\(\)](#)方法中调用draw()方法就可以了。当你运行这个应用时，它看上去会像是这样：



在这个代码样例中，还存在一些问题。首先，它无法给你的朋友带来什么深刻的印象。其次，这个三角形看上去有一些扁，另外当你改变屏幕方向时，它的形状也会随之改变。形状发生形变的原因是因为对象的顶点没有根据显示[GLSurfaceView](#)的屏幕区域的比例进行修正。你可以在下一节课中使用投影（projection）或者相机视图（camera view）来解决这个问题。

最后，这个三角形是静止的，这看上去有些无聊。在[添加移动](#)课程当中（后续课程），你会让这个形状发生旋转，并使用一些OpenGL ES图形处理流程的更加新奇的用法。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/projection.html>

# 运用投影与相机视图

在OpenGL ES环境中，投影和相机视图允许你显示绘图对象时，其效果更加酷似于你用肉眼看到的真实物体。这个物理视图的仿真是使用绘制对象坐标的数学变换实现的：

- **投影（Projection）**：这个变换会基于显示它们的[GLSurfaceView](#)的长和宽，来调整绘图对象的坐标。如果没有该计算，那么用OpenGL ES绘制的对象会由于视图窗口比例的不匹配而发生形变。一个投影变换一般仅需要在渲染器的[onSurfaceChanged\(\)](#)方法中，OpenGL视图的比例建立时或发生变化时才被计算。关于更多OpenGL ES投影和坐标映射的知识，可以阅读[Mapping Coordinates for Drawn Objects](#)。
- **相机视图（camera view）**：这个变化会基于一个虚拟相机位置改变绘图对象的坐标。注意到OpenGL ES并没有定义一个实际的相机对象，但是取而代之的，它提供了一些辅助方法，通过变化绘图对象的显示来模拟相机。一个相机视图变换可能仅在你建立你的[GLSurfaceView](#)时计算一次，也可能根据用户的行为或者你的应用的功能进行动态调整。

这节课将解释如何创建一个投影和一个相机视图，并应用它们到[GLSurfaceView](#)中的绘制图像上。

## 定义一个投影

投影变换的数据会在你的[GLSurfaceView.Renderer](#)类中的[onSurfaceChanged\(\)](#)方法中被计算。下面的代码首先接收[GLSurfaceView](#)的高和宽，然后用它来填充一个投影变换矩阵([Matrix](#))，使用[Matrix.frustumM\(\)](#)方法：

```
@Override  
public void onSurfaceChanged(GL10 unused, int width, int height) {  
    GLES20.glViewport(0, 0, width, height);  
  
    float ratio = (float) width / height;  
  
    // this projection matrix is applied to object coordinates  
    // in the onDrawFrame() method  
    Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1, 1, 3,  
}
```

该代码填充了一个投影矩阵：mProjectionMatrix，在下一节中，你可以在[onDrawFrame\(\)](#)将它和一个相机视图变换结合起来。

**Note:** 在你的绘图对象上只应用一个投影变换会导致一个看上去很空的显示效果。一般而言，你必须同时为每一个要在屏幕上显示的任何东西实现一个相机视图。

## 定义一个相机视图

通过添加一个相机视图变换作为绘图过程的一部分，以此来完成你的绘图对象变换的所有步骤。在下面的代码中，使用[Matrix.setLookAtM\(\)](#)方法来计算相机视图变换，然后与之前计算的投影矩阵结合起来。结合后的变换矩阵传递给绘制图像：

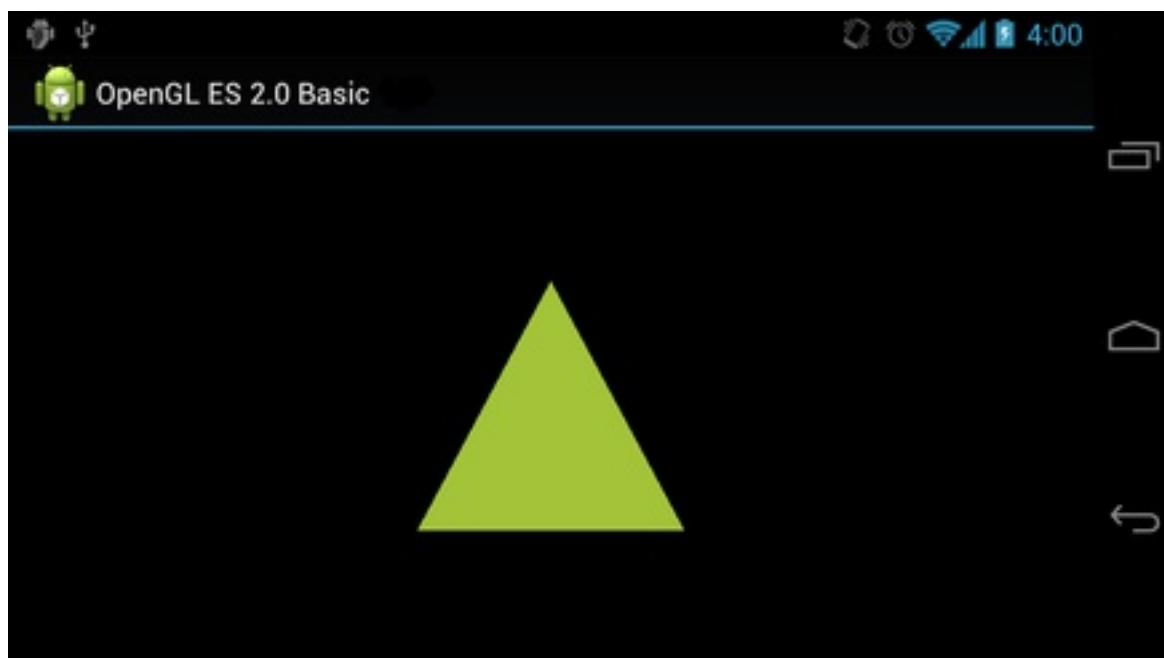
```
@Override  
public void onDrawFrame(GL10 unused) {  
    ...  
    // Set the camera position (View matrix)  
    Matrix.setLookAtM(mViewMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.  
  
    // Calculate the projection and view transformation  
    Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix, 0, mViewMa  
  
    // Draw shape  
    mTriangle.draw(mMVPMatrix);  
}
```

## 应用投影和相机变换

为了使用在之前章节中结合了的相机视图变换和投影变换，修改你的图形对象的draw()方法，接收结合的变换并将其应用到图形上：

```
public void draw(float[] mvpMatrix) { // pass in the calculated tr
    ...
    // get handle to shape's transformation matrix
    mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVP
    // Pass the projection and view transformation to the shader
    GLES20 glUniformMatrix4fv(mMVPMatrixHandle, 1, false, mvpMatrix);
    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);
    ...
}
```

一旦你正确地计算了投影变换和相机视图变换，并应用了它们，你的图形就会以正确的比例画出，看上去会像是这样：



现在你有了一个能以正确的比例显示你的图形的应用了，下面就该为图形添加一些动画效果了！

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/motion.html>

# 添加移动

在屏幕上绘制图形是OpenGL的一个基本特性，但你也可以通过其它的Android图形框架类做这些事情，包括[Canvas](#)和[Drawable](#)对象。OpenGL ES提供额外的功能，能够在三维空间对绘制图形进行移动和变换操作，或者还可以通过其它独有的方法创建出引人入胜的用户体验。

在这节课中，一会更深入的学习OpenGL ES的知识：对一个形状添加旋转动画。

## 旋转一个形状

使用OpenGL ES 2.0 旋转一个绘制图形是比较简单的。首先创建一个变换矩阵（一个旋转矩阵）并且将它和你的投影变换矩阵和相机试图变换矩阵结合在一起：

```
private float[] mRotationMatrix = new float[16];
public void onDrawFrame(GL10 gl) {
    ...
    float[] scratch = new float[16];

    // Create a rotation transformation for the triangle
    long time = SystemClock.uptimeMillis() % 4000L;
    float angle = 0.090f * ((int) time);
    Matrix.setRotateM(mRotationMatrix, 0, angle, 0, 0, -1.0f);

    // Combine the rotation matrix with the projection and camera
    // Note that the mMVPMatrix factor *must be first* in order
    // for the matrix multiplication product to be correct.
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix,
        0);

    // Draw triangle
    mTriangle.draw(scratch);
}
```

如果完成了这些变更以后，你的三角形还是没有旋转的话，确认一下你是否将[GLSurfaceView.RENDERMODE\\_WHEN\\_DIRTY](#)的配置注释掉了，有关该方面的知识会在下一节中展开。

## 启用连续渲染

如果你严格按照这节课的样例代码走到了现在这一步，那么请确定您将设置渲染模式为“RENDERMODE\_WHEN\_DIRTY”的这一行注释了，不然的话OpenGL只会对这个形状执行一个增量的旋转，然后就等待[GLSurfaceView](#)容器的[requestRender\(\)](#)方法被调用。

```
public MyGLSurfaceView(Context context) {  
    ...  
    // Render the view only when there is a change in the drawing  
    // To allow the triangle to rotate automatically, this line is  
    //setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);  
}
```

除非你的某个对象，它的变化和用户的交互无关，不然的话一般还是建议将这个配置打开。在下一节课将会将这个注释放开，并且再次调用。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/graphics/opengl/touch.html>

# 响应触摸事件

让对象根据预设的程序运动，如让一个三角形旋转可以有效地让人引起注意，但是如果你希望可以让OpenGL ES与用户交互呢？让你的OpenGL ES应用可以与触摸交互的关键点在于，拓展你的[GLSurfaceView](#)的实现，覆写[onTouchEvent\(\)](#)方法来监听触摸事件。

这节课将会向你展示如何监听触摸事件，让用户旋转一个OpenGL ES对象。

## 配置触摸监听器

为了让你的OpenGL ES应用响应触摸事件，你必须实现在[GLSurfaceView](#)类中的[onTouchEvent\(\)](#)方法。下述实现的样例展示了如何监听[MotionEvent.ACTION\\_MOVE](#)事件，并将它们转换为形状旋转的角度：

```
@Override
public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. In this case, you are only
    // interested in events where the touch position changed.

    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
        case MotionEvent.ACTION_MOVE:

            float dx = x - mPreviousX;
            float dy = y - mPreviousY;

            // reverse direction of rotation above the mid-line
            if (y > getHeight() / 2) {
                dx = dx * -1;
            }

            // reverse direction of rotation to left of the mid-line
            if (x < getWidth() / 2) {
                dy = dy * -1;
            }

            mRenderer.setAngle(
                mRenderer.getAngle() +
                ((dx + dy) * TOUCH_SCALE_FACTOR); // = 180.0f
            requestRender();
        }

        mPreviousX = x;
        mPreviousY = y;
        return true;
    }
}
```

注意在计算旋转角度后，该方法会调用[requestRender\(\)](#)来告诉渲染器现在可以进行渲染了。该方法对于这个例子来说是最有效的，因为图形并不需要重新绘制，除非有一个旋转角度的变化。然而，它对于执行效率并没有任何影响，除非你需要渲染器仅在数据变化时才会重新绘制（使用[setRenderMode\(\)](#)方法），所以请确保这一行没有被注释掉：

```
public MyGLSurfaceView(Context context) {
    ...
    // Render the view only when there is a change in the drawing
    setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
}
```



## 公开旋转角度

上述样例代码需要你公开旋转的角度，方法是在你的渲染器中添加一个共有成员。由于渲染器代码运行在一个独立的线程中（非主UI线程），你必须将你的这个公共变量声明为 volatile。请看下面的代码：

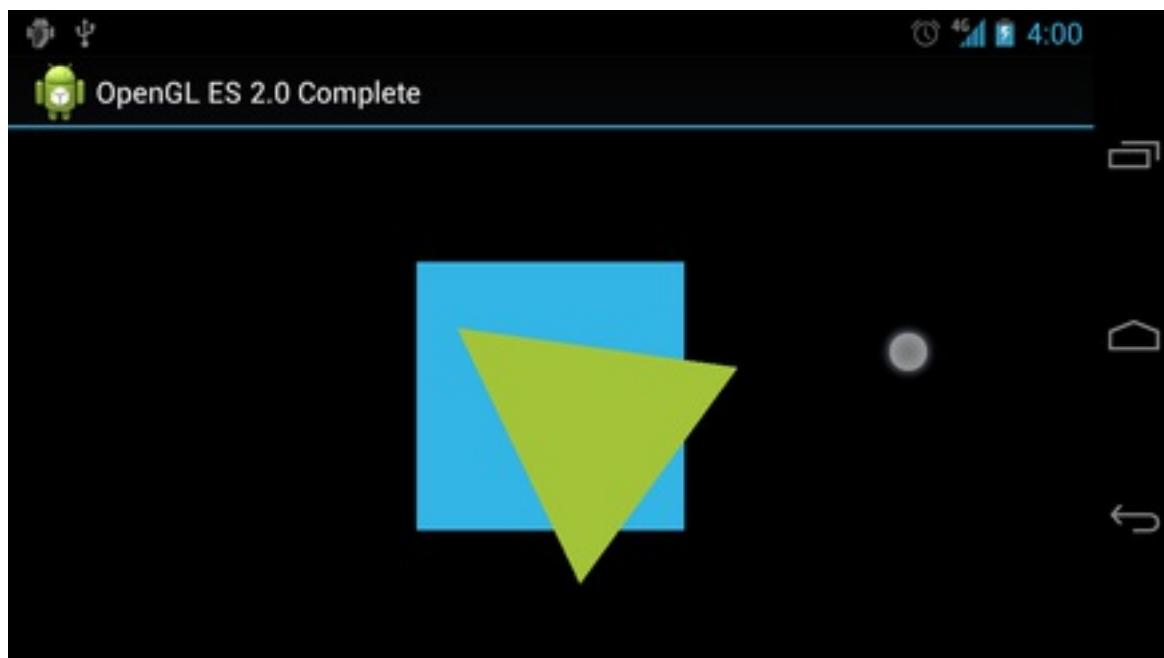
```
public class MyGLRenderer implements GLSurfaceView.Renderer {  
    ...  
    public volatile float mAngle;
```

## 应用旋转

为了应用触摸输入所导致的旋转，注释掉创建一个旋转角度的代码，然后添加mAngle，该变量包含了输入所导致的角度：

```
public void onDrawFrame(GL10 gl) {  
    ...  
    float[] scratch = new float[16];  
  
    // Create a rotation for the triangle  
    // long time = SystemClock.uptimeMillis() % 4000L;  
    // float angle = 0.090f * ((int) time);  
    Matrix.setRotateM(mRotationMatrix, 0, mAngle, 0, 0, -1.0f);  
  
    // Combine the rotation matrix with the projection and camera  
    // Note that the mMVPMatrix factor *must be first* in order  
    // for the matrix multiplication product to be correct.  
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix);  
  
    // Draw triangle  
    mTriangle.draw(scratch);  
}
```

当你完成了上述的步骤，运行这个程序并用你的手指在屏幕上拖动，来旋转三角形：



编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/index.html>

# 添加动画

动画为你 App 上将要发生的事件增加精细的提醒，并且能改进你 App 界面的思维模型。当界面改变其状态时，例如加载内容或新操作可用时，动画特别有帮助。动画也能为你的 App 增添优雅的外观，提供给你的 App 一种优质体验。

但是记住：滥用动画或者在错误时机使用动画也是有害的，例如：他们造成了延迟。这节课告诉你如何应用常用动画类型来提升易用性，在不打扰用户情况下提升性能。

# Lessons

- [View间渐变](#)

学习在重叠 view 间淡入淡出。这节课将展示你如何使用一个进度提示来淡入淡出包含文本信息的 view。

- [使用ViewPager实现屏幕滑动](#)

学习怎样为水平相邻界面间提供渐变动画。

- [展示卡片（Card）翻转动画](#)

学习怎样实现两个视图的翻转动画

- [缩放View](#)

学习怎样通过触控放大视图

- [布局变更动画](#)

学习当你增加、移除或更新子 View 时怎样使用内置动画

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/crossfade.html>

# View间渐变

渐变动画（也叫消失）通常指渐渐的淡出某个 UI 组件，同时同步地淡入另一个。在你 App 想切换内容或 view 的情况下，这种动画很有用。渐变简短不易察觉，它也能提供从一个界面到下一个之间流畅的转换。当你不使用它们，不管怎么样转换经常感到生硬而仓促。

下面是一个利用进度指示渐变到一些文本内容的例子。



如果你想跳过看整个例子，[下载](#) App 样例然后运行渐变例子。查看下列文件中的代码实现：

- src/CrossfadeActivity.java
- layout/activity\_crossfade.xml
- menu/activity\_crossfade.xml

## 创建View

创建两个你想相互渐变的 view。下面的例子创建了一个进度提示圈和可滑动文本 view。

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

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

        <TextView style="?android:textAppearanceMedium"
            android:lineSpacingMultiplier="1.2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/lorem_ipsum"
            android:padding="16dp" />

    </ScrollView>

    <ProgressBar android:id="@+id/loading_spinner"
        style="?android:progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</FrameLayout>
```

## 设置动画

为设置动画，你需要：

1. 为你想渐变的 view 创建成员变量。之后动画应用途中修改 View 的时候你会需要这些引用的。
2. 对于被淡出的 view，设置它的 visibility 为 GONE。这样防止 view 再占据布局的空间，而且也能在布局计算中将其忽略，加速处理过程。
3. 将 config\_shortAnimTime 系统属性暂存到一个成员变量里。这个属性为动画定义了一个标准的“短”持续时间。对于微妙或者快速发生的动画，这是个很理想的时间段。config\_longAnimTime 和 config\_mediumAnimTime 也行，如果你想用的话。

下面是个内容 view 的 activity 例子，它使用了前面所述代码片段中的布局。

```
public class CrossfadeActivity extends Activity {

    private View mContentView;
    private View mLoadingView;
    private int mShortAnimationDuration;

    ...

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

        mContentView = findViewById(R.id.content);
        mLoadingView = findViewById(R.id.loading_spinner);

        // Initially hide the content view.
        mContentView.setVisibility(View.GONE);

        // Retrieve and cache the system's default "short" animation
        mShortAnimationDuration = getResources().getInteger(
            android.R.integer.config_shortAnimTime);
    }
}
```

# 渐变View

既然正确地设置了那些 view，做下面这些事情来渐变他们吧：

1. 对于淡入的 view，设置 alpha 值为 0 并且设置 visibility 为 [VISIBLE](#)（要记得他起初被设置成了 [GONE](#)）。这让 view 可见了但是它是透明的。
2. 对于淡入的 view，把 alpha 值从 0 动态改变到 1。同时，对于淡出的 view，把 alpha 值从 1 动态变到 0。
3. 使用 [Animator.AnimatorListener](#) 中的 [onAnimationEnd\(\)](#)，设置淡出 view 的 visibility 为 [GONE](#)。即使 alpha 值为 0，也要把 view 的 visibility 设置成 [GONE](#) 来防止 view 占据布局空间，还能把它从布局计算中忽略，加速处理过程。

下面方法展示如何去做：

```
private View mContentView;
private View mLoadingView;
private int mShortAnimationDuration;

...

private void crossfade() {

    // Set the content view to 0% opacity but visible, so that it
    // (but fully transparent) during the animation.
    mContentView.setAlpha(0f);
    mContentView.setVisibility(View.VISIBLE);

    // Animate the content view to 100% opacity, and clear any ani-
    // listener set on the view.
    mContentView.animate()
        .alpha(1f)
        .setDuration(mShortAnimationDuration)
        .setListener(null);

    // Animate the loading view to 0% opacity. After the animation
    // set its visibility to GONE as an optimization step (it won't
    // participate in layout passes, etc.)
    mLoadingView.animate()
        .alpha(0f)
        .setDuration(mShortAnimationDuration)
        .setListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                mLoadingView.setVisibility(View.GONE);
            }
        });
}
```

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/screen-slide.html>

# 使用ViewPager实现屏幕滑动

滑屏是在两个完整界面间的转换，它在一些 UI 中很常见，比如设置导向和幻灯放映。这节课将告诉你怎样通过[支持库 \(support library\)](#)提供的[ViewPager](#)实现滑屏。[ViewPager](#)能自动实现滑屏动画。下面展示了从一个内容界面到一下界面的滑屏转换是什么样子。



如果你想跳过看整个例子，[下载](#) App 样例然后运行屏幕滑动例子。查看下列文件中的代码实现：

- src/ScreenSlidePageFragment.java
- src/ScreenSlideActivity.java
- layout/activity\_screen\_slide.xml
- layout/fragment\_screen\_slide\_page.xml

## 创建View

创建你之后会为 Fragment 内容使用的布局文件。下面例子包含一个展示文本的 text view：

```
<!-- fragment_screen_slide_page.xml -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android
    android:id="@+id/content"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView style="?android:textAppearanceMedium"
        android:padding="16dp"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/lorem_ipsum" />
</ScrollView>
```

同时也为 fragment 内容定义了一个字符串。

## 创建Fragment

创建一个 [Fragment](#) 子类，它从 [onCreateView\(\)](#) 方法中返回你刚创建的布局。无论什么时候，如果你需要为用户展示一个页面，你可以在它的父 activity 中为这个 fragment 新建实例：

```
import android.support.v4.app.Fragment;
...
public class ScreenSlidePageFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup co
        Bundle savedInstanceState) {
        ViewGroup rootView = (ViewGroup) inflater.inflate(
            R.layout.fragment_screen_slide_page, container, fa

        return rootView;
    }
}
```

## 添加ViewPager

[ViewPager](#) 有内建的滑动手势用来在页面间转换，并且他默认使用滑屏动画，所以你不用自己创建。[ViewPager](#) 使用 [PagerAdapter](#) 来补充新页面，所以 [PagerAdapter](#) 会用到你之前新建 fragment 类。

开始之前，创建一个包含 [ViewPager](#) 的布局：

```
<!-- activity_screen_slide.xml -->
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

创建一个 activity 做下面这些事情：

- 把 content view 设置成这个包含 [ViewPager](#) 的布局。
- 创建一个继承自 [FragmentStatePagerAdapter](#) 抽象类的类，然后实现 [getItem\(\)](#) 方法来把 ScreenSlidePageFragment 实例作为新页面补充进来。pager adapter还需要实现 [getCount\(\)](#) 方法，它返回 adapter 将要创建页面的总数（例如5个）。
- 把 [PagerAdapter](#) 和 [ViewPager](#) 关联起来。
- 处理 Back 按钮，按下变为在虚拟的 fragment 栈中回退。如果用户已经在第一个页面了，则在 activity 的 back stack 中回退。

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
...
public class ScreenSlidePagerAdapterActivity extends FragmentActivity {
    /**
     * The number of pages (wizard steps) to show in this demo.
     */
    private static final int NUM_PAGES = 5;

    /**
     * The pager widget, which handles animation and allows swiping
     * and next wizard steps.
     */
    private ViewPager mPager;

    /**
     * The pager adapter, which provides the pages to the view pager
     */
    private PagerAdapter mPagerAdapter;

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

```
// Instantiate a ViewPager and a PagerAdapter.  
mPager = (ViewPager) findViewById(R.id.pager);  
mPagerAdapter = new ScreenSlidePagerAdapter(getSupportFragmentManager());  
mPager.setAdapter(mPagerAdapter);  
  
}  
  
@Override  
public void onBackPressed() {  
    if (mPager.getCurrentItem() == 0) {  
        // If the user is currently looking at the first step,  
        // Back button. This calls finish() on this activity a  
        super.onBackPressed();  
    } else {  
        // Otherwise, select the previous step.  
        mPager.setCurrentItem(mPager.getCurrentItem() - 1);  
    }  
}  
  
/**  
 * A simple pager adapter that represents 5 ScreenSlidePageFra  
 * sequence.  
 */  
private class ScreenSlidePagerAdapter extends FragmentStatePage  
    public ScreenSlidePagerAdapter(FragmentManager fm) {  
        super(fm);  
    }  
  
    @Override  
    public Fragment getItem(int position) {  
        return new ScreenSlidePageFragment();  
    }  
  
    @Override  
    public int getCount() {  
        return NUM_PAGES;  
    }  
}  
}
```

## 用PageTransformer自定义动画

为展示不同于默认滑屏效果的动画，实现 [ViewPager.PageTransformer](#) 接口，然后把它补充到 view pager 里。这接口只暴露了一个方法，[transformPage\(\)](#)。每次界面切换，这个方法都会为每个可见页面和界面中消失的相邻界面调用一次（通常只有一个页面可见）。例如，第三页可见而且用户向第四页拖动，，[transformPage\(\)](#) 在手势的各个阶段为第二，三，四页分别调用。

在你，[transformPage\(\)](#) 的实现中，基于当前界面上页面的 position (position 由 [transformPage\(\)](#) 方法的参数给出) 决定哪些页面需要被动画转换，通过这样你就能新建自己的动画。

position 参数表示给定页面相对于屏幕中的页面的位置。它的值在用户滑动页面过程中动态变化。当页面填充屏幕，它的值为 0。当页面刚从屏幕右边拖走，它的值为 1。如果用户滑动到半路，那么左边 position 为 -0.5 并且右边 position 为 0.5。根据屏幕上页面的 position，你可以通过[setAlpha\(\)](#)，[setTranslationX\(\)](#) 或 [setScaleY\(\)](#) 这些方法设定页面属性来自定义滑动动画。

当你有了一个 [PageTransformer](#) 的实现后，用你的实现调用 [setPageTransformer\(\)](#) 来应用这些自定义动画。例如，如果你有一个 [PageTransformer](#) 叫做 ZoomOutPageTransformer，你可以这样设置自定义动画：

```
ViewPager mPager = (ViewPager) findViewById(R.id.pager);  
...  
mPager.setPageTransformer(true, new ZoomOutPageTransformer());
```

详情查看[放大型 Page Transformer \(页面转换动画\)](#) 和[潜藏型 Page Transformer \(页面转换动画\)](#) 部分和 [PageTransformer](#) 视屏。

### 放大型PageTransformer (页面转换动画)

当在相邻界面滑动时，这个page transformer缩放和渐变动画。当页面越靠近中心，它将渐渐还原到正常大小并且渐入屏幕。



```

public class ZoomOutPageTransformer implements ViewPager.PageTransformer {
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.5f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();
        int pageHeight = view.getHeight();

        if (position < -1) { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 1) { // [-1,1]
            // Modify the default slide transition to shrink the page
            float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(position));
            float vertMargin = pageHeight * (1 - scaleFactor) / 2;
            float horzMargin = pageWidth * (1 - scaleFactor) / 2;
            if (position < 0) {
                view.setTranslationX(horzMargin - vertMargin / 2);
            } else {
                view.setTranslationX(-horzMargin + vertMargin / 2);
            }

            // Scale the page down (between MIN_SCALE and 1)
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

            // Fade the page relative to its size.
            view.setAlpha(MIN_ALPHA +
                    (scaleFactor - MIN_SCALE) /
                    (1 - MIN_SCALE) * (1 - MIN_ALPHA));
        } else { // (1,+Infinity]
            // This page is way off-screen to the right.
            view.setAlpha(0);
        }
    }
}

```

## 潜藏型PageTransformer（页面转换动画）

这个page transformer使用默认动画的屏幕左滑动画。但是为右滑使用一种“潜藏”效果的动画。潜藏动画淡出页面，并且线性缩小它。



注意：在潜藏过程中，默认动画（屏幕滑动）是仍旧发生的，所以你必须用负的 X 平移来抵消它。例如：

```
view.setTranslationX(-1 * view.getWidth() * position);
```

下面的例子展示了如何抵消默认滑屏动画：

```
public class DepthPageTransformer implements ViewPager.PageTransformer {
    private static final float MIN_SCALE = 0.75f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();

        if (position < -1) { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 0) { // [-1,0]
            // Use the default slide transition when moving to the
            view.setAlpha(1);
            view.setTranslationX(0);
            view.setScaleX(1);
            view.setScaleY(1);

        } else if (position <= 1) { // (0,1]
            // Fade the page out.
            view.setAlpha(1 - position);

            // Counteract the default slide transition
            view.setTranslationX(pageWidth * -position);

            // Scale the page down (between MIN_SCALE and 1)
        }
    }
}
```

```
    float scaleFactor = MIN_SCALE
        + (1 - MIN_SCALE) * (1 - Math.abs(position));
    view.setScaleX(scaleFactor);
    view.setScaleY(scaleFactor);

} else { // (1,+Infinity]
    // This page is way off-screen to the right.
    view.setAlpha(0);
}
}
```

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/cardflip.html>

# 展示卡片（Card）翻转动画

这节课展示怎样自定义 fragment 动画和实现卡片翻转。view 内容间的卡片翻转动画模拟卡片翻转的效果。

下面是卡片翻转动画的样子：



如果你想跳过看整个例子，[下载](#) App 样例然后运行卡片翻转例子。查看下列文件中的代码实现：

- src/CardFlipActivity.java
- animator/card\_flip\_right\_in.xml
- animator/card\_flip\_right\_out.xml
- animator/card\_flip\_left\_in.xml
- animator/card\_flip\_left\_out.xml
- layout/fragment\_card\_back.xml
- layout/fragment\_card\_front.xml

## 创建Animator (动画者)

创建卡片翻转动画，你需要两个 animator 来让卡片正面其一：右侧部分弹出，然后向左转动，其二左侧部分陷入，然后向右转动。你还需要两个 animator 将卡片背面其一：右侧部分陷入，然后向左转动，其二左侧弹出，然后向右转动。

### card\_flip\_left\_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="-180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
        alpha back to 1.0. -->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

### card\_flip\_left\_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
        alpha back to 1.0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

## **card\_flip\_right\_in.xml**

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dec
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
```

## **card\_flip\_right\_out.xml**

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="-180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dec
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

## 创建View

卡片的每一面是一个能包含你想要内容的布局，比如两屏文字，两张图片，或者任何view的组合。然后你将在应用动画的fragment里面用到这俩布局。下面的布局创建了一面展示文本的布局：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#a6c"
    android:padding="16dp"
    android:gravity="bottom">

    <TextView android:id="@+id/text1"
        style="?android:textAppearanceLarge"
        android:textStyle="bold"
        android:textColor="#fff"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_title" />

    <TextView style="?android:textAppearanceSmall"
        android:textAllCaps="true"
        android:textColor="#80ffffff"
        android:textStyle="bold"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_description" />

</LinearLayout>
```

卡片另一面显示一个 [ImageView](#)：

```
<ImageView xmlns:android="http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/image1"
    android:scaleType="centerCrop"
    android:contentDescription="@string/description_image_1" />
```

## 创建Fragment

为正反面创建fragment，这些类从[onCreateView\(\)](#)方法中分别为每个framgent返回你之前创建的布局。你可以在展示卡片的父activity中新建他们的实例。下面的例子展示父activity内嵌套的fragment：

```
public class CardFlipActivity extends Activity {  
    ...  
    /**  
     * A fragment representing the front of the card.  
     */  
    public class CardFrontFragment extends Fragment {  
        @Override  
        public View onCreateView(LayoutInflater inflater, ViewGroup  
            Bundle savedInstanceState) {  
            return inflater.inflate(R.layout.fragment_card_front,  
        }  
    }  
  
    /**  
     * A fragment representing the back of the card.  
     */  
    public class CardBackFragment extends Fragment {  
        @Override  
        public View onCreateView(LayoutInflater inflater, ViewGroup  
            Bundle savedInstanceState) {  
            return inflater.inflate(R.layout.fragment_card_back, c  
        }  
    }  
}
```

## 应用卡片翻转动画

现在，你需要在父activity中展示fragment。为做这件事，首先创建你activity的布局。下面例子创建了一个你可以在运行时添加fragment的 [FrameLayout](#)。

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

在activity代码中，把刚创建的布局设置成content view。当activity创建时展示一个默认的fragment是个不错的注意。所以下面的activity样例告诉你如何默认显示卡片正面：

```
public class CardFlipActivity extends Activity {

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

        if (savedInstanceState == null) {
            getFragmentManager()
                .beginTransaction()
                .add(R.id.container, new CardFrontFragment())
                .commit();
        }
        ...
    }
}
```

既然现在显示了卡片的正面，你可以在合适时机用翻转动画显示背面了。创建一个方法来显示背面需要做下面这些事情：

- 为fragment转换设置你刚做的自定义动画
- 用新fragment替换当前显示的fragment，并且应用你刚创建的动画到这个事件中。
- 添加之前显示的fragment到fragment的back stack中，所以当你摁 Back 键时，卡片会翻转回来。

```
private void flipCard() {
    if (mShowingBack) {
        getFragmentManager().popBackStack();
        return;
    }

    // Flip to the back.

    mShowingBack = true;

    // Create and commit a new fragment transaction that adds the
```

```
// the card, uses custom animations, and is part of the fragme

getFragmentManager()
    .beginTransaction()

    // Replace the default fragment animations with animat
    // rotations when switching to the back of the card, a
    // resources representing rotations when flipping back
    // the system Back button is pressed).
    .setCustomAnimations(
        R.animator.card_flip_right_in, R.animator.card
        R.animator.card_flip_left_in, R.animator.card

    // Replace any fragments currently in the container vi
    // representing the next page (indicated by the just-i
    // variable).
    .replace(R.id.container, new CardBackFragment())

    // Add this transaction to the back stack, allowing us
    // to get to the front of the card.
    .addToBackStack(null)

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

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/zoom.html>

# 缩放View

这节课程师范怎样实现点击缩放动画，这对相册很有用，他能允许相片从缩略图转换成原图并填充屏幕提供动画。

下面展示了触摸缩放动画效果是什么样子，它将缩略图扩大并填充屏幕。



如果你想跳过看整个例子，[下载](#) App 样例然后运行缩放的例子。查看下列文件中的代码实现：

- src/TouchHighlightImageButton.java (简单的helper类，当image button被按下它显示蓝色高亮)
- src/ZoomActivity.java
- layout/activity\_zoom.xml

## 创建View

为你想缩放的内容创建一大一小两个版本布局文件。下面的例子为可点击的缩略图新建了一个 [ImageButton](#) 和一个 [ImageView](#) 来展示原图：

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

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <ImageButton
            android:id="@+id/thumb_button_1"
            android:layout_width="100dp"
            android:layout_height="75dp"
            android:layout_marginRight="1dp"
            android:src="@drawable/thumb1"
            android:scaleType="centerCrop"
            android:contentDescription="@string/description_image_>

    </LinearLayout>

    <!-- This initially-hidden ImageView will hold the expanded/zoomed
        the images above. Without transformations applied, it takes up
        screen. To achieve the "zoom" animation, this view's bounds
        from the bounds of the thumbnail button above, to its final
        bounds.
    -->

    <ImageView
        android:id="@+id/expanded_image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="invisible"
        android:contentDescription="@string/description_zoom_touchable">

</FrameLayout>
```

## 设置缩放动画

一旦实现了布局，你需要设置触发缩放事件handler。下面的例子为ImageButton添加了一个View.OnClickListener，当用户点击按钮时它执行放大动画。

```
public class ZoomActivity extends FragmentActivity {
    // Hold a reference to the current animator,
    // so that it can be canceled mid-way.
    private Animator mCurrentAnimator;

    // The system "short" animation time duration, in milliseconds
    // duration is ideal for subtle animations or animations that
    // very frequently.
    private int mShortAnimationDuration;

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

        // Hook up clicks on the thumbnail views.

        final View thumb1View = findViewById(R.id.thumb_button_1);
        thumb1View.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                zoomImageFromThumb(thumb1View, R.drawable.image1);
            }
        });

        // Retrieve and cache the system's default "short" animation
        mShortAnimationDuration = getResources().getInteger(
            android.R.integer.config_shortAnimTime);
    }
    ...
}
```

## 缩放View

你现在需要适时应用放大动画了。通常来说，你需要按边界来从小号View放大到大号View。下面的方法告诉你如何实现缩放动画：

1. 把高清图像设置到“放大版”隐藏的[ImageView](#)中。为表简单，下面的例子在UI线程中加载了一张大图。但是你需要在一个单独的线程中来加载以免阻塞UI线程，然后再回到UI线程中设置。理想状况下，图片不要大过屏幕。
2. 计算[ImageView](#)开始和结束时的边界。
3. 同步地动态改变四个位置和大小属性[X](#), [Y](#) ([SCALE\\_X](#) 和 [SCALE\\_Y](#))，从起始点到结束点。这四个动画被加入到了[AnimatorSet](#)，所以你可以一起开始。
4. 缩回则运行相同的动画，但是是用户点击屏幕放大时的逆向效果。你可以在[ImageView](#)中添加一个[View.OnClickListener](#)来实现它。当点击时，[ImageView](#)缩回到原来缩略图的大小，然后设置它的visibility为[GONE](#)来隐藏。

```
private void zoomImageFromThumb(final View thumbView, int imageResId) {
    // If there's an animation in progress, cancel it
    // immediately and proceed with this one.
    if (mCurrentAnimator != null) {
        mCurrentAnimator.cancel();
    }

    // Load the high-resolution "zoomed-in" image.
    final ImageView expandedImageView = (ImageView) findViewById(
        R.id.expanded_image);
    expandedImageView.setImageResource(imageResId);

    // Calculate the starting and ending bounds for the zoomed-in view.
    // This step involves lots of math. Yay, math.
    final Rect startBounds = new Rect();
    final Rect finalBounds = new Rect();
    final Point globalOffset = new Point();

    // The start bounds are the global visible rectangle of the thumbnail,
    // and the final bounds are the global visible rectangle of the
    // view. Also set the container view's offset as the origin for
    // bounds, since that's the origin for the positioning animation
    // properties (X, Y).
    thumbView.getGlobalVisibleRect(startBounds);
    findViewById(R.id.container)
        .getGlobalVisibleRect(finalBounds, globalOffset);
    startBounds.offset(-globalOffset.x, -globalOffset.y);
    finalBounds.offset(-globalOffset.x, -globalOffset.y);

    // Adjust the start bounds to be the same aspect ratio as the
    // bounds using the "center crop" technique. This prevents undifferentiated
    // stretching during the animation. Also calculate the start scale
    // factor (the end scaling factor is always 1.0).
    float startScale;
    if ((float) finalBounds.width() / finalBounds.height() >
        (float) startBounds.width() / startBounds.height())
        startScale = (float) startBounds.height() /
            (float) finalBounds.height();
    else
        startScale = (float) finalBounds.width() /
            (float) startBounds.width();
    mCurrentAnimator = AnimatorSet.of(ObjectAnimator
        .ofFloat(expandedImageView, View.SCALE_X, 0f, 1f),
        ObjectAnimator
        .ofFloat(expandedImageView, View.SCALE_Y, 0f, 1f),
        ObjectAnimator
        .ofFloat(expandedImageView, View.LEFT, startBounds.left,
            finalBounds.left),
        ObjectAnimator
        .ofFloat(expandedImageView, View.TOP, startBounds.top,
            finalBounds.top))
        .start();
}
```

```
        > (float) startBounds.width() / startBounds.height())
    // Extend start bounds horizontally
    startScale = (float) startBounds.height() / finalBounds.height();
    float startWidth = startScale * finalBounds.width();
    float deltaWidth = (startWidth - startBounds.width()) / 2;
    startBounds.left -= deltaWidth;
    startBounds.right += deltaWidth;
} else {
    // Extend start bounds vertically
    startScale = (float) startBounds.width() / finalBounds.width();
    float startHeight = startScale * finalBounds.height();
    float deltaHeight = (startHeight - startBounds.height()) / 2;
    startBounds.top -= deltaHeight;
    startBounds.bottom += deltaHeight;
}

// Hide the thumbnail and show the zoomed-in view. When the animation begins, it will position the zoomed-in view in the place of the thumbnail.
thumbView.setAlpha(0f);
expandedImageView.setVisibility(View.VISIBLE);

// Set the pivot point for SCALE_X and SCALE_Y transformations to the top-left corner of the zoomed-in view (the default is the center of the view).
expandedImageView.setPivotX(0f);
expandedImageView.setPivotY(0f);

// Construct and run the parallel animation of the four translation and scale properties (X, Y, SCALE_X, and SCALE_Y).
AnimatorSet set = new AnimatorSet();
set
    .play(ObjectAnimator.ofFloat(expandedImageView, View.X,
                                 startBounds.left, finalBounds.left))
    .with(ObjectAnimator.ofFloat(expandedImageView, View.Y,
                                 startBounds.top, finalBounds.top))
    .with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_X,
                                 startScale, 1f)).with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_Y, startScale, 1f));
set.setDuration(mShortAnimationDuration);
set.setInterpolator(new DecelerateInterpolator());
set.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        mCurrentAnimator = null;
    }

    @Override
    public void onAnimationCancel(Animator animation) {
        mCurrentAnimator = null;
    }
});
set.start();
mCurrentAnimator = set;
```

```
// Upon clicking the zoomed-in image, it should zoom back down
// to the original bounds and show the thumbnail instead of
// the expanded image.
final float startScaleFinal = startScale;
expandedImageView.setOnClickListener(new View.OnClickListener(
    @Override
    public void onClick(View view) {
        if (mCurrentAnimator != null) {
            mCurrentAnimator.cancel();
        }

        // Animate the four positioning/sizing properties in p
        // back to their original values.
        AnimatorSet set = new AnimatorSet();
        set.play(ObjectAnimator
            .ofFloat(expandedImageView, View.X, startB
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.Y,startBounds.top))
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.SCALE_X, startScaleFi
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.SCALE_Y, startScaleFi
        set.setDuration(mShortAnimationDuration);
        set.setInterpolator(new DecelerateInterpolator());
        set.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                thumbView.setAlpha(1f);
                expandedImageView.setVisibility(View.GONE);
                mCurrentAnimator = null;
            }

            @Override
            public void onAnimationCancel(Animator animation) {
                thumbView.setAlpha(1f);
                expandedImageView.setVisibility(View.GONE);
                mCurrentAnimator = null;
            }
        });
        set.start();
        mCurrentAnimator = set;
    }
));
}
```

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/animation/layout.html>

# 布局变更动画

布局动画是一种预加载动画，系统在你每次改变的布局配置时运行它。你需要做的仅是在布局文件里设置属性告诉Android系统为你这些布局的变更应用动画。系统动画时为你默认执行的。

**小贴士:** 如果你想补充自定义布局动画，创建 [LayoutTransition](#) 对象和然后用 [setLayoutTransition\(\)](#) 方法把它加到布局中。

下面是在一个list中添加一项的默认布局动画：



如果你想跳过看整个例子，[下载](#) App 样例然后运行布局渐变的例子。查看下列文件中的代码实现：

1. src/LayoutChangesActivity.java
2. layout/activity\_layout\_changes.xml
3. menu/activity\_layout\_changes.xml

## 创建布局

在你的activity的XML布局文件中，为你想开启动画的布局设置 android:animateLayoutChanges 属性为真。例如：

```
<LinearLayout android:id="@+id/container"
    android:animateLayoutChanges="true"
    ...
/>
```

## 从布局中添加，更新或删除项目

现在，你需要做的就是添加，删除或更新布局里的项目，然后这些项目就会自动显示动画啦：

```
private ViewGroup mContainerView;  
...  
private void addItem() {  
    View newView;  
    ...  
    mContainerView.addView(newView, 0);  
}
```

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/training/building-connectivity.html>

# 连接

These classes teach you how to connect your app to the world beyond the user's device. You'll learn how to connect to other devices in the area, connect to the Internet, backup and sync your app's data, and more.

## [无线连接设备 - Connecting Devices Wirelessly](#)

How to find and connect to local devices using Network Service Discovery and how to create peer-to-peer connections with Wi-Fi.

## [网络连接操作 - Performing Network Operations](#)

How to create a network connection, monitor the connection for changes in connectivity, and perform transactions with XML data.

## [高效下载 - Transferring Data Without Draining the Battery](#)

How to minimize your app's impact on the battery when performing downloads and other network transactions.

## [使用Sync Adapter传输数据 - Transferring Data Using Sync Adapters](#)

How to transfer data between the cloud and the device using the Android sync adapter framework

## [Volley](#)

### [云同步： Syncing to the Cloud](#)

### [解决云同步的保存冲突： Resolving Cloud Save Conflicts](#)

编写:[acenodie](#), 校对:

原文:<http://developer.android.com/training/connect-devices-wirelessly/index.html>

# 无线连接设备

除了能够在云端通信，Android的无线接口（wireless APIs）也允许同一局域网中的设备进行通信，甚至没有连接到网络上，而是物理上隔得很近，也可以相互通信。此外，网络服务发现（Network Service Discovery，简称NSD）可以进一步通过允许应用程序运行能相互通信的服务去寻找附近运行相同服务的设备。把这个功能整合到你的应用中可以提供一个范围广泛的特点，如在同一个房间，用户玩游戏，可以利用NSD实现从一个网络摄像头获取图像，或远程登录到在同一网络中的其他机器。

本节课介绍了一些使你的应用程序能够寻找和连接其他设备的主要的接口（APIs）。具体地说，它介绍了用于发现可用服务的NSD API和能实现点对点无线连接的无线点对点（the Wi-Fi Peer-to-Peer，简称Wi-Fi P2P）API。本节课也将告诉我们怎样将NSD和Wi-Fi P2P结合起来去检测其他设备所提供的服务，当检测到时，连接到相应的设备上。

# Lessons

- 使得网络服务可发现

学习如何广播由你自己的应用程序提供的服务，如何发现在本地网络上提供的服务并用NSD获取你将要连接的服务的详细信息。

- 使用WiFi建立P2P连接

学习如何获取附近附近的对等设备，如何创建一个设备接入点，如何连接到其他具有Wi-Fi P2P连接功能的设备。

- 使用WiFi P2P发现服务

学习如何使用WiFi P2P服务去发现附近的不在同一个网络的服务。

编写: [naizhengtan](#) - 校对

原文:

# 使得网络服务可发现

网络服务发现（Network Service Discovery）是一种在局域网内可以辨识并使用其他设备上提供的服务的技术。这种技术在端对端应用中能够提供大量帮助，例如文件共享、联机游戏等。Android提供了网络服务发现（NSD）相应的API，大大降低了其实现难度。

本讲将简要介绍如何创建NSD应用，使其能够在本地网络内广播自己的名称和链接信息，并且扫描网络发现其他NSD设备。最后，将介绍如何连接到运行着同样应用的另一台设备上。

## 注册NSD服务

**Note:** 这一步骤是选做的。如果你并不关心自己的服务是否被广播，你可以跳过这一步，直接尝试[发现网络中的服务](#)。

在局域网内注册自己服务的第一步是创建[NsdServiceInfo](#)对象。此对象包含的信息能够帮助其他设备决定是否要连接到你所提供的服务。

```
public void registerService(int port) {  
    // Create the NsdServiceInfo object, and populate it.  
    NsdServiceInfo serviceInfo = new NsdServiceInfo();  
  
    // The name is subject to change based on conflicts  
    // with other services advertised on the same network.  
    serviceInfo.setServiceName("NsdChat");  
    serviceInfo.setServiceType("_http._tcp.");  
    serviceInfo.setPort(port);  
    ....  
}
```

这段代码将所提供的服务命名为“NsdChat”。该名称将对所有局域网络中的设备可见。需要注意的是，在网络内该名称必须是独一无二的。Android系统会自动处理冲突的服务名称。如果同时有两个名为“NsdChat”的应用，其中一个会被自动转换为“NsdChat(1)”。

第二个参数设置了服务类型，即，使用的通信协议和传输层协议，语法是“< protocol >.\_< transportlayer >”。在上面的代码中，服务使用了TCP协议上的HTTP协议。如果应用想要提供打印服务（例如，一台网络打印机）应该将服务的类型设置为“\_ipp.\_tcp”。

**Note:** 互联网编号分配机构（International Assigned Numbers Authority，简称IANA）提供用于服务发现协议（例如NSD和Bonjour）的官方服务种类列表。你可以下载该列表了解相应的服务名称和端口号。如果你想用新的服务种类，应该向IANA官方提交申请。

当为你的服务设置端口号时，应该尽量避免将其硬编码在代码中，防止与其他应用产生冲突。例如，如果你的应用仅仅使用端口1337，就可能与其他使用1337端口的应用发生冲突。解决方法是，不要硬编码，使用下一个可用的端口。不必担心其他应用无法知晓服务的端口号，因为该信息将包含在服务的广播包中。接收到广播后，其他应用将从广播包中得知服务端口号，并通过端口连接到你的服务上。

如果你使用的是socket，你可以将端口号初始值设置为0来使用下一个可用端口。

```
public void initializeServerSocket() {  
    // Initialize a server socket on the next available port.  
    mServerSocket = new ServerSocket(0);  
  
    // Store the chosen port.
```

```
mLocalPort = mServerSocket.getLocalPort();  
...  
}
```

现在，你已经成功的创建了[NsdServiceInfo](#)对象，接下来要做的是实现[RegistrationListener](#)接口。该接口包含了注册在Android系统中的一系列回调函数，作用是通知应用程序服务注册/注销的成功和失败。

```
public void initializeRegistrationListener() {  
    mRegistrationListener = new NsdManager.RegistrationListener()  
  
        @Override  
        public void onServiceRegistered(NsdServiceInfo NsdServiceI  
            // Save the service name.  Android may have changed it  
            // resolve a conflict, so update the name you initiall  
            // with the name Android actually used.  
            mServiceName = NsdServiceInfo.getServiceName();  
        }  
  
        @Override  
        public void onRegistrationFailed(NsdServiceInfo serviceInf  
            // Registration failed!  Put debugging code here to de  
        }  
  
        @Override  
        public void onServiceUnregistered(NsdServiceInfo arg0) {  
            // Service has been unregistered.  This only happens w  
            // NsdManager.unregisterService() and pass in this lis  
        }  
  
        @Override  
        public void onUnregistrationFailed(NsdServiceInfo serviceI  
            // Unregistration failed.  Put debugging code here to  
        }  
    };  
}
```

万事俱备只欠东风，调用[registerService\(\)](#)方法，真正注册服务。

因为该方法是异步的，所以进一步的操作需要在[onServiceRegistered\(\)](#)方法中进行。

```
public void registerService(int port) {  
    NsdServiceInfo serviceInfo = new NsdServiceInfo();  
    serviceInfo.setServiceName("NsdChat");  
    serviceInfo.setServiceType("_http._tcp.");  
    serviceInfo.setPort(port);  
  
    mNsdManager = Context.getSystemService(Context.NSD_SERVICE);  
  
    mNsdManager.registerService(  
        serviceInfo, NsdManager.PROTOCOL_DNS_SD, mRegistration  
    }
```

# 发现网络中的服务

网络充斥着我们的生活，从网络打印机到网络摄像头，再到联网井字棋。网络服务发现是能让你的应用融入这一切功能的关键。你的应用需要侦听网络内服务的广播，发现可用的服务，过滤无效的信息。

与注册网络服务类似，服务发现需要两步骤：正确配置发现监听者（Discover Listener），以及调用[discoverServices\(\)](#)这个异步API。

首先，实例化一个实现[NsdManager.DiscoveryListener](#)接口的匿名类。下列代码是一个简单的范例：

```
public void initializeDiscoveryListener() {  
  
    // Instantiate a new DiscoveryListener  
    mDiscoveryListener = new NsdManager.DiscoveryListener() {  
  
        // Called as soon as service discovery begins.  
        @Override  
        public void onDiscoveryStarted(String regType) {  
            Log.d(TAG, "Service discovery started");  
        }  
  
        @Override  
        public void onServiceFound(NsdServiceInfo service) {  
            // A service was found! Do something with it.  
            Log.d(TAG, "Service discovery success" + service);  
            if (!service.getServiceType().equals(SERVICE_TYPE)) {  
                // Service type is the string containing the proto  
                // transport layer for this service.  
                Log.d(TAG, "Unknown Service Type: " + service.getS  
            } else if (service.getServiceName().equals(mServiceNam  
                // The name of the service tells the user what the  
                // connecting to. It could be "Bob's Chat App".  
                Log.d(TAG, "Same machine: " + mServiceName);  
            } else if (service.getServiceName().contains("NsdChat"))  
                mNsdManager.resolveService(service, mResolveListen  
            }  
        }  
  
        @Override  
        public void onServiceLost(NsdServiceInfo service) {  
            // When the network service is no longer available.  
            // Internal bookkeeping code goes here.  
            Log.e(TAG, "service lost" + service);  
        }  
  
        @Override  
        public void onDiscoveryStopped(String serviceType) {  
            Log.i(TAG, "Discovery stopped: " + serviceType);  
        }  
  
        @Override
```

```
public void onStartDiscoveryFailed(String serviceType, int
        Log.e(TAG, "Discovery failed: Error code:" + errorCode
        mNsdManager.stopServiceDiscovery(this);
    }

    @Override
    public void onStopDiscoveryFailed(String serviceType, int
        Log.e(TAG, "Discovery failed: Error code:" + errorCode
        mNsdManager.stopServiceDiscovery(this);
    }
}
```

NSD API通过使用该接口中的方法通知用户程序何时发现开始、何时发现失败、何时找到可用服务、何时服务丢失（丢失意味着“不再可用”）。在上述代码中，当发现了可用的服务时，程序做了几次检查。

1. 比较发现的服务名称与本地的服务，判断是否发现的是本机的服务（这是合法的）。
2. 检查服务的类型，确认是否可以接入。
3. 检查服务的名称，确认接入了正确的应用。

我们并不需要每次都检查服务名称，仅当你想要接入特定的应用时，再去判断。例如，应用只想与运行在其他设备上的相同应用通信。然而，如果应用仅仅想接入到一台网络打印机，那么看到服务类型是“\_ipp.\_tcp”的服务就足够了。

当配置好发现回调函数后，调用[discoverService\(\)](#)函数，其参数包括试图发现的服务种类、发现使用的协议、以及上一步创建的监听者。

```
mNsdManager.discoverServices(
    SERVICE_TYPE, NsdManager.PROTOCOL_DNS_SD, mDiscoveryListen
```

## 连接到网内的服务

当你的代码发现了网上可接入的服务，第一件必须做的事情是确认服务的连接信息。调用[resolveService\(\)](#)方法，并将实现了[NsdManager.ResolveListener](#)的对象和在[服务发现](#)过程中得到的[NsdServiceInfo](#)对象传入。通过该方法，可以将连接信息从[NsdServiceInfo](#)对象中解析出来。

```
public void initializeResolveListener() {
    mResolveListener = new NsdManager.ResolveListener() {

        @Override
        public void onResolveFailed(NsdServiceInfo serviceInfo, int
            // Called when the resolve fails. Use the error code
            Log.e(TAG, "Resolve failed" + errorCode);
    }

    @Override
    public void onServiceResolved(NsdServiceInfo serviceInfo)
        Log.e(TAG, "Resolve Succeeded. " + serviceInfo);

        if (serviceInfo.getServiceName().equals(mServiceName))
            Log.d(TAG, "Same IP.");
            return;
        }
        mService = serviceInfo;
        int port = mService.getPort();
        InetAddress host = mService.getHost();
    }
};

}
```

当服务解析完成后，你将获得服务的详细资料，包括其IP地址和端口号。此时，你有了服务的地址和端口，可以通过创建自己网络连接与服务进行通讯。

## 当程序退出时注销服务

在应用的生命周期中正确的开启和关闭NSD服务是十分关键的。在程序退出时注销服务可以防止其他程序因为不知道服务退出而反复尝试连接的行为。另外，服务发现是一种开销很大的操作，应该随着父Activity的暂停而停止，当用户返回该界面是再开启。因此，开发者应该重载Activity的生命周期函数，并正确操作服务的广播和发现。

```
//In your application's Activity

@Override
protected void onPause() {
    if (mNsdHelper != null) {
        mNsdHelper.tearDown();
    }
    super.onPause();
}

@Override
protected void onResume() {
    super.onResume();
    if (mNsdHelper != null) {
        mNsdHelper.registerService(mConnection.getLocalPort());
        mNsdHelper.discoverServices();
    }
}

@Override
protected void onDestroy() {
    mNsdHelper.tearDown();
    mConnection.tearDown();
    super.onDestroy();
}

// NsdHelper's tearDown method
public void tearDown() {
    mNsdManager.unregisterService(mRegistrationListener);
    mNsdManager.stopServiceDiscovery(mDiscoveryListener);
}
```

编写: [naizhengtan](#) - 校对

原文:

# 使用WiFi建立P2P连接

Android提供的Wi-Fi端对端（P2P）技术允许应用程序无需连接到网络和热点的情况下连接到附近的设备。（Android Wi-Fi P2P框架遵循[Wi-Fi Direct™](#)证书程序）Wi-Fi P2P技术使得应用程序可以快速发现附近的设备并与之交互。相比于蓝牙技术，Wi-Fi P2P的优势是具有较大的连接范围。

本节主要内容是使用Wi-Fi P2P技术发现并连接到附近的设备。

## 配置应用权限

使用Wi-Fi P2P技术，需要添加`CHANGE_WIFI_STATE`, `ACCESS_WIFI_STATE`以及`INTERNET`三种权限到应用的manifest文件。Wi-Fi P2P技术虽然不需要访问互联网，但是它会使用Java中的标准socket。而使用socket需要具有`INTERNET`权限，这也是Wi-Fi P2P技术需要申请该权限的原因。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
}
```

## BroadCast Receiver和Peer-to-peer Manager

使用Wi-Fi P2P的时候需要侦听相关的广播事件（broadcast intent）。所以在应用中需要实例化一个[IntentFilter](#)，并将其设置为侦听下列事件：

- [WIFI\\_P2P\\_STATE\\_CHANGED\\_ACTION](#)  
指示Wi-Fi P2P是否开启
- [WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#)  
代表可达的同伴（peer）列表发生了变化
- [WIFI\\_P2P\\_CONNECTION\\_CHANGED\\_ACTION](#)  
表明Wi-Fi P2P的连接状态发生了改变
- [WIFI\\_P2P\\_THIS\\_DEVICE\\_CHANGED\\_ACTION](#)  
指示设备的详细配置发生了变化

```
private final IntentFilter intentFilter = new IntentFilter();
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Indicates a change in the Wi-Fi P2P status.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_A

    // Indicates a change in the list of available peers.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_A

    // Indicates the state of Wi-Fi P2P connectivity has changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHA

    // Indicates this device's details have changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHA

    ...
}
```

在[onCreate\(\)](#)方法的最后，需要获得[WifiPpManager](#)的实例，并调用它的[initialize\(\)](#)方法。该方法将返回[WifiP2pManager.Channel](#)对象。你的应用将使用该对象与Wi-Fi P2P框架进行交互。

```
@Override
Channel mChannel;

public void onCreate(Bundle savedInstanceState) {
    ....
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_
    mChannel = mManager.initialize(this, getMainLooper(), null);
}
```

接下来，创建一个新的BroadcastReceiver类侦听系统中Wi-Fi P2P的状态变化。在onReceive()方法中，加入对上述四种不同P2P状态变化的处理。

```
@Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action))
            // Determine if Wifi P2P mode is enabled or not, alert
            // the Activity.
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, 0);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                activity.setIsWifiP2pEnabled(true);
            } else {
                activity.setIsWifiP2pEnabled(false);
            }
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action))
            // The peer list has changed! We should probably do something
            // with that.

        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action))
            // Connection state changed! We should probably do something
            // with that.

        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action))
            DeviceListFragment fragment = (DeviceListFragment) activity
                .findFragmentById(R.id.frag_list);
            fragment.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
    }
}
```

最后，在主界面开启时，加入注册intent filter和broadcast receiver的代码，并在暂停或关闭时，注销它们。最好的位置是在onResume()和onPause()方法中。

```
/** register the BroadcastReceiver with the intent values to be monitored */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(mManager, mChannel);
    registerReceiver(receiver, intentFilter);
}

@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

## 初始化同伴发现（Peer Discovery）过程

在Wi-Fi P2P中，应用通过调用[discoverPeers\(\)](#)搜寻附近的设备。该方法需要以下参数：

- 上节中调用WifiP2pManager的initialize()函数获得的[WifiP2pManager.Channel](#)对象
- 一个对[WifiP2pManager.ActionListener](#)接口的实现，包括了当系统成功/失败发现所调用的方法

```
mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener

    @Override
    public void onSuccess() {
        // Code for when the discovery initiation is successful
        // No services have actually been discovered yet, so this
        // can often be left blank. Code for peer discovery goes in
        // onReceive method, detailed below.
    }

    @Override
    public void onFailure(int reasonCode) {
        // Code for when the discovery initiation fails goes here
        // Alert the user that something went wrong.
    }
});
```

需要注意的是，在此的成功仅仅表示对同伴发现（Peer Discovery）的过程完成初始化。方法discoverPeers()开启了发现过程并且立即返回。系统会通过调用WifiP2pManager.ActionListener中的方法通知应用同伴发现过程初始化是否正确。同时，同伴发现过程本身仍然继续运行，直到一条连接或者一个P2P小组建立。

## 获取同伴列表

在完成同伴发现过程的初始化后，我们需要进一步获取附近的同伴列表。第一步是实现[WifiP2pManager.PeerListListener](#)接口。该接口提供了Wi-Fi P2P框架发现的同伴信息。下列代码实现了相应功能：

```
private List peers = new ArrayList();
...
private PeerListListener peerListListener = new PeerListListener
@Override
public void onPeersAvailable(WifiP2pDeviceList peerList) {

    // Out with the old, in with the new.
    peers.clear();
    peers.addAll(peerList.getDeviceList());

    // If an AdapterView is backed by this data, notify it
    // of the change. For instance, if you have a ListView
    // peers, trigger an update.
    ((WiFiPeerListAdapter) getListAdapter()).notifyDataSetChanged();
    if (peers.size() == 0) {
        Log.d(WiFiDirectActivity.TAG, "No devices found");
        return;
    }
}
}
```

接下来，完善上文广播接收者（Broadcast Receiver）的onReceiver()方法。当收到[WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#)事件时，调用[requestPeer\(\)](#)方法获取同伴列表。在此，需要将WifiP2pManager.PeerListListener对象传递给该方法。一种方法是在广播接收者构造时，就将对象作为参数传入。

现在，一个[WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#)事件将触发应用对同伴列表的更新了。

## 连接到同伴

为了连接到一个同伴，你需要创一个新的对象[WifiP2pConfig](#)，并将设备信息从[WifiP2pDevice](#)拷贝到其中，最后调用[connect\(\)](#)方法。

```
@Override
public void connect() {
    // Picking the first device found on the network.
    WifiP2pDevice device = peers.get(0);

    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;
    config.wps.setup = WpsInfo.PBC;

    mManager.connect(mChannel, config, new ActionListener() {

        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ign
        }

        @Override
        public void onFailure(int reason) {
            Toast.makeText(WiFiDirectActivity.this, "Connect f
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

在本段代码中的[WifiP2pManager.ActionListener](#)仅能通知你初始化的成功或失败。想要侦听连接状态的变化，需要实现[WifiP2pManager.ConnectionInfoListener](#)接口。接口中的[onConnectionInfoAvailable\(\)](#)回调函数会在连接状态发生改变时通知应用程序。当有多个设备同时试图连接到一台设备时（例如多人游戏或者聊天群），这一台设备将被指定为“群主”（group owner）。

```
@Override
public void onConnectionInfoAvailable(final WifiP2pInfo info)

    // InetAddress from WifiP2pInfo struct.
    InetAddress groupOwnerAddress = info.groupOwnerAddress.get

    // After the group negotiation, we can determine the group
    if (info.groupFormed && info.isGroupOwner) {
        // Do whatever tasks are specific to the group owner.
        // One common case is creating a server thread and acc
        // incoming connections.
    } else if (info.groupFormed) {
        // The other device acts as the client. In this case,
        // you'll want to create a client thread that connects
        // owner.
    }
```

```
}
```

此时，回头继续完善广播接收者的onReceive()方法，并将对WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION事件的处理补全。当该事件发生后，调用requestConnectionInfo()方法。此方法为异步，所以结果将会被你提供的WifiP2pManager.ConnectionInfoListener)所获取。

```
...
} else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(intent.getAction())) {
    if (mManager == null) {
        return;
    }

    NetworkInfo networkInfo = (NetworkInfo) intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
    if (networkInfo.isConnected()) {
        // We are connected with the other device, request
        // info to find group owner IP
        mManager.requestConnectionInfo(mChannel, connectionInfoListener);
    }
}
```

编写: [naizhengtan](#) - 校对

原文:

# 使用WiFi P2P发现服务

在本章第一节“[使得网络服务可发现](#)”中介绍了如何在局域网中发现并连接到其他设备的服务上。然而，即使在不接入网络的环境中，Wi-Fi P2P的发现服务也可以使你的应用直接连接到附近的设备。与此同时，你也可以向外公布自己设备上的服务。Wi-Fi P2P发现服务的这种能力可以在没有局域网或者网络热点的情况下，帮助不同设备上的应用进行通信。

虽然本节所述的API与第一节NSD（Network Service Discovery）的API相似，但是具体的实现代码却截然不同。本节将讲述如何通过Wi-Fi P2P技术发现附近可用设备中的服务。假设读者已经对Wi-Fi P2P的API有一定了解。

## 配置Manifest

使用Wi-Fi P2P技术，需要添加[CHANGE\\_WIFI\\_STATE](#), [ACCESS\\_WIFI\\_STATE](#)以及[INTERNET](#)三种权限到应用的manifest文件。 Wi-Fi P2P技术虽然不需要访问互联网，但是它会使用Java中的标准socket。而使用socket需要具有INTERNET权限，这也是Wi-Fi P2P技术需要申请该权限的原因。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
}
```

## 添加本地服务

如果你想提供一个本地服务，就需要在服务发现框架中注册该服务。当本地服务被成功注册，系统将自动回复所有来自附近的服务发现请求。

三步创建本地服务：

1. 新建[WifiP2pServiceInfo](#)对象
2. 加入想创立服务的详细信息
3. 调用[addLocalService\(\)](#)注册该服务

```
private void startRegistration() {
    // Create a string map containing information about your
    Map record = new HashMap();
    record.put("listenport", String.valueOf(SERVER_PORT));
    record.put("buddynname", "John Doe" + (int) (Math.random() * 100));
    record.put("available", "visible");

    // Service information. Pass it an instance name, service
    // _protocol._transportlayer , and the map containing
    // information other devices will want once they connect to
    WifiP2pDnsSdServiceInfo serviceInfo =
        WifiP2pDnsSdServiceInfo.newInstance("_test", "_pre");

    // Add the local service, sending the service info, network
    // and listener that will be used to indicate success or failure
    // of the request.
    mManager.addLocalService(channel, serviceInfo, new ActionListener<Boolean>() {
        @Override
        public void onSuccess() {
            // Command successful! Code isn't necessarily needed
            // Unless you want to update the UI or add logging
        }

        @Override
        public void onFailure(int arg0) {
            // Command failed. Check for P2P_UNSUPPORTED, ERROR
        }
    });
}
```

## 发现附近的服务

Android使用回调函数通知应用程序附近可用的服务，因此发现服务的第一步是设置这些回调函数。新建一个[WifiP2pManager.DnsSdTxtRecordListener](#)实例侦听实时收到的记录（record）。这些记录是来自其他设备的广播。当收到记录时，将其中的设备地址和其他相关信息拷贝出，供之后使用。下面的例子假设这条记录不仅包含设备的身份，还包含一个名为“buddynname”的域（field）。

```
final HashMap<String, String> buddies = new HashMap<String, String>();
...
private void discoverService() {
    DnsSdTxtRecordListener txtListener = new DnsSdTxtRecordListener() {
        @Override
        /* Callback includes:
         * fullDomain: full domain name: e.g "printer._ipp._tcp.lo
         * record: TXT record data as a map of key/value pairs.
         * device: The device running the advertised service.
         */
        public void onDnsSdTxtRecordAvailable(
            String fullDomain, Map record, WifiP2pDevice device) {
            Log.d(TAG, "DnsSdTxtRecord available - " + record.toString());
            buddies.put(device.deviceAddress, record.get("buddynname"));
        }
    };
    ...
}
```

接下来创建[WifiP2pManager.DnsSdServiceResponseListener](#)对象，用以获取目标服务的信息。这个对象将接受服务的实际描述以及连接信息。上一段代码构建了一个包含设备地址和“buddynname”键值对的Map对象。[WifiP2pManager.DnsSdServiceResponseListener](#)对象使用这些配对信息将DNS记录和对应的服务信息对应起来。当上述两个侦听对象构建完成了，调用[setDnsSdResponseListeners\(\)](#)将他们加入[WifiP2pManager](#)。

```
private void discoverService() {
    ...
    DnsSdServiceResponseListener servListener = new DnsSdServiceResponseListener() {
        @Override
        public void onDnsSdServiceAvailable(String instanceName, String type,
            WifiP2pDevice resourceType) {
            // Update the device name with the human-friendly
            // the DnsTxtRecord, assuming one arrived.
            resourceType.deviceName = buddies
                .containsKey(resourceType.deviceAddress) ?
                .get(resourceType.deviceAddress) : resourceType.deviceName;
            ...
            // Add to the custom adapter defined specifically
            // wifi devices.
            WiFiDirectServicesList fragment = (WiFiDirectServicesList)
                .findFragmentById(R.id.frag_peerlist);
        }
    };
}
```

```
        WiFiDevicesAdapter adapter = ((WiFiDevicesAdapter)
            .getListAdapter());

        adapter.add(resourceType);
        adapter.notifyDataSetChanged();
        Log.d(TAG, "onBonjourServiceAvailable " + instance
    }
};

mManager.setDnsSdResponseListeners(channel, servListener, txtL
...
}
```

然后创建服务请求，并调用[addServiceRequest\(\)](#)方法。这个方法也需要一个侦听者（Listener）报告请求成功与失败。

```
serviceRequest = WifiP2pDnsSdServiceRequest.newInstance();
mManager.addServiceRequest(channel,
    serviceRequest,
    new ActionListener() {
        @Override
        public void onSuccess() {
            // Success!
        }

        @Override
        public void onFailure(int code) {
            // Command failed. Check for P2P_UNSUPPORTED
        }
    });
}
```

最后调用[discoverServices\(\)](#)。

```
mManager.discoverServices(channel, new ActionListener() {

    @Override
    public void onSuccess() {
        // Success!
    }

    @Override
    public void onFailure(int code) {
        // Command failed. Check for P2P_UNSUPPORTED, ERR
        if (code == WifiP2pManager.P2P_UNSUPPORTED) {
            Log.d(TAG, "P2P isn't supported on this device"
        } else if(...)

        ...
    }
});
```

如果所有部分都配置正确，你应该就能看到正确的结果了！如果遇到了问题，你可以查看[WifiP2pManager.ActionListener](#)中的回调函数。它们能够指示操作是否成功。你可以将

debug的代码放置在[onFailure\(\)](#)中来诊断问题。其中的一些错误码（Error Code）也许能为你带来不小启发。下面是一些常见的错误：

- [P2P\\_UNSUPPORTED](#)

Wi-Fi P2P 不被现在的设备支持。

- [BUSY](#)

系统忙

- [ERROR](#)

内部错误

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/basics/network-ops/index.html>

# 网络连接操作

这一章会介绍一些基本的网络操作，监视网络链接（包括网络改变），让用户来控制app对网络的选择使用。还会介绍如何解析与使用XML数据。

[NetworkUsage.zip](#)

通过学习这章节的课程，你会学习一些基础知识，如何在最小化网络阻塞的情况下，创建一个高效的app，用来下载数据与解析数据。

你还可以参考下面文章进阶学习：

- [Optimizing Battery Life](#)
- [Transferring Data Without Draining the Battery](#)
- [Web Apps Overview](#)

# Lessons

- [连接到网络 - Connecting to the Network](#)

Learn how to connect to the network, choose an HTTP client, and perform network operations outside of the UI thread.

- [管理使用的网络 - Managing Network Usage](#)

Learn how to check a device's network connection, create a preferences UI for controlling network usage, and respond to connection changes.

- [解析XML数据 - Parsing XML Data](#)

Learn how to parse and consume XML data.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/basics/network-ops/connecting.html>

# 连接到网络Connecting to the Network

这一课会演示如何实现一个简单的连接到网络的程序。它提供了一些你应该follow的最好示例，用来创建最简单的网络连接程序。请注意，想要执行网络操作首先需要在程序的manifest文件中添加下面的permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_S
```

## Choose an HTTP Client(选择一个HTTP Client)

大多数连接网络的Android app会使用HTTP来发送与接受数据。Android提供了两种HTTP clients: [HttpURLConnection](#) 与Apache [HttpClient](#)。他们二者均支持HTTPS，都以流方式进行上传与下载，都有可配置timeout, IPv6 与连接池（connection pooling）。推荐从**Gingerbread**版本开始使用 **HttpURLConnection**。关于这部分的更多详情，请参考 [Android's HTTP Clients](#)。

## Check the Network Connection(检测网络连接)

在你的app尝试进行网络连接之前，需要检测当前是否有可用的网络。请注意，设备可能会不在网络覆盖范围内，或者用户可能关闭Wi-Fi与移动网络连接。

```
public void myClickHandler(View view) {  
    ...  
    ConnectivityManager connMgr = (ConnectivityManager)  
        getSystemService(Context.CONNECTIVITY_SERVICE);  
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();  
    if (networkInfo != null && networkInfo.isConnected()) {  
        // fetch data  
    } else {  
        // display error  
    }  
    ...  
}
```

## Perform Network Operations on a Separate Thread(在另外一个Thread执行网络操作)

网络操作会遇到不可预期的延迟。显然为了避免一个不好的用户体验，总是在UI Thread之外去执行网络操作。AsyncTask类提供了一种简单的方式来处理这个问题。关于更多的详情，请参考[Multithreading For Performance](#)。

在下面的代码示例中，myClickHandler()方法会触发一个新的DownloadWebpageTask().execute(stringUrl)。它继承自AsyncTask，实现了下面两个方法：

- [doInBackground\(\)](#) 执行 downloadUrl()方法。Web URL作为参数，方法downloadUrl()获取并处理网页返回的数据，执行完毕后，传递结果到onPostExecute()。参数类型为String。
- [onPostExecute\(\)](#) 获取到返回数据并显示到UI上。

```
public class HttpExampleActivity extends Activity {  
    private static final String DEBUG_TAG = "HttpExample";  
    private EditText urlText;  
    private TextView textView;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        urlText = (EditText) findViewById(R.id.myUrl);  
        textView = (TextView) findViewById(R.id.myText);  
    }  
  
    // When user clicks button, calls AsyncTask.  
    // Before attempting to fetch the URL, makes sure that there is  
    public void myClickHandler(View view) {  
        // Gets the URL from the UI's text field.  
        String urlString = urlText.getText().toString();  
        ConnectivityManager connMgr = (ConnectivityManager)  
            getSystemService(Context.CONNECTIVITY_SERVICE);  
        NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();  
        if (networkInfo != null && networkInfo.isConnected()) {  
            new DownloadWebpageText().execute(stringUrl);  
        } else {  
            textView.setText("No network connection available.");  
        }  
    }  
  
    // Uses AsyncTask to create a task away from the main UI thread.  
    // URL string and uses it to create an HttpURLConnection. Once  
    // has been established, the AsyncTask downloads the contents  
    // an InputStream. Finally, the InputStream is converted into  
    // displayed in the UI by the AsyncTask's onPostExecute method.  
    private class DownloadWebpageText extends AsyncTask {  
        @Override  
        protected String doInBackground(String... urls) {  
  
            // params comes from the execute() call: params[0] is
```

```
        try {
            return downloadUrl(urls[0]);
        } catch (IOException e) {
            return "Unable to retrieve web page. URL may be invalid";
        }
    }
    // onPostExecute displays the results of the AsyncTask.
    @Override
    protected void onPostExecute(String result) {
        textView.setText(result);
    }
}
...
}
```

关于上面那段代码的示例详解，请参考下面：

- When users click the button that invokes myClickHandler(), the app passes the specified URL to the AsyncTask subclass DownloadWebpageTask.
- The AsyncTask method doInBackground() calls the downloadUrl() method.
- The downloadUrl() method takes a URL string as a parameter and uses it to create a URL object.
- The URL object is used to establish an HttpURLConnection.
- Once the connection has been established, the HttpURLConnection object fetches the web page content as an InputStream.
- The InputStream is passed to the readIt() method, which converts the stream to a string.
- Finally, the AsyncTask's onPostExecute() method displays the string in the main activity's UI.

## Connect and Download Data(连接并下载数据)

在执行网络交互的线程里面，你可以使用 HttpURLConnection 来执行一个 GET 类型的操作并下载数据。在你调用 connect()之后，你可以通过调用getInputStream()来得到一个包含数据的[InputStream](#) 对象。

在下面的代码示例中，doInBackground() 方法会调用downloadUrl(). 这个 downloadUrl() 方法使用给予的URL，通过 HttpURLConnection 连接到网络。一旦建立连接，app 使用 getInputStream() 来获取数据。

```
// Given a URL, establishes an HttpURLConnection and retrieves
// the web page content as a InputStream, which it returns as
// a string.
private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
    // Only display the first 500 characters of the retrieved
    // web page content.
    int len = 500;

    try {
        URL url = new URL(myurl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        // Starts the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        is = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString = readIt(is, len);
        return contentAsString;

        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (is != null) {
            is.close();
        }
    }
}
```

请注意，getResponseCode() 会返回连接状态码( status code). 这是一种获知额外网络连接信息的有效方式。status code 是 200 则意味着连接成功。

## Convert the InputStream to a String(把InputStream的数据转换为String)

InputStream 是一种可读的byte数据源。如果你获得了一个 InputStream, 通常会进行decode或者转换为制定的数据类型。例如, 如果你是在下载一张image数据, 你可能需要像下面一下进行decode:

```
InputStream is = null;
...
Bitmap bitmap = BitmapFactory.decodeStream(is);
ImageView imageView = (ImageView) findViewById(R.id.image_view);
imageView.setImageBitmap(bitmap);
```

在上面演示的示例中, InputStream 包含的是web页面的文本内容。下面会演示如何把 InputStream 转换为string, 以便显示在UI上。

```
// Reads an InputStream and converts it to a String.
public String readIt(InputStream stream, int len) throws IOException {
    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/basics/network-ops/managing.html>

# 管理使用的网络Managing Network Usage

这一课会介绍如何细化管理使用的网络资源。如果你的程序需要执行很多网络操作，你应该提供用户设置选项来允许用户控制程序的数据偏好。例如，同步数据的频率，是否只在连接到WiFi才进行下载与上传操作，是否在漫游时使用套餐数据流量等等。这样用户才能在快到达流量上限时关闭你的程序获取数据功能。

关于如何编写一个最小化下载与网络操作对电量影响的程序，请参考：

- [Optimizing Battery Life:](#)
- [Transferring Data Without Draining the Battery:](#)

## Check a Device's Network Connection(检查设备的网络连接信息)

设备可以有许多种网络连接。关于所有可能的网络连接类型，请看[ConnectivityManager](#).

通常Wi-Fi是比较快的。移动数据通常都是需要按流量计费，会比较贵。通常我们会选择让app在连接到WiFi时去获取大量的数据。

那么，我们就需要在执行网络操作之前检查当前连接的网络信息。这样可以防止你的程序不经意连接使用了非意向的网络频道。为了实现这个目的，我们需要使用到下面两个类：

- [ConnectivityManager](#): Answers queries about the state of network connectivity. It also notifies applications when network connectivity changes.
- [NetworkInfo](#): Describes the status of a network interface of a given type (currently either Mobile or Wi-Fi).

下面示例了检查WiFi与Mobile是否连接上(请注意available与isConnected的区别)：

```
private static final String DEBUG_TAG = "NetworkStatusExample";
...
ConnectivityManager connMgr = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

一个更简单的检查网络是否可用的示例如下：

```
public boolean isOnline() {
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```

你可以使用[NetworkInfo.DetailedState](#), 来获取更加详细的网络信息。

# Manage Network Usage(管理网络使用)

你可以实现一个偏好设置的activity，来允许用户设置程序的网络资源的使用。例如：

- 你可以允许用户在仅仅连接到WiFi时上传视频。
- 你可以根据诸如网络可用等条件来选择是否做同步的操作。

网络操作需要添加下面的权限：

- [android.permission.INTERNET](#)—Allows applications to open network sockets.
- [android.permission.ACCESS\\_NETWORK\\_STATE](#)—Allows applications to access information about networks.

你可以为你的设置Activity声明intent filter for the ACTION\_NETWORK\_USAGE action (introduced in Android 4.0),这样你的这个activity就可以提供数据控制的选项了。在章节概览提供的Sample中，这个action is handled by the class SettingsActivity, 它提供了偏好设置UI来让用户决定何时进行下载。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.networkusage"
    ...>

    <uses-sdk android:minSdkVersion="4"
              android:targetSdkVersion="14" />

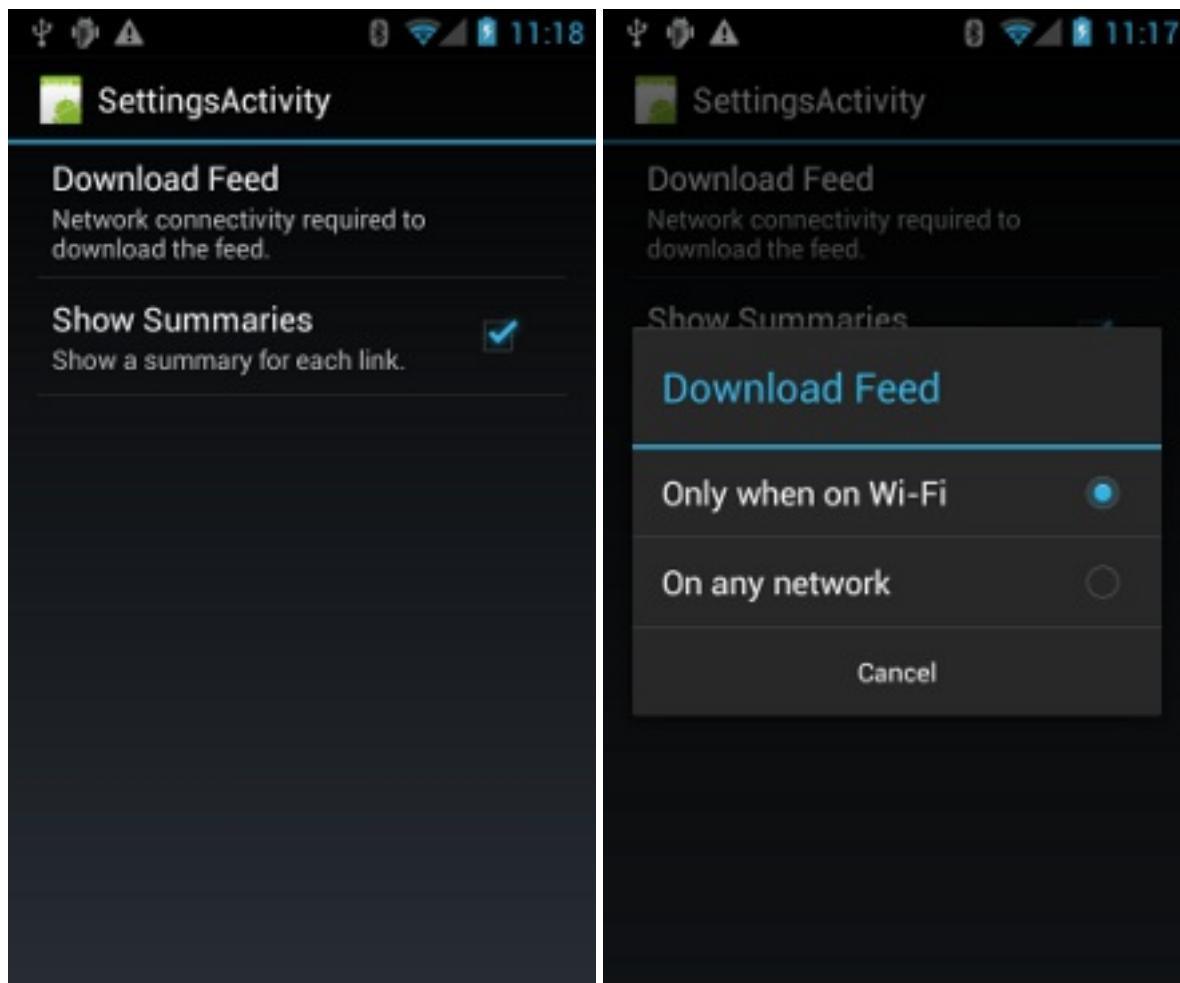
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        ...>
        ...
        <activity android:label="SettingsActivity" android:name=".SettingsActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Implement a Preferences Activity(实现一个偏好设置activity)

正如上面看到的那样， SettingsActivity 是一个 PreferenceActivity 的子类。

所实现的功能见下图：



下面是一个 SettingsActivity。请注意它实现了 OnSharedPreferenceChangeListener。当用户改变了他的偏好，就会触发 onSharedPreferenceChanged()，这个方法会设置 refreshDisplay 为 true(这里的变量存在于自己定义的 activity，见下一部分的代码示例)。这会使的当用户返回到 main activity 的时候进行 refresh。(请注意，代码中的注释，不得不说，Googler 写的 Code 看起来就是舒服)

```
public class SettingsActivity extends PreferenceActivity implements  
    OnSharedPreferenceChangeListener {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Loads the XML preferences file  
        addPreferencesFromResource(R.xml.preferences);  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        refreshDisplay();  
    }  
}
```

```
// Registers a listener whenever a key changes
getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);

@Override
protected void onPause() {
    super.onPause();

    // Unregisters the listener set in onResume().
    // It's best practice to unregister listeners when your app
    // has unnecessary system overhead. You do this in onPause().
    getPreferenceScreen().getSharedPreferences().unregisterOnSharedPreferenceChangeListener(this);

    // When the user changes the preferences selection,
    // onSharedPreferenceChanged() restarts the main activity as a
    // task. Sets the refreshDisplay flag to "true" to indicate
    // the main activity should update its display.
    // The main activity queries the PreferenceManager to get the
    // new preference values.

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPrefs, String key) {
        // Sets refreshDisplay to true so that when the user returns
        // to the activity, the display refreshes to reflect the new setting.
        NetworkActivity.refreshDisplay = true;
    }
}
```

## Respond to Preference Changes(对偏好改变进行响应)

当用户在设置界面改变了偏好，它通常都会对app的行为产生影响。在下面的代码示例中，app会在onStart()方法里面检查偏好设置。如果设置的类型与当前设备的网络连接类型相一致，那么程序就会下载数据并刷新显示。(for example, if the setting is "Wi-Fi" and the device has a Wi-Fi connection)。(这是一个很好的代码示例，如何选择合适的网络类型进行下载操作)

```
public class NetworkActivity extends Activity {  
    public static final String WIFI = "Wi-Fi";  
    public static final String ANY = "Any";  
    private static final String URL = "http://stackoverflow.com/fe  
  
    // Whether there is a Wi-Fi connection.  
    private static boolean wifiConnected = false;  
    // Whether there is a mobile connection.  
    private static boolean mobileConnected = false;  
    // Whether the display should be refreshed.  
    public static boolean refreshDisplay = true;  
  
    // The user's current network preference setting.  
    public static String sPref = null;  
  
    // The BroadcastReceiver that tracks network connectivity chan  
    private NetworkReceiver receiver = new NetworkReceiver();  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Registers BroadcastReceiver to track network connection  
        IntentFilter filter = new IntentFilter(ConnectivityManager  
        receiver = new NetworkReceiver();  
        this.registerReceiver(receiver, filter);  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        // Unregisters BroadcastReceiver when app is destroyed.  
        if (receiver != null) {  
            this.unregisterReceiver(receiver);  
        }  
    }  
  
    // Refreshes the display if the network connection and the  
    // pref settings allow it.  
  
    @Override  
    public void onStart () {  
        super.onStart();  
  
        // Gets the user's network preference settings
```

```
SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);

// Retrieves a string value for the preferences. The second
// is the default value to use if a preference value is not
// found.
String sPref = sharedPrefs.getString("listPref", "Wi-Fi");

updateConnectedFlags();

if(refreshDisplay) {
    loadPage();
}

// Checks the network connection and sets the wifiConnected and
// mobileConnected variables accordingly.
public void updateConnectedFlags() {
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);

    NetworkInfo activeInfo = connMgr.getActiveNetworkInfo();
    if (activeInfo != null && activeInfo.isConnected()) {
        wifiConnected = activeInfo.getType() == ConnectivityManager.TYPE_WIFI;
        mobileConnected = activeInfo.getType() == ConnectivityManager.TYPE_MOBILE;
    } else {
        wifiConnected = false;
        mobileConnected = false;
    }
}

// Uses AsyncTask subclass to download the XML feed from stackoverflow.com
public void loadPage() {
    if (((sPref.equals(ANY)) && (wifiConnected || mobileConnected)) ||
        ((sPref.equals(WIFI)) && (wifiConnected))) {
        // AsyncTask subclass
        new DownloadXmlTask().execute(URL);
    } else {
        showErrorPage();
    }
}

...
}
```

## Detect Connection Changes(监测网络连接的改变)

最后一部分是关于 BroadcastReceiver 的子类： NetworkReceiver. 当设备网络连接改变时， NetworkReceiver会监听到 CONNECTIVITY\_ACTION, 这时需要判断当前网络连接类型并相应的设置好 wifiConnected 与 mobileConnected .

我们需要控制好BroadcastReceiver的使用，不必要的声明注册会浪费系统资源。通常是在onCreate()去registers这个BroadcastReceiver，在onPause()或者onDestroy()时unregisters它。这样做会比直接在manifest里面直接注册更轻量。当你在manifest里面注册了一个，你的程序可以在任何时候被唤醒，即使你已经好几个星期没有使用这个程序了。而通过前面的办法进行注册，可以确保用户离开你的程序之后，不会因为那个Broadcast而被唤起。如果你确保知道何时需要使用到它，你可以在合适的地方使用[setComponentEnabledSetting\(\)](#)来开启或者关闭它。

```
public class NetworkReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        ConnectivityManager conn = (ConnectivityManager)
            context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = conn.getActiveNetworkInfo();

        // Checks the user prefs and the network connection. Based on
        // to refresh the display or keep the current display.
        // If the userpref is Wi-Fi only, checks to see if the device
        if (WIFI.equals(sPref) && networkInfo != null && networkInfo.g
            // If device has its Wi-Fi connection, sets refreshDisplay
            // to true. This causes the display to be refreshed when t
            // returns to the app.
            refreshDisplay = true;
        Toast.makeText(context, R.string.wifi_connected, Toast.LEN

        // If the setting is ANY network and there is a network connec
        // (which by process of elimination would be mobile), sets ref
        } else if (ANY.equals(sPref) && networkInfo != null) {
            refreshDisplay = true;

            // Otherwise, the app can't download content--either because t
            // connection (mobile or Wi-Fi), or because the pref setting i
            // is no Wi-Fi connection.
            // Sets refreshDisplay to false.
        } else {
            refreshDisplay = false;
            Toast.makeText(context, R.string.lost_connection, Toast.LE
        }
    }
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/basics/network-ops/xml.html>

# 解析XML数据 Parsing XML Data

Extensible Markup Language (XML) .很多网站或博客上都提供XML feed来记录更新的信息，以便用户进行订阅阅读取。

那么上传[?]与解析XML数据就成了app的一个常见的功能。这一课会介绍如何解析XML文档并使用他们的数据。

([?]这里很奇怪，为什么是Upload，看文章最后一段代码示例的注释，应该是Download才对)

## Choose a Parser(选择一个解析器)

我们推荐[XmlPullParser](#), 它是在Android上一个高效且可维护的解析XML方法。 Android 上有这个接口的两种实现方式：

- [KXmlParser](#) via [XmlPullParserFactory.newPullParser\(\)](#).
- ExpatPullParser, via [Xml.newPullParser\(\)](#).

两个选择都是比较好的。下面的示例中是使用ExpatPullParser, via Xml.newPullParser().

## Analyze the Feed(分析Feed)

解析一个feed的第一步是决定需要获取哪些字段。这样解析器才去抽取出那些需要的字段而忽视剩下的。下面一段章节概览Sample app中截取的一段代码示例。

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="h
<title type="text">newest questions tagged android - Stack Overflow
...
<entry>
  ...
</entry>
<entry>
  <id>http://stackoverflow.com/q/9439999</id>
  <re:rank scheme="http://stackoverflow.com">0</re:rank>
  <title type="text">Where is my data file?</title>
  <category scheme="http://stackoverflow.com/feeds/tag?tagname">
  <category scheme="http://stackoverflow.com/feeds/tag?tagname">
    <author>
      <name>cliff2310</name>
      <uri>http://stackoverflow.com/users/1128925</uri>
    </author>
    <link rel="alternate" href="http://stackoverflow.com/question/9439999">
    <published>2012-02-25T00:30:54Z</published>
    <updated>2012-02-25T00:30:54Z</updated>
    <summary type="html">
      <p>I have an Application that requires a data file...</p>
    </summary>
  </entry>
  <entry>
  ...
</entry>
...
</feed>
```

在sample app中抽取了entry 标签与它的子标签 title, link,summary.

## Instantiate the Parser(实例化解析器)

下一步就是实例化一个parser并开始解析的操作。请看下面的示例：

```
public class StackOverflowXmlParser {  
    // We don't use namespaces  
    private static final String ns = null;  
  
    public List parse(InputStream in) throws XmlPullParserException  
        try {  
            XmlPullParser parser = Xml.newPullParser();  
            parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);  
            parser.setInput(in, null);  
            parser.nextTag();  
            return readFeed(parser);  
        } finally {  
            in.close();  
        }  
    }  
    ...  
}
```

## Read the Feed(读取Feed)

The `readFeed()` 实际上并没有处理feed的内容。它只是在寻找一个 "entry" 的标签作为递归 (recursively) 处理整个feed的起点。如果一个标签它不是"entry", `readFeed()`方法会跳过它. 当整个feed都被递归处理后, `readFeed()` 会返回一个包含了entry标签 (包括里面的数据成员) 的 List。

```
private List readFeed(XmlPullParser parser) throws XmlPullParserException {
    List entries = new ArrayList();

    parser.require(XmlPullParser.START_TAG, ns, "feed");
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        // Starts by looking for the entry tag
        if (name.equals("entry")) {
            entries.add(readEntry(parser));
        } else {
            skip(parser);
        }
    }
    return entries;
}
```

## Parse XML(解析XML)

解析步骤如下：

- 正如在上面“分析Feed”所说的，判断出你想要的tag。这个example抽取了 entry 标签与它的内部标签 title, link,summary.
- 创建下面的方法：
  - 为每一个你想要获取的标签创建一个 "read" 方法。例如 readEntry(), readTitle() 等等. 解析器从input stream中读取tag . 当读取到 entry, title, link 或者 summary 标签时，它会为那些标签调用相应的方法，否则，跳过这个标签。
  - 为每一个不同的标签的提取数据方法进行优化，例如：
    - 对于 title and summary tags, 解析器调用 readText(). 通过调用 parser.getText(). 来获取返回数据。
    - 对于 link tag, 解析器先判断这个link是否是我们想要的类型，然后再读取数据。可以使用 parser.getAttributeValue() 来获取返回数据。
    - 对于 entry tag, 解析器调用 readEntry(). 这个方法解析entry的内部标签并返回一个带有title, link, and summary数据成员的Entry对象。
  - 一个帮助方法： skip(). 关于这部分的讨论，请看下面一部分内容： Skip Tags You Don't Care About

下面的代码演示了如何解析 entries, titles, links, 与 summaries.

```
public static class Entry {  
    public final String title;  
    public final String link;  
    public final String summary;  
  
    private Entry(String title, String summary, String link) {  
        this.title = title;  
        this.summary = summary;  
        this.link = link;  
    }  
}  
  
// Parses the contents of an entry. If it encounters a title, summ  
// to their respective "read" methods for processing. Otherwise, s  
private Entry readEntry(XmlPullParser parser) throws XmlPullParser  
parser.require(XmlPullParser.START_TAG, ns, "entry");  
String title = null;  
String summary = null;  
String link = null;  
while (parser.next() != XmlPullParser.END_TAG) {  
    if (parser.getEventType() != XmlPullParser.START_TAG) {  
        continue;  
    }  
    String name = parser.getName();  
    if (name.equals("title")) {  
        title = readTitle(parser);  
    } else if (name.equals("summary")) {  
        summary = readSummary(parser);  
    } else if (name.equals("link")) {  
        link = readLink(parser);  
    }  
}
```

```

        } else {
            skip(parser);
        }
    }
    return new Entry(title, summary, link);
}

// Processes title tags in the feed.
private String readTitle(XmlPullParser parser) throws IOException,
parser.require(XmlPullParser.START_TAG, ns, "title");
String title = readText(parser);
parser.require(XmlPullParser.END_TAG, ns, "title");
return title;
}

// Processes link tags in the feed.
private String readLink(XmlPullParser parser) throws IOException,
String link = "";
parser.require(XmlPullParser.START_TAG, ns, "link");
String tag = parser.getName();
String relType = parser.getAttributeValue(null, "rel");
if (tag.equals("link")) {
    if (relType.equals("alternate")){
        link = parser.getAttributeValue(null, "href");
        parser.nextTag();
    }
}
parser.require(XmlPullParser.END_TAG, ns, "link");
return link;
}

// Processes summary tags in the feed.
private String readSummary(XmlPullParser parser) throws IOException
parser.require(XmlPullParser.START_TAG, ns, "summary");
String summary = readText(parser);
parser.require(XmlPullParser.END_TAG, ns, "summary");
return summary;
}

// For the tags title and summary, extracts their text values.
private String readText(XmlPullParser parser) throws IOException,
String result = "";
if (parser.next() == XmlPullParser.TEXT) {
    result = parser.getText();
    parser.nextTag();
}
return result;
}
...
}

```

## Skip Tags You Don't Care About(跳过你不在意标签)

下面演示解析器的 skip() 方法:

```
private void skip(XmlPullParser parser) throws XmlPullParserException {
    if (parser.getEventType() != XmlPullParser.START_TAG) {
        throw new IllegalStateException();
    }
    int depth = 1;
    while (depth != 0) {
        switch (parser.next()) {
            case XmlPullParser.END_TAG:
                depth--;
                break;
            case XmlPullParser.START_TAG:
                depth++;
                break;
        }
    }
}
```

上面这个方法是如何工作的呢?

- It throws an exception if the current event isn't a START\_TAG.
- It consumes the START\_TAG, and all events up to and including the matching END\_TAG.
- To make sure that it stops at the correct END\_TAG and not at the first tag it encounters after the original START\_TAG, it keeps track of the nesting depth.

因此如果目前的标签有子标签, depth 的值就不会为 0, 直到解析器已经处理了所有位于 START\_TAG 与 END\_TAG 之间的事件。例如, 看解析器如何跳过 标签, 它有2个子标签, 与 :

- The first time through the while loop, the next tag the parser encounters after is the START\_TAG for . The value for depth is incremented to 2.
- The second time through the while loop, the next tag the parser encounters is the END\_TAG . The value for depth is decremented to 1.
- The third time through the while loop, the next tag the parser encounters is the START\_TAG . The value for depth is incremented to 2.
- The fourth time through the while loop, the next tag the parser encounters is the END\_TAG . The value for depth is decremented to 1.
- The fifth time and final time through the while loop, the next tag the parser encounters is the END\_TAG . The value for depth is decremented to 0, indicating that the element has been successfully skipped.

## Consume XML Data(使用XML数据)

示例程序是在 AsyncTask 中获取与解析XML数据的。当获取到数据后，程序会在main activity(NetworkActivity)里面进行更新操作。

在下面示例代码中，loadPage() 方法做了下面的事情：

- 初始化一个带有URL地址的String变量，用来订阅XML feed。
- 如果用户设置与网络连接都允许，会触发 new DownloadXmlTask().execute(url). 这会初始化一个新的 DownloadXmlTask(AsyncTask subclass) 对象并且开始执行它的 execute() 方法。

```
public class NetworkActivity extends Activity {  
    public static final String WIFI = "Wi-Fi";  
    public static final String ANY = "Any";  
    private static final String URL = "http://stackoverflow.com/fe  
  
    // Whether there is a Wi-Fi connection.  
    private static boolean wifiConnected = false;  
    // Whether there is a mobile connection.  
    private static boolean mobileConnected = false;  
    // Whether the display should be refreshed.  
    public static boolean refreshDisplay = true;  
    public static String sPref = null;  
  
    ...  
  
    // Uses AsyncTask to download the XML feed from stackoverflow.  
    public void loadPage() {  
  
        if ((sPref.equals(ANY)) && (wifiConnected || mobileConnecte  
            new DownloadXmlTask().execute(URL);  
        }  
        else if ((sPref.equals(WIFI)) && (wifiConnected)) {  
            new DownloadXmlTask().execute(URL);  
        } else {  
            // show error  
        }  
    }  
}
```

下面是DownloadXmlTask是怎么工作的：

```
// Implementation of AsyncTask used to download XML feed from stac  
private class DownloadXmlTask extends AsyncTask<String, Void, Stri  
    @Override  
    protected String doInBackground(String... urls) {  
        try {  
            return loadXmlFromNetwork(urls[0]);  
        } catch (IOException e) {  
            return getResources().getString(R.string.connection_er  
        } catch (XmlPullParserException e) {  
            return getResources().getString(R.string.xml_error);  
    }
```

```

        }

    }

    @Override
    protected void onPostExecute(String result) {
        setContentView(R.layout.main);
        // Displays the HTML string in the UI via a WebView
        WebView myWebView = (WebView) findViewById(R.id.webview);
        myWebView.loadData(result, "text/html", null);
    }
}

```

下面是loadXmlFromNetwork是怎么工作的：

```

// Uploads XML from stackoverflow.com, parses it, and combines it
// HTML markup. Returns HTML string. 【这里可以看出应该是Download】
private String loadXmlFromNetwork(String urlString) throws XmlPullParserException, IOException {
    InputStream stream = null;
    // Instantiate the parser
    StackOverflowXmlParser stackOverflowXmlParser = new StackOverflowXmlParser();
    List<Entry> entries = null;
    String title = null;
    String url = null;
    String summary = null;
    Calendar rightNow = Calendar.getInstance();
    DateFormat formatter = new SimpleDateFormat("MMM dd h:mm:ss");

    // Checks whether the user set the preference to include summary
    SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences();
    boolean pref = sharedPrefs.getBoolean("summaryPref", false);

    StringBuilder htmlString = new StringBuilder();
    htmlString.append("<h3>" + getResources().getString(R.string.post_title));
    htmlString.append("<em>" + getResources().getString(R.string.post_url));
    htmlString.append(" " + formatter.format(rightNow.getTime()) + "</em>");

    try {
        stream = downloadUrl(urlString);
        entries = stackOverflowXmlParser.parse(stream);
        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (stream != null) {
            stream.close();
        }
    }

    // StackOverflowXmlParser returns a List (called "entries") of
    // Each Entry object represents a single post in the XML feed.
    // This section processes the entries list to combine each entry
    // Each entry is displayed in the UI as a link that optionally
    // has a text summary.
    for (Entry entry : entries) {

```

```
        htmlString.append("<p><a href='");
        htmlString.append(entry.link);
        htmlString.append(">" + entry.title + "</a></p>");
        // If the user set the preference to include summary text,
        // adds it to the display.
        if (pref) {
            htmlString.append(entry.summary);
        }
    }
    return htmlString.toString();
}

// Given a string representation of a URL, sets up a connection and
// an input stream.
【关于Timeout具体应该设置多少，可以借鉴这里的数据，当然前提是一般情况下】
// Given a string representation of a URL, sets up a connection and
// an input stream.
private InputStream downloadUrl(String urlString) throws IOException
{
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* milliseconds */);
    conn.setConnectTimeout(15000 /* milliseconds */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Starts the query
    conn.connect();
    return conn.getInputStream();
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/efficient-downloads/index.html>

# 高效下载

在这一章，我们将学习为了最小化某些操作对电量的影响是如何处理下载，网络连接，尤其是无线电连接的。

下面几节课会演示了如何使用缓存caching，轮询polling，预取prefetching等技术来计划与执行下载操作。我们还会学习无线电波的power-use属性配置是如何影响我们对于在何时，用什么，以何种方式来传输数据的选择。当然这些选择是为了最小化对电池寿命的影响。

You should also read [Optimizing Battery Life](#)

# Lessons

- [Optimizing Downloads for Efficient Network Access - 使用有效的网络连接方式来最优化下载](#)

This lesson introduces the wireless radio state machine, explains how your app's connectivity model interacts with it, and how you can minimize your data connection and use prefetching and bundling to minimize the battery drain associated with your data transfers.

- [Minimizing the Effect of Regular Updates - 优化常规更新操作的效果](#)

This lesson will examine how your refresh frequency can be varied to best mitigate the effect of background updates on the underlying wireless radio state machine.

- [Redundant Downloads are Redundant - 重复的下载是冗余的](#)

The most fundamental way to reduce your downloads is to download only what you need. This lesson introduces some best practices to eliminate redundant downloads.

- [Modifying your Download Patterns Based on the Connectivity Type - 根据网络连接类型来更改下载模式](#)

When it comes to impact on battery life, not all connection types are created equal. Not only does the Wi-Fi radio use significantly less battery than its wireless radio counterparts, but the radios used in different wireless radio technologies have different battery implications.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/efficient-downloads/efficient-network-access.html>

# **Optimizing Downloads for Efficient Network Access(用有效的网络访问来最优化下载)**

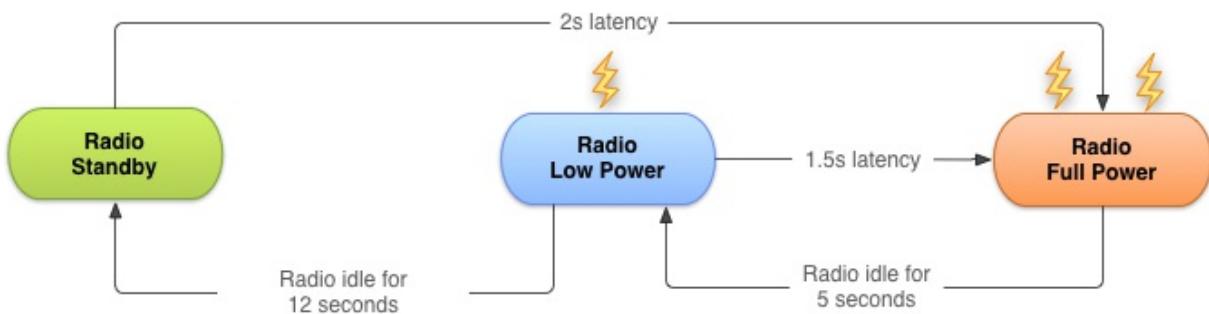
也许使用无线电波(wireless radio)进行传输数据会是我们app最耗电的操作之一。所以为了最小化网络连接的电量消耗，懂得连接模式(connectivity model)会如何影响底层的音频硬件设备是至关重要的。这节课介绍了无线电波状态机(wireless radio state machine)，并解释了app的connectivity model是如何与状态机进行交互的。然后会提出建议的方法来最小化我们的数据连接，使用预取(prefetching)与捆绑(bundle)的方式进行数据的传输，这些操作都是为了最小化电量的消耗。

# The Radio State Machine(无线电状态机)

一个完全活动的无线电会消耗很大部分的电量，因此需要学习如何在不同状态下进行过渡，这样能够避免电量的浪费。典型的3G无线电网络有三种能量状态：

- **Full power**: 当无线连接被激活的时候，允许设备以最大的传输速率进行操作。
- **Low power**: 相对Full power来说，算是一种中间状态，差不多50%的传输速率。
- **Standby**: 最低的状态，没有数据连接需要传输。

在最低并且空闲的状态下，电量消耗相对来说是少的。这里需要介绍一延时(latency)的机制，从low status返回到full status大概需要花费1.5秒，从idle status返回到full status需要花费2秒。为了最小化延迟，状态机使用了一种后滞过渡到更低能量状态的机制。下图是一个典型的3G无线电波状态机的图示(AT&T电信的一种制式)。



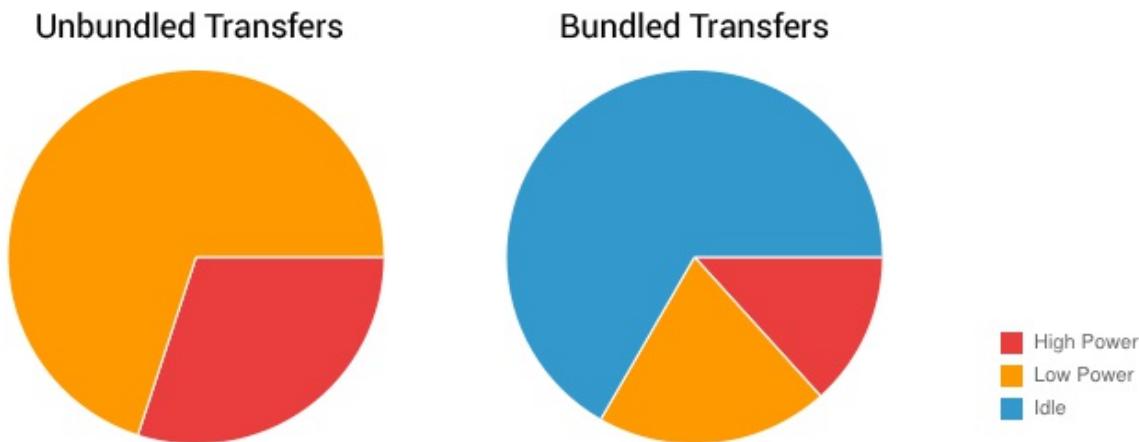
- 在每一台设备上的无线电状态机都会根据无线电波的制式(2G,3G,LTE等)而改变，并且由设备本身自己所使用的网络进行定义与配置。
- 这一课描述了一种典型的3G无线电波状态机，[data provided by AT&T](#)。这些原理是具有通用性的，在其他的无线电波上同样适用。
- 这种方法在浏览通常的网页操作上是特别有效的，因为它可以阻止一些不必要的浪费。而且相对较短的后期处理时间也保证了当一个session结束的时候，无线电波可以转移到相对较低的能量状态。
- 不幸的是，这个方法会导致在现代的智能机系统例如Android上的apps效率低下。因为Android上的apps不仅仅可以在前台运行，也可以在后台运行。(无线电波的状态改变会影响到本来的设计，有些想在前台运行的可能会因为切换到低能量状态而影响程序效率。坊间说手机在电量低的状态下无线电波的强度会增大好几倍来保证信号，可能与这个有关。)

## How Apps Impact the Radio State Machine[看apps如何影响无线状态机(使用bundle与unbundle传输数据的差异)]

每一次新创建一个网络连接，无线电波就切换到full power状态。在上面典型的3G无线电波状态机情况下，无线电波会在传输数据时保持在full power的状态，结束之后会有一个附加的5秒时间切换到low power,再之后会经过12秒进入到low energy的状态。因此对于典型的3G设备，每一次数据传输的会话都会引起无线电波都会持续消耗大概20秒的能量。

实际上，这意味着一个app传递1秒钟的unbundled data会使得无线电波持续活动18秒(18=1秒的传输数据+5秒过渡时间回到low power+12秒过渡时间回到standby)。因此每一分钟，它会消耗18秒high power的电量，42秒的low power的电量。

通过比较，如果每分钟app会传输bundle的数据持续3秒的话，其中会使得无线电波持续在high power状态仅仅8秒钟，在low power状态仅仅12秒钟。上面第二种传输bundle data的例子，可以看到减少了大量的电量消耗。图示如下：



## Prefetch Data(预取数据)

预取(Prefetching)数据是一种减少独立数据传输会话数量的有效方法。预取技术允许你在单次操作的时候，通过一次连接，在最大能力下，根据给出的时间下载到所有的数据。

通过前面的传输数据的技术，你减少了大量的无线电波激活时间。这样的话，不仅仅是保存了电量，也提高了潜在风险，降低了带宽，减少了下载时间。

预取技术提供了一种提高用户体验的方法，通过减少可能因为下载时间过长而导致预览后者后续操作等待漫长。

然而，使用预取技术过于频繁，不仅仅会导致电量消耗快速增长，还有可能预取到一些并不需要的数据。同样，确保app不会因为等待预取全部完成而卡到程序的开始播放也是非常重要的。从实践的角度，那意味着需要逐步处理数据，并且按照有优先级的顺序开始进行数据传递，这样能确保不卡到程序的开始播放的同时数据也能够得到持续的下载。

那么应该如何控制预取的操作呢？这需要根据正在下载的数据大小与可能被用到的数据量来决定。一个基于上面状态机情况的比较大概的建议是：对于数据来说，大概有50%的机会可能用在当前用户的会话中，那么我们可以预取大约6秒(大约1-2Mb)，这大概使得潜在可能要用的数据量与可能已经下载好的数据量相一致。

通常来说，预取1-5Mb会比较好，这种情况下，我们仅仅只需要每隔2-5分钟开始另一段下载。根据这个原理，大数据的下载，比如视频文件，应该每隔2-5秒开始另一段下载，这样能有效的预取到下面几分钟内的数据进行预览。

值得注意的是，下载需要是bundled的形式，而且上面那些大概的数据与时间可能会根据网络连接的类型与速度有所变化，这些都将在下面两部分内容讲到。

让我们来看一些例子：

**A music player** 你可以选择预取整个专辑，然而这样用户在第一首歌曲之后停止监听，那么就浪费了大量的带宽与电量。一个比较好的方法是维护一首歌曲的缓冲区。对于流媒体音乐，不应该去维护一段连续的数据流，因为这样会使得无线电波一直保持激活状态，应该考虑把HTTP的数据流集中一次传输到音频流，就像上面描述的预取技术一样(下载好2Mb，然后开始一次取出，再去下载下面的2Mb)。

**A news reader** 许多news apps尝试通过只下载新闻标题来减少带宽，完整的文章仅在用户想要读取的时候再去读取，而且文章也会因为太长而刚开始只显示部分信息，等用户下滑时再去读取完整信息。使用这个方法，无线电波仅仅会在用户点击更多信息的时候才会被激活。但是，在切换文章分类预阅读文章的时候仍然会造成大量潜在的消耗。

一个比较好的方法是在启动的时候预取一个合理数量的数据，比如在启动的时候预取一些文章的标题与缩略图信息。之后开始获取剩余的标题预缩略信息。

另一个方法是预取所有的标题，缩略信息，文章文字，甚至是所有文章的图片-根据既设的后台程序进行逐一获取。这样做的风险是花费了大量的带宽与电量去下载一些不会阅读到的内容，因此这需要比较小心思考是否合适。其中的一个解决方案是，当在连接至Wi-Fi时有计划的下载所有的内容，并且如果有可能最好是设备正在充电的时候。关于这个的细节的实现，我们将在后面的课程中涉及到。【这让我想起了网易新闻的离线下载，在连接到Wi-Fi的时候，可以选择下载所有的内容到本地，之后直接打开阅读】

## **Batch Transfers and Connections(批量传输与连接)**

使用典型3G无线网络制式的时候，每一次初始化一个连接(与需要传输的数据量无关)，你都有可能导致无线电波持续花费大约20秒的电量。

一个app，若是每20秒进行一次ping server的操作，假设这个app是正在运行且对用户可见，那么这会导致无线电波不确定什么时候被开启，最终可能使得电量花费在没有实际传输数据的情况下。

因此，对数据进行bundle操作并且创建一个序列来存放这些bundle好的数据就显得非常重要。操作正确的话，可以使得大量的数据集中进行发送，这样使得无线电波的激活时间尽可能的少，同时减少大部分电量的花费。这样做的潜在好处是尽可能在每次传输数据的会话中尽可能多的传输数据而且减少了会话的次数。

## Reduce Connections(减少连接次数)

重用已经存在的网络连接比起重新建立一个新的连接通常来说是更有效率的。重用网络连接同样可以使得在拥挤不堪的网络环境中进行更加智能的互动。当可以捆绑所有请求在一个GET里面的时候不要同时创建多个网络连接或者把多个GET请求进行串联。

例如，可以一起请求所有文章的情况下，不要根据多个栏目进行多次请求。无线电波会在等待接受返回信息或者timeout信息之前保持激活状态，所以如果不需要的连接请立即关闭而不是等待他们timeout。

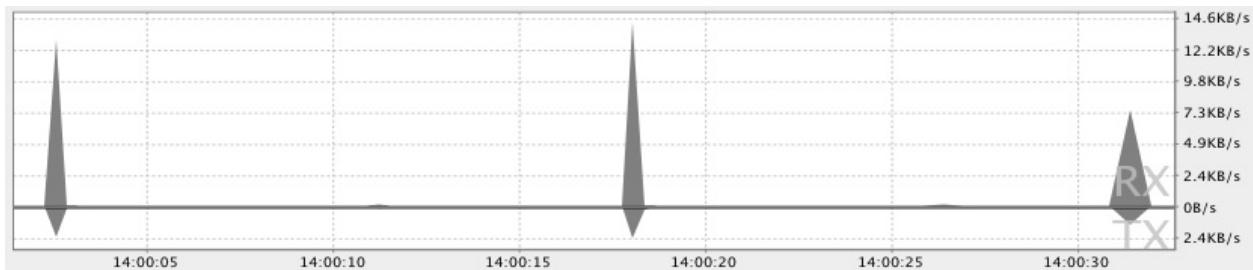
之前说道，如果关闭一个连接过于及时，会导致后面再次请求时重新建立一个在Server与Client之间的连接，而我们说过要尽量避免建立重复的连接，那么有个有效的折中办法是不要立即关闭，而是在timeout之前关闭(即稍微晚点却又不至于到timeout)。

**Note:** 使用HttpURLConnection，而不是Apache的HttpClient，前者有做response cache.

# Use the DDMS Network Traffic Tool to Identify Areas of Concern[使用DDMS网络通信工具来检测网络使用情况]

The Android [DDMS \(Dalvik Debug Monitor Server\)](#) 包含了一个查看网络使用详情的栏目来允许跟踪app的网络请求。使用这个工具，可以监测app是在何时，如何传输数据的，从而可以进行代码的优化。

下图显示了传输少量的网络模型，可以看到每次差不多相隔15秒，这意味着可以通过预取技术或者批量上传来大幅提高效率。



通过监测数据传输的频率与每次传输的数据量，可以查看出哪些位置应该进行优化，通常的，图中显示的短小的类似钉子形状的位置，可以进行与附近位置的请求进行做merge的动作。

为了更好的检测出问题所在，**Traffic Status API**允许你使用**TrafficStats.setThreadStatsTag()**的方法标记数据传输发生在某个Thread里面，然后可以手动的使用**tagSocket()**进行标记到或者使用**untagSocket()**来取消标记，例如：

```
TrafficStats.setThreadStatsTag(0xF00D);
TrafficStats.tagSocket(outputSocket);
// Transfer data using socket
TrafficStats.untagSocket(outputSocket);
```

Apache的HttpClient与URLConnection库可以自动tag sockets使用当前getThreadStatusTag()的值。那些库在通过keep-alive pools循环的时候也会tag与untag sockets。

```
TrafficStats.setThreadStatsTag(0xF00D);
try {
    // Make network request using HttpClient.execute()
} finally {
    TrafficStats.clearThreadStatsTag();
}
```

Socket tagging 是在Android 4.0上才被支持的，但是实际情况是仅仅会在运行Android 4.0.3 or higher的设备上才会显示。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/efficient-downloads/regular-update.html>

# Minimizing the Effect of Regular Updates(最小化定期更新操作的副作用)

最佳的定时更新频率是不确定的，通常由设备状态，网络连接状态，用户行为与用户定义明确的偏好而决定。

[Optimizing Battery Life](#)这一章有讨论如何根据设备状态来修改更新频率。里面介绍了当断开网络连接的时候去关闭后台服务，在电量比较低的时候减少更新的频率。

这一课会介绍更新频率是多少才会使得更新操作对无线电状态机的影响最小。(C2DM与指数退避算法的使用)

## **Use Google Cloud Messaging as an Alternative to Polling[使用C2DM作为轮询方式之一]**

关于Android Cloud to Device Messaging (C2DM)详情 ,请参考:<http://code.google.com/intl/zh-CN/android/c2dm/>

每次app去向server询问检查是否有更新操作的时候会激活无线电，这样造成了不必要的能量消耗(在3G情况下，会差不多消耗20秒的能量)。

C2DM是一个用来从server到特定app传输数据的轻量级的机制。使用C2DM,server会在某个app有需要获取新数据的时候通知app有这个消息。

比起轮询方式(app为了即时拿到最新的数据需要定时向server请求数据)，C2DM这种有事件驱动的模式会在仅有有数据更新的时候通知app去创建网络连接来获取数据(很显然这样减少了app的大量操作，当然也减少了很多电量)。

C2DM需要通过使用固定TCP/IP来实现操作。当在你的设备上可以实现固定IP的时候，最好使用C2DM。(这个地方应该不是传统意义上的固定IP，可以理解为某个会话情况下)。很明显，使用C2DM既减少了网络连接次数，也优化了带宽，还减少了对电量的消耗。

**Ps:**大陆的Google框架通常被移除掉，这导致C2DM实际上根本没有办法在大陆的App上使用

# Optimize Polling with Inexact Repeating Alarms and Exponential Backoffs(通过不定时的重复提醒与指数退避来优化轮询操作)

如果需要使用轮询机制，在不影响用户体验的前提下，当然设置默认更新频率是越低越好(减少电量的浪费)。

一个简单的方法是给用户提供更新频率的选择，允许用户自己来处理如何平衡数据及时性与电量的消耗。

当设置安排好更新操作后，可以使用不确定重复提醒的方式来允许系统把当前这个操作进行定向移动(比如推迟一会)。

```
int alarmType = AlarmManager.ELAPSED_REALTIME;
long interval = AlarmManager.INTERVAL_HOUR;
long start = System.currentTimeMillis() + interval;

alarmManager.setInexactRepeating(alarmType, start, interval, pi);
```

若是多个提醒都安排在某个点同时被触发，那么这样就可以使得多个操作在同一个无线电状态下操作完。

如果可以，请设置提醒的类型为ELAPSED\_REALTIME or RTC而不是\_WAKEUP。这样能够更进一步的减少电量的消耗。

我们还可以通过根据app被使用的频率来有选择性的减少更新的频率。

另一个方法在app在上一次更新操作之后还未被使用的情况下，使用指数退避算法exponential back-off algorithm来减少更新频率。当然我们也可以使用一些类似指数退避的方法。

```
SharedPreferences sp =
    context.getSharedPreferences(PREFS, Context.MODE_WORLD_READABLE)

boolean appUsed = sp.getBoolean(PREFS_APPUSED, false);
long updateInterval = sp.getLong(PREFS_INTERVAL, DEFAULT_REFRESH_I

if (!appUsed)
    if ((updateInterval *= 2) > MAX_REFRESH_INTERVAL)
        updateInterval = MAX_REFRESH_INTERVAL;

Editor spEdit = sp.edit();
spEdit.putBoolean(PREFS_APPUSED, false);
spEdit.putLong(PREFS_INTERVAL, updateInterval);
spEdit.apply();

rescheduleUpdates(updateInterval);
executeUpdateOrPrefetch();
```

初始化一个网络连接的花费不会因为是否成功下载了数据而改变。我们可以使用指数退避算法来减少重复尝试(retry)的次数，这样能够避免浪费电量。例如：

```
private void retryIn(long interval) {  
    boolean success = attemptTransfer();  
  
    if (!success) {  
        retryIn(interval*2 < MAX_RETRY_INTERVAL ?  
                interval*2 : MAX_RETRY_INTERVAL);  
    }  
}
```

---

笔者结语:这一课讲到C2DM与指数退避算法等,其实这些细节很值得我们注意,如果能在实际项目中加以应用,很明显程序的质量上升了一个档次!

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/efficient-downloads/redundant-redundant.html>

# **Redundant Downloads are Redundant(重复的 下载是冗余的)**

减少下载的最基本方法是仅仅下载那些你需要的。从数据的角度看，我们可以通过传递类似上次更新时间这样的参数来制定查询数据的条件。同样，在下载图片的时候，server那边最好能够减少图片的大小，而不是让我们下载完整大小的图片。

## 1) Cache Files Locally(缓存文件到本地)

避免下载重复的数据是很重要的。可以使用缓存机制来处理这个问题。缓存static的资源，例如完整的图片。这些缓存的资源需要分开存放。为了保证app不会因为缓存而导致显示的是旧数据，请从缓存中获取最新的数据，当数据过期的时候，会提示进行刷新。

```
long currentTime = System.currentTimeMillis();

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

long expires = conn.getHeaderFieldDate("Expires", currentTime);
long lastModified = conn.getHeaderFieldDate("Last-Modified", current

setDataExpirationDate(expires);

if (lastModified < lastUpdateTime) {
    // Skip update
} else {
    // Parse update
}
```

使用这种方法，可以有效保证缓存里面一直是最新的数据。

可以使用下面的方法来获取External缓存的目录：(目录会是sdcard下面的Android/data/data/com.xxx.xxx/cache)

```
Context.getExternalCacheDir();
```

下面是获取内部缓存的方法，请注意，存放在内存中的数据有可能因内部空间不够而被清除。(类似:system/data/data/com.xxx.xxx./cache)

```
Context.getCache();
```

上面两个Cache的文件都会在app卸载的时候被清除。

**Ps:**请注意这点:发现很多应用总是随便在sdcard下面创建一个目录用来存放缓存，可是这些缓存又不会随着程序的卸载而被删除，这其实是很令人讨厌的，程序都被卸载了，为何还要留那么多垃圾文件，而且这些文件有可能会泄漏一些隐私信息。除非你的程序是音乐下载，拍照程序等等，这些确定程序生成的文件是会被用户需要留下的，不然都应该使用上面的那种方式来获取Cache目录

## 2) Use the HttpURLConnection Response Cache(使用HttpURLConnection Response缓存)

在Android 4.0里面为HttpURLConnection增加了一个response cache(这是一个很好的减少http请求次数的机制，Android官方推荐使用HttpURLConnection而不是Apache的DefaultHttpClient，就是因为前者不仅仅有针对android做http请求的优化，还在4.0上增加了Reponse Cache，这进一步提高了效率)

我们可以使用反射机制开启HTTP response cache，看下面的例子：

```
private void enableHttpResponseBodyCache() {  
    try {  
        long httpCacheSize = 10 * 1024 * 1024; // 10 MiB  
        File httpCacheDir = new File(getCacheDir(), "http");  
        Class.forName("android.net.http.HttpResponseCache")  
            .getMethod("install", File.class, long.class)  
            .invoke(null, httpCacheDir, httpCacheSize);  
    } catch (Exception httpResponseCacheNotAvailable) {  
        Log.d(TAG, "HTTP response cache is unavailable.");  
    }  
}
```

上面的sample code会在Android 4.0以上的设备上开启response cache，同时不会影响到之前的程序。在cache被开启之后，所有cache中的HTTP请求都可以直接在本地存储中进行响应，并不需要开启一个新的网络连接。被cache起来的response可以被server所确保没有过期，这样就减少了带宽。没有被cached的response会因方便下次请求而被存储在response cache中。

---

Ps:Cache机制在很多实际项目上都有使用到，实际操作会复杂许多，有机会希望能够分享一个Cache的例子。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/efficient-downloads/connectivity-patterns.html>

# 根据网络类型改变下载模式Modifying your Download Patterns Based on the Connectivity Type

并不是所有的网络类型(Wi-Fi,3G,2G,etc)对电量的消耗是同等的。不仅仅Wi-Fi电波比无线电波消耗的电量要少很多，而且不同的无线电波(3G,2G,LTE.....)也存在使用不同电量的区别。

## 1) Use Wi-Fi[使用Wi-Fi]

在大多数情况下，Wi-Fi电波会在使用相对较低的电量的情况下提供一个相对较大的带宽。因此，我们需要努力争取尽量使用Wi-Fi来传递数据。我们可以使用Broadcast Receiver来监听当网络连接切换为Wi-Fi，这个时候我们可以进行大量的数据传递操作，例如下载，执行定时的更新操作，甚至是在这个时候加大更新的频率。这些内容都可以在前面的课程中找到。

## 2) Use Greater Bandwidth to Download More Data Less Often[使用更大的带宽来下载更多的数据，而不是经常去下载]

当通过无线电进行连接的时候，更大的带宽通常伴随着更多的电量消耗。这意味着LTE(一种4G网络制式)会比3G制式消耗更多，当然比起2G更甚。

从Lesson 1我们知道了无线电状态机是怎么回事，通常来说相对更宽的带宽网络制式会有更长的状态切换时间(也就是从full power过渡到standby有更长一段时间的延迟)。同时，更高的带宽意味着可以更贪婪的进行prefetch，下载更多的数据。也许这个说法不是很直观，因为过渡的时间比较长，而过渡时间的长短我们无法控制，也就是过渡时间的电量消耗差不多是固定了，既然这样，我们在每次传输会话中为了减少更新的频率而把无线电激活的时间拉长，这样显的更有效率。也就是尽量一次性把事情做完，而不是断断续续的请求。

例如：如果LTE无线电的带宽与电量消耗都是3G无线电的2倍，我们应该在每次会话的时候都下载4倍于3G的数据量，或者是差不多10Mb(前面文章有说明3G一般每次下载2Mb)。当然，下载到这么多数据的时候，我们需要好好考虑prefetch本地存储的效率并且需要经常刷新预取的cache。我们可以使用connectivity manager来判断当前激活的无线电波，并且根据不同结果来修改prefetch操作。

```
ConnectivityManager cm =  
    (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);  
  
TelephonyManager tm =  
    (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);  
  
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();  
  
int PrefetchCacheSize = DEFAULT_PREFETCH_CACHE;  
  
switch (activeNetwork.getType()) {  
    case ConnectivityManager.TYPE_WIFI:  
        PrefetchCacheSize = MAX_PREFETCH_CACHE; break;  
    case ConnectivityManager.TYPE_MOBILE: {  
        switch (tm.getNetworkType()) {  
            case TelephonyManager.NETWORK_TYPE_LTE |  
                TelephonyManager.NETWORK_TYPE_HSPAP):  
                PrefetchCacheSize *= 4;  
            break;  
            case (TelephonyManager.NETWORK_TYPE_EDGE |  
                TelephonyManager.NETWORK_TYPE_GPRS):  
                PrefetchCacheSize /= 2;  
            break;  
            default: break;  
        }  
        break;  
    }  
    default: break;  
}
```

Ps：想要最大化效率与最小化电量的消耗，需要考虑的东西太多了，通常来说，会根据app的功能需求来选择有所侧重，那么前提就是需要了解到底哪些对效率的影响比较大，这有利于我们做出最优选择。

编写:[jdneo](#)

校对:

# 使用Sync Adapter传输数据

在一台Android设备和网络服务器之间同步数据，可以让你的应用更加实用，更加吸引用户的注意。例如，将数据传输到服务器可以实现一个有用的备份，另一方面，将数据从服务器中获取可以让用户随时随地都能使用你的应用。在一些情况中，用户可能会发觉在线编辑他们的数据并将其发送到设备上，会是一件很方便的事情；或者他们有时会希望将收集的数据上传到一个统一的存储区域中。

尽管你可以设计你自己的系统来实现你应用中的数据传输，但你可以考虑一下使用Android的Sync Adapter框架（Android's sync adapter framework）。这个框架可以帮助管理并自动传输数据，并且协调不同应用间的同步问题。当你使用这个框架时，你可以利用它的一些特性，而这些特性可能是你自己设计的传输方案中所没有的：

## 插件架构（Plug-in architecture）：

允许你以可调用组件的形式，将传输代码添加到系统中。

## 自动执行（Automated execution）：

允许你可以给予不同的准则自动地执行数据传输，包括数据变更，经过的时间，传输时间等。另外，系统会把当前无法执行的传输添加到一个队列中，并且在合适的时候运行它们。

## 自动网络监测（Automated network checking）：

系统只在有网络连接的时候才会运行数据传输。

## 提升电池使用效率：

允许你将所有的数据传输任务统一地进行一次性批量传输，这样的话数据传输任务会在同一时间运行。你的应用的数据传输也会和其它应用的传输任务相结合，并一起传输。这样做可以减少系统连接网络的次数，进而减少电量的使用。

## 账户管理和授权：

如果你的应用需要用户登录，那么你可以将账户管理和授权的功能集成到你的数据传输中。

本系列课程将向你展示如何创建一个Sync Adapter，以及它所封装和绑定的服务（[Service](#)），如何提供其它组件来帮助你将Sync Adapter添加到框架中，以及如何通过不同的方法来运行Sync Adapter。

**Note:** Sync Adapter是异步执行的，所以你应该在期望它可以定期地有效地而不是瞬间地传输数据时使用它。如果你想要实时地传输数据，那么你应该在 [AsyncTask](#)或[IntentService](#)完成这一任务。

## **Sample Code**

[BasicSyncAdapter.zip](#)

# Lessons

- [创建Stub授权器](#)

学习如何添加一个账户处理组件。Sync Adapter框架要求应用中需要具备这样的一个组件。这节课将向你展示如何简单的创建一个Stub授权器组件。

- [创建Stub Content Provider](#)

学习如何添加一个Content Provider组件Sync Adapter框架要求应用中需要具备这样的一个组件。这节课中我们假设你的应用实际上不需要使用Content Provider，所以它将教你如何添加一个Stub Provider。如果你在应用中已经有了一个Content Provider组件，那么可以跳过这节课。

- [创建Sync Adapter](#)

学习如何将你的数据传输代码封装到组件当中，以此使得Sync Adapter框架可以自动执行。

- [执行Sync Adapter](#)

学习如何使用Sync Adapter框架激活并调度数据传输。

编写:[jdneo](#)

校对:

# 创建Stub授权器

Sync Adapter框架假定你的Sync Adapter在同步数据时，设备存储会有一个账户，服务器存储端会有登录验证。因此，框架期望你提供一个叫做授权器的组件作为你的Sync Adapter的一部分。该组件会植入Android账户及认证框架，并提供一个标准的接口来处理用户凭据，比如登录信息。

甚至，如果你的应用不使用账户，你仍然需要提供一个授权器组件。如果你不使用账户或者服务器登录，授权器所处理的信息将被忽略，所以你可以提供一个授权器组件，它包括了一个“空”的实现（译者注：也即标题中的Stub）。同时你需要提供一个捆绑的[Service](#)，来允许Sync Adapter框架来调用授权器的方法。

这节课将向你展示如何定义一个Stub授权器的所有满足其实现要求的部件。如果你想要提供一个真实的处理用户账户的授权器，可以阅读：[AbstractAccountAuthenticator](#)。

## 添加一个Stub授权器组件

要在你的应用中添加一个Stub授权器，创建一个继承[AbstractAccountAuthenticator](#)的类，并将要覆写的方法置空（这样就不会做任何处理了），返回null或者抛出异常。

下面的代码片段是一个Stub授权器的例子：

```
/*
 * Implement AbstractAccountAuthenticator and stub out all
 * of its methods
 */
public class Authenticator extends AbstractAccountAuthenticator {
    // Simple constructor
    public Authenticator(Context context) {
        super(context);
    }
    // Editing properties is not supported
    @Override
    public Bundle editProperties(
            AccountAuthenticatorResponse r, String s) {
        throw new UnsupportedOperationException();
    }
    // Don't add additional accounts
    @Override
    public Bundle addAccount(
            AccountAuthenticatorResponse r,
            String s,
            String s2,
            String[] strings,
            Bundle bundle) throws NetworkErrorException {
        return null;
    }
    // Ignore attempts to confirm credentials
    @Override
    public Bundle confirmCredentials(
            AccountAuthenticatorResponse r,
            Account account,
            Bundle bundle) throws NetworkErrorException {
        return null;
    }
    // Getting an authentication token is not supported
    @Override
    public Bundle getAuthToken(
            AccountAuthenticatorResponse r,
            Account account,
            String s,
            Bundle bundle) throws NetworkErrorException {
        throw new UnsupportedOperationException();
    }
    // Getting a label for the auth token is not supported
    @Override
    public String getAuthTokenLabel(String s) {
        throw new UnsupportedOperationException();
    }
}
```

```
// Updating user credentials is not supported
@Override
public Bundle updateCredentials(
    AccountAuthenticatorResponse r,
    Account account,
    String s, Bundle bundle) throws NetworkErrorException
throw new UnsupportedOperationException();

// Checking features for the account is not supported
@Override
public Bundle hasFeatures(
    AccountAuthenticatorResponse r,
    Account account, String[] strings) throws NetworkErrorException
throw new UnsupportedOperationException();
}

}
```

## 将授权器绑定到框架

为了让Sync Adapter框架可以访问你的授权器，你必须为它创建一个捆绑服务。这一服务提供一个Android binder对象，允许框架调用你的授权器，并且在授权器和框架间传输数据。

因为框架会在它第一次访问授权器时启动[Service](#)，你也可以使用服务来实例化授权器，方法是通过在服务的[Service.onCreate\(\)](#)方法中调用授权器的构造函数。

下面的代码样例展示了如何定义绑定[Service](#):

```
/**  
 * A bound Service that instantiates the authenticator  
 * when started.  
 */  
public class AuthenticatorService extends Service {  
    ...  
    // Instance field that stores the authenticator object  
    private Authenticator mAuthenticator;  
    @Override  
    public void onCreate() {  
        // Create a new authenticator object  
        mAuthenticator = new Authenticator(this);  
    }  
    /*  
     * When the system binds to this Service to make the RPC call  
     * return the authenticator's IBinder.  
     */  
    @Override  
    public IBinder onBind(Intent intent) {  
        return mAuthenticator.getIBinder();  
    }  
}
```

# 添加授权器的元数据文件

要将你的授权器组件插入到Sync Adapter和账户框架中，你需要为框架提供带有描述组件的元数据。该元数据声明了你创建的Sync Adapter的账户类型以及系统所显示的用户接口元素（如果你希望将你的账户类型对用户可见）。在你的项目目录：“/res/xml/”下，将元数据声明于一个XML文件中。你可以随便为它起一个名字，一般来说，可以叫“authenticator.xml”

在这个XML文件中，包含单个元素<account-authenticator>，它有下列一些属性：

## **android:accountType**

Sync Adapter框架需要每一个适配器以域名的形式拥有一个账户类型。框架会将它作为其内部的标识。对于需要登录的服务器，账户类型会和账户一起发送到服务端作为登录凭据的一部分。

如果你的服务不需要登录，你仍然需要提供一个账户类型。值的话就用你能控制的一个域名即可。由于框架会使用它来管理Sync Adapter，所以值不会发送到服务器上。

## **android:icon**

指向一个包含一个图标的[Drawable](#)资源的指针。如果你在“res/xml/syncadapter.xml”中通过指定`android:userVisible="true"`让Sync Adapter可见，那么你必须提供图标资源。它会在系统的设置中的账户这一栏内显示。

## **android:smallIcon**

指向一个包含一个微小版本图标的[Drawable](#)资源的指针。结合具体的屏幕大小，这一资源可能会替代“`android:icon`”中所指定的图标资源。

## **android:label**

将指明了用户账户类型的string本地化。如果你在“res/xml/syncadapter.xml”中通过指定`android:userVisible="true"`让Sync Adapter可见，那么你需要提供这个string。它会在系统的设置中的账户这一栏内显示，就在你定义的图标旁边。

下面的代码样例展示了你之前为授权器创建的XML文件：

```
<?xml version="1.0" encoding="utf-8"?>
<account-authenticator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="example.com"
    android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher"
    android:label="@string/app_name"/>
```

## 在清单文件中声明授权器

在之前的步骤中，你创建了一个捆绑服务，将授权器和Sync Adapter框架连接起来。要标识这个服务，你需要再清单文件中添加`<service>`标签，将它作为`<application>`的子标签：

```
<service
    android:name="com.example.android.syncadapter.Authenti
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenti
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/authenticator" />
</service>
```

标签`<intent-filter>`配置了一个由`android.accounts.AccountAuthenticator`的intent所激活的过滤器，这一intent会在系统要运行授权器时由系统发出。当过滤器被激活，系统会启动`AuthenticatorService`，它是你之前用来封装授权器的捆绑[Service](#)。

`<meta-data>`标签声明了授权器的元数据。`android:name`属性将元数据和授权器框架连接起来。`android:resource`指定了你之前所创建的授权器元数据文件的名字。

除了一个授权器，一个Sync Adapter框架需要一个内容提供器（content provider）。如果你的应用不适用内容提供器，可以阅读下一节课程，在下节课中将会创建一个空的内容提供器；如果你的应用适用的话，可以直接阅读：[Creating a Sync Adapter](#)。

编写:[jdneo](#)

校对:

# 创建Stub Content Provider

Sync Adapter框架是设计成用来和设备数据一起工作的，这些设备的数据被灵活且高安全的Content Provider所管理。因此，Sync Adapter框架会期望应用所使用的框架已经为它的本地数据定义了Content Provider。如果Sync Adapter框架尝试去运行你的Sync Adapter，而你的应用没有一个Content Provider的话，那么你的Sync Adapter将会崩溃。

如果你正在开发一个新的应用，它将数据从服务器传输到一台设备上，那么你务必应该考虑将本地数据存储于Content Provider中。因为它对于Sync Adapter来说是很重要的，另外Content Provider可以给予许多安全上的好处，并且是专门被设计成在Android设备上处理数据存储的。要学习如何创建一个Content Provider，可以阅读：[Creating a Content Provider](#)。

然而，如果你已经通过别的形式来存储本地数据，你仍然可以使用Sync Adapter来处理数据传输。为了满足Sync Adapter框架对于Content Provider的要求，可以在你的应用中添加一个空的Content Provider（Stub Content Provider）。一个Stub Content Provider实现了Content Provider类，但是所有的方法都返回null或者0。如果你添加了一个空提供器，你可以使用Sync Adapter从任何你选择的存储机制来传输数据。

如果你在你的应用中已经有了一个Content Provider，那么你就不需要一个Stub Content Provider了。在这种情况下，你可以略过这节课，直接进入：[创建Sync Adapter](#)。如果你还没有一个Content Provider，这节课将向你展示如何添加一个Stub Content Provider，来允许你将你的Sync Adapter添加到框架中。

## 添加一个Stub Content Provider

要为你的应用创建一个Stub Content Provider，继承[ContentProvider](#)并且置空它需要的方法。下面的代码片段展示了你应该如何创建Stub Content Provider：

```
/*
 * Define an implementation of ContentProvider that stubs out
 * all methods
 */
public class StubProvider extends ContentProvider {
    /*
     * Always return true, indicating that the
     * provider loaded correctly.
     */
    @Override
    public boolean onCreate() {
        return true;
    }
    /*
     * Return an empty String for MIME type
     */
    @Override
    public String getType() {
        return new String();
    }
    /*
     * query() always returns no results
     */
    @Override
    public Cursor query(
            Uri uri,
            String[] projection,
            String selection,
            String[] selectionArgs,
            String sortOrder) {
        return null;
    }
    /*
     * insert() always returns null (no URI)
     */
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }
    /*
     * delete() always returns "no rows affected" (0)
     */
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
    /*

```

```
* update() always returns "no rows affected" (0)
*/
public int update(
    Uri uri,
    ContentValues values,
    String selection,
    String[] selectionArgs) {
    return 0;
}
```

## 在清单文件中声明提供器

Sync Adapter框架会检查你的应用在清单文件中是否声明了一个Provider来验证你的应用是否有Content Provider。为了在清单文件中声明Stub Content Provider，添加一个<provider>标签，并让它拥有下列属性字段：

**android:name="com.example.android.datasync.provider.StubProvider"**

指定一个实现了Stub Content Provider的类的完整包名。

**android:authorities="com.example.android.datasync.provider"**

一个URI Authority来指定Stub Content Provider。让它的值是你的应用包名加上字符串“.provider”。虽然你在这向系统声明了你的空提供器，但是这并不会导致对提供器的访问。

**android:exported="false"**

确定其它应用是否可以访问Content Provider。对于你的Stub Content Provider，由于没有让其它应用访问提供器的必要，将值设置为“false”。该值并不会影响Sync Adapter框架和Content Provider之间的交互。

**android:syncable="true"**

设置一个指明该提供器是可同步的标识。如果将这个值设置为“true”，你不需要在你的代码中调用[setIsSyncable\(\)](#)。这一标识将会允许Sync Adapter框架和Content Provider进行数据传输，但是仅仅在你显式地执行这一传输时才会进行。

下面的代码片段展示了你应该如何将<provider>添加到应用清单文件中：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.network.sync.BasicSyncAdapter"
    android:versionCode="1"
    android:versionName="1.0" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        ...
        <provider
            android:name="com.example.android.datasync.provider.StubProvider"
            android:authorities="com.example.android.datasync.provider"
            android:export="false"
            android:syncable="true"/>
        ...
    </application>
</manifest>
```

现在你已经创建了Sync Adapter框架所需要的依赖关系，你可以创建封装你的数据传输代码的组件了。该组件就叫做Sync Adapter。下节课将会展示如何将它添加到你的应用中。

编写:[jdneo](#)

校对:

# 创建Sync Adapter

在你应用中的Sync Adapter组件会封装在设备和服务器之间传输数据的任务代码。基于你提供的调度和触发器，Sync Adapter框架会在Sync Adapter组件中运行你的代码。要将同步适配组件添加到你的应用，你需要添加下列部件：

## Sync Adapter类

这将你的数据传输代码封装到一个接口中，该接口与Sync Adapter框架兼容。

## 捆绑Service

一个组件，它可以允许Sync Adapter框架在你的Sync Adapter类中运行代码。

## Sync Adapter的XML元数据文件

一个文件，包含了你的Sync Adapter信息。框架会读取该文件并确定应该如何加载并调度你的数据传输任务。

## 应用清单文件的声明

在XML文件中声明的捆绑服务，并指出Sync Adapter的元数据。

这节课将会向你展示如何定义这些元素。

# 创建一个Sync Adapter类

在这部分课程中，你将会学习如何创建Sync Adapter类，该类封装了数据传输的代码。创建该类并继承Sync Adapter的基类，为该类定义构造函数，并实现你定义的数据传输任务的方法。

## 继承Sync Adapter基类：AbstractThreadedSyncAdapter

要创建Sync Adapter组件，首先继承AbstractThreadedSyncAdapter，然后编写它的构造函数。每次你的Sync Adapter组件创建的时候，构造函数就会执行配置任务，和你使用[Activity.onCreate\(\)](#)配置Activity是一样的。例如，如果你的应用使用一个Content Provider来存储数据，那么使用构造函数来获取一个[ContentResolver](#)实例。由于从Android 3.0开始添加了第二种形式的构造函数，来支持parallelSyncs参数，所以你需要创建两种形式的构造函数来保证兼容性。

**Note:** Sync Adapter框架是设计成和Sync Adapter组件的单例一起工作的。实例化Sync Adapter组件的更多细节，可以阅读：[Bind the Sync Adapter to the Framework](#)。

下面的代码展示了如何实现AbstractThreadedSyncAdapter和它的构造函数：

```
/*
 * Handle the transfer of data between a server and an
 * app, using the Android sync adapter framework.
 */
public class SyncAdapter extends AbstractThreadedSyncAdapter {
    ...
    // Global variables
    // Define a variable to contain a content resolver instance
    ContentResolver mContentResolver;
    /**
     * Set up the sync adapter
     */
    public SyncAdapter(Context context, boolean autoInitialize) {
        super(context, autoInitialize);
        /*
         * If your app uses a content resolver, get an instance of
         * from the incoming Context
         */
        mContentResolver = context.getContentResolver();
    }
    ...
}
/**
 * Set up the sync adapter. This form of the
 * constructor maintains compatibility with Android 3.0
 * and later platform versions
 */
public SyncAdapter(
    Context context,
    boolean autoInitialize,
    boolean allowParallelSyncs) {
```

```
super(context, autoInitialize, allowParallelSyncs);
/*
 * If your app uses a content resolver, get an instance of
 * from the incoming Context
 */
mContentResolver = context.getContentResolver();
...
}
```

## 在onPerformSync()中添加数据传输代码

Sync Adapter组件并不会自动地执行数据传输。相反地，它只是对你的数据传输代码进行封装，所以Sync Adapter框架可以在后台执行数据传输，而不会牵连到你的应用。当框架准备同步你的应用数据时，它会调用你的[onPerformSync\(\)](#)方法的实现。

为了便于将你的数据从你的应用程序代码转移到Sync Adapter组件，Sync Adapter框架调用[onPerformSync\(\)](#)，它具有下面的参数：

### 账户 (Account)

该[Account](#)对象是和激活Sync Adapter的事件相关联的。如果你的服务不需要使用账户，你不需要使用这个对象内的信息。

### 额外数据 (Extras)

一个Bundle对象，它包含了激活Sync Adapter的事件所发送的标识。

### 权限 (Authority)

系统Content Provider的Authority。你的应用必须要有访问它的权限。通常，Authority对应于你应用的Content Provider。

### Content Provider客户端 (Content provider client)

一个Content Provider的[ContentProviderClient](#)对象是由Authority参数所指定的。一个[ContentProviderClient](#)是一个Content Provider的轻量级共有接口。它的基本功能和一个[ContentResolver](#)一样。如果你正在使用一个Content Provider来存储你的应用的数据，你可以用该对象和提供器连接。否则的话你可以忽略它。

### 同步结果 (Sync result)

一个[SyncResult](#)对象，你可以使用它来将信息发送到Sync Adapter框架。

下面的代码片段展示了[onPerformSync\(\)](#)函数的整体架构：

```
/*
 * Specify the code you want to run in the sync adapter. The e
 * sync adapter runs in a background thread, so you don't have
 * up your own background processing.
 */
@Override
public void onPerformSync(
    Account account,
```

```
        Bundle extras,
        String authority,
        ContentProviderClient provider,
        SyncResult syncResult) {
    /*
     * Put the data transfer code here.
     */
    ...
}
```

但是实际的[onPerformSync\(\)](#)实现是要根据你的应用数据的同步需求以及服务器的连接协议来制定的，有一些你应该要实现的基本任务，如下所示：

### 连接到一个服务器

尽管你可以假定当你开始传输数据时，可以获取到网络连接，但是Sync Adapter框架并不会自动地连接到一个服务器。

### 下载和上传数据

一个Sync Adapter不会自动执行数据传输。如果你想要从一个服务器下载数据并将它存储到一个Content Provider中，你必须提供请求数据，下载数据和将数据插入到提供器里的代码。同样地，如果你想把数据发送到一个服务器，你必须要从一个文件，数据库或者Provider中读取数据，并且发送必需的上传请求。你也需要处理在你执行数据传输时所发生的网络错误。

### 处理数据冲突或者确定当前的数据是怎样的

一个Sync Adapter不会自动地解决服务器数据与设备数据的冲突。同时，它也不会检测服务器上的数据是否比设备上的数据要新，反之亦然。因此，你必须提供处理此状况的算法。

### 清理

在数据传输的尾声，记得要关闭网络连接，清除临时文件和缓存。

**Note:** Sync Adapter框架在一个后台线程中执行[onPerformSync\(\)](#)方法，所以你不需要配置你自己的后台处理任务。

另外，你应该尝试将你的定期网络相关的任务结合起来，并将它们添加到[onPerformSync\(\)](#)中。通过将所有网络任务集中到该方法中，你可以节省由启动和停止网络接口所造成的电量损失。有关更多如何在进行网络访问时更高效地使用电池，可以阅读：[Transferring Data Without Draining the Battery](#)，它描述了一些你的数据传输代码可以包含的网络访问任务。

## 将Sync Adapter和框架进行绑定

你现在在一个Sync Adapter框架中已经封装了你的数据传输代码，但是你必须向框架提供你的代码。为了做这一点，你需要创建一个捆绑[Service](#)，它将一个特殊的Android binder对象从Sync Adapter组件传递给框架。有了这一binder对象，框架可以激活[onPerformSync\(\)](#)方法并将数据传递给binder对象。

在你的服务的[onCreate\(\)](#)方法中将你的Sync Adapter组件实例化为一个单例。通过在[onCreate\(\)](#)方法中实例化该组件，你可以延迟到服务启动后再创建它，这会在框架第一次尝试执行你的数据传输时发生。你需要通过一种线程安全的方法来实例化组件，来防止Sync Adapter框架在响应激活和调度时会将Sync Adapter的执行排成多个队列。

作为例子，下面的代码片段展示了你应该如何创建一个捆绑[Service](#)的类的实现，实例化你的Sync Adapter组件，并获取Android binder对象：

```
package com.example.android.syncadapter;
/**
 * Define a Service that returns an IBinder for the
 * sync adapter class, allowing the sync adapter framework to call
 * onPerformSync().
 */
public class SyncService extends Service {
    // Storage for an instance of the sync adapter
    private static SyncAdapter sSyncAdapter = null;
    // Object to use as a thread-safe lock
    private static final Object sSyncAdapterLock = new Object();
    /*
     * Instantiate the sync adapter object.
     */
    @Override
    public void onCreate() {
        /*
         * Create the sync adapter as a singleton.
         * Set the sync adapter as syncable
         * Disallow parallel syncs
         */
        synchronized (sSyncAdapterLock) {
            if (sSyncAdapter == null) {
                sSyncAdapter = new SyncAdapter(getApplicationContext())
            }
        }
    }
    /**
     * Return an object that allows the system to invoke
     * the sync adapter.
     */
    @Override
    public IBinder onBind(Intent intent) {
        /*
         * Get the object that allows external processes
         * to call onPerformSync(). The object is created
         * in the base class code when the SyncAdapter
         */
```

```
    * constructors call super()
    */
    return sSyncAdapter.getSyncAdapterBinder();
}
}
```

**Note:** 要看更多Sync Adapter的捆绑服务的例子，可以阅读样例代码。

## 添加框架所需的账户

Sync Adapter框架需要每个Sync Adapter拥有一个账户类型。在[创建Stub授权器](#)章节中，你声明了账户类型的值。现在你需要在Android系统中配置该账户类型。要配置账户类型，通过调用[addAccountExplicitly\(\)](#)添加一个假的账户并使用其账户类型。

最佳的调用该方法的地方是在你的应用的启动Activity的[onCreate\(\)](#)方法中。下面的代码样例展示了你应该怎么做：

```
public class MainActivity extends FragmentActivity {  
    ...  
    ...  
    // Constants  
    // The authority for the sync adapter's content provider  
    public static final String AUTHORITY = "com.example.android.da  
    // An account type, in the form of a domain name  
    public static final String ACCOUNT_TYPE = "example.com";  
    // The account name  
    public static final String ACCOUNT = "dummyaccount";  
    // Instance fields  
    Account mAccount;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        // Create the dummy account  
        mAccount = CreateSyncAccount(this);  
        ...  
    }  
    ...  
    /**  
     * Create a new dummy account for the sync adapter  
     *  
     * @param context The application context  
     */  
    public static Account CreateSyncAccount(Context context) {  
        // Create the account type and default account  
        Account newAccount = new Account(  
            ACCOUNT, ACCOUNT_TYPE);  
        // Get an instance of the Android account manager  
        AccountManager accountManager =  
            (AccountManager) context.getSystemService(  
                ACCOUNT_SERVICE);  
        /*  
         * Add the account and account type, no password or user d  
         * If successful, return the Account object, otherwise rep  
         */  
        if (accountManager.addAccountExplicitly(newAccount, null,  
            /*  
             * If you don't set android:syncable="true" in  
             * in your <provider> element in the manifest,  
             * then call context.setIsSyncable(account, AUTHORITY,  
             */  
            ...  
        )  
    }  
}
```

```
    * here.  
    */  
} else {  
/*  
* The account exists or some other error occurred. Lo  
* or handle it internally.  
*/  
}  
}  
...  
}
```

# 添加Sync Adapter的元数据文件

要将你的Sync Adapter组件添加到框架中，你需要向框架提供描述组件的元数据，以及额外的标识信息。元数据指定了你为你的Sync Adapter所创建的账户类型，声明了一个和你的应用相关联的Content Provider Authority，对和Sync Adapter相关的一部分系统用户接口进行控制，并声明了其它同步相关的标识。在你的项目中的“/res/xml/”目录下的一个特定的文件内声明这一元数据，你可以为这个文件任意起一个名字，不过通常都叫做：“syncadapter.xml”。

在这一文件中包含了一个单一的XML元素`<sync-adapter>`，并且它包含了下列的属性字段：

## **android:contentAuthority**

你的Content Provider的URI Authority。如果你在前一节课程中为你的应用创建了一个Stub Content Provider，使用你在清单文件中添加的`<provider>`标签内的**android:authorities**属性的值。这一属性的更多细节在章节[创建Stub Content Provider](#)中有更多的介绍。

如果你正在使用Sync Adapter，从Content Provider将数据传输到服务器，这个值应该和你的数据的URI Authority的值是一样的。这个值也是你在清单文件中添加的`<provider>`标签内的**android:authorities**属性的值。

## **android:accountType**

Sync Adapter框架所需要的账户类型。这个值必须和你创建验证器的元数据文件中所提供的的一致（详细内容可以阅读：[创建Stub授权器](#)）。这也是你在上一节中代码片段里的常量“ACCOUNT\_TYPE”的值。

## 配置相关属性

- **android:userVisible**: 指的是Sync Adapter框架所需要的账户类型。默认地，和账户类型相关联的账户图标和标签在系统的设置里的账户选项中可以看见，所以你需要将你的Sync Adapter对用户不可见，除非你拥有一个账户类型或者域名，它们可以轻松地和你的应用相关联。如果你将你的账户类型设置为不可见，你仍然可以允许用户通过应用的一个activity内的用户接口来控制你的Sync Adapter。
- **android:supportsUploading**: 允许你将数据上传到云。如果你的应用仅仅下载数据，那么设置为“false”。
- **android:allowParallelSyncs**: 允许在同一时间你的Sync Adapter组件的多个实例运行。如果你的应用支持多个用户账户并且你希望多个用户并行地传输数据，那么使用这个属性。如果你从不执行多个数据传输，这个选项是没用的。
- **android:isAlwaysSyncable**: 指明Sync Adapter框架可以在任何你指定的时间运行你的Sync Adapter。如果你希望通过代码来控制Sync Adapter的运行，将这个标识设置为“false”，然后调用[requestSync\(\)](#)来执行Sync Adapter。要学习更多关于运行一个Sync Adapter的知识，可以阅读：[执行Sync Adapter](#)。

```
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:contentAuthority="com.example.android.datasync.pro"
    android:accountType="com.android.example.datasync"
    android:userVisible="false"
    android:supportsUploading="false"
```

```
    android:allowParallelSyncs="false"  
    android:isAlwaysSyncable="true"/>
```

# 在清单文件中声明Sync Adapter

一旦你将Sync Adapter组件添加到了你的应用中，你需要声明相关的权限来使用它，并且你需要声明你所添加的捆绑[Service](#)。

由于Sync Adapter组件运行网络与设备之间传输数据的代码，你需要使用网络的权限。另外，你的应用需要权限来读写Sync Adapter的配置信息，这样你才能通过你应用中的其它组件去控制Sync Adapter。你还需要一个特殊的权限允许你的应用使用你在[创建Stub授权器](#)中所创建的授权器组件。

要请求这些权限，将下列内容添加到你的应用清单文件中，并作为[`<manifest>`](#)标签的子标签：

## [android.permission.INTERNET](#)

允许Sync Adapter访问网络，使得它可以从设备下载和上传数据到服务器。如果之前请求了该权限，那么你就不需要重复请求了。

## [android.permission.READ\\_SYNC\\_SETTINGS](#)

允许你的应用读取当前的Sync Adapter配置。例如，你需要该权限来调用[getIsSyncable\(\)](#)。

## [android.permission.WRITE\\_SYNC\\_SETTINGS](#)

允许你的应用对Sync Adapter的配置进行控制。你需要这一权限来通过[addPeriodicSync\(\)](#)设置执行同步的时间间隔。另外，调用[requestSync\(\)](#)不需要用到该权限。更多信息可以阅读：[执行Sync Adapter](#)。

## [android.permission.AUTHENTICATE\\_ACCOUNTS](#)

允许你使用在[创建Stub授权器](#)中所创建的验证器组件。

下面的代码片段展示了如何添加权限：

```
<manifest>
...
<uses-permission
    android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.READ_SYNC_SETTINGS"/>
<uses-permission
    android:name="android.permission.WRITE_SYNC_SETTINGS"/>
<uses-permission
    android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
...
</manifest>
```

最后，要声明框架使用的捆绑[Service](#)和你的Sync Adapter进行交互，添加下列的XML代码到你的应用清单文件中，作为[`<application>`](#)标签的子标签：

```
<service
    android:name="com.example.android.datasync.SyncSer
    android:exported="true"
```

```
        android:process=":sync">
<intent-filter>com.example.android.datasync.provider
    <action android:name="android.content.SyncAdapter"
    </intent-filter>
<meta-data android:name="android.content.SyncAdapter"
            android:resource="@xml/syncadapter" />
</service>
```

<intent-filter>标签配置了一个过滤器，它会被带有“android.content.SyncAdapter”这一action的intent所激活，而这一intent一般是由系统为了运行Sync Adapter而发出的。当过滤器被激活时，系统会启动你所创建的捆绑服务，在例子中它叫做“SyncService”。属性`android:exported="true"`允许你应用之外的其它进程（包括系统）访问这一Service。属性`android:process=":sync"`告诉系统在一个全局共享，且称之为“sync”的进程中运行Service。如果你的应用中有多个Sync Adapter，那么它们可以共享该进程，这有助于减少开销。

<meta-data>标签提供了你之前为Sync Adapter所创建的元数据文件。属性`android:name`指出这一元数据是针对于Sync Adapter框架的。而`android:resource`标签则指定了元数据文件的文字。

现在你拥有了所有Sync Adapter的相关组件。下一节课将讲授如何让Sync Adapter框架运行你的Sync Adapter，既可以通过响应一个事件的方式，也可以通过执行一个定期任务调度的方式。

编写:[jdneo](#)

校对:

# 执行Sync Adapter

在这系列课程中之前的一些课程中，你学习了如何创建一个封装数据传输代码的Sync Adapter组件，以及如何添加其它的组件，来允许你将Sync Adapter集成到系统当中。现在我们已经拥有了所有需要的东西，来安装包含有一个Sync Adapter的应用，但是这里是没有任何代码是去运行Sync Adapter的。

你应该基于计划任务的调度或者一些事件的间接结果来执行Sync Adapter。例如，你可能希望你的Sync Adapter运行一个定期的计划任务，每隔一段时间或每天的一个固定的时间。或者你还希望当设备上的数据发生变化后，执行你的Sync Adapter。你应该避免将运行Sync Adapter作为用户某个行为的直接结果，因为这样做的话你就无法利用Sync Adapter框架可以按计划调度的特性。例如，你应该在UI中避免使用刷新按钮。

下列情况可以作为运行Sync Adapter的时机：

当服务器数据变更时：

运行Sync Adapter以响应来自服务器的消息，指明服务端的数据变化了。这一选项允许从服务器刷新数据到设备上，这一方法可以避免降低性能，或者由于轮询服务器所造成的电量损耗。

当设备的数据变更时：

当设备上的数据发生变化时，运行Sync Adapter。这一选项允许你将修改后的数据从设备发送给服务器，如果你需要保证服务器端的数据一直保持最新，那么这一选项非常有用。如果你将数据存储于你的Content Provider，那么这一选项的实现将会非常直接。如果你使用的是一个Stub Content Provider，检测数据的变化可能会比较困难。

当系统发送了一个网络消息：

当Android系统发送了一个网络消息来保持TCP/IP连接开启时，运行Sync Adapter。这个消息是网络框架的一个基本部分。使用这一选项是自动运行Sync Adapter的一个方法。可以考虑配合基于时间间隔的Sync Adapter一起使用。

每隔固定的时间间隔后：

在你定的时间间隔过了之后，运行Sync Adapter，或者在每天的固定时间运行它。

按照要求：

运行Sync Adapter以响应用户的行为。然而，为了提供最佳的用户体验，你应该主要依赖更多自动类型的选项。使用自动化的选项，你可以节省大量的电量以及网络资源。

本课程的后续部分会详细介绍每个选项。

## 当服务器数据变化时，运行Sync Adapter

如果你的应用从服务器传输数据，且服务器的数据频繁的发生变化，你可以使用一个Sync Adapter通过下载数据来响应服务端数据的变化。要运行Sync Adapter，让服务端向你的应用的BroadcastReceiver发送一条特殊的消息。要响应这条消息，可以调用ContentResolver.requestSync()方法，来向Sync Adapter框架发出信号，让它运行你的Sync Adapter。

谷歌云消息（[Google Cloud Messaging](#)，GCM）提供了你需要的服务端组件和设备端组件，来让这一消息提供能够运行。使用GCM激活数据传输比通过向服务器轮询的方式要更加可靠，也更加有效。因为轮询需要一个一直处于活跃状态的Service，而GCM使用的BroadcastReceiver仅在消息到达时会激活。另外，即使没有更新的内容，定期的轮询也会消耗大量的电池电量，而GCM仅在需要时才会发出消息。

**Note:** 如果你使用GCM，通过一个到所有安装了你的应用的设备的广播，来激活你的Sync Adapter，要记住他们会在同一时间（粗略地）收到你的消息。这会导致在同一时间有多个Sync Adapter的实例在运行，进而导致服务器和网络的负载过重。要避免这一情况，你应该考虑让每个设备的Sync Adapter启动的时间有所差异。

下面的代码展示了如何运行requestSync()以响应一个接收到的GCM消息：

```
public class GcmBroadcastReceiver extends BroadcastReceiver {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.da
    // Account type
    public static final String ACCOUNT_TYPE = "com.example.android
    // Account
    public static final String ACCOUNT = "default_account";
    // Incoming Intent key for extended data
    public static final String KEY_SYNC_REQUEST =
        "com.example.android.datasync.KEY_SYNC_REQUEST";
    ...
    @Override
    public void onReceive(Context context, Intent intent) {
        // Get a GCM object instance
        GoogleCloudMessaging gcm =
            GoogleCloudMessaging.getInstance(context);
        // Get the type of GCM message
        String messageType = gcm.getMessageType(intent);
        /*
         * Test the message type and examine the message contents.
         * Since GCM is a general-purpose messaging system, you
         * may receive normal messages that don't require a sync
         * adapter run.
         * The following code tests for a boolean flag indicating
         * that the message is requesting a transfer from the devi
        */
    }
}
```

```
if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType) && intent.getBooleanExtra(KEY_SYNC_REQUEST)) {
    /*
     * Signal the framework to run your sync adapter. Assume
     * app initialization has already created the account.
     */
    ContentResolver.requestSync(ACCOUNT, AUTHORITY, null);
    ...
}
...
}
```

## 当Content Provider的数据变化时，运行Sync Adapter

如果你的应用在一个Content Provider中收集数据，并且你希望当你更新提供器的时候一起更新服务器的数据，你可以配置你的Sync Adapter来让它自动运行。要做到这一点，你首先应该为Content Provider注册一个观察器（observer）。当你的Content Provider的数据发生了变化以后，Content Provider框架会调用观察器。在观察器中，调用[requestSync\(\)](#)来告诉框架运行你的Sync Adapter。

**Note:** 如果你使用的是一个空的Content Provider，那么你在Content Provider中没有任何数据，并且[onChange\(\)](#)方法从来没有被调用。在这种情况下，你不得不提供你自己的机制来检测设备数据的变化。这一机制还要负责当数据发生变化时调用[requestSync\(\)](#)。

为了给你的Content Provider创建一个观察器，继承[ContentObserver](#)类，并且实现[onChange\(\)](#)方法的几种形式。在[onChange\(\)](#)中，调用[requestSync\(\)](#)来启动Sync Adapter。

要注册观察器，将它作为参数传递给[registerContentObserver\(\)](#)。在这个调用中，你还要传递一个你想要监视的内容URI。Content Provider框架会将这个监视的URI和通过[ContentResolver](#)方法（如[ContentResolver.insert\(\)](#)）所传递过来的修改了你的提供器的URI进行对比，如果匹配上了，那么你所实现的[ContentObserver.onChange\(\)](#)将会被调用。

下面的代码片段展示了如何定义一个[ContentObserver](#)，当表发生变化时调用[requestSync\(\)](#)：

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider scheme
    public static final String SCHEME = "content://";
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.da
    // Path for the content provider table
    public static final String TABLE_PATH = "data_table";
    // Account
    public static final String ACCOUNT = "default_account";
    // Global variables
    // A content URI for the content provider's data table
    Uri mUri;
    // A content resolver for accessing the provider
    ContentResolver mResolver;
    ...
    public class TableObserver extends ContentObserver {
        /*
         * Define a method that's called when data in the
         * observed content provider changes.
         * This method signature is provided for compatibility with
         * older platforms.
         */
        @Override
        public void onChange(boolean selfChange) {
```

```
    /*
     * Invoke the method signature available as of
     * Android platform version 4.1, with a null Uri.
     */
    onChange(selfChange, null);
}
/*
 * Define a method that's called when data in the
 * observed content provider changes.
*/
@Override
public void onChange(boolean selfChange, Uri changeUri) {
    /*
     * Ask the framework to run your sync adapter.
     * To maintain backward compatibility, assume that
     * changeUri is null.
     ContentResolver.requestSync(ACCOUNT, AUTHORITY, null);
    }
    ...
}

...
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    // Get the content resolver object for your app
    mResolver = getContentResolver();
    // Construct a URI that points to the content provider data
    mUri = new Uri.Builder()
        .scheme(SCHEME)
        .authority(AUTHORITY)
        .path(TABLE_PATH)
        .build();
    /*
     * Create a content observer object.
     * Its code does not mutate the provider, so set
     * selfChange to "false"
     */
    TableObserver observer = new TableObserver(false);
    /*
     * Register the observer for the data table. The table's p
     * and any of its subpaths trigger the observer.
     */
    mResolver.registerContentObserver(mUri, true, observer);
    ...
}
...
}
```

## 在一个网络消息之后，运行Sync Adapter

当一个网络连接可获得时，Android系统会每隔几秒发送一条消息来保持TCP/IP连接打开。这一消息也会传递到每个应用的[ContentResolver](#)中。通过调用[setSyncAutomatically\(\)](#)，你可以在[ContentResolver](#)收到消息后，运行Sync Adapter。

当网络可获得的时候，通过调度你的Sync Adapter运行，来保证你的Sync Adapter在可以获得网络时都会被调度。如果不是每次数据变化时就要以数据传输来响应，但是又希望自己的数据会被定期地更新，那么可以用这一选项。类似地，如果你不想要给你的Sync Adapter配置一个定期调度，但你希望经常运行它，你也可以使用这一选项。

由于[setSyncAutomatically\(\)](#)方法不会禁用[addPeriodicSync\(\)](#)，你的Sync Adapter可能会在一小段时间内重复地被激活。如果你想要定期地运行你的Sync Adapter，你应该禁用[setSyncAutomatically\(\)](#)。

下面的代码片段向你展示如何配置你的[ContentResolver](#)来运行你的Sync Adapter，响应网络消息：

```
public class MainActivity extends FragmentActivity {  
    ...  
    // Constants  
    // Content provider authority  
    public static final String AUTHORITY = "com.example.android.da  
    // Account  
    public static final String ACCOUNT = "default_account";  
    // Global variables  
    // A content resolver for accessing the provider  
    ContentResolver mResolver;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        // Get the content resolver for your app  
        mResolver = getContentResolver();  
        // Turn on automatic syncing for the default account and a  
        mResolver.setSyncAutomatically(ACCOUNT, AUTHORITY, true);  
        ...  
    }  
    ...  
}
```

## 定期地运行Sync Adapter

你可以设置一个每次运行期间的间隔时间来定期运行你的Sync Adapter，或者在每天的固定时间运行，或者两者都有。定期地运行你的Sync Adapter可以让你与你的服务器更新间隔粗略地保持一致。

同样地，当你的服务器相对来说比较空闲时，你可以从设备更新数据，方法可以是在夜间定期调用Sync Adapter。大多数用户晚上会不关机并对收集充电，所以这一方法是可行的。而且，那个时间设备不会运行其他的任务除了你的Sync Adapter。如果你使用这个方法的话，你需要注意每台设备会在略微不同的时间激活数据传输。如果所有设备在同一时间运行你的Sync Adapter，那么你的服务器将很有可能负载过重。

一般来说，如果你的用户不需要实时更新，但希望定期更新，定期运行会很有用。如果你希望在获取实时数据和一个不过度使用用户设备资源的，高效的且更微型的Sync Adapter这两者之间进行一个平衡，那么定期执行是一个不错的选择。

要定期运行你的Sync Adapter，调用[addPeriodicSync\(\)](#)。这样每隔一段时间，Sync Adapter就会运行。由于Sync Adapter框架会考虑其他Sync Adapter的执行，并尝试最大化电池效率，间隔时间会动态做出细微调整。同时，如果网络不可获得，框架不会运行你的Sync Adapter。

注意，[addPeriodicSync\(\)](#)方法不会每天某个时间自动运行。要让你的Sync Adapter每天某个时间内自动执行，使用一个重复计时器作为激发器。重复计时器的更多细节可以阅读：[AlarmManager](#)。如果你使用[setInexactRepeating\(\)](#)方法来设置每天激活的时间具有一些变化，你仍然应该将不同设备的Sync Adapter的运行时间随机化，使得它们的执行交错开来。

[addPeriodicSync\(\)](#)方法不会禁用[setSyncAutomatically\(\)](#)，所以你可能会在一时间段内获取多个同步执行。同样，仅有一些Sync Adapter的控制标识会在[addPeriodicSync\(\)](#)方法中被允许。不允许的标识在该方法的[文档](#)中可以查看。

下面的代码样例展示了如何定期执行Sync Adapter：

```
public class MainActivity extends FragmentActivity {  
    ...  
    // Constants  
    // Content provider authority  
    public static final String AUTHORITY = "com.example.android.da  
    // Account  
    public static final String ACCOUNT = "default_account";  
    // Sync interval constants  
    public static final long MILLISECONDS_PER_SECOND = 1000L;  
    public static final long SECONDS_PER_MINUTE = 60L;  
    public static final long SYNC_INTERVAL_IN_MINUTES = 60L;  
    public static final long SYNC_INTERVAL =  
        SYNC_INTERVAL_IN_MINUTES *  
        SECONDS_PER_MINUTE *  
        MILLISECONDS_PER_SECOND;  
    // Global variables  
    // A content resolver for accessing the provider  
    ContentResolver mResolver;  
    ...  
    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    // Get the content resolver for your app
    mResolver = getContentResolver();
    /*
     * Turn on periodic syncing
     */
    ContentResolver.addPeriodicSync(
        ACCOUNT,
        AUTHORITY,
        null,
        SYNC_INTERVAL);
    ...
}
...
}
```

## 按需求执行Sync Adapter

运行你的Sync Adapter来响应一个用户需求是运行一个Sync Adapter最不推荐的策略。要知道，框架是被特别设计成根据计划运行Sync Adapter时可以最大化保留电量。既然更新数据的过程后损耗电量，那么在数据变化时响应一个Sync Adapter的同步选项应该有效地使用电量。

相比之下，允许用户按照需求运行Sync Adapter意味着Sync Adapter会自己运行，这对电量和网络来说会导致使用效率的下降。同时，向用户提供同步，会让用户甚至没有证据表明数据发生了变化以后也请求一个更新，这会导致对电量的低效率使用，一般来说，你的应用应该使用其它信号来激活一个同步更新或者定期地去做它们，而不是依赖于用户的输入。

不过，如果你仍然想要按照需求运行Sync Adapter，将Sync Adapter标识设置为人为运行的Sync Adapter，之后调用[ContentResolver.requestSync\(\)](#)。

使用下列标识来执行按需求的数据传输：

### SYNC\_EXTRAS\_MANUAL

强制执行人为的同步更新。Sync Adapter框架会忽略当前的设置，如被[setSyncAutomatically\(\)](#)方法设置的标识。

### SYNC\_EXTRAS\_EXPEDITED

强制同步立即执行。如果你不设置此项，系统可能会在运行同步需求之前等待一小段时间，因为它会尝试通过将多个请求在一小段时间内调度来尝试优化电量。

下面的代码片段将向你展示如何调用[requestSync\(\)](#)来响应一个按钮的点击：

```
public class MainActivity extends FragmentActivity {  
    ...  
    // Constants  
    // Content provider authority  
    public static final String AUTHORITY =  
        "com.example.android.datasync.provider"  
    // Account type  
    public static final String ACCOUNT_TYPE = "com.example.android.  
    // Account  
    public static final String ACCOUNT = "default_account";  
    // Instance fields  
    Account mAccount;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        /*  
         * Create the dummy account. The code for CreateSyncAccoun  
         * is listed in the lesson Creating a Sync Adapter  
         */  
  
        mAccount = CreateSyncAccount(this);  
    }  
}
```

```
    ...
}

/**
 * Respond to a button click by calling requestSync(). This is
 * asynchronous operation.
 *
 * This method is attached to the refresh button in the layout
 * XML file
 *
 * @param v The View associated with the method call,
 * in this case a Button
 */
public void onRefreshButtonClick(View v) {
    ...
    // Pass the settings flags by inserting them in a bundle
    Bundle settingsBundle = new Bundle();
    settingsBundle.putBoolean(
        ContentResolver.SYNC_EXTRAS_MANUAL, true);
    settingsBundle.putBoolean(
        ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
    /*
     * Request the sync for the default account, authority,
     * and manual sync settings
     */
    ContentResolver.requestSync(mAccount, AUTHORITY, settingsB
}
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/volley/index.html>

# 使用Volley传输网络数据(Transmitting Network Data Using Volley)

Volley是一个HTTP库，它能够帮助Android apps更方便的执行网络操作，最重要的是，它更快速高效。可以通过开源的[AOSP](#)仓库获取到Volley。

## YOU SHOULD ALSO SEE

使用Volley来编写一个app，请参考[2013 Google I/O schedule app](#)。另外需要特别关注下面2个部分：

- [ImageLoader](#)
- [BitmapCache](#)

## [VIDEO - Volley:Easy,Fast Networking for Android](#)

---

Volley有如下的优点：

- 自动执行网络请求。
- 高并发网络连接。
- 通过标准的HTTP的[cache coherence](#)(高速缓存一致性)使得磁盘与内存缓存不可见(Transparent)。
- 支持指定请求的优先级。
- 支持取消已经发出的请求。你可以取消单个请求，或者指定取消请求队列中的一个区域。
- 框架容易被定制，例如，定制重试或者回退功能。
- 强大的指令(Strong ordering)可以使得异步加载网络数据并显示到UI的操作更加简单。
- 包含了Debugging与tracing工具。

Volley擅长执行用来显示UI的RPC操作，例如获取搜索结果的数据。它轻松的整合了任何协议，并输出操作结果的数据，可以是raw strings，也可以是images，或者是JSON。通过提供内置你可能使用到得功能，Volley可以使得你免去重复编写样板代码，使你可以把关注点放在你的app的功能逻辑上。

Volley不适合用来下载大的数据文件。因为Volley会在解析的过程中保留持有所有的响应数据在内存中。对于下载大量的数据操作，请考虑使用[DownloadManager](#)。

Volley框架的核心代码是托管在AOSP仓库的frameworks/volley中，相关的工具放在toolbox下。把Volley添加到你的项目中的最简便的方法是Clone仓库然后把它设置为一个library project：

- 通过下面的命令来Clone仓库：

```
git clone  
https://android.googlesource.com/platform/frameworks/volley
```

- 以一个Android library project的方式导入下载的源代码到你的项目中。(如果你是使用Eclipse，请参考[Managing Projects from Eclipse with ADT](#))，或者编译成一个.jar文件。

# Lessons

- [发送一个简单的网络请求\(Sending a Simple Request\)](#)

学习如何通过Volley默认的行为发送一个简单的请求，以及如何取消一个请求。

- [建立一个请求队列\(Setting Up a RequestQueue\)](#)

学习如何建立一个请求队列，以及如何实现一个单例模式来创建一个请求队列。

- [生成一个标准的请求\(Making a Standard Request\)](#)

学习如何使用Volley的out-of-the-box的请求类型(raw strings, images, and JSON)来发送一个请求。

- [实现自定义的请求\(Implementing a Custom Request\)](#)

学习如何实现一个自定义的请求

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/volley/simple.html>

# 发送简单的网络请求(Sending a Simple Request)

使用Volley的方式是，你通过创建一个RequestQueue并传递Request对象给它。RequestQueue管理工作线程用来执行网络操作，从Cache中读取与写入数据，以及解析Http的响应内容。Requests执行raw responses的解析，Volley会把响应的数据分发给主线程。

这节课会介绍如何使用Volley.newRequestQueue这个建立请求队列的方法来发送一个请求，在下一节课[建立一个请求队列Setting Up a RequestQueue](#)中会介绍你自己如何建立一个请求队列。

这节课也会介绍如何添加一个请求到RequestQueue以及如何取消一个请求。

## Add the INTERNET Permission

为了使用Volley，你必须添加`android.permission.INTERNET`权限到你的manifest文件中。没有这个权限，你的app将无法访问网络。

## Use newRequestQueue

Volley提供了一个简便的方法：Volley.newRequestQueue用来为你建立一个RequestQueue，使用默认值，并启动这个队列。例如：

```
final TextView mTextView = (TextView) findViewById(R.id.text);
...
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET
        new Response.Listener() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string
        mTextView.setText("Response is: "+ response.substring(0,50)
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        mTextView.setText("That didn't work!");
    }
});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Volley总是把解析过后的数据返回到主线程中。在主线程中更加合适使用接收到的数据用来操作UI控件，这样你可以在响应的handler中轻松的修改UI，但是对于库提供的一些其他方法是有些特殊的，例如与取消有关的。

关于如何创建你自己的请求队列，不要使用Volley.newRequestQueue方法，请查看[建立一个请求队列Setting Up a RequestQueue](#)。

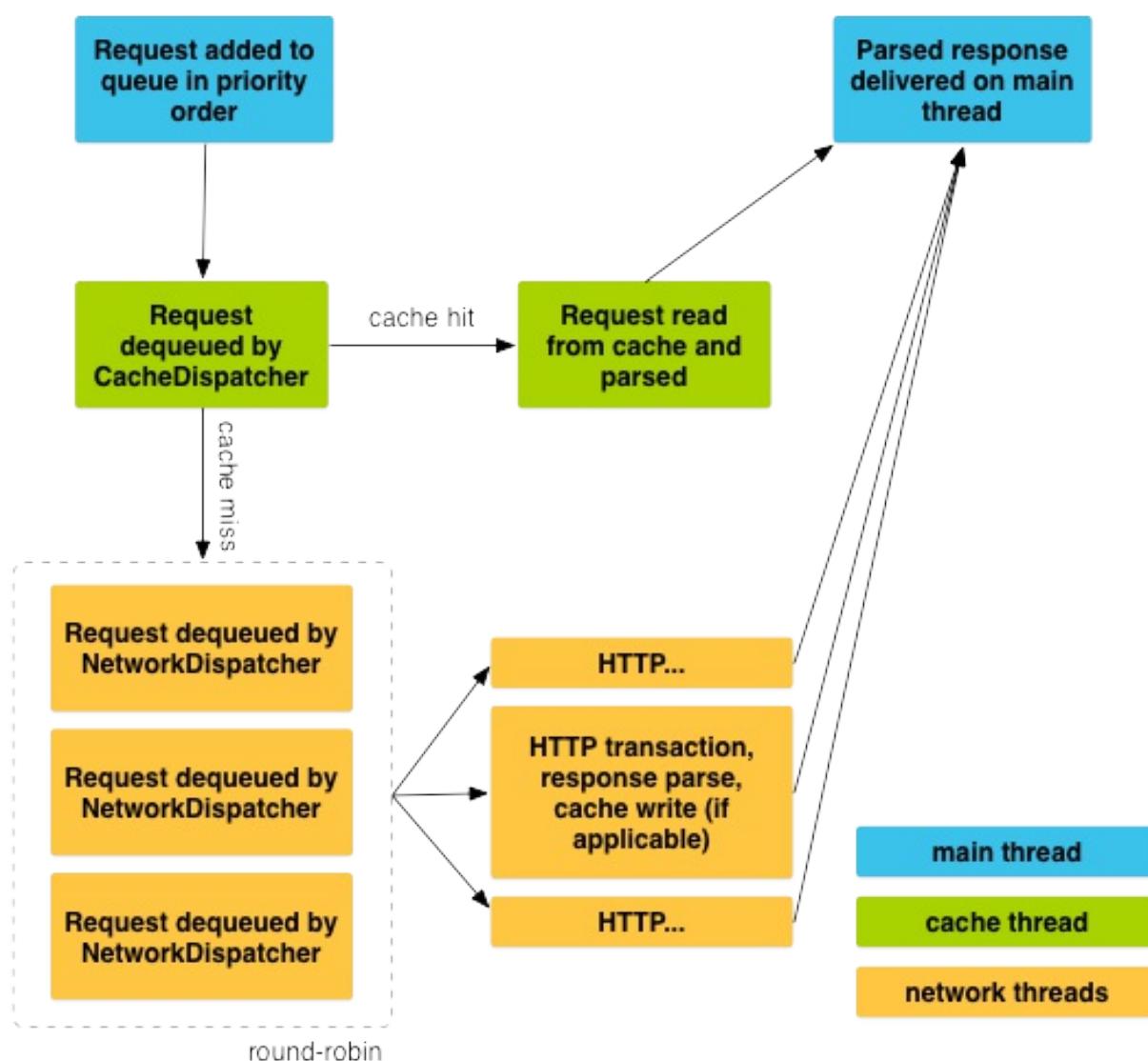
## Send a Request

为了发送一个请求，你只需要构造一个请求并通过add()方法添加到RequestQueue中。一旦你添加了这个请求，它会通过队列，得到处理，然后得到原始的响应数据并返回。

当你执行add()方法时，Volley触发执行一个缓存处理线程以及网络一系列的网络处理线程。当你添加一个请求到队列中，它将被缓存线程所捕获并触发：如果这个请求可以被缓存处理，那么会在缓存线程中执行响应数据的解析并返回到主线程。如果请求不能被缓存所处理，它会被放到网络队列中。网络线程池中的第一个可用的网络线程会从队列中获取到这个请求并执行HTTP操作，解析响应数据，把数据写到缓存中之后再把解析之后的数据返回到主线程。

请注意那些比较耗时的操作，例如I/O与解析parsing/decoding都是执行在工作线程。你可以在任何线程中添加一个请求，但是响应结果都是返回到主线程的。

下图1，演示了一个请求的生命周期：



## Cancel a Request

为了取消一个请求，对你的请求对象执行`cancel()`方法。一旦取消，Volley会确保你的响应Handler不会被执行。这意味着在实际操作中你可以在activity的`onStop()`方法中取消所有pending在队列中的请求。你不需要通过检测`getActivity() == null`来丢弃你的响应handler，其他类似`onSaveInstanceState()`等保护性的方法里面也都不需要检测。

为了利用这种优势，你应该跟踪所有已经发送的请求，以便在需要的时候，可以取消他们。有一个简便的方法：你可以为每一个请求对象都绑定一个tag对象。你可以使用这个tag来提供取消的范围。例如，你可以为你的所有请求都绑定到执行的Activity上，然后你可以在`onStop()`方法执行`requestQueue.cancelAll(this)`。同样的，你可以为ViewPager中的所有请求缩略图Request对象分别打上对应Tab的tag。并在滑动时取消这些请求，用来确保新生成的tab不会被前面tab的请求任务所卡到。

下面一个使用String来打Tag的例子：

1. 定义你的tag并添加到你的请求任务中。

```
public static final String TAG = "MyTag";
StringRequest stringRequest; // Assume this exists.
RequestQueue mRequestQueue; // Assume this exists.

// Set the tag on the request.
stringRequest.setTag(TAG);

// Add the request to the RequestQueue.
mRequestQueue.add(stringRequest);
```

1. 在activity的`onStop()`方法里面，取消所有的包含这个tag的请求任务。

```
@Override
protected void onStop () {
    super.onStop ();
    if (mRequestQueue != null) {
        mRequestQueue.cancelAll(TAG);
    }
}
```

当取消请求时请注意：如果你依赖你的响应handler来标记状态或者触发另外一个进程，你需要为此给出有力的解释。再说一次，response handler是不会被执行的。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/volley/request-queue.html>

# 建立请求队列(Setting Up a RequestQueue)

前一节课演示了如何使用Volley.newRequestQueue这一简便的方法来建立一个RequestQueue，这是利用了Volley默认的优势。这节课会介绍如何显式的建立一个RequestQueue，以便满足你自定义的需求。

这节课同样会介绍一种推荐的实现方式：创建一个单例的RequestQueue，这使得RequestQueue能够持续保持在你的app的生命周期中。

## Set Up a Network and Cache

一个RequestQueue需要两部分来支持它的工作：一部分是网络操作用来执行请求的数据传输，另外一个是用来处理缓存操作的Cache。在Volley的工具箱中包含了标准的实现方式：DiskBasedCache提供了每个文件与对应响应数据一一映射的缓存实现。

BasicNetwork提供了一个网络传输的实现，连接方式可以是[AndroidHttpClient](#)或者是[HttpURLConnection](#)。

BasicNetwork是Volley默认的网络操作实现方式。一个BasicNetwork必须使用HTTP Client进行初始化。这个Client通常是AndroidHttpClient或者 HttpURLConnection：

- 对于app target API level低于API 9(Gingerbread)的使用AndroidHttpClient。在 Gingerbread之前，HttpURLConnection是不可靠的。对于这个的细节，请参考[Android's HTTP Clients](#)。
- 对于API Level 9以及以上的，会使用HttpURLConnection。

为了创建一个能够执行在所有Android版本上的应用，你可以通过检查系统版本选择合适的HTTP Client。例如：

```
HttpStack stack;
...
// If the device is running a version >= Gingerbread...
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
    // ...use HttpURLConnection for stack.
} else {
    // ...use AndroidHttpClient for stack.
}
Network network = new BasicNetwork(stack);
```

下面的代码片段会掩饰如何一步步建立一个RequestQueue：

```
RequestQueue mRequestQueue;

// Instantiate the cache
Cache cache = new DiskBasedCache(getCacheDir(), 1024 * 1024); // 1 MB

// Set up the network to use HttpURLConnection as the HTTP client.
Network network = new BasicNetwork(new HurlStack());

// Instantiate the RequestQueue with the cache and network.
mRequestQueue = new RequestQueue(cache, network);

// Start the queue
mRequestQueue.start();

String url ="http://www.myurl.com";

// Formulate the request and handle the response.
StringRequest stringRequest = new StringRequest(Request.Method.GET,
        new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
```

```
// Do something with the response
}
},
new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        // Handle error
    }
});

// Add the request to the RequestQueue.
mRequestQueue.add(stringRequest);
...
```

如果你仅仅是想做一个单次的请求并且不想要线程池一直保留，你可以通过使用在前面一课：[发送一个简单的请求\(Sending a Simple Request\)](#)文章中提到Volley.newRequestQueue()方法在任何需要的时刻创建RequestQueue，然后在你的响应回调里面执行stop()方法来停止操作。但是更通常的做法是创建一个RequestQueue并设置为一个单例。下面将演示这种做法。

## Use a Singleton Pattern

如果你的程序需要持续的使用网络，更加高效的方式应该是建立一个RequestQueue的单例，这样它能够持续保持在整个app的生命周期中。你可以通过多种方式来实现这个单例。推荐的方式是实现一个单例类，里面封装了RequestQueue对象与其他Volley的方法。另外一个方法是继承Application类，并在Application.OnCreate()方法里面建立RequestQueue。但是这个方法是不推荐的。因为一个static的单例能够以一种更加模块化的方式提供同样的功能。

一个关键的概念是RequestQueue必须和Application context所关联的。而不是Activity的context。这可以确保RequestQueue可以在你的app生命周期中一直存活，而不会因为activity的重新创建而重新创建RequestQueue。(例如，当用户旋转设备时)。

下面是一个单例类，提供了RequestQueue与ImageLoader的功能：

```
private static MySingleton mInstance;
private RequestQueue mRequestQueue;
private ImageLoader mImageLoader;
private static Context mCtx;

private MySingleton(Context context) {
    mCtx = context;
    mRequestQueue = getRequestQueue();

    mImageLoader = new ImageLoader(mRequestQueue,
        new ImageLoader.ImageCache() {
            private final LruCache<String, Bitmap>
                cache = new LruCache<String, Bitmap>(20);

            @Override
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }

            @Override
            public void putBitmap(String url, Bitmap bitmap) {
                cache.put(url, bitmap);
            }
        });
}

public static synchronized MySingleton getInstance(Context con
    if (mInstance == null) {
        mInstance = new MySingleton(context);
    }
    return mInstance;
}

public RequestQueue getRequestQueue() {
    if (mRequestQueue == null) {
        // getApplicationContext() is key, it keeps you from l
        // Activity or BroadcastReceiver if someone passes one
        mRequestQueue = Volley.newRequestQueue(mCtx.getAppli
```

```
        }
        return mRequestQueue;
    }

    public <T> void addToRequestQueue(Request<T> req) {
        getRequestQueue().add(req);
    }

    public ImageLoader getImageLoader() {
        return mImageLoader;
    }
}
```

下面演示了利用单例类来执行RequestQueue的操作：

```
// Get a RequestQueue
RequestQueue queue = MySingleton.getInstance(this.getApplicationCo
    getRequestQueue();
...
// Add a request (in this example, called stringRequest) to your R
MySingleton.getInstance(this).addToRequestQueue(stringRequest);
```

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/volley/request.html>

# 创建标准的网络请求(Making a Standard Request)

这一课会介绍如何使用Volley支持的常用请求类型：

- `StringRequest`。指定一个URL并在相应回调中接受一个原始的raw string数据。  
请参考前一课的示例。
- `ImageRequest`。指定一个URL并在相应回调中接受一个image。
- `JsonObjectRequest`与`JsonArrayRequest` (均为`JsonRequest`的子类)。指定一个URL并在相应回调中获取到一个JSON对象或者JSON数组。

如果你需要的是上面演示的请求类型，那么你应该不需要自己实现一个自定义的请求。这节课会演示如何使用那些标准的请求类型。关于如何实现自定义的请求，请看下一课：[实现自定义的请求](#)。

# 1) Request an Image

Volley为请求图片提供了如下的类。这些类依次有着依赖关系，用来支持在不同的层级进行图片处理：

- `ImageRequest` - 一个封装好的，用来处理URL请求图片并且返回一张decode好的bitmap的类。它同样提供了一些简便的接口方法，例如指定一个大小进行重新裁剪。它的主要好处是Volley回确保类似decode, resize等耗时的操作执行在工作线程中。
- `ImageLoader` - 一个用来处理加载与缓存从网络上获取到的图片的帮助类。  
`ImageLoader`是管理协调大量的`ImageRequest`的类。例如，在`ListView`中需要显示大量缩略图的时候。`ImageLoader`为通常的Volley cache提供了更加前瞻的内存缓存，这个缓存对于防止图片抖动非常有用。。这还使得能够在避免阻挡或者延迟主线程的前提下在缓存中能够被Hit到。`ImageLoader`还能够实现响应联合Coalescing，每一个响应回调里面都可以设置bitmap到view上面。联合Coalescing使得能够同时提交多个响应，这提升了性能。
- `NetworkImageView` - 在`ImageLoader`的基础上建立，替换`ImageView`进行使用。对于需要对`ImageView`设置网络图片的情况下使用很有效。`NetworkImageView`同样可以在view被detached的时候取消pending的请求。

## 1.1) Use `ImageRequest`

下面是一个使用`ImageRequest`的示例。它会获取指定URL的image病显示到app上。里面演示的RequestQueue是通过上一课提到的单例类实现的。

```
ImageView mImageView;
String url = "http://i.imgur.com/7spzG.png";
mImageView = (ImageView) findViewById(R.id.myImage);
...

// Retrieves an image specified by the URL, displays it in the UI.
ImageRequest request = new ImageRequest(url,
    new Response.Listener() {
        @Override
        public void onResponse(Bitmap bitmap) {
            mImageView.setImageBitmap(bitmap);
        }
    }, 0, 0, null,
    new Response.ErrorListener() {
        public void onErrorResponse(VolleyError error) {
            mImageView.setImageResource(R.drawable.image_load_error);
        }
    });
// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(request);
```

## 1.2) Use `ImageLoader` and `NetworkImageView`

你可以使用`ImageLoader`与`NetworkImageView`来处理类似`ListView`等大量显示图片的情况。在你的layout XML文件中，你可以使用`NetworkImageView`来替代通常的`ImageView`，

例如：

```
<com.android.volley.toolbox.NetworkImageView  
    android:id="@+id/networkImageView"  
    android:layout_width="150dp"  
    android:layout_height="170dp"  
    android:layout_centerHorizontal="true" />
```

你可以使用ImageLoader来显示一张图片，例如：

```
ImageLoader mImageLoader;  
ImageView mImageView;  
// The URL for the image that is being loaded.  
private static final String IMAGE_URL =  
    "http://developer.android.com/images/training/system-ui.png";  
...  
mImageView = (ImageView) findViewById(R.id.regularImageView);  
  
// Get the ImageLoader through your singleton class.  
mImageLoader = MySingleton.getInstance(this).getImageLoader();  
mImageLoader.get(IMAGE_URL, ImageLoader.getImageListener(mImageView,  
    R.drawable.def_image, R.drawable.err_image));
```

然而，如果你要做得是为ImageView进行图片设置，你可以使用NetworkImageView来实现，例如：

```
ImageLoader mImageLoader;  
NetworkImageView mNetworkImageView;  
private static final String IMAGE_URL =  
    "http://developer.android.com/images/training/system-ui.png";  
...  
  
// Get the NetworkImageView that will display the image.  
mNetworkImageView = (NetworkImageView) findViewById(R.id.networkImage);  
  
// Get the ImageLoader through your singleton class.  
mImageLoader = MySingleton.getInstance(this).getImageLoader();  
  
// Set the URL of the image that should be loaded into this view,  
// specify the ImageLoader that will be used to make the request.  
mNetworkImageView.setImageUrl(IMAGE_URL, mImageLoader);
```

上面的代码是通过前一节课的单例模式来实现访问到RequestQueue与ImageLoader的。之所以这样做原因是：对于ImageLoader(一个用来处理加载与缓存图片的帮助类)来说，单例模式可以避免旋转所带来的抖动。使用单例模式可以使得bitmap的缓存与activity的生命周期无关。如果你在activity中创建ImageLoader，这个ImageLoader有可能会在手机进行旋转的时候被重新创建。这可能会导致抖动。

### 1.3) Example LRU cache

Volley工具箱中提供了通过DiskBasedCache实现的一种标准缓存。这个类能够缓存文件到

磁盘的制定目录。但是为了使用ImageLoader，你应该提供一个自定义的内存LRC缓存，这个缓存需要实现ImageLoader.ImageCache的接口。你可能想把你的缓存设置成一个单例。关于更多的有关内容，请参考[建立请求队列Setting Up a RequestQueue](#)。

下面是一个内存LRU Cache的实例。它继承自LruCache并实现了ImageLoader.ImageCache的接口：

```
import android.graphics.Bitmap;
import android.support.v4.util.LruCache;
import android.util.DisplayMetrics;
import com.android.volley.toolbox.ImageLoader.ImageCache;

public class LruBitmapCache extends LruCache<String, Bitmap>
    implements ImageCache {

    public LruBitmapCache(int maxSize) {
        super(maxSize);
    }

    public LruBitmapCache(Context ctx) {
        this(getCacheSize(ctx));
    }

    @Override
    protected int sizeOf(String key, Bitmap value) {
        return value.getRowBytes() * value.getHeight();
    }

    @Override
    public Bitmap getBitmap(String url) {
        return get(url);
    }

    @Override
    public void putBitmap(String url, Bitmap bitmap) {
        put(url, bitmap);
    }

    // Returns a cache size equal to approximately three screens worth of
    public static int getCacheSize(Context ctx) {
        final DisplayMetrics displayMetrics = ctx.getResources().
            getDisplayMetrics();
        final int screenWidth = displayMetrics.widthPixels;
        final int screenHeight = displayMetrics.heightPixels;
        // 4 bytes per pixel
        final int screenBytes = screenWidth * screenHeight * 4;

        return screenBytes * 3;
    }
}
```

下面是如何初始化ImageLoader并使用cache的实例：

```
RequestQueue mRequestQueue; // assume this exists.  
ImageLoader mImageLoader = new ImageLoader(mRequestQueue, new LruB
```

## 2) Request JSON

Volley提供了以下的类用来执行JSON请求：

- `JsonArrayRequest` - 一个为了获取`JSONArray`返回数据的请求。
- `JsonObjectRequest` - 一个为了获取`JSONObject`返回数据的请求。允许把一个`JSONObject`作为请求参数。

这两个类都是继承自`JsonRequest`的。你可以使用类似的方法来处理这两种类型的请求。如下演示了如果获取一个JSON feed并显示到UI上：

```
TextView mTxtDisplay;
ImageView mImageView;
mTxtDisplay = (TextView) findViewById(R.id.txtDisplay);
String url = "http://my-json-feed";

JsonObjectRequest jsObjRequest = new JsonObjectRequest
        (Request.Method.GET, url, null, new Response.Listener() {

    @Override
    public void onResponse(JSONObject response) {
        mTxtDisplay.setText("Response: " + response.toString());
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        // TODO Auto-generated method stub
    }
});

// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(jsObjRequest);
```

关于基于[Gson](#)实现一个自定义的JSON请求对象，请参考下一节课：[实现一个自定义的请求](#)[Implementing a Custom Request](#)。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/volley/request-custom.html>

# 实现自定义的网络请求Implementing a Custom Request

这节课会介绍如何实现你自定义的请求类型，这些自定义的类型不属于Volley内置支持包里面。

# 编写一个自定义的请求Write a Custom Request

大多数的请求类型都已经包含在Volley的工具箱里面。如果你的请求返回数值是一个string, image或者JSON, 那么你是不需要自己去实现请求类的。

对于那些你需要自定义的请求类型, 下面是你需要做得步骤:

- 继承Request<T>类, <T>表示了请求返回的数据类型。因此如果你需要解析的响应类型是一个String, 可以通过继承Request<String>来创建你自定义的请求。请参考Volley工具类中的StringRequest与ImageRequest来学习如何继承Request。
- 实现抽象方法parseNetworkResponse()与deliverResponse(), 下面会详细介绍。

## parseNetworkResponse

为了能够提交一种指定类型的数据(例如, string, image, JSON等), 需要对解析后的结果进行封装。下面会演示如何实现parseNetworkResponse()。

```
@Override  
protected Response<T> parseNetworkResponse(  
    NetworkResponse response) {  
    try {  
        String json = new String(response.data,  
            HttpHeaderParser.parseCharset(response.headers));  
        return Response.success(gson.fromJson(json, clazz),  
            HttpHeaderParser.parseCacheHeaders(response));  
    }  
    // handle errors  
    ...  
}
```

请注意:

- parseNetworkResponse()的参数是类型是NetworkResponse, 这种参数包含了的响应数据内容有一个byte[], HTTP status code以及response headers.
- 你实现的方法必须返回一个Response, 它包含了你响应对象与缓存metadata或者是一个错误。

如果你的协议没有标准的cache机制, 你可以自己建立一个Cache.Entry, 但是大多数请求都可以用下面的方式来处理:

```
return Response.success(myDecodedObject,  
    HttpHeaderParser.parseCacheHeaders(response));
```

Volley在工作线程中执行parseNetworkResponse()方法。这确保了耗时的解析操作, 例如decode一张JPEG图片成bitmap, 不会阻塞UI线程。

## deliverResponse

Volley会把parseNetworkResponse()方法返回的数据带到主线程的回调中。如下所示:

```
protected void deliverResponse(T response) {
    listener.onResponse(response);
```

## Example: GsonRequest

[Gson](#)是一个使用映射支持JSON与Java对象之间相互转换的库文件。你可以定义和JSON keys想对应名称的Java对象。把对象传递给传递Gson，然后Gson会帮你为对象填充字段值。下面是一个完整的示例：演示了使用Gson解析Volley数据：

```
public class GsonRequest<T> extends Request<T> {
    private final Gson gson = new Gson();
    private final Class<T> clazz;
    private final Map<String, String> headers;
    private final Listener<T> listener;

    /**
     * Make a GET request and return a parsed object from JSON.
     *
     * @param url URL of the request to make
     * @param clazz Relevant class object, for Gson's reflection
     * @param headers Map of request headers
     */
    public GsonRequest(String url, Class<T> clazz, Map<String, String> headers,
                       Listener<T> listener, ErrorListener errorListener) {
        super(Method.GET, url, errorListener);
        this.clazz = clazz;
        this.headers = headers;
        this.listener = listener;
    }

    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        return headers != null ? headers : super.getHeaders();
    }

    @Override
    protected void deliverResponse(T response) {
        listener.onResponse(response);
    }

    @Override
    protected Response<T> parseNetworkResponse(NetworkResponse response)
        try {
            String json = new String(
                response.data,
                HttpHeaderParser.parseCharset(response.headers));
            return Response.success(
                gson.fromJson(json, clazz),
                HttpHeaderParser.parseCacheHeaders(response));
        } catch (UnsupportedEncodingException e) {
            return Response.error(new ParseError(e));
        } catch (JsonSyntaxException e) {
            return Response.error(new ParseError(e));
        }
    }
}
```

```
        }  
    }  
}
```

如果你愿意使用的话，Volley提供了现成的JSONArrayRequest与JSONArrayObject类。参考上一课[创建标准的网络请求](#)

编写:[kesenhoo](#), [jdneo](#), 校对:

原文:<http://developer.android.com/training/cloudsync/index.html>

# 云同步

通过为网络连接提供强大的APIs，Android Framework帮助你建立丰富的，具有云功能的App，这些App可以同步数据到远程服务器端，这使得所有你的设备都保持数据同步，并且重要的数据都能够备份在云端。

这章节会介绍几种不同的策略来实现具有云功能的App。这样用当用户安装你的app到新的设备上的时候能够恢复之前的使用记录。

# Lessons

- [使用备份API](#)

学习如何集成Backup API到你的应用中。这样使得例如Preference，笔记与最高分记录等数据都能够无缝在用户的多台设备上进行同步更新。

- [使用Google Cloud Messaging](#)

学习如何高效的发送多路广播，如何正确的响应接收到的Google Cloud Messaging (GCM) 消息，以及如何使用GCM消息来与服务器进行同步。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/cloudsync/backupapi.html>

# Using the Backup API[使用Backup API]

当一个用户购买了新的设备或者是把当前的设备做了的恢复出厂设置的操作，用户希望在进行初始化设置的时候，Google Play能够把之前安装过的应用恢复到设备上。默认情况是，那些操作不会发生，用户之前的设置与数据都会丢失。

对于一些数据量相对较少的情况下(通常少于1MB)，例如用户偏好设置，笔记，游戏分数或者是其他的一些状态数据，可以使用Backup API来提供一个轻量级的解决方案。这一课会介绍如何使用Backup API。

## 1) Register for the Android Backup Service[为Android备份服务进行注册]

这一课会使用Android Backup Service, 它需要进行注册. 点击这个链接进行注册:[register here](#).  
注册成功后, 服务器会提供一段类似下面的代码用来添加到程序的Manifest文件中:

```
<meta-data android:name="com.google.android.backup.api_key"  
        android:value="ABcDe1FGHij2KlmN3oPQRs4TUvW5xYZ" />
```

请注意, 每一个备份key都只能在特定的包名下工作, 如果你有不同的程序需要使用这个方法进行备份, 那么需要为他们分别进行注册。

## 2) Configure Your Manifest[确认你的Manifest]

使用Android的备份服务需要添加2个内容到你的程序Manifest中，首先，声明作为你的备份代理的类名，然后添加一段类似上面的代码作为Application标签的根标签。假设你的备份代理是TheBackupAgent，下面演示里如何在Manifest中添加上面这些信息：

```
<application android:label="MyApp"  
            android:backupAgent="TheBackupAgent">  
    ...  
    <meta-data android:name="com.google.android.backup.api_key"  
              android:value="ABcDe1FGHij2KlmN3oPQRs4TUvW5xYZ" />  
    ...  
</application>
```

### 3) Write Your Backup Agent[编写你的备份代理]

最简单的创建你的备份代理的方法是继承[BackupAgentHelper](#). 创建这个帮助类实际上是非常简单。仅仅是创建一个你上面Manifest文件中声明的类去继承BackupAgentHelper. 然后重写onCreate(). 在onCreate() 创建一个[BackupHelper](#). 目前Android framework包含了两种那样的帮助类: [FileBackupHelper](#) 与 [SharedPreferencesBackupHelper](#). 在你创建一个帮助类并且指向需要备份的数据的时候, 仅仅需要使用 addHelper() 方法来添加到BackupAgentHelper, 在后面再增加一个key用来retrieve数据. 大多数情况下, 完整的实现差不多只需要10行代码.

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.FileBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    // The name of the SharedPreferences file
    static final String HIGH_SCORES_FILENAME = "scores";

    // A key to uniquely identify the set of backup data
    static final String FILES_BACKUP_KEY = "myfiles";

    // Allocate a helper and add it to the backup agent
    @Override
    void onCreate() {
        FileBackupHelper helper = new FileBackupHelper(this, HIGH_
            addHelper(FILES_BACKUP_KEY, helper);
    }
}
```

为了使得程序更加灵活, FileBackupHelper的constructor可以带有一些文件名, 你可以简单的通过增加一个额外的参数实现备份最高分文件与游戏程序文件, 像下面一样:

```
@Override
void onCreate() {
    FileBackupHelper helper = new FileBackupHelper(this, HIGH_SCORES_
        addHelper(FILES_BACKUP_KEY, helper);
}
```

备份用户偏好同样比较简单. 像创建FileBackupHelper一样来创建一个SharedPreferencesBackupHelper。在这种情况下, 不是添加文件名到constructor, 而是添加被你的程序所用的shared preference groups的名称. 请看示例:

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.SharedPreferencesBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    // The names of the SharedPreferences groups that the applicat
    // are the same strings that are passed to getSharedPreference
    static final String PREFS_DISPLAY = "displayprefs";
    static final String PREFS_SCORES = "highscores";

    // An arbitrary string used within the BackupAgentHelper imple
```

```
// identify the SharedPreferencesBackupHelper's data.  
static final String MY_PREFS_BACKUP_KEY = "myprefs";  
  
// Simply allocate a helper and install it  
void onCreate() {  
    SharedPreferencesBackupHelper helper =  
        new SharedPreferencesBackupHelper(this, PREFS_DISP  
    addHelper(MY_PREFS_BACKUP_KEY, helper);  
}  
}
```

你可以根据你的喜好增加许多备份帮助类,但是请记住你仅仅需要为每一类添加一个既可。一个FileBackupHelper 处理了所有的你想要备份的文件,一个SharedPreferencesBackupHelper 则处理了所有的你想要备份的shared preference groups.

## 4) Request a Backup[请求一个备份]

为了请求一个备份，仅仅需要创建一个BackupManager的实例，然后调用它的dataChanged()方法既可。

```
import android.app.backup.BackupManager;  
...  
  
public void requestBackup() {  
    BackupManager bm = new BackupManager(this);  
    bm.dataChanged();  
}
```

执行这个调用通知了backup manager 即将有数据会被备份到云端。在之后的某个时间点，backup manager会执行备份代理的 onBackup() 方法。无论任何时候，只要你的数据有发生改变的都可以去调用它，不用担心会导致过度的网络活动。如果你在上一个备份还没有发生之前再次请求了备份，那么这个备份操作仅仅会出现一次。

## **5)Restore from a Backup[从备份中恢复]**

通常情况下，你不应该手动去请求一个恢复，而是应该在你的程序安装到设备上的时候自动进行恢复。然而，如果那确实有必要手动去触发恢复，只需要调 requestRestore() 方法。

编写:[jdneo](#) - 校对:

原文:<http://developer.android.com/training/cloudsync/gcm.html>

# 使用Google Cloud Messaging

谷歌云消息（GCM）是一个用来给Android设备发送消息的免费服务。GCM消息可以极大地提升用户体验。它可以你的应用一直保持更新的状态，同时不会使你的设备由于唤醒无线电或者在没有更新时对服务器发起询问而消耗电量。同时，GCM可以让你最多一次性将一条消息发送给1000个人，使得你可以在恰当地时候很轻松地联系大量的用户，同时大量地减轻你的服务器负担。

这节课将包含把GCM集成到你的应用中的一些最佳实践方法，前提是假定你已经对该服务的基本实现有了一个了解。如果不是这样的话，你可以先阅读一下：[GCM demo app tutorial](#)。

## 高效地发送多播消息

一个GCM所支持的最有用的特性是单条消息最多可以发送给1,000个接收者。这个功能可以更加简单地将重要消息发送给你的所有用户群体。例如，比方说你有一条消息需要发送给1,000,000个人，而你的服务器每秒能发送500条消息。如果你每次只给一个接受者发送消息，那么将会耗时 $1,000,000/500=2,000$ 秒，大约半小时。然而，如果一条消息可以一次性地发送给1,000个人的话，那么耗时将会是 $1,000,000/1,000/5,00=2$ 秒。这不仅仅体现在功能的实用性上，对于具有高时效性的消息而言，比如灾难预警或者体育比分播报，如果延迟了30分钟，消息的价值就大打折扣了。

想要利用这一功能非常简单。如果你使用Java的[GCM helper library](#)，只需要向“send”或者“sendNoRetry”方法提供一个注册ID的List就行了（不要只给单个的注册ID）。

```
// This method name is completely fabricated, but you get the idea
List regIds = whoShouldISendThisTo(message);

// If you want the SDK to automatically retry a certain number of
// standard send method.
MulticastResult result = sender.send(message, regIds, 5);

// Otherwise, use sendNoRetry.
MulticastResult result = sender.sendNoRetry(message, regIds);
```

对于除了Java之外的语言，要实现GCM的支持，可以构建一个带有下列头部信息的HTTP POST请求：

- Authorization: key=YOUR\_API\_KEY
- Content-type: application/json

之后将你想要的参数编码成一个JSON对象，列出所有在“registration\_ids”这个key下的注册ID。下面的代码片段是一个例子。除了“registration\_ids”之外的所有参数都是可选的，在“data”内的项目代表了用户定义的载荷数据，而非GCM定义的参数。这个HTTP POST消息将会发送到：<https://android.googleapis.com/gcm/send>:

```
{
  "collapse_key": "score_update",
  "time_to_live": 108,
  "delay_while_idle": true,
  "data": {
    "score": "4 x 8",
    "time": "15:16.2342"
  },
  "registration_ids": ["4", "8", "15", "16", "23", "42"]
}
```

关于更多GCM多播的消息格式，可以阅读：[Sending Messages](#)。

## 对可替换的消息执行折叠

GCM经常被用作为一个触发器，告诉移动应用向服务器发起链接并刷新数据。在GCM中，可以（也推荐）在新消息要替代旧消息时，使用可折叠的消息（collapsible messages）。我们用体育比赛作为例子，如果你向所有用户发送了一条消息包含了比赛的比分，然后再15分钟后，又发送了一条消息更新比分，那么第一条消息就没有意义了。对于那些还没有收到第一条消息的用户，就没有必要接收两条消息，并且如果接收了两条消息，那么设备不得不响应两次（比如对用户发出通知或警告），但实际上两条消息中只有一条是重要的。

当你定义了一个折叠键，此时如果有多个消息在GCM服务器中，对于相同的用户形成了一个队列，那么只有最后的那一条消息会被发出。对于之前所说的体育比分的例子，这样做能让设备免于处理不必要的任务，也不会让设备对用户造成太多打扰。对于其他的一些场景比如与服务器同步数据（检查邮件接收），这样做的话可以减少设备需要执行的同步次数。例如，如果有10封邮件在服务器中等待被接收，那么实际上只需要发送一个GCM，让设备一次性把10封邮件都同步了。

为了使用这一特性，只需要在你要发出的消息中添加一个消息折叠key。如果你在使用[GCM helper library](#)，那么就使用Message类的collapseKey(String key)方法。

```
Message message = new Message.Builder(regId)
    .collapseKey("game4_scores") // The key for game 4.
    .ttl(600) // Time in seconds to keep message queued if device
    .delayWhileIdle(true) // Wait for device to become active before
    .addPayload("key1", "value1")
    .addPayload("key2", "value2")
    .build();
```

如果你没有使用[GCM helper library](#)，那么就直接在你要构建的POST头部中添加一个变量。collapse\_key作为变量名，你要更新的字段以字符串的形式作为值。

## 在GCM消息中嵌入数据

通常， GCM消息作为一个激活器，或者用来告诉设备，有一些待更新的数据需要去服务器或者别的地方去获取。然而，一个GCM消息的大小最大可以有4kb，有时候可以在GCM消息中放置一些简单的数据，这样的话设备就不需要再去和服务器发起连接了。在下列情形都满足的情况下，我们可以将数据放置在GCM消息中：

- 数据的总大小在4kb以内。
- 每一条消息都很重要，应该保留。
- 这些消息不适用于消息折叠的使用情形。

例如，短消息或者回合制网游中玩家的移动数据等都是将数据直接嵌入在GCM消息中的例子。而电子邮件就是反面例子了。因为电子邮件的数据量一般都大于4kb，且用户不需要对每个邮件都收到一个GCM提醒的消息。

同时在发送多播消息时，也可以考虑这一方法，这样的话就不会导致大量用户在接收到GCM的更新提醒后，同时向你的服务器发起连接。

这一策略不适用于发送大量的数据（你可能会想要用这种方法将数据分割后发送），有这么一些原因：

- 为了防止恶意软件发送垃圾消息，GCM有发送频率的限制。
- 无法保证消息按照既定的顺序到达。
- 无法保证消息可以在你发送后立即到达。假设设备每一秒都接收一条消息，最大为1K，即8kbps，或者说是1990年代的家庭拨号上网的速度。那么如此大量的消息，一定会让你的应用在Google Play上的评分非常尴尬。

如果恰当地使用，直接将数据嵌入到GCM消息中，可以加速你的应用的“感知速度”，因为它不必再去服务器获取数据了。

## 智能地响应GCM消息

你的应用不应该仅仅对收到的GCM消息进行响应就够了，还应该响应地更智能一些。至于要如何响应需要结合具体情况而定。

### 不要太过激进

当提醒用户区更新数据时，很容易不小心从“有用的消息”变成“干扰消息”。如果你的应用使用状态栏通知，那么应该[更新现有的通知](#)，而不是创建第二个。如果你通过铃声或者震动的方式提醒用户，一定要设置一个计时器。不要让应用的提醒时间超过1分钟，不然的话用户很可能为不堪其扰而卸载你的应用，关机，或者把手机扔到河里：）

### 用聪明的办法同步数据，别用笨办法

当使用GCM告知设备有数据需要从服务器下载时，记住你有4kb的数据大小和消息一起发出，这可以帮助你的应用做出更智能地响应。例如，如果你有一个源阅读应用，而你的用户订阅了100个源，那么这就可以帮助你的应用更智能地决定应该去服务器下载什么数据。下面的例子说明了在GCM载荷中可以发送那些数据，以及设备可以做出什么样的反应：

- refresh - 你的应用被告知向每一个源请求数据。此时你的应用可以向100个不同的服务器发起获取源的请求，或者如果你在你的服务器上有一个聚合服务，那么可以只发送一个请求，将100个源的数据进行打包并获取，这样一次性完成更新。
- refresh, freshID - 一种更好的解决方案，你的应用可以有针对性的完成更新。
- refresh, freshID, timestamp - 一种更好的解决方案，如果正好用户在GCM消息收到之前手动做了更新，那么应用可以利用时间戳和当前的更新时间进行对比，并决定是否要执行下一步的行动。

编写:[jdneo](#), 校对:

原文:<http://developer.android.com/training/cloudsave/conflict-res.html>

# 解决云同步的保存冲突

这篇文章介绍了当应用使用[Cloud Save service](#)存储数据到云端时，如何设计一个鲁棒性较高的冲突解决策略。云存储服务允许你为每一个在Google服务上的应用用户，存储他们的应用数据。你的应用可以通过使用云存储API，从Android设备，iOS设备或者web应用恢复或更新这些数据。

在云存储过程中的保存和加载是很直接的：它只是一个数据和byte数组之间的相互转换，并将这些数组存储在云端。然而，当你的用户有多个设备，并且两个以上的设备尝试将它们的数据存储在云端时，这一保存可能会引起冲突，因此你必须决定应该如何处理。你在云端存储的数据结构在很大程度上决定了你的冲突解决方案的鲁棒性，所以小心地设计你的数据，使得你的冲突检测解决方案的逻辑可以正确地处理每一种情况。

本篇文章从描述一些有缺陷的方法入手，并解释他们为何具有缺陷。之后呈现一个解决方案来避免冲突。用于讨论的例子关注于游戏，但解决问题的宗旨是可以适用于任何将数据存储于云端的应用的。

## 冲突时获得通知

[OnStateLoadedListener](#)方法负责从Google服务器下载应用的状态数据。回调函数[OnStateLoadedListener.onStateConflict](#)为你的应用在本地状态和云端存储的状态发生冲突时，提供了一个解决机制：

```
@Override  
public void onStateConflict(int stateKey, String resolvedVersion,  
    byte[] localData, byte[] serverData) {  
    // resolve conflict, then call mAppStateClient.resolveConflict  
    ...  
}
```

此时你的应用必须决定要保留哪一个数据，或者它自己提交一个新的数据来表示合并后的数据状态，解决冲突的逻辑由你来实现。

我们必须要意识到云存储服务是在后台执行同步的。所以你应该确保你的应用能够在你创建这一数据的context之外接回调。特别地，如果Google Play服务应用在后台检测到了一个冲突，该回调函数可以在你下一次加载数据时被调用，而不是下一次用户启动该应用时。

因此，你的云存储代码和冲突解决代码的设计必须是和当前context无关的：即给两个冲突的数据，你必须仅通过数据集中获取的数据区解决冲突，而不依赖于任何其它外部环境。

## 处理简单情况

下面列举一些冲突解决的简单例子。对于很多应用而言，用这些策略或者其变体就足够解决大多数问题了：

**新的比旧的更有效：**在一些情况下，新的数据总是替代老数据。例如，如果数据代表了用户选择角色的衣服颜色，那么最近的新选择就应该覆盖老的选择。在这种情况下，你可能会选择在云存储数据中存储时间戳。当处理这些冲突时，选择时间戳最新的数据（记住要选择一个可靠的时钟，并注意对不同时区的处理）。

**一个数据好于其他数据：**在一些情况下，我们是可以有方法在若干数据集中选取一个最好的。例如，如果数据代表了玩家在赛车比赛中的最佳时间，那么显然，在冲突发生时，你应该保留成绩最好的那个数据。

**进行合并：**有可能通过计算两个数据集的合并版本来解决冲突。例如，如果你的数据代表了用户解锁关卡的进度，那么我们需要的数据就是冲突集的并集。通过这个方法，用户不会丢失任何他的游戏进度。这里的[例子](#)使用了这一操作的一个变形。

# 为更复杂的情况设计一个策略

一个更复杂的情况是当你的游戏允许玩家收集可以互换的东西时（比如金币或者经验点数），我们来假想一个游戏，叫做“金币跑酷”，一个无限跑步的角色，其目标是不断地收集金币使自己变的富有。每个收集到的金币都会加入到玩家的储蓄罐中。

下面的章节将展示三种在多个设备间解决冲突的方案：有两个听上去很不错，可惜最终还是不能适用于所有的场景，最后一个解决方案可以解决多个设备间的冲突。

## 第一个尝试：只保存总数

首先，这个问题看上去像是说：云存储的数据只要存储金币的数量就行了。但是如果就只有这些数据是可用的，那么解决冲突的方案将会严重受到限制。此时最佳的方案就是在冲突发生时存储数值最大的数据。

想一下表1中所展现的场景。假设玩家一开始有20枚硬币，然后在设备A上收集了10个，在设备B上收集了15个。然后设备B将数据存储到了云端。当设备A尝试去存储的时候，冲突发生了。“只存储总数”的冲突解决方案会存储35作为这一数据的值（两数之间最大的）。

表1. 值保存最大的数（不佳的策略）

| 事件                  | 设备A的数据 | 设备B的数据 | 云端的数据 | 实际的总数 |
|---------------------|--------|--------|-------|-------|
| 开始阶段                | 20     | 20     | 20    | 20    |
| 玩家在A设备上收集了10个硬币     | 30     | 20     | 20    | 30    |
| 玩家在B设备上收集了15个硬币     | 30     | 35     | 20    | 45    |
| 设备B将数据存储至云端         | 30     | 35     | 35    | 45    |
| 设备A尝试将数据存储至云端，发生冲突  | 30     | 35     | 35    | 45    |
| 设备A通过选择两数中最大的数来解决冲突 | 35     | 35     | 35    | 45    |

这一策略会失败：玩家的金币数从20变成35，但实际上玩家总共收集了25个硬币（A设备10个，B设备15个）。所以有10个硬币丢失了。只在云端存储硬币的总数是不足以实现一个鲁棒的冲突解决算法的。

## 第二个尝试：存储总数和变化值

另一个方法是在存储数据中包括一些额外的数据：自上次提交后硬币增加的数量（ $\Delta$ ）。在这一方法中，存储的数据可以用一个二元组来表示  $(T, \Delta)$ ，其中  $T$  是硬币的总数，而  $\Delta$  是硬币增加的数量。

在这个结构中，你的冲突检测算法在鲁棒性上有更大的提升空间。但是这个方法还是无法给出一个可靠的玩家最终的状态。

下面是包含  $\Delta$  的冲突解决算法过程：

- 本地数据：  $(T, \Delta)$
- 云端数据：  $(T', \Delta')$
- 解决后的数据：  $(T + \Delta, \Delta')$

例如，当你在本地状态  $(T, d)$  和云端状态  $(T, d)$  之间发生了冲突时，你可以将它们合并成  $(T+d, d)$ 。意味着你从本地拿出delta数据，并将它和云端的数据结合起来，乍一看，这种方法可以很好的计量多个设备所收集的金币。

看上去很可靠的方法，但这个方法在移动环境中难以适用：

- 用户可能在设备不在线时存储数据。这些改变会以队列形式等待手机联网后提交。
- 这个方法的同步机制是用最新的变化覆盖掉任何之前的变化。换句话说，第二次写入的变化会提交到云端（当设备联网了以后），而第一次写入的变化就被忽略了。

为了进一步说明，我们考虑一下表2所列的场景。在表2的一系列操作后，云端的状态将是  $(130, +5)$ ，之后最终冲突解决后的状态是  $(140, +10)$ 。这是不正确的，因为从总体上而言，用户一共在A上收集了110枚硬币而在B上收集了120枚硬币。总数应该为250。

表2.“总数+增量”策略的失败案例

| 事件                       | 设备A的数<br>据    | 设备B的数<br>据    | 云端的数<br>据    | 实际的总<br>数 |
|--------------------------|---------------|---------------|--------------|-----------|
| 开始阶段                     | $(20, x)$     | $(20, x)$     | $(20, x)$    | 20        |
| 玩家在A设备上收集了100个硬币         | $(120, +100)$ | $(20, x)$     | $(20, x)$    | 120       |
| 玩家在A设备上又收集了10个硬币         | $(130, +10)$  | $(20, x)$     | $(20, x)$    | 130       |
| 玩家在B设备上收集了115个硬币         | $(130, +10)$  | $(125, +115)$ | $(20, x)$    | 245       |
| 玩家在B设备上又收集了5个硬币          | $(130, +10)$  | $(130, +5)$   | $(20, x)$    | 250       |
| 设备B将数据存储至云端              | $(130, +10)$  | $(130, +5)$   | $(130, +5)$  | 250       |
| 设备A尝试将数据存储至云端，发生冲突       | $(130, +10)$  | $(130, +5)$   | $(130, +5)$  | 250       |
| 设备A通过将本地的增量和云端的总数相加来解决冲突 | $(140, +10)$  | $(130, +5)$   | $(140, +10)$ | 250       |

注：x代表与该场景无关的数据

你可能会尝试在每次保存后不重置增量数据来解决此问题，这样的话在每个设备上的第二次存储所收集到的硬币将不会产生问题。这样的话设备A在第二次本地存储完成后，数据将是  $(130, +110)$  而不是  $(130, +10)$ 。然而，这样做的话就会发生如表3所述的情况：

表3. 算法改进后的失败案例

| 事件               | 设备A的数<br>据    | 设备B的数<br>据 | 云端的数<br>据     | 实际的总<br>数 |
|------------------|---------------|------------|---------------|-----------|
| 开始阶段             | $(20, x)$     | $(20, x)$  | $(20, x)$     | 20        |
| 玩家在A设备上收集了100个硬币 | $(120, +100)$ | $(20, x)$  | $(20, x)$     | 120       |
| 设备A将状态存储到云端      | $(120, +100)$ | $(20, x)$  | $(120, +100)$ | 120       |
| 玩家在A设备上又收集了10个硬币 | $(130, +110)$ | $(20, x)$  | $(120, +100)$ | 130       |
| 玩家在B设备上收集了1个硬币   | $(130, +110)$ | $(21, +1)$ | $(120, +100)$ | 131       |

|                          |                       |  |     |
|--------------------------|-----------------------|--|-----|
| 设备B尝试向云端存储数据，发生冲突        | (130, +110) (21, +1)  | $\begin{pmatrix} +100 \\ (120, \\ +100) \end{pmatrix}$ | 131 |
| 设备B通过将本地的增量和云端的总数相加来解决冲突 | (130, +110) (121, +1) | (121, +1)  | 131 |
| 设备A尝试将数据存储至云端，发生冲突       | (130, +110) (121, +1) | (121, +1)  | 131 |
| 设备A通过将本地的增量和云端的总数相加来解决冲突 | (231, +110) (121, +1) | $\begin{pmatrix} (231, \\ +110) \end{pmatrix}$         | 131 |

注：x代表与该场景无关的数据

现在你碰到了另一个问题：你给予了玩家过多的硬币。这个玩家拿到了211枚硬币，但实际上他只收集了111枚。

### 解决办法：

分析之前的几次尝试，我们发现这些策略都没有这样一个能力：知晓哪些硬币已经计数了，哪些硬币没有被计数，尤其是当多个设备连续提交的时候，算法会出现混乱。

该问题的解决办法将你云端的存储结构改为字段，使用字符串+整形的键值对。每一个键值对都会代表一个包含硬币的“委托”，而总数就应该是将所有值加起来。这一设计的宗旨是每个设备有它自己的委托，并且只有设备自己可以把硬币放到其委托中。

字典的结构是：(A:a, B:b, C:c, ...)，其中a代表了委托A所拥有的硬币，b是委托B所拥有的硬币，以此类推。

这样的话，新的冲突解决策略算法将如下所示：

- 本地数据：(A:a, B:b, C:c, ...)
- 云端数据：(A:a', B:b', C:c', ...)
- 解决后的数据：(A:max(a,a'), B:max(b,b'), C:max(c,c'), ...)

例如，如果本地数据是(A:20, B:4, C:7)并且云端数据是(B:10, C:2, D:14)，这样的话解决冲突后的数据将会是(A:20, B:10, C:7, D:14)。注意，应用的冲突解决逻辑会根据具体的场景可能有所差异。比如，有一些应用你可能希望挑选最小的值。

为了测试新的算法，将它应用于任何一个之前提到过的场景。你将会发现它都能取得正确地结果。

表4阐述了这一点，它基于表3的场景。注意下面所列的：

在初始状态，玩家有20枚硬币。此数值在所有设备和云端都是正确的，我们用(X:20)这一元组代表它，其中X我们不用太多关心，我们不去追求这个初始化的数据是哪儿来的。

当玩家在设备A上收集了100枚硬币，这一变化会作为一个元组保存到云端。它的值是100是因为这就是玩家在设备A上收集的硬币数量。在这一过程中，没有要执行数据的计算（设备A仅仅是将玩家所收集的数据汇报给了云端）。

每一个新的硬币提交会打包成一个与设备关联的元组并保存到云端。例如，假设玩家又在设备A上收集了100枚硬币，那么元组的值被更新为110。

最终的结果就是，应用知道了玩家在每个设备上收集硬币的总数。这样它就能轻易地计算总数了。

表4. 键值对策略的成功应用案例

| 事件                 | 设备A的数据             | 设备B的数据             | 云端的数据                                | 实际的总数 |
|--------------------|--------------------|--------------------|--------------------------------------|-------|
| 开始阶段               | (X:20, x)          | (X:20, x)          | (X:20, x)                            | 20    |
| 玩家在A设备上收集了100个硬币   | (X:20, A:100)      | (X:20)             | (X:20)                               | 120   |
| 设备A将状态存储到云端        | (X:20, A:100)      | (X:20)             | (X:20, A:100)                        | 120   |
| 玩家在A设备上又收集了10个硬币   | (X:20, A:110)      | (X:20)             | (X:20, A:100)                        | 130   |
| 玩家在B设备上收集了1个硬币     | (X:20, A:110)      | (X:20, B:1)        | (X:20, A:100)                        | 131   |
| 设备B尝试向云端存储数据，发生冲突  | (X:20, A:110)      | (X:20, B:1)        | (X:20, A:100)                        | 131   |
| 设备B解决冲突            | (X:20, A:110)      | (X:20, A:100, B:1) | (X:20, A:100, B:1)                   | 131   |
| 设备A尝试将数据存储至云端，发生冲突 | (X:20, A:110)      | (X:20, A:100, B:1) | (X:20, A:100, B:1)                   | 131   |
| 设备A解决冲突            | (X:20, A:110, B:1) | (X:20, A:100, B:1) | (X:20, A:110, B:1), <b>total 131</b> | 131   |

## 清除你的数据

在云端存储数据的大小是有限制的，所以在后续的论述中，我们将会关注如何避免创建过大的词典。一开始，看上去每个设备只会有一个词典字段，即使是非常激进的用户也不太会拥有上千条字段。然而，获取设备ID的方法很难，并且我们认为这是一种不好的实践方式，所以你应该使用一个安装ID，这更容易获取也更可靠。这样的话就意味着，每一次用户在每台设备安装一次就会产生一个ID。假设每个键值对占据32字节，由于一个个人云存储缓存最多可以有128K的大小，那么你最多可以存储4096个字段。

在现实场景中，你的数据可能更加复杂。在这种情况下，存储的数据字段数也会进一步受到限制。具体而言则需要取决于实现，比如可能需要添加时间戳来指明每个字段是何时修改的。当你检测到有一个字段在过去几个礼拜或者几个月的时间内都没有被修改，那么就可以安全地将它转移到另一个字段中并删除老的字段。

编写: [spencer198711](#) - 校对:

原文:

# 联系人信息与位置信息

These classes teach you how to add user personalization to your app. Some of the ways you can do this is by identifying users, providing information that's relevant to them, and providing information about the world around them.

# Lessons

## [访问联系人数据 - Accessing Contacts Data](#)

How to use Android's central address book, the Contacts Provider, to display contacts and their details and modify contact information.

## [位置信息- Making Your App Location-Aware](#)

How to add location-aware features to your app by getting the user's current location.

编写: [spencer198711](#) - 校对:

原文:

# 联系人信息

[Contacts Provider](#)是用户联系人信息的集中仓库，它包含了来自联系人应用与社交应用的联系人数据。在你的应用中，你可以通过调用[ContentResolver](#)的方法或者通过发送Intent给联系人应用来访问Contacts Provider的信息。

这个章节会讲解获取联系人列表，显示指定联系人详细以及通过intent来修改联系人信息。这些基础技能能够进行扩展执行更复杂的任务。同时，这个章节也会帮助你了解Contacts Provider的整个架构与操作方法。

# Lessons

- [获取联系人列表 - Retrieving a List of Contacts](#)

学习如何获取联系人列表。你可以使用下面的技术来筛选需要的信息：

- 通过联系人名字进行筛选
- 通过联系人类型进行筛选
- 通过类似电话号码等指定的一类信息进行筛选。

- [获取联系人详情 - Retrieving Details for a Contact](#)

学习如何获取单个联系人的详情。一个联系人的详细信息包括电话号码与邮件地址等等。你可以获取所有的详细信息，也有可以只获取指定的详细数据，例如邮件地址。

- [修改联系人信息 - Modifying Contacts Using Intents](#)

学习如何通过发送intent给联系人应用来修改联系人信息。

- [显示联系人头像 - Displaying the Quick Contact Badge](#)

学习如何显示**QuickContactBadge**小组件。当用户点击联系人臂章(头像)组件时，会显示一个联系人详情的对话框，并提供给应用可以进行的操作。例如，如果联系人信息有邮件地址，这个对话框可以显示一个启动默认邮件应用的操作按钮。

编写: [spencer198711](#) - 校对:

原文:

# 获取联系人列表

这一课展示了如何根据要搜索的字符串去匹配联系人的数据，从而得到联系人列表，你可以使用以下方法去实现：

## 匹配联系人名字

根据搜索字符串来匹配部分或者全部联系人的名字来获得联系人列表，联系人数据库允许多人拥有相同的名字，所以这种方法能够取得相匹配的的列表。

## 匹配特定的数据类型，比如电话号码

根据搜索字符串来匹配联系人的特定类型的数据，比如电子邮件，来取得符合要求的联系人列表。比如说，这种方法可以让你取得联系人列表，他们的电子邮件于搜索字符相匹配。

## 匹配任意类型的数据

根据搜索字符串来匹配联系人详情的所有类型的数据，包括名字、电话号码、地址、电子邮件地址等等。比如说，这种方法可以让你根据任意类型的数据，去获取与联系人详情数据相匹配的列表。

提示：这一课的所有例子都使用了CursorLoader去从ContactsProvider中获取数据。CursorLoader在一个工作线程中去运行查询操作，使得能够与UI线程分开，这保证了数据查询不会降低UI响应的时间，以免引起糟糕的用户体验。更多信息，请参照在后台加载数据。

## 请求读取联系人的权限

为了能够在联系人数据库中做任意类型的搜索，你的应用必须拥有READ\_CONTACTS的权限，为了拥有这个权限，你需要向项目的清单文件中添加以下结点作为的子结点

```
<uses-permission android:name="android.permission.READ_CONTACTS">
```

## 根据名字取得联系人并列出结果列表

这种方法试图通过根据一个搜索字符串，去匹配联系人数据库ContactsContract.Contacts表中的联系人名字，从而取得一个或者多个联系人。通常希望在ListView中展示结果，去让用户在所有匹配的联系人中做选择。

### 定义列表和列表项的布局

为了能够将搜索结果展示在列表中，你需要一个包含ListView以及其他布局控件的主布局文件，和定义列表中每一项的布局文件。例如，你可以使用以下的XML代码去创建主布局文件res/layout/contacts\_list\_view.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

这个XML代码使用了Android内建的ListView控件，他的id是android:id/list。

使用以下XML代码定义列表项布局文件contacts\_list\_item.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:clickable="true"/>
```

这个XML代码使用了Android内建的TextView控件，他的id是android:text1。

提示：本课并不会描述如何从用户那里获取搜索字符串的界面，因为你可能会间接地获取这个字符串。比如说，你可能会给用户一个选项，让他从收到的短信中的部分内容作为名字去搜索匹配的联系人。

刚刚写的这两个布局文件定义了展示在ListView的用户界面，下一步是编写使用这些界面显示联系人列表的代码。

### 定义显示联系人列表的Fragment

为了显示联系人列表，需要定义一个由Activity加载的Fragment。使用Fragment是一个比较灵活的方法，因为你可以使用一个Fragment去显示列表，当用户选择列表中的某一个联系人的时候，用第二个Fragment显示此联系人的详情。使用这种方式，你可以结合本课程中展示的方法和另外一课“获取联系人详情”。

想要学习如何在Activity中使用一个或者多个Fragment，请阅读培训课程“使用Fragment构建灵活的用户界面”。

为了帮你编写对联系人数据库的查询，android框架提供了一个叫做ContactsContract的契约类，这个类定义了一些对查询数据库很有用的常量和方法。当你使用这个类的时候，你不用自己定义内容URI、表名、列名等常量。使用这个类，你需要引入以下类声明：

```
import android.provider.ContactsContract;
```

由于代码中使用了CursorLoader去从provider中获取数据，你必须实现加载器接口 LoaderManager.LoaderCallbacks。同时，为了检测用户从结果列表中选择了哪一个联系人，必须实现适配器接口AdapterView.OnItemClickListener。例如：

```
...
import android.support.v4.app.Fragment;
import android.support.v4.app.LoaderManager.LoaderCallbacks;
import android.widget.AdapterView;
...
public class ContactsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor>,
    AdapterView.OnItemClickListener {
```

## 定义全局变量

定义在其他代部分码中使用的全局变量：

```
...
/*
 * Defines an array that contains column names to move from
 * the Cursor to the ListView.
 */
@SuppressWarnings("InlinedApi")
private final static String[] FROM_COLUMNS = {
    Build.VERSION.SDK_INT
        >= Build.VERSION_CODES.HONEYCOMB ?
        Contacts.DISPLAY_NAME_PRIMARY :
        Contacts.DISPLAY_NAME
};
/*
 * Defines an array that contains resource ids for the layout
 * that get the Cursor column contents. The id is pre-defined
 * the Android framework, so it is prefaced with "android.R.id"
 */
private final static int[] TO_IDS = {
    android.R.id.text1
};
// Define global mutable variables
// Define a ListView object
ListView mContactsList;
// Define variables for the contact the user selects
// The contact's _ID value
long mContactId;
// The contact's LOOKUP_KEY
String mContactKey;
```

```
// A content URI for the selected contact
Uri mContactUri;
// An adapter that binds the result Cursor to the ListView
private SimpleCursorAdapter mCursorAdapter;
...
```

提示：由于Contacts.DISPLAY\_NAME\_PRIMARY需要在android 3.0 (API版本11) 之后才能使用，如果你的应用的minSdkVersion是10或者更小，会在eclipse中产生警告信息。为了关闭这个警告，你可以在FROM\_COLUMNS定义之前加上@SuppressLint("InlinedApi")注解。

## 初始化Fragment

为了初始化Fragment，android系统需要你为这个Fragment添加空的、公有的构造方法，同时在回调方法onCreateView()中绑定界面。例如：

```
// Empty public constructor, required by the system
public ContactsFragment() {}
// A UI Fragment must inflate its View
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup co
    Bundle savedInstanceState) {
    // Inflate the fragment layout
    return inflater.inflate(R.layout.contact_list_fragment,
        container, false);
}
```

## 为ListView绑定CursorAdapter数据

将绑定到搜索结果的SimpleCursorAdapter设置到ListView。为了获得显示联系人列表的ListView控件，需要使用Fragment的父Activity调用Activity.findViewById()。当你调用setAdapter()的时候，需要使用父Activity的上下文（Context）。

```
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    ...
    // Gets the ListView from the View list of the parent acti
    mContactsList =
        (ListView) getActivity().findViewById(R.layout.contact_
    // Gets a CursorAdapter
    mCursorAdapter = new SimpleCursorAdapter(
        getActivity(),
        R.layout.contact_list_item,
        null,
        FROM_COLUMNS, TO_IDS,
        0);
    // Sets the adapter for the ListView
    mContactsList.setAdapter(mCursorAdapter);
}
```

## 为选择的联系人设置监听器

当你显示搜索列表结果的时候，你通常会让用户选择某一个联系人去做进一步的处理。例如，当用户选择某一个联系人的时候，你可以在地图上显示这个人的地址。为了能够提供这个功能。你需要定义当前的Fragment为一个点击监听器，这需要这个类实现 AdapterView.OnItemClickListener接口，就像“定义显示联系人列表的Fragment”那一节展示的那样。

继续设置这个监听器，需要在onActivityCreated()方法中调用setOnItemClickListener()以使得这个监听器绑定到ListView。例如：

```
public void onActivityCreated(Bundle savedInstanceState) {  
    ...  
    // Set the item click listener to be the current fragment.  
    mContactsList.setOnItemClickListener(this);  
    ...  
}
```

由于指定了当前的Fragment作为ListView的点击监听器，现在你需要实现处理点击事件的onItemClick()方法。这个会在随后讨论。

## 定义查询映射

定义一个常量，这个常量是你想要的查询返回值所包含的列。Listview中得每一行显示了一个联系人的“显示名字”，它包含了联系人名字的主要部分。在android 3.0之后，这个列的名字是Contacts.DISPLAY\_NAME\_PRIMARY,在android 3.0之前，这个列的名字是Contacts.DISPLAY\_NAME。

Contacts.\_ID列在SimpleCursorAdapter绑定过程中会用到。Contacts.\_ID和LOOKUP\_KEY一同用来构建用户选择的联系人的内容URI。

```
...  
@SuppressLint("InlinedApi")  
private static final String[] PROJECTION =  
{  
    Contacts._ID,  
    Contacts.LOOKUP_KEY,  
    Build.VERSION.SDK_INT  
        >= Build.VERSION_CODES.HONEYCOMB ?  
        Contacts.DISPLAY_NAME_PRIMARY :  
        Contacts.DISPLAY_NAME  
};
```

## 定义Cursor的列索引常量

为了从Cursor中获得单独某一列的数据，你需要知道这一列在Cursor中的索引值。你需要定义Cursor列的索引值，这些索引值同你定义的查询映射的列的顺序是一样的。例如：

```
// The column index for the _ID column  
private static final int CONTACT_ID_INDEX = 0;
```

```
// The column index for the LOOKUP_KEY column
private static final int LOOKUP_KEY_INDEX = 1;
```

## 指定查询标准

为了指定你想要查询的数据，你需要创建一个包含字符串表达式和变量组成的条件，去告诉provider你需要的数据列和想要的值。

对于字符串表达式，你需要定义一个所有列要满足的条件的常量。尽管这个表达式可以包含变量值，但是一个比较好的建议是用"?"占位符来替代这个值，在搜索的时候，占位符里的值会被数组里的值所取代。使用"?"占位符确保了搜索条件是由绑定产生而不是有SQL编译产生。这条实践消除了恶意SQL注入的可能。例如：

```
// Defines the text expression
@SuppressLint("InlinedApi")
private static final String SELECTION =
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
        Contacts.DISPLAY_NAME_PRIMARY + " LIKE ?" :
        Contacts.DISPLAY_NAME + " LIKE ?";
// Defines a variable for the search string
private String mSearchString;
// Defines the array to hold values that replace the ?
private String[] mSelectionArgs = { mSearchString };
```

## 定义onItemClick()方法

在之前的内容中，你为Listview设置了列表项点击监听器，现在需要定义AdapterView.OnItemClickListener.onItemClick()方法以实现监听器行为：

```
@Override
public void onItemClick(
    AdapterView<?> parent, View item, int position, long rowID
) {
    // Get the Cursor
    Cursor cursor = parent.getAdapter().getCursor();
    // Move to the selected contact
    cursor.moveToPosition(position);
    // Get the _ID value
    mContactId = getLong(CONTACT_ID_INDEX);
    // Get the selected LOOKUP KEY
    mContactKey = getString(CONTACT_KEY_INDEX);
    // Create the contact's content Uri
    mContactUri = Contacts.getLookupUri(mContactId, mContactKey);
    /*
     * You can use mContactUri as the content URI for retrieving
     * the details for a contact.
     */
}
```

## 初始化loader

由于使用了CursorLoader获取数据，你必须初始化后台线程和其他的控制异步获取数据的

变量。需要在onActivityCreated()方法中做初始化的工作，这个方法是在Fragment的界面显示之前调用的，相关代码展示如下：

```
public class ContactsFragment extends Fragment implements  
    LoaderManager.LoaderCallbacks<Cursor> {  
    ...  
    // Called just before the Fragment displays its UI  
    @Override  
    public void onActivityCreated(Bundle savedInstanceState) {  
        // Always call the super method first  
        super.onActivityCreated(savedInstanceState);  
        ...  
        // Initializes the loader  
        getLoaderManager().initLoader(0, null, this);  
    }  
}
```

## 实现onCreateLoader()方法

你需要实现onCreateLoader()方法，这个方法是在你调用initLoader后被loader框架直接调用的。

在onCreateLoader()方法中，设置搜索字符串模式。为了让一个字符串符合一个模式，可以插入 "%" 字符代表 0 个或多个字符或者插入 "\_" 代表单独一个字符。例如，模式 "%Jefferson%" 将会匹配“Thomas Jefferson”和“Jefferson Davis”。

这个方法返回一个 CursorLoader 对象。对于内容 URI，则使用了 Contacts.CONTENT\_URI，这个 URI 关联到整个表，例子如下所示：

```
...  
@Override  
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args  
/*  
 * Makes search string into pattern and  
 * stores it in the selection array  
 */  
mSelectionArgs[0] = "%" + mSearchString + "%";  
// Starts the query  
return new CursorLoader(  
    getActivity(),  
    Contacts.CONTENT_URI,  
    PROJECTION,  
    SELECTION,  
    mSelectionArgs,  
    null  
);  
}
```

## 实现 onLoadFinished() 方法和 onLoaderReset() 方法

实现 onLoadFinished() 方法。当联系人 provider 返回查询结果的时候，Android loader 框架会调用 onLoadFinished() 方法。在这个方法中，将查询结果 Cursor 传给 SimpleCursorAdapter，这将会使用这个搜索结果自动更新 ListView。

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor
    // Put the result Cursor in the adapter for the ListView
    mCursorAdapter.swapCursor(cursor);
}
```

当loader框架检测到结果集Cursor包含过时的数据时，它会调用onLoaderReset()。你需要删除SimpleCursorAdapter对已经存在Cursor的引用。如果不这么做的话，loader框架将不会回收Cursor对象，这将会导致内存泄漏。例如：

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // Delete the reference to the existing Cursor
    mCursorAdapter.swapCursor(null);
}
```

你现在已经实现了根据搜索字符串匹配联系人名字，并将获得的结果展示在ListView中的关键部分。用户可以点击选择一个联系人名字，这将会触发一个监听器，在监听器的回调函数中，你可以使用此联系人的数据做进一步的处理。例如，你可以进一步获取此联系人的详情，想要知道如何获取联系人详情，请继续学习下一课——获取联系人详情。

想要了解更多搜索用户界面的知识，请参考API指导——创建搜索界面。

这一课的以下内容展示了在联系人数据库中查找联系人的其他方法。

# 根据特定类型的数据匹配联系人

这种方法可以让你指定你想要匹配的数据类型，根据名字去检索是这种类型的一个具体的例子。但也可以用任何与联系人详情数据相关的数据类型去做查询。例如，您可以检索具有特定邮政编码联系人，在这种情况下，搜索字符串将会去匹配存储在一个邮政编码列中的数据。

为了实现这种类型的检索，首先实现以下的代码，正如之前的内容所展示的：

- 请求读取联系人的权限
- 定义列表和列表项的布局
- 定义显示联系人列表的Fragment
- 定义全局变量
- 初始化Fragment
- 为ListView绑定CursorAdapter数据
- 设置选择联系人的监听器
- 定义Cursor的列索引常量

尽管你现在从不同的表中取数据，检索列的映射顺序是一样的，所以你可以为这个Cursor使用同样的索引常量。

- 定义onItemClick()方法
- 初始化loader
- 实现onLoadFinished()方法和onLoaderReset()方法

为了将搜索字符串匹配特定类型的详情数据并显示结果，以下的步骤展示了你需要做的额外的代码。

## 选择要查询的数据类型和数据库表

为了从特定类型的详情数据中查询，你必须知道的数据类型的自定义MIME类型的值。每一个数据类型拥有唯一的MIME类型值，这个值在ContactsContract.CommonDataKinds的子类中被定义为常量CONTENT\_ITEM\_TYPE，并且与实际的数据类型相关。子类的名字会表明它们的实际数据类型，例如，email数据的子类是ContactsContract.CommonDataKinds.Email，并且email的自定义MIME类型是Email.CONTENT\_ITEM\_TYPE。

在你的搜索中需要使用ContactsContract.Data类，同时所有需要的常量，包括数据映射、选择字句、排序规则都是由这个类定义或继承自此类。

## 定义查询映射

为了定义一个查询映射，请选择一个或者多个由ContactsContract.Data或其子类定义的列名称。Contacts Provider在返回行结果集之前，隐式的连接了ContactsContract.Data表和其他表。例如：

```
@SuppressLint("InlinedApi")
private static final String[] PROJECTION =
{
    /*
     * The detail data row ID. To make a ListView work,

```

```
* this column is required.  
*/  
Data._ID,  
// The primary display name  
Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB  
    Data.DISPLAY_NAME_PRIMARY :  
    Data.DISPLAY_NAME,  
// The contact's _ID, to construct a content URI  
Data.CONTACT_ID  
// The contact's LOOKUP_KEY, to construct a content UR  
Data.LOOKUP_KEY (a permanent link to the contact  
};
```

## 定义查询标准

为了能在特定类型的联系人数据中查询字符串，请按照以下方法构建查询选择子句：

- 列名称包含你要搜索的字符串。这个名字根据数据类型所变化，所以你需要找到与你搜索的数据类型有关的ContactsContract.CommonDataKinds的子类，并从这个子类中选择列名称。例如，想要搜索email地址，需要使用Email.ADDRESS列。
- 搜索字符串本身，请在查询选择子句里使用"?"表示。
- 列名字包含自定义的MIME类型值。这个列名字总是Data.MIMETYPE。
- 自定义MIME类型值的数据类型。如之前描述，这需要使用ContactsContract.CommonDataKinds子类中的CONTENT\_ITEM\_TYPE常量。例如，email数据的MIME类型值是Email.CONTENT\_ITEM\_TYPE。需要在这个常量值的开头和结尾加上单引号，否则的话，provider会把这个值翻译成一个变量而不是一个字符串。你不需要为这个值提供占位符，因为你在使用一个常量而不是用户提供的值。例如：

```
/*  
 * Constructs search criteria from the search string  
 * and email MIME type  
 */  
private static final String SELECTION =  
    /*  
     * Searches for an email address  
     * that matches the search string  
     */  
    Email.ADDRESS + " LIKE ? " + "AND " +  
    /*  
     * Searches for a MIME type that matches  
     * the value of the constant  
     * Email.CONTENT_ITEM_TYPE. Note the  
     * single quotes surrounding Email.CONTENT_ITEM_TYPE.  
     */  
    Data.MIMETYPE + " = '" + Email.CONTENT_ITEM_TYPE + "'";
```

下一步，定义包含选择字符串的变量：

```
String mSearchString;  
String[] mSelectionArgs = { "" };
```

## 实现onCreateLoader()方法

现在，你已经指定了你想要的数据和如何找到这些数据。然后需要在onCreateLoader方法中定义一个查询，使用你的数据映射、查询选择表达式和一个数组作为选择表达式的参数，并从这个方法中返回一个新的CursorLoader对象。而内容URI需要使用Data.CONTENT\_URI，例如：

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args
    // OPTIONAL: Makes search string into pattern
    mSearchString = "%" + mSearchString + "%";
    // Puts the search string into the selection criteria
    mSelectionArgs[0] = mSearchString;
    // Starts the query
    return new CursorLoader(
        getActivity(),
        Data.CONTENT_URI,
        PROJECTION,
        SELECTION,
        mSelectionArgs,
        null
    );
}
```

这段代码片段是基于一种特定类型的联系人详情数据的简单反向查找。如果你的应用关注于某一种特定类型的数据，比如说email地址，并且允许用户获得与此数据相关的联系人名字，这种形式的查询是最好的方法。

# 根据任意类型的数据匹配联系人

根据任意类型的数据获取联系人，如果它们的数据能匹配要搜索的字符串。这些数据包括名字、email地址、邮件地址和电话号码等等。这种搜索结果会比较广泛。例如，如果搜索字符串是"Doe"，搜索任意类型的数据将会返回名字为"Jone Doe"的联系人，也会返回一个住在"Doe Street"的联系人。

为了完成这种类型的查询，就像之前展示的那样，首先需要实现以下代码：

- 请求读取联系人的权限
- 定义列表和列表项的布局
- 定义显示联系人列表的Fragment
- 定义全局变量
- 初始化Fragment
- 为ListView绑定CursorAdapter数据
- 设置选择联系人的监听器
- 定义Cursor的列索引常量

对于这种形式的查询，你需要使用与在“使用特定类型的数据匹配联系人”那一节中相同的表，也可以使用相同的列索引。

- 定义onItemClick()方法
- 初始化loader
- 实现onLoadFinished()方法和onLoaderReset()方法

以下的步骤展示了为了能够根据任意类型的数据去匹配查询字符串并显示结果列表，你需要做的额外代码。

## 去除查询标准

不需要为mSelectionArgs定义查询标准常量SELECTION。这些内容在根据任意类型的数据匹配联系人数据不会用到。

## 实现onCreateLoader()方法

实现onCreateLoader()方法，返回一个新的CursorLoader对象。你不需要把搜索字符串转化成一个搜索模式，因为Contacts Provider会自动做这件事。使用Contacts.CONTENT\_FILTER\_URI作为基础查询URI，并使用Uri.withAppendedPath()方法将搜索字符串添加到基础URI中。使用这个URI会自动触发对任意数据类型的搜索，就像以下例子所示：

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args
{
    /*
     * Appends the search string to the base URI. Always
     * encode search strings to ensure they're in proper
     * format.
     */
    Uri contentUri = Uri.withAppendedPath(
        Contacts.CONTENT_FILTER_URI,
        Uri.encode(mSearchString));
```

```
// Starts the query
return new CursorLoader(
    getActivity(),
    contentUri,
    PROJECTION,
    null,
    null,
    null
);
}
```

这段代码片段，是想要在Contacts Provider中建立广泛搜索类型的应用的基础部分。这种方法对那些想要实现与通讯录应用联系人列表中相似的搜索功能的应用，会很有帮助。

编写: [spencer198711](#) - 校对:

原文:

# 获取联系人详情

这一课展示了如何取得一个联系人的详细信息，比如email地址、电话号码等。当使用者去获取联系人信息的时候，这些信息正是他们所查找的。你可以给他们关于一个联系人的所有信息，或者仅仅显示一个特定的数据类型，比如email地址。

这一课假设你已经获取到了一个用户所选取的联系人的[ContactsContract.Contacts](#)数据项。在[获取联系人名字](#)那一课展示了如何获取联系人列表。

## 获取联系人的所有详细信息

为了取得一个联系人的所有详情，需要去搜索[ContactsContract.Data](#)表，寻找包含联系人[LOOKUP\\_KEY](#)的任意一行。列名称可以从[ContactsContract.Data](#)表中查到。因为[ContactsProvider](#)隐示地连接了[ContactsContract.Contacts](#)表和[ContactsContract.Data](#)表。关于[LOOKUP\\_KEY](#)列，在[获取联系人名字](#)那一课有详细的描述。

提示：取得一个联系人的所有信息会降低设备的性能，因为这需要获取[ContactsContract.Data](#)表的所有列，在你使用这种方法之前，请认真考虑对性能影响。

## 请求权限

为了能够读联系人数据库，你的应用必须拥有[READ\\_CONTACTS](#)权限，为了请求这个权限，需要在清单文件里边添加以下子节点

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

## 设置查询映射

根据一行数据的数据类型的不同，它可能会使用很多列或者只使用几列。另外，数据会根据不同的数据类型而在不同的列中。为了确保能够获取所有可能的数据类型的所有可能的数据列，需要在查询映射中添加所有列的名字。如果你要把Cursor绑定到ListView，记得永远都需要获取Data.\_ID，否则的话，界面绑定就不会起作用。同时你也需要获取[Data.MIMETYPE](#)列，这样你才能识别你获取到的每一行数据的数据类型。例如：

```
private static final String PROJECTION =
{
    Data._ID,
    Data.MIMETYPE,
    Data.DATA1,
    Data.DATA2,
    Data.DATA3,
    Data.DATA4,
    Data.DATA5,
    Data.DATA6,
    Data.DATA7,
    Data.DATA8,
    Data.DATA9,
    Data.DATA10,
    Data.DATA11,
    Data.DATA12,
    Data.DATA13,
    Data.DATA14,
    Data.DATA15
};
```

这个查询映射使用了[ContactsContract.Data](#)类中定义的列名字，获取了[ContactsContract.Data](#)

表的所有数据列。

你也可以使用由ContactsContract.Data或其子类定义的列常量去设置查询映射。需要注意的是，从SYNC1到SYNC4的数据列是sync adapter同步数据所使用的，它们的值对我们没有意义。

## 定义查询标准

为查询选择子句定义一个常量，一个包含查询选择参数的数组，以及一个保存查询选择值的变量。使用数据列LOOKUP\_KEY去查找这个联系人。例如：

```
// Defines the selection clause
private static final String SELECTION = Data.LOOKUP_KEY + " =
// Defines the array to hold the search criteria
private String[] mSelectionArgs = { "" };
/*
 * Defines a variable to contain the selection value. Once you
 * have the Cursor from the Contacts table, and you've selected
 * the desired row, move the row's LOOKUP_KEY value into this
 * variable.
 */
private String mLookupKey;
```

在查询选择表达式中使用“?”占位符，确保了搜索是由绑定生成而不是由SQL编译生成。这种方法消除了恶意SQL注入的可能性。

## 定义排序顺序

定义在查询结果Cursor中希望的排序顺序。为了让特定类型的数据列聚集在一起，需要按照Data.MIMETYPE去排序。这种形式的查询排序参数让所有的email信息排在一起，所有的电话信息排在一起等等。例如：

```
/*
 * Defines a string that specifies a sort order of MIME type
 */
private static final String SORT_ORDER = Data.MIMETYPE;
```

一些数据类型不使用子类型，所以你不能按照子类型来排序。作为替代方法，你不得不遍历返回的cursor，去判定当前行的数据类型，为那些使用子类型的数据行保存数据。当读取完cursor后，你可以根据子类型去排序每一个数据类型，然后显示结果。

## 初始化查询loader

永远在后台线程中去从Contacts Provider中获取数据(或者其他content provider)。使用loader框架中的LoaderManager类和LoaderManager.LoaderCallbacks在后台去做获取数据的工作。

当你已经准备好了去获取数据行，需要通过调用initLoader()方法去初始化loader框架。传递

一个Integer类型的标示符给initLoader()方法，这个标示符会传递给 LoaderManager.LoaderCallbacks的相关方法。当在一个应用中使用多个loader时，这个标示符能够帮助你区分它们。

以下的代码片段展示了如何初始化loader框架：

```
public class DetailsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor> {
    ...
    // Defines a constant that identifies the loader
    DETAILS_QUERY_ID = 0;
    ...
    /*
     * Invoked when the parent Activity is instantiated
     * and the Fragment's UI is ready. Put final initialization
     * steps here.
     */
    @Override
    onActivityCreated(Bundle savedInstanceState) {
        ...
        // Initializes the loader framework
        getLoaderManager().initLoader(DETAILS_QUERY_ID, null, this
    }
}
```

## 实现onCreateLoader方法

实现onCreateLoader方法。loader框架会在你调用initLoader()方法后立即调用onCreateLoader方法，并返回一个CursorLoader对象。由于你是要搜索ContactsContract.Data表，所以需要使用常量Data.CONTENT\_URI作为内容URI。例如：

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args
    // Choose the proper action
    switch (loaderId) {
        case DETAILS_QUERY_ID:
            // Assigns the selection parameter
            mSelectionArgs[0] = mLookupKey;
            // Starts the query
            CursorLoader mLoader =
                new CursorLoader(
                    getActivity(),
                    Data.CONTENT_URI,
                    PROJECTION,
                    SELECTION,
                    mSelectionArgs,
                    SORT_ORDER
                );
            ...
    }
}
```

## 实现onLoadFinished()方法和onLoaderReset()方法

实现onLoadFinished()方法。当Contacts Provider返回查询结果的时候，loader框架会调用

onLoadFinished()方法。例如：

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {  
    switch (loader.getId()) {  
        case DETAILS_QUERY_ID:  
            /*  
             * Process the resulting Cursor here.  
             */  
            break;  
        ...  
    }  
}
```

当loader框架检测到结果集Cursor所对应的数据已经发生变化的时候，会调用onLoaderReset()方法。这时，需要通过把Cursor设置为null来移除对已经存在Cursor对象的引用，如果不这样做的话，loader框架就不会销毁旧的Cursor对象，就回发生内存泄漏。例如：

```
@Override  
public void onLoaderReset(Loader<Cursor> loader) {  
    switch (loader.getId()) {  
        case DETAILS_QUERY_ID:  
            /*  
             * If you have current references to the Cursor,  
             * remove them here.  
             */  
            break;  
    }  
}
```

# 获取联系人的特定类型的信息

获取联系人的特定类型的信息，例如所有的email信息，跟获取联系人的所有详细信息类似。你仅需要修改在[获取联系人的所有详细信息](#)列举的以下部分代码：

- 映射

修改查询映射使得能够针对特定类型的数据查询。同时需要修改查询映射的列名称，要根据查询的数据类型使用在ContactsContract.CommonDataKinds的子类中定义的常量。

- 查询选择

根据查询的数据类型，修改查询选择子句去搜索特定类型的MIMETYPE值。

- 排序顺序

由于仅仅搜索一种类型的详细数据，所以不需要根据返回的Cursor的Data.MIMETYPE去将结果分组。

这些修改将会在以下章节中详细描述。

## 设置查询映射

使用ContactsContract.CommonDataKinds的特定类型子类所定义的列名称常量，定义你想要获取的数据列。如果你打算把结果Cursor绑定到ListView，确保要获取\_ID列。例如，为了获取email数据，需要定义以下数据映射：

```
private static final String[] PROJECTION =
{
    Email._ID,
    Email.ADDRESS,
    Email.TYPE,
    Email.LABEL
};
```

需要注意的是，这个查询映射使用了在ContactsContract.CommonDataKinds.Email类中定义的列名称，替代了ContactsContract.Data类中定义的列名称。使用email类型特定的列名称使得代码更具可读性。

在查询投影中，你还可以使用其他ContactsContract.CommonDataKinds子类所定义的数据列。

## 定义查询标准

根据你想要找的特定联系人的LOOKUP\_KEY和联系人详细信息的Data.MIMETYPE定义一个搜索表达式，去获取数据。把MIMETYPE的值从头到尾用单引号括住，否则的话，content provider将会把这个常量翻译成变量名，而不是翻译成字符串值。不需要为这个值使用占位符，因为你使用的是常量，而不是用户提供的值。例如：

```
/*
 * Defines the selection clause. Search for a lookup key
```

```
* and the Email MIME type
*/
private static final String SELECTION =
    Data.LOOKUP_KEY + " = ?" +
    " AND " +
    Data.MIMETYPE + " = " +
    "\"" + Email.CONTENT_ITEM_TYPE + "\"";
// Defines the array to hold the search criteria
private String[] mSelectionArgs = { "" };
```

## 定义排序规则

为查询返回的[Cursor](#)定义一个排序规则。由于是为了获取特定类型的数据，请忽略在[MIMETYPE](#)上的排序。作为替代，如果查询的详细数据类型包含子类型，可以根据这个子类型去排序。例如，对于email数据你可以根据[Email.TYPE](#)排序：

```
private static final String SORT_ORDER = Email.TYPE + " ASC";
```

编写: [spencer198711](#) - 校对:

原文:

# 使用Intent修改联系人信息

这一课向你展示了如何使用[Intent](#)去插入一个新的联系人或者修改联系人的数据。我们不是直接访问Contacts Provider，而是通过Intent去启动Contacts应用的适当的[Activity](#)。对于这一课中描述的数据修改行为，如果你向Intent发送扩展的数据，它会自动填充进启动的Activity页面中。

使用Intent去插入或者更新一个联系人是比较推荐的修改Contacts Provider的做法。原因如下：

- 节省了你去开发自己的UI和代码的时间和精力
- 避免了由于不符合Contacts Provider规则的修改，从而引入错误
- 减少你的应用所需要申请的权限数量。你的应用不需要去申请写Contacts Provider的权限，因为它把修改行为委托给了已经拥有这个权限的Contacts应用。

# 使用Intent插入新的联系人

当你的应用接收到新的数据时，你通常会允许用户去插入一个新的联系人。例如，一个餐馆评论应用可以允许用户在评论餐馆的时候，把这个餐馆添加为一个联系人。可以使用Intent去做这个任务，使用你所拥有的尽可能多的数据去创建对应的Intent，然后发送这个Intent到Contacts应用。

使用Contacts应用去插入一个联系人将会向Contacts Provider中的[ContactsContract.RawContact](#)表中插入一个原始联系人。必要的情况下，在创建原始联系人的时候，Contacts应用将会提示用户选择账户类型和要使用的账户。如果联系人已经存在，Contacts应用也会告知用户。用户将会有取消插入的选项，在这种情况下不会有联系人创建。想要知道更多关于原始联系人的信息，请参阅[Contacts Provider](#)的API指导。

开始需要创建一个拥有Intents.Insert.ACTION的Intent对象，并设置其MIME类型为[RawContacts.CONTENT\\_TYPE](#)。例如：

```
...
// Creates a new Intent to insert a contact
Intent intent = new Intent(Intents.Insert.ACTION);
// Sets the MIME type to match the Contacts Provider
intent.setType(ContactsContract.RawContacts.CONTENT_TYPE);
```

如果你已经获得了此联系人的详细信息，比如说电话号码或者email地址，你可以把它们作为扩展数据添加到Intent中。对于键值，需要使用[Intents.Insert](#)中对应的常量。Contacts应用将会在插入界面显示这些数据，以便用户作进一步的数据编辑和数据添加。

```
/* Assumes EditText fields in your UI contain an email address
 * and a phone number.
 */
private EditText mEmailAddress = (EditText) findViewById(R.id.email);
private EditText mPhoneNumber = (EditText) findViewById(R.id.phone)
...
/*
 * Inserts new data into the Intent. This data is passed to the
 * contacts app's Insert screen
*/
// Inserts an email address
intent.putExtra(Intents.Insert.EMAIL, mEmailAddress.getText())
/*
 * In this example, sets the email type to be a work email.
 * You can set other email types as necessary.
*/
    .putExtra(Intents.Insert.EMAIL_TYPE, CommonDataKinds.Email.T
// Inserts a phone number
    .putExtra(Intents.Insert.PHONE, mPhoneNumber.getText())
/*
 * In this example, sets the phone type to be a work phone.
 * You can set other phone types as necessary.
*/
    .putExtra(Intents.Insert.PHONE_TYPE, Phone.TYPE_WORK);
```

一旦你已经创建好Intent，调用startActivity()发送到Contacts应用。

```
/* Sends the Intent
 */
startActivity(intent);
```

这个调用将会打开Contacts应用的界面，并允许用户进入一个新的联系人。这个联系人可用的账户类型和账户名字列在屏幕的上方。一旦用户输入数据并点击确定。Contacts应用的联系人列表则会显示出来。用户可以点击Back键返回。

# 使用Intent编辑已经存在的联系人

如果用户已经选择了一个感兴趣的联系人，使用Intent去编辑这个已存在的联系人会很有用。例如，一个用来查找拥有邮政地址但是缺少邮政编码的联系人的应用，可以给用户查找邮政编码的选项，然后把它添加到这个联系人中。

使用Intent编辑已经存在的联系人，同插入一个联系人的步骤类似。创建一个在[使用Intent插入新的联系人](#)那一节课描述的Intent，但是需要给这个Intent添加对应联系人的[Contacts.CONTENT\\_LOOKUP\\_URI](#)和MIME类型[Contacts.CONTENT\\_ITEM\\_TYPE](#)。如果你想要使用已经拥有的详情信息编辑这个联系人，你需要把这些数据放到Intent的扩展数据中。同时注意有些列是不能使用Intent编辑的，这些不可编辑的列在[ContactsContract.Contacts](#)的API参考文档中的摘要部分的“Update”标题下有列出。

最后发送这个Intent。作为回应，Contacts应用会显示一个编辑界面。当用户编辑完成并保存，Contacts应用会显示一个联系人列表。当用户点击Back，就回显示你自己的应用。

## 创建Intent

为了能够编辑一个联系人，需要调用Intent(action)去创建一个拥有ACTION\_EDIT行为的Intent，调用setDataAndType()去设置这个Intent要编辑的联系人的Contacts.CONTENT\_LOOKUP\_URI和MIME类型Contacts.CONTENT\_ITEM\_TYPE。因为对setType()的调用会重写Intent的当前data数据，你必须同时设置data数据和MIME类型。

为了得到联系人的Contacts.CONTENT\_LOOKUP\_URI，需要调用Contacts.getLookupUri(id, lookupkey)方法，并用这个联系人的Contacts.\_ID和Contacts.LOOKUP\_KEY作为参数。

以下的代码片段展示了如何创建这个Intent：

```
// The Cursor that contains the Contact row
public Cursor mCursor;
// The index of the lookup key column in the cursor
public int mLookupKeyIndex;
// The index of the contact's _ID value
public int mIdIndex;
// The lookup key from the Cursor
public String mCurrentLookupKey;
// The _ID value from the Cursor
public long mCurrentId;
// A content URI pointing to the contact
Uri mSelectedContactUri;
...
/*
 * Once the user has selected a contact to edit,
 * this gets the contact's lookup key and _ID values from the
 * cursor and creates the necessary URI.
 */
// Gets the lookup key column index
mLookupKeyIndex = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Gets the lookup key value
mCurrentLookupKey = mCursor.getString(mLookupKeyIndex);
// Gets the _ID column index
mIdIndex = mCursor.getColumnIndex(Contacts._ID);
mCurrentId = mCursor.getLong(mIdIndex);
```

```
mSelectedContactUri =
    Contacts.getLookupUri(mCurrentId, mCurrentLookupKey);
...
// Creates a new Intent to edit a contact
Intent editIntent = new Intent(Intent.ACTION_EDIT);
/*
 * Sets the contact URI to edit, and the data type that the
 * Intent must match
*/
editIntent.setDataAndType(mSelectedContactUri, Contacts.CONTENT
```

## 添加导航标志

在android 4.0 (API版本14) 之后, Contacts应用中的一个问题会导致错误的页面导航。当你的应用发送一个编辑联系人的Intent到Contacts应用, 然后用户编辑并保存这个联系人, 当用户点击Back键的时候会看到联系人列表页面, 为了能够导航回到你自己的应用, 用户不得不点击最近使用的应用, 然后选择你的应用, 才能回到之前的页面。

要在android 4.0.3 (API版本15) 及以后的版本解决此问题, 需要添加 `finishActivityOnSaveCompleted` 扩展数据参数到这个Intent, 并将它的值设置为true。Android 4.0之前的版本也能够接受这个参数, 但是不起作用。为了设置扩展数据, 请按照以下方式去做:

```
// Sets the special extended data for navigation
editIntent.putExtra("finishActivityOnSaveCompleted", true);
```

## 添加其他的扩展数据

对Intent添加额外的扩展数据, 需要调用`putExtra()`。可以为常见的联系人数据字段添加扩展数据, 这些常见字段的key值可以从Intents.InsertAPI参考文档中查到。记住 `ContactsContract.Contacts` 表中的有些列是不能编辑的, 这列在 `ContactsContract.Contacts` 的 API参考文档中的摘要部分的“Update”标题下有列出。

## 发送Intent

最后, 发送你已经构建好的Intent。例如:

```
// Sends the Intent
startActivity(editIntent);
```

## 使用Intent让用户去选择是插入还是编辑联系人

你可以通过发送ACTION\_INSERT\_OR\_EDIT行为的Intent，让用户去选择是插入联系人还是编辑已有的联系人。例如，一个email客户端应用会允许用户添加一个收件地址到新的联系人，或者仅仅作为额外的邮件地址添加到已有的联系人。需要为这个Intent设置MIME类型Contacts.CONTENT\_ITEM\_TYPE，但是不需要设置数据URI。

当你发送这个Intent后，Contacts应用会展示一个联系人列表，用户可以选择是插入一个新的联系人还是挑选一个存在的联系人去编辑。任何你添加到Intent中得扩展数据字段都会填充在界面上。你可以使用任何在Intents.InsertAPI参考文档中制定的的key值。以下的代码片段展示了如何构建和发送这个Intent：

```
// Creates a new Intent to insert or edit a contact
Intent intentInsertEdit = new Intent(Intent.ACTION_INSERT_OR_EDIT);
// Sets the MIME type
intentInsertEdit.setType(Contacts.CONTENT_ITEM_TYPE);
// Add code here to insert extended data, if desired
...
// Sends the Intent with an request ID
startActivity(intentInsertEdit);
```

编写: [spencer198711](#) - 校对:

原文:

# 显示快速联系人徽章(头像)

这一课展示了如何在你的应用界面上添加一个[QuickContactBadge](#), 以及如何为它绑定数据。 QuickContactBadge是一个在初始情况下显示联系人缩略图头像的widget。尽管你可以使用任何[Bitmap](#)作为缩略图头像, 但是通常你会使用从联系人照片缩略图中解码出来的Bitmap。

这个小的图片作为一个控件, 当用户点击它时, QuickContactBadge会伸展成一个包含以下内容的对话框:

- 一个大的联系人头像  
与这个联系人关联的大的头像, 如果此人没有设置头像, 则显示默认的头像。
- 应用程序图标

根据联系人详情数据, 显示每一个能够被手机中的应用所处理的数据的图标。例如, 如果联系人的数据包含一个或多个email地址, 就会显示email应用的图标。当用户点击这个图标的时候, 这个联系人所有的email地址都会显示出来。当用户点击其中一个email地址时, email应用将会显示一个界面, 让用户为这个选择的地址去编辑邮件。

QuickContactBadge视图提供了对联系人数据的即时访问, 是作为一种与联系人沟通的快捷方式。用户不用查询一个联系人, 查找并复制信息, 然后把信息粘贴到合适的应用中。他们可以点击QuickContactBadge, 选择他们想要的沟通方式, 然后直接把信息发送到对应的应用中。

## 添加一个**QuickContactBadge**视图

为了添加一个**QuickContactBadge**视图，需要在你的布局文件中插入一个**QuickContactBadge**。例如：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    ...  
    <QuickContactBadge  
        android:id="@+id/quickbadge"  
        android:layout_height="wrap_content"  
        android:layout_width="wrap_content"  
        android:scaleType="centerCrop"/>  
    ...  
</RelativeLayout>
```

## 获取数据库数据

为了能在QuickContactBadge中显示联系人，你需要这个联系人的内容URI和显示头像的Bitmap。你可以从Contacts Provider中获取到的相关数据列中生成这两个数据。需要指定这些列作为查询映射去把数据加载到Cursor中。

对于Android 3.0 (API版本为11) 以及以后的版本，需要在查询映射中添加以下列：

\*[Contacts\\_ID](#)

\*[Contacts\\_LOOKUP\\_KEY](#)

\*[Contacts\\_PHOTO\\_THUMBNAIL\\_URI](#)

对于Android 2.3.3 (API版本为10) 以及之前的版本，则使用以下列：

\*[Contacts\\_ID](#)

\*[Contacts\\_LOOKUP\\_KEY](#)

这一课的剩余部分假设你已经获取到了包含这些以及其他你可能选择的数据列的Cursor对象。想要学习如何获取这些列对象的Cursor，请参阅课程[获取联系人列表](#)。

## 设置联系人URI和缩略图象

一旦你已经拥有了所需的数据列，你就可以为QuickContactBadge视图绑定数据了。

### 设置联系人URI

为了设置联系人URI，需要调用[getLookupUri\(id.lookupKey\)](#)去获取[CONTENT\\_LOOKUP\\_URI](#)，然后调用[assignContactUri\(\)](#)去为QuickContactBadge设置对应的联系人。例如：

```
// The Cursor that contains contact rows
Cursor mCursor;
// The index of the _ID column in the Cursor
int mIdColumn;
// The index of the LOOKUP_KEY column in the Cursor
int mLookupKeyColumn;
// A content URI for the desired contact
Uri mContactUri;
// A handle to the QuickContactBadge view
QuickContactBadge mBadge;
...
mBadge = (QuickContactBadge) findViewById(R.id.quickbadge);
/*
 * Insert code here to move to the desired cursor row
 */
// Gets the _ID column index
mIdColumn = mCursor.getColumnIndex(Contacts._ID);
// Gets the LOOKUP_KEY index
mLookupKeyColumn = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Gets a content URI for the contact
mContactUri =
    Contacts.getLookupUri(
        mCursor.getLong(mIdColumn),
        mCursor.getString(mLookupKeyColumn)
    );
mBadge.assignContactUri(mContactUri);
```

当用户点击QuickContactBadge图标的时候，这个联系人的详细信息将会自动展现在对话框中。

### 设置联系人照片的缩略图

为QuickContactBadge设置联系人URI并不会自动加载联系人的缩略图照片。为了加载联系人照片，需要从联系人的Cursor对象的一行数据中获取照片的URI，使用这个URI去打开包含压缩的缩略图照片的文件，并把这个文件读到Bitmap对象中。

[PHOTO\_THUMBNAIL\_URI]这一列在Android 3.0之前的版本是不存在的，对于这些版本，你必须从[Contacts.Photo]表中获取照片的URI。

首先，为包含Contacts.\_ID和Contacts.LOOKUP\_KEY的Cursor数据列设置对应的变量，这

在之前已经有描述：

```
// The column in which to find the thumbnail ID
int mThumbnailColumn;
/*
 * The thumbnail URI, expressed as a String.
 * Contacts Provider stores URIs as String values.
 */
String mThumbnailUri;
...
/*
 * Gets the photo thumbnail column index if
 * platform version >= Honeycomb
 */
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    mThumbnailColumn =
        mCursor.getColumnIndex(Contacts.PHOTO_THUMBNAIL_URI)
    // Otherwise, sets the thumbnail column to the _ID column
} else {
    mThumbnailColumn = mIdColumn;
}
/*
 * Assuming the current Cursor position is the contact you want
 * gets the thumbnail ID
*/
mThumbnailUri = mCursor.getString(mThumbnailColumn);
...
```

首先定义一个方法，使用这个联系人的照片有关的数据和目标视图的大小作为参数，返回一个大小合适的Bitmap对象的缩略图。先构建一个指向这个缩略图的URI：

```
/**
 * Load a contact photo thumbnail and return it as a Bitmap,
 * resizing the image to the provided image dimensions as needed
 * @param photoData photo ID Prior to Honeycomb, the contact's
 * For Honeycomb and later, the value of PHOTO_THUMBNAIL_URI.
 * @return A thumbnail Bitmap, sized to the provided width and
 * Returns null if the thumbnail is not found.
 */
private Bitmap loadContactPhotoThumbnail(String photoData) {
    // Creates an asset file descriptor for the thumbnail file
    AssetFileDescriptor afd = null;
    // try-catch block for file not found
    try {
        // Creates a holder for the URI.
        Uri thumbUri;
        // If Android 3.0 or later
        if (Build.VERSION.SDK_INT
            >=
            Build.VERSION_CODES.HONEYCOMB) {
            // Sets the URI from the incoming PHOTO_THUMBNAIL_
            thumbUri = Uri.parse(photoData);
        } else {
```

```

        // Prior to Android 3.0, constructs a photo Uri using
        /*
         * Creates a contact URI from the Contacts content
         * incoming photoData (_ID)
         */
        final Uri contactUri = Uri.withAppendedPath(
                Contacts.CONTENT_URI, photoData);
        /*
         * Creates a photo URI by appending the content UR
         * Contacts.Photo.
         */
        thumbUri =
                Uri.withAppendedPath(
                        contactUri, Photo.CONTENT_DIRECTORY
                }

/*
 * Retrieves an AssetFileDescriptor object for the thumbnail
 * URI
 * using ContentResolver.openAssetFileDescriptor
 */
afd = getActivity().getContentResolver().
        openAssetFileDescriptor(thumbUri, "r");
/*
 * Gets a file descriptor from the asset file descriptor.
 * This object can be used across processes.
 */
FileDescriptor fileDescriptor = afd.getFileDescriptor();
// Decode the photo file and return the result as a Bitmap
// If the file descriptor is valid
if (fileDescriptor != null) {
    // Decodes the bitmap
    return BitmapFactory.decodeFileDescriptor(
            fileDescriptor, null, null);
}
// If the file isn't found
} catch (FileNotFoundException e) {
    /*
     * Handle file not found errors
     */
}
// In all cases, close the asset file descriptor
} finally {
    if (afd != null) {
        try {
            afd.close();
        } catch (IOException e) {}
    }
}
return null;
}

```

然后在你的代码中调用loadContactPhotoThumbnail()去获取缩略图Bitmap对象，使用获取的Bitmap对象去设置QuickContactBadge头像缩略图。

```
...
/*
 * Decodes the thumbnail file to a Bitmap.
 */
Bitmap mThumbnail =
    loadContactPhotoThumbnail(mThumbnailUri);
/*
 * Sets the image in the QuickContactBadge
 * QuickContactBadge inherits from ImageView, so
 */
mBadge.setImageBitmap(mThumbnail);
```

# 把QuickContactBadge添加到ListView

QuickContactBadge对于一个展示联系人列表的ListView来说是一个非常有用的功能。使用QuickContactBadge去为每一个联系人显示一个缩略图，当用户点击这个缩略图时，QuickContactBadge对话框将会显示。

## 为ListView添加QuickContactBadge

首先，在你的列表项布局文件中添加QuickContactBadge视图元素。例如，如果你想为获取到的每一个联系人显示QuickContactBadge和名字，把以下的XML内容放到对应的布局文件中：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <QuickContactBadge
        android:id="@+id/quickcontact"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:scaleType="centerCrop"/>
    <TextView android:id="@+id/displayname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/quickcontact"
        android:gravity="center_vertical"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"/>
</RelativeLayout>
```

在以下的章节中，这个文件被称为contact\_item\_layout.xml

## 设置自定义的CursorAdapter

为了能够绑定CursorAdapter到一个包含QuickContactBadge的ListView中，需要一个继承自 CursorAdapter 的自定义 adapter。这种方式允许你在绑定数据到QuickContactBadge之前对 Cursor 中的数据进行处理，同时也能绑定多个Cursor中得列到QuickContactBadge。而这些操作使用普通的CursorAdapter是不能完成的。

你定义的CursorAdapter的子类必须重写以下方法：

- CursorAdapter.newView() 填充一个View对象去持有列表项布局。在重写这个方法的过程中，需要保存这个布局的子View的引用，包括QuickContactBadge的引用。通过采用这种方法，避免了每次在填充新的布局的时候都去获取子View的引用。你必须重写这个方法以便能够获取每个子View的引用，这种方法允许你控制这些控件在CursorAdapter.bindView()方法中的数据绑定过程。
- CursorAdapter.bindView() 将数据从当前Cursor行绑定到列表项布局的子View对象中。必须重写这个方法以便能够将联系人的URI和缩略图信息绑定到 QuickContactBadge。这个方法的默认实现仅仅允许在数据行和View之间的一对一的映射。

以下的代码片段包含了自定义CursorAdapter子类的一个例子。

## 定义自定义的列表Adapter

定义CursorAdapter的子类包括编写这个类的构造方法，以及重写newView()和bindView():

```
private class ContactsAdapter extends CursorAdapter {
    private LayoutInflater mInflater;
    ...
    public ContactsAdapter(Context context) {
        super(context, null, 0);

        /*
         * Gets an inflator that can instantiate
         * the ListView layout from the file.
         */
        mInflater = LayoutInflater.from(context);
        ...
    }
    ...
    /**
     * Defines a class that hold resource IDs of each item lay
     * row to prevent having to look them up each time data is
     * bound to a row.
     */
    private class ViewHolder {
        TextView displayname;
        QuickContactBadge quickcontact;
    }
    ...
    @Override
    public View newView(
            Context context,
            Cursor cursor,
            ViewGroup viewGroup) {
        /* Inflates the item layout. Stores resource IDs in a
         * in a ViewHolder class to prevent having to look
         * them up each time bindView() is called.
         */
        final View itemView =
            mInflater.inflate(
                R.layout.contact_list_layout,
                viewGroup,
                false
            );
        final ViewHolder holder = new ViewHolder();
        holder.displayname =
            (TextView) view.findViewById(R.id.displayname)
        holder.quickcontact =
            (QuickContactBadge)
                view.findViewById(R.id.quickcontact);
        view.setTag(holder);
        return view;
    }
    ...
    @Override
```

```
public void bindView(
    View view,
    Context context,
    Cursor cursor) {
    final ViewHolder holder = (ViewHolder) view.getTag();
    final String photoData =
        cursor.getString(mPhotoDataIndex);
    final String displayName =
        cursor.getString(mDisplayNameIndex);

    ...
    // Sets the display name in the layout
    holder.displayname = cursor.getString(mDisplayNameIndex);
    ...
    /*
     * Generates a contact URI for the QuickContactBadge.
     */
    final Uri contactUri = Contacts.getLookupUri(
        cursor.getLong(mIdIndex),
        cursor.getString(mLookupKeyIndex));
    holder.quickcontact.assignContactUri(contactUri);
    String photoData = cursor.getString(mPhotoDataIndex);
    /*
     * Decodes the thumbnail file to a Bitmap.
     * The method loadContactPhotoThumbnail() is defined
     * in the section "Set the Contact URI and Thumbnail"
     */
    Bitmap thumbnailBitmap =
        loadContactPhotoThumbnail(photoData);
    /*
     * Sets the image in the QuickContactBadge
     * QuickContactBadge inherits from ImageView
     */
    holder.quickcontact.setImageBitmap(thumbnailBitmap);
}
```

## 设置查询变量

在你的代码中设置相关变量，包括必须要查询的数据列的Cursor投影。

以下的代码片段使用了方法loadContactPhotoThumbnail()，这个方法是在[设置联系人URI和缩略图象](#)那一节中定义的。

例如：

```
public class ContactsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor> {

    ...
    // Defines a ListView
    private ListView mListview;
    // Defines a ContactsAdapter
```

```
private ContactsAdapter mAdapter;
...
// Defines a Cursor to contain the retrieved data
private Cursor mCursor;
/*
 * Defines a projection based on platform version. This ensure
 * that you retrieve the correct columns.
 */
private static final String[] PROJECTION =
{
    Contacts._ID,
    Contacts.LOOKUP_KEY,
    (Build.VERSION.SDK_INT >=
        Build.VERSION_CODES.HONEYCOMB) ?
        Contacts.DISPLAY_NAME_PRIMARY :
        Contacts.DISPLAY_NAME
    (Build.VERSION.SDK_INT >=
        Build.VERSION_CODES.HONEYCOMB) ?
        Contacts.PHOTO_THUMBNAIL_ID :
    /*
     * Although it's not necessary to include
     * column twice, this keeps the number of
     * columns the same regardless of version
     */
    Contacts.ID
    ...
};

/*
 * As a shortcut, defines constants for the
 * column indexes in the Cursor. The index is
 * 0-based and always matches the column order
 * in the projection.
*/
// Column index of the _ID column
private int mIdIndex = 0;
// Column index of the LOOKUP_KEY column
private int mLookupKeyIndex = 1;
// Column index of the display name column
private int mDisplayNameIndex = 3;
/*
 * Column index of the photo data column.
 * It's PHOTO_THUMBNAIL_URI for Honeycomb and later,
 * and _ID for previous versions.
*/
private int mPhotoDataIndex =
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB
    3 :
    0;
...
```

## 设置ListView的参数

在[Fragment.onCreate\(\)](#)方法中，实例化自定义的adapter对象，获得一个对ListView的引用。

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        /*
         * Instantiates the subclass of
         * CursorAdapter
         */
        ContactsAdapter mContactsAdapter =
            new ContactsAdapter(getActivity());
        /*
         * Gets a handle to the ListView in the file
         * contact_list_layout.xml
         */
        mListview = (ListView) findViewById(R.layout.contact_list_
        ...
    }
    ...
}
```

在onActivityCreated()方法中，将ContactsAdapter绑定到ListView。

```
@Override
    public void onActivityCreated(Bundle savedInstanceState) {
        ...
        // Sets up the adapter for the ListView
        mListview.setAdapter(mAdapter);
        ...
    }
    ...
}
```

你通常会在onLoadFinished()方法中获取一个包含联系人数据的Cursor对象，这个时候你需要调用swapCursor()，这个方法会把Cursor中的数据绑定到ListView。这将会为联系人列表中的每一项都显示一个QuickContactBadge。

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    // When the loader has completed, swap the cursor into the
    mContactsAdapter.swapCursor(cursor);
}
```

当你使用CursorAdapter或其子类中将Cursor中的数据绑定到ListView，并且使用了CursorLoader去加载Cursor数据，记得要在onLoaderReset()方法的实现中清理对Cursor对象的引用。例如：

```
@Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Removes remaining reference to the previous Cursor
        mContactsAdapter.swapCursor(null);
    }
```

编写:[penkzhou](#)

校对:

# 位置信息

移动应用一个独特的特征就是对地址的感知。移动用户把他们的设备带到各个地方，这时将位置的感知能力添加到你的应用里可以让用户有更多的地理位置相关的体验。最新的位置服务API集成在Google Play服务里面，内置有自动位置记录，地理围栏，用户活动识别。这个API让Android平台的位置API优势更加突出了。

这个课程教你如何在你的应用里使用位置服务，获取周期性的位置更新，查询地址，创建并监视地理围栏以及探测用户的活动。这个课程包括示例应用和代码片段，你可以使用它们让你的应用拥有位置感知能力。

注意: 因为这个课程基于Google Play services client library，在使用这些示例应用和代码段之前确保你安装了最新版本的Google Play services client library。要想学习如何安装最新版的client library，请参考安装Google Play services 向导。

# Lessons

- [获取当前的位置](#)

学习如何获取用户当前的位置。

- [接收位置更新](#)

学习如何请求和接收周期性的位置更新。

- [显示一个地点位置](#)

学习如何将一个位置的经纬度转化成一个地址（反向 geocoding）。

- [创建和监视Geofences](#)

学习如何将一个或多个地理区域定义成一个兴趣位置集合，称为地理围栏。学习如何探测用户靠近或者进入地理围栏事件。

- [识别用户当前的活动](#)

学习如何识别用户当前的活动，比如步行，骑行，或者驾车行驶。学习如何使用这些信息去更改你应用的位置策略。

- [使用Mock Locations测试你的应用](#)

学习如何使用虚拟的位置数据来测试一个位置应用。在mock模式里面，位置服务将会发送一些虚拟的位置数据。

编写:[penkzhou](#)

校对:

# 获取当前位置

位置服务会自动持有用户当前的位置信息，你的应用在需要位置的时候获取一下即可。位置的精确度基于你所请求的位置权限以及当前设备已经激活的位置传感器。

位置服务通过一个[LocationClient](#)（位置服务类LocationClient的一个实例）将当前的位置发送给你的应用。关于位置信息的所有请求都是通过这个类发送。

**注意:** 在开始这个课程之前，确定你的开发环境和测试设备处于正常可用状态。要了解更多，请阅读Google Play services 引导。

## 确定应用的权限

使用位置服务的应用必须用户位置权限。Android拥有两种位置权限：[ACCESS\\_COARSE\\_LOCATION](#) 和 [ACCESS\\_FINE\\_LOCATION](#)。选择不同的权限决定你的应用最后获取的位置信息的精度。如果你只请求了一个精度比较低的位置权限，位置服务会对返回的位置信息处理成一个相当于城市级别精确度的位置。

请求[ACCESS\\_FINE\\_LOCATION](#)权限时也包含了[ACCESS\\_COARSE\\_LOCATION](#)权限。

举个例子，如果你要添加[ACCESS\\_COARSE\\_LOCATION](#)权限，你需要将下面的权限添加到<manifest>标签中：

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
```

# 检测Google Play Services

位置服务是Google Play services 中的一部分。由于很难预料用户设备的状态，所以你在尝试连接位置服务之前应该要检测你的设备是否安装了Google Play services安装包。为了检测这个安装包是否被安装，你可以调

用[GooglePlayServicesUtil.isGooglePlayServicesAvailable\(\)](#)，这个方法将会返回一个结果代码。你可以通过查询[ConnectionResult](#)的参考文档中结果代码列表来理解对应的结果代码。如果你碰到了错误，你可以调用[GooglePlayServicesUtil.getErrorDialog\(\)](#)获取本地化的对话框来提示用户采取适当地行为，接着你需要将这个对话框置于一个[DialogFragment](#)中显示。这个对话框可以让用户去纠正这个问题，这个时候Google Services可以将结果返回给你的activity。为了处理这个结果，重写[onActivityResult\(\)](#)即可。

因为你的代码里通常会不止一次地检测Google Play services是否安装，为了方便，可以定义一个方法来封装这种检测行为。下面的代码片段包含了所有检测Google Play services是否安装需要用到的代码：

```
public class MainActivity extends FragmentActivity {  
    ...  
    //全局变量  
    /*  
     * 定义一个发送给Google Play services的请求代码  
     * 这个代码将会在Activity.onActivityResult的方法中返回  
     */  
    private final static int  
        CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;  
    ...  
    // 定义一个显示错误对话框的DialogFragment  
    public static class ErrorDialogFragment extends DialogFragment  
        // 表示错误对话框的全局属性  
        private Dialog mDialog;  
        // 默认的构造函数，将 dialog 属性设为空  
        public ErrorDialogFragment() {  
            super();  
            mDialog = null;  
        }  
        // 设置要显示的dialog  
        public void setDialog(Dialog dialog) {  
            mDialog = dialog;  
        }  
        // 返回一个 Dialog 给 DialogFragment.  
        @Override  
        public Dialog onCreateDialog(Bundle savedInstanceState) {  
            return mDialog;  
        }  
    }  
    ...  
    /*  
     * 处理来自Google Play services 发给FragmentActivity的结果  
     *  
     */  
    @Override  
    protected void onActivityResult(  
        int requestCode, int resultCode, Intent data) {
```



```
}
```

```
...
```

下面的代码片段使用了这个方法来检查Google Play services是否可用。

# 定义位置服务回调函数

为了获取当前的位置，你需要创建一个location client，将它连接到Location Services，然后调用它的 [getLastLocation\(\)](#) 方法。最后返回的值是基于你应用请求的权限以及当时启用的位置传感器的最佳位置信息。

在你创建location client之前，你必须实现一些被 Location Services 用来同你的应用通信的接口

## [ConnectionCallbacks](#)

- 设置了当一个location client连接成功或者断开连接时 Location Services 必须调用了方法。

## [OnConnectionFailedListener](#)

- 设置了当一个错误出现而需要去连接location client时 Location Services 需要调用的方法。这个方法用到了之前定义好的 `showErrorDialog` 方法来显示一个 error dialog，并尝试用 Google Play services 来修复这个问题。

下面的代码片段展示了如何实现这些接口并定义对应的方法：

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {

    ...
    /**
     * 当连接到client的请求成功结束时被Location Services 调用。这时你可以
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        // 显示连接状态
        Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show()

    }
    ...
    /**
     * 当连接因为错误被location client丢弃时，Location Services调用此方
     */
    @Override
    public void onDisconnected() {
        // 显示连接状态
        Toast.makeText(this, "Disconnected. Please re-connect.",
                      Toast.LENGTH_SHORT).show();
    }
    ...
    /**
     * 尝试连接Location Services失败后被 Location Services调用的方法
     */
    @Override
    public void onConnectionFailed(ConnectionResult connectionResu
        /*
         * Google Play services 可以解决它探测到的一些错误。
         * 如果这个错误有一个解决方案，这个方法会试着发送一个Intent去启动-
         */
    
```

```
if (connectionResult.hasResolution()) {
    try {
        // 启动一个尝试解决问题的Activity
        connectionResult.startResolutionForResult(
            this,
            CONNECTION_FAILURE_RESOLUTION_REQUEST);
        /*
         * 如果Google Play services 取消这个最初的PendingIntent
         */
    } catch (IntentSender.SendIntentException e) {
        // 记录错误
        e.printStackTrace();
    }
} else {
    /*
     * 如果没有可用的解决方案，将错误通过一个 dialog 显示给用户
     */
    showErrorDialog(connectionResult.getErrorCode());
}
...
}
```

## 连接 Location Client

既然这个回调函数已经写好了，现在我们可以创建location client并将它连接到Location Services。

首先你要在onCreate()方法里面创建location client，然后在onStart()方法里面连接它 then connect it in，这样当你的activity对用户可见时 Location Services 就保存着当前的位置信息了。你需要在onStop()方法里面断开连接，这样当你的activity不可见时，Location Services 就不会保存你的位置信息。下面连接和断开连接的方式对节省电池很有帮助。例如：

**注意:** 当前的位置信息只有在location client 连接到 Location Service时才会被保存。假设没有其他应用连接到 Location Services，如果你断开 client 的连接，那么这时你调用 [getLastLocation\(\)](#) 所获取到的位置信息可能已经过时。

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        /*
         * Create a new location client, using the enclosing class
         * handle callbacks.
         */
        mLocationClient = new LocationClient(this, this, this);
        ...
    }
    ...
    /**
     * 当Activity可见时调用
     */
    @Override
    protected void onStart() {
        super.onStart();
        // 连接 client.
        mLocationClient.connect();
    }
    ...
    /**
     * Called when the Activity is no longer visible.
     */
    @Override
    protected void onStop() {
        // 断开 client 与 Location Services 的连接，使 client 失效。
        mLocationClient.disconnect();
        super.onStop();
    }
    ...
}
```

## 获取当前的位置信息

为了获取当前的位置信息，调用[getLastLocation\(\)](#)方法。例如：

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {

    ...
    // 保存当前位置信息的全局变量
    Location mCurrentLocation;
    ...
    mCurrentLocation = mLocationClient.getLastLocation();
    ...
}
```

下一课，[获取位置更新](#)，教你如果周期性地从Location Services获取位置信息更新。

编写:[penkzhou](#)

校对:

# 获取位置更新

如果你的应用需要导航或者记录路径，你可能会周期性地去获取用户的位置信息。此时你可以使用Location Services 里面的 [LocationClient.getLastLocation\(\)](#) 来进行周期性的位置信息更新。使用这个方法之后，Location Services 会基于当前可用的位置信息提供源（比如 WiFi 和 GPS）返回最准确的位置信息更新。

你可以使用一个location client从 Location Services 那里请求周期性的位置更新。根据不同请求的形式，Location Services 要么调用一个回调函数并传入一个 [Location](#) 对象，或者发送一个包含位置信息的 [Intent](#)。位置更新的精度和频率与你的应用所申请的权限相关联。

## 确定应用的权限

使用位置服务的应用必须用户位置权限。Android拥有两种位置权限：[ACCESS\\_COARSE\\_LOCATION](#) 和 [ACCESS\\_FINE\\_LOCATION](#)。选择不同的权限决定你的应用最后获取的位置信息的精度。如果你只请求了一个精度比较低的位置权限，位置服务会对返回的位置信息处理成一个相当于城市级别精确度的位置。

请求[ACCESS\\_FINE\\_LOCATION](#)权限时也包含了[ACCESS\\_COARSE\\_LOCATION](#)权限。

举个例子，如果你要添加[ACCESS\\_COARSE\\_LOCATION](#)权限，你需要将下面的权限添加到<manifest>标签中：

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
```

# 检测Google Play Services

位置服务是Google Play services 中的一部分。由于很难预料用户设备的状态，所以你在尝试连接位置服务之前应该要检测你的设备是否安装了Google Play services安装包。为了检测这个安装包是否被安装，你可以调

用[GooglePlayServicesUtil.isGooglePlayServicesAvailable\(\)](#)，这个方法将会返回一个结果代码。你可以通过查询[ConnectionResult](#)的参考文档中结果代码列表来理解对应的结果代码。如果你碰到了错误，你可以调用[GooglePlayServicesUtil.getErrorDialog\(\)](#)获取本地化的对话框来提示用户采取适当地行为，接着你需要将这个对话框置于一个[DialogFragment](#)中显示。这个对话框可以让用户去纠正这个问题，这个时候Google Services可以将结果返回给你的activity。为了处理这个结果，重写[onActivityResult\(\)](#)即可。

**注意:** 为了让你的应用能够兼容 Android 1.6 之后的版本，用来显示DialogFragment 的必须是FragmentActivity而不是之前的Activity。使用FragmentActivity同样可以调用 `getSupportFragmentManager()` 方法来显示 DialogFragment。

因为你的代码里通常会不止一次地检测Google Play services是否安装，为了方便，可以定义一个方法来封装这种检测行为。下面的代码片段包含了所有检测Google Play services是否安装需要用到的代码：

```
public class MainActivity extends FragmentActivity {  
    ...  
    //全局变量  
    /*  
     * 定义一个发送给Google Play services的请求代码  
     * 这个代码将会在Activity.onActivityResult的方法中返回  
     */  
    private final static int  
        CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;  
    ...  
    // 定义一个显示错误对话框的DialogFragment  
    public static class ErrorDialogFragment extends DialogFragment  
        // 表示错误对话框的全局属性  
        private Dialog mDialog;  
        // 默认的构造函数，将 dialog 属性设为空  
        public ErrorDialogFragment() {  
            super();  
            mDialog = null;  
        }  
        // 设置要显示的dialog  
        public void setDialog(Dialog dialog) {  
            mDialog = dialog;  
        }  
        // 返回一个 Dialog 给 DialogFragment.  
        @Override  
        public Dialog onCreateDialog(Bundle savedInstanceState) {  
            return mDialog;  
        }  
    }  
    ...
```

```
/*
 * 处理来自Google Play services 发给FragmentActivity的结果
 *
 */
@Override
protected void onActivityResult(
    int requestCode, int resultCode, Intent data) {
    // 根据请求代码来决定做什么
    switch (requestCode) {
        ...
        case CONNECTION_FAILURE_RESOLUTION_REQUEST :
        /*
         * 如果结果代码是 Activity.RESULT_OK, 尝试重新连接
         *
        */
        switch (resultCode) {
            case Activity.RESULT_OK :
            /*
             * 尝试重新请求
             */
            ...
            break;
        }
        ...
    }
}

private boolean servicesConnected() {
    // 检测Google Play services 是否可用
    int resultCode =
        GooglePlayServicesUtil.
            isGooglePlayServicesAvailable(this);
    // 如果 Google Play services 可用
    if (ConnectionResult.SUCCESS == resultCode) {
        // 在 debug 模式下, 记录程序日志
        Log.d("Location Updates",
            "Google Play services is available.");
        // Continue
        return true;
    }
    // 因为某些原因Google Play services 不可用
} else {
    // 获取error code
    int errorCode = connectionResult.getErrorCode();
    // 从Google Play services 获取 error dialog
    Dialog errorDialog = GooglePlayServicesUtil.getErrorDi-
        errorCode,
        this,
        CONNECTION_FAILURE_RESOLUTION_REQUEST);

    // 如果 Google Play services可以提供一个error dialog
    if (errorDialog != null) {
        // 为这个error dialog 创建一个新的DialogFragment
        ErrorDialogFragment errorFragment =
            new ErrorDialogFragment();
        // 在DialogFragment中设置dialog
```

```
        errorFragment.setDialog(errorDialog);
        // 在DialogFragment中显示error dialog
        errorFragment.show(getSupportFragmentManager(),
                            "Location Updates");
    }
}
...
}
```

下面的代码片段使用了这个方法来检查Google Play services是否可用。

# 定义位置服务回调函数

在你创建location client之前, 你必须实现一些被 Location Services用来同你的应用通信的接口

## [ConnectionCallbacks](#)

- 设置了当一个location client连接成功或者断开连接时 Location Services必须调用了方法。

## [OnConnectionFailedListener](#)

- 设置了当一个错误出现而需要去连接location client时Location Services需要调用的方法。这个方法用到了之前定义好的 showErrorDialog 方法来显示一个error dialog, 并尝试用Google Play services来修复这个问题。

下面的代码片段展示了如何实现这些接口并定义对应的方法:

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {

    ...
    /**
     * 当连接到client的请求成功结束时被Location Services 调用。这时你可以
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        // 显示连接状态
        Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show();

    }
    ...
    /**
     * 当连接因为错误被location client丢弃时, Location Services调用此方
     */
    @Override
    public void onDisconnected() {
        // 显示连接状态
        Toast.makeText(this, "Disconnected. Please re-connect.",
                      Toast.LENGTH_SHORT).show();
    }
    ...
    /**
     * 尝试连接Location Services失败后被 Location Services调用的方法
     */
    @Override
    public void onConnectionFailed(ConnectionResult connectionResu
        /*
         * Google Play services 可以解决它探测到的一些错误。
         * 如果这个错误有一个解决方案, 这个方法会试着发送一个Intent去启动-
         */
        if (connectionResult.hasResolution()) {
            try {
                // 启动一个尝试解决问题的Activity
                connectionResult.startResolutionForResult(

```

```
        this,
        CONNECTION_FAILURE_RESOLUTION_REQUEST);
    /*
     * 如果Google Play services 取消这个最初的PendingIntent
     */
} catch (IntentSender.SendIntentException e) {
    // 记录错误
    e.printStackTrace();
}
} else {
    /*
     * 如果没有可用的解决方案，将错误通过一个 dialog 显示给用户
     */
    showErrorDialog(connectionResult.getErrorCode());
}
...
}
```

现在你已经写好了回调函数，你可以设置位置更新的请求了。第一步就是确定可以控制位置更新的参数。

## 确定位置更新参数

Location Services可以让你通过设置[LocationRequest](#)里面的值来控制位置更新的频率和精度，然后把[LocationRequest](#)这个对象作为更新请求的一部分发送出去，接着就可以开始更新位置信息了。

首先，设置下面的周期参数：

更新频率

- 更新频率通过[LocationRequest.setInterval\(\)](#)方法来设置。这个方法设置的毫秒数表示你的应用在这个时间内尽可能的接受位置更新信息。如果当时没有其他应用从Location Services获取位置更新，那么你的应用就会以设置的频率接收位置更新。

最快更新频率

- 最快更新频率通过[LocationRequest.setFastestInterval\(\)](#)方法设置。这个方法设置你的应用能够接收位置更新最快的频率。你必须设置这个频率因为其他应用也会影响位置更新的频率。Location Services会以选择所有请求位置更新的应用中频率最快的发送位置更新。。如果这个频率比你的应用能处理的频率还要快，那么你的应用可能会出现UI闪烁或者数据溢出。为了防止这样的情况出现，调用[LocationRequest.setFastestInterval\(\)](#)方法来设置位置更新频率的上限，同时还可以节约电量。当你通过[LocationRequest.setInterval\(\)](#)方法设置理想的更新频率，通过[LocationRequest.setFastestInterval\(\)](#)设置更新频率的上限，然后你的应用就会在系统中获得最快的位置更新频率。如果其他应用设置的更新频率更快，那么你的应用也跟着受益。如果其他应用的更新频率没有你的频率快，那么你的应用将会以你通过[LocationRequest.setInterval\(\)](#)方法设置的频率更新位置信息。

接着，设置精度参数。在一个前台应用（foreground app）中，你需要不断地获取高精度的位置更新，因此需要使用[LocationRequest.PRIORITY\\_HIGH\\_ACCURACY](#)。

下面的代码片段展示了如何在onCreate()方法里面设置更新频率和精度：

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener,
    LocationListener {

    ...
    // Global constants
    ...
    // Milliseconds per second
    private static final int MILLISECONDS_PER_SECOND = 1000;
    // Update frequency in seconds
    public static final int UPDATE_INTERVAL_IN_SECONDS = 5;
    // Update frequency in milliseconds
    private static final long UPDATE_INTERVAL =
        MILLISECONDS_PER_SECOND * UPDATE_INTERVAL_IN_SECONDS;
    // The fastest update frequency, in seconds
    private static final int FASTEST_INTERVAL_IN_SECONDS = 1;
    // A fast frequency ceiling in milliseconds
    private static final long FASTEST_INTERVAL =
        MILLISECONDS_PER_SECOND * FASTEST_INTERVAL_IN_SECONDS;
    ...
}
```

```
// 定义一个包含定位精度和定位频率的对象
LocationRequest mLocationRequest;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create the LocationRequest object
    mLocationRequest = LocationRequest.create();
    // 使用高精度
    mLocationRequest.setPriority(
        LocationRequest.PRIORITY_HIGH_ACCURACY);
    // 设置更新频率为 5 seconds
    mLocationRequest.setInterval(UPDATE_INTERVAL);
    // 设置最快更新频率为 1 second
    mLocationRequest.setFastestInterval(FAATEST_INTERVAL);
    ...
}
...
}
```

注意：如果你的应用在获取位置更新后需要访问网络或者进行长时的操作，你可以将最快更新频率调整至更慢的值。这样可以让你的应用不会接受无法使用的位置更新。一旦这样的长时操作完成，将最快更新频率设回原值。

## 开始进行位置更新

To send the request for location updates, create a location client in `onCreate()`, then connect it and make the request by calling `requestLocationUpdates()`. Since your client must be connected for your app to receive updates, you should connect the client in `onStart()`. This ensures that you always have a valid, connected client while your app is visible. Since you need a connection before you can request updates, make the update request in `ConnectionCallbacks.onConnected()`

Remember that the user may want to turn off location updates for various reasons. You should provide a way for the user to do this, and you should ensure that you don't start updates in `onStart()` if updates were previously turned off. To track the user's preference, store it in your app's Shared Preferences in `onPause()` and retrieve it in `onResume()`.

The following snippet shows how to set up the client in `onCreate()`, and how to connect it and request updates in `onStart()`:

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener,
    LocationListener {

    ...
    // Global variables
    ...
    LocationClient mLocationClient;
    boolean mUpdatesRequested;
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Open the shared preferences
        mPrefs = getSharedPreferences("SharedPreferences",
            Context.MODE_PRIVATE);
        // Get a SharedPreferences editor
        mEditor = mPrefs.edit();
        /*
         * Create a new location client, using the enclosing class
         * handle callbacks.
         */
        mLocationClient = new LocationClient(this, this, this);
        // Start with updates turned off
        mUpdatesRequested = false;
        ...
    }
    ...
    @Override
    protected void onPause() {
        // Save the current setting for updates
        mEditor.putBoolean("KEY_UPDATES_ON", mUpdatesRequested);
        mEditor.commit();
        super.onPause();
    }
    ...
    @Override
```

```
protected void onStart() {
    ...
    mLocationClient.connect();
}

...
@Override
protected void onResume() {
    /*
     * Get any previous setting for location updates
     * Gets "false" if an error occurs
     */
    if (mPrefs.contains("KEY_UPDATES_ON")) {
        mUpdatesRequested =
            mPrefs.getBoolean("KEY_UPDATES_ON", false);

        // Otherwise, turn off location updates
    } else {
        mEditor.putBoolean("KEY_UPDATES_ON", false);
        mEditor.commit();
    }
}
...
/*
 * Called by Location Services when the request to connect the
 * client finishes successfully. At this point, you can
 * request the current location or start periodic updates
 */
@Override
public void onConnected(Bundle dataBundle) {
    // Display the connection status
    Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show
    // If already requested, start periodic updates
    if (mUpdatesRequested) {
        mLocationClient.requestLocationUpdates(mLocationReques
    }
}
...
}
```

For more information about saving preferences, read [Saving Key-Value Sets](#).

## Stop Location Updates

To stop location updates, save the state of the update flag in onPause(), and stop updates in onStop() by calling removeLocationUpdates(LocationListener). For example:

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener,
    LocationListener {

    ...
    /*
     * Called when the Activity is no longer visible at all.
     * Stop updates and disconnect.
     */
    @Override
    protected void onStop() {
        // If the client is connected
        if (mLocationClient.isConnected()) {
            /*
             * Remove location updates for a listener.
             * The current Activity is the listener, so
             * the argument is "this".
             */
            removeLocationUpdates(this);
        }
        /*
         * After disconnect() is called, the client is
         * considered "dead".
         */
        mLocationClient.disconnect();
        super.onStop();
    }
    ...
}
```

You now have the basic structure of an app that requests and receives periodic location updates. You can combine the features described in this lesson with the geofencing, activity recognition, or reverse geocoding features described in other lessons in this class.

The next lesson, Displaying a Location Address, shows you how to use the current location to display the current street address.

编写:[penkzhou](#)

校对:

# 显示位置地址

获取当前位置和接收位置更新课程描述了如何以一个[Location](#)对象的形式获取用户当前的位置信息，这个位置信息包括了经纬度。尽管经纬度对计算地理距离和在地图上显示位置很有帮助，但是更多情况下位置信息的地址更有用。

Android平台API提供一个根据地理经纬度返回一个大概的街道地址信息这一课教你如何使用这个地址检索功能。

注意：地址检索需要一个后台服务，然后这个后台服务是不包含在核心的Android框架里面的。如果这个后台服务不可用，[Geocoder.getFromLocation\(\)](#)方法将会返回一个空列表。在Android API 9以上的API里面有一个辅助方法[isPresent\(\)](#)可以检查这个后台服务是否可用。

下面的代码假设你已经获取到了位置信息并将位置信息以[Location](#)对象的形式保存到全局变量mLocation里面。

## 定义地址检索任务

为了通过给定的经纬度获取地址信息，你需要调用[Geocoder.getFromLocation\(\)](#)方法并返回一个地址列表。由于这个方法是同步的，所以在获取地址信息的时候可能耗时较长，因此你需要通过 [AsyncTask 的 doInBackground\(\)](#) 方法来执行这个方法。

当你的应用在获取地址信息的时候，可以使用一个进度条之类的控件来表示你的应用正在后台工作中。你可以将这个控件的初始状态设为`android:visibility="gone"`，这样可以让这个控件不可见。当你开始进行地址检索的时候，你需要将这个控件的可见属性设为`"visible"`。

下面的代码教你如何在你的布局文件里面添加一个进度条：

```
<ProgressBar  
    android:id="@+id/address_progress"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:indeterminate="true"  
    android:visibility="gone" />
```

要创建一个后台任务，首先要定义一个`AsyncTask`的子类来调用`getFromLocation()`方法，然后返回地址。定义一个`TextView`对象`mAddress`来显示返回的地址信息，进度条则用来显示请求的进度过程。例如：

```
public class MainActivity extends FragmentActivity {  
    ...  
    private TextView mAddress;  
    private ProgressBar mActivityIndicator;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        mAddress = (TextView) findViewById(R.id.address);  
        mActivityIndicator =  
            (ProgressBar) findViewById(R.id.address_progress);  
    }  
    ...  
    /**  
     * A subclass of AsyncTask that calls getFromLocation() in the  
     * background. The class definition has these generic types:  
     * Location - A Location object containing  
     * the current location.  
     * Void - indicates that progress units are not used  
     * String - An address passed to onPostExecute()  
     */  
    private class GetAddressTask extends  
        AsyncTask<Location, Void, String> {  
        Context mContext;  
        public GetAddressTask(Context context) {  
            super();
```

```
mContext = context;
}
...
/***
 * Get a Geocoder instance, get the latitude and longitude
 * look up the address, and return it
 *
 * @params params One or more Location objects
 * @return A string containing the address of the current
 * location, or an empty string if no address can be found
 * or an error message
 */
@Override
protected String doInBackground(Location... params) {
    Geocoder geocoder =
        new Geocoder(mContext, Locale.getDefault());
    // Get the current location from the input parameter
    Location loc = params[0];
    // Create a list to contain the result address
    List<Address> addresses = null;
    try {
        /*
         * Return 1 address.
         */
        addresses = geocoder.getFromLocation(loc.getLatitude(),
            loc.getLongitude(), 1);
    } catch (IOException e1) {
        Log.e("LocationSampleActivity",
            "IO Exception in getFromLocation()");
        e1.printStackTrace();
        return ("IO Exception trying to get address");
    } catch (IllegalArgumentException e2) {
        // Error message to post in the log
        String errorString = "Illegal arguments " +
            Double.toString(loc.getLatitude()) +
            " , " +
            Double.toString(loc.getLongitude()) +
            " passed to address service";
        Log.e("LocationSampleActivity", errorString);
        e2.printStackTrace();
        return errorString;
    }
    // If the reverse geocode returned an address
    if (addresses != null && addresses.size() > 0) {
        // Get the first address
        Address address = addresses.get(0);
        /*
         * Format the first line of address (if available)
         * city, and country name.
         */
        String addressText = String.format(
            "%s, %s, %s",
            // If there's a street address, add it
            address.getMaxAddressLineIndex() > 0 ?
                address.getAddressLine(0) : "",
```

```
        // Locality is usually a city
        address.getLocality(),
        // The country of the address
        address.getCountryName());
    // Return the text
    return addressText;
} else {
    return "No address found";
}
...
}
...
}
```

下一部分教你如何在用户界面上显示地址信息。

## 定义显示结果的方法

[doInBackground\(\)](#)方法返回一个包含地址检索结果的字符串。这个值会被传入[onPostExecute\(\)](#)方法，通过这个方法你可以对结果进行更深的处理。因为[onPostExecute\(\)](#)运行在UI主线程上面，它可以更新用户界面；例如，它可以隐藏进度条然后显示返回的地址结果给用户：

```
private class GetAddressTask extends
    AsyncTask<Location, Void, String> {
    ...
    /**
     * A method that's called once doInBackground() completes.
     * off the indeterminate activity indicator and set
     * the text of the UI element that shows the address. If t
     * lookup failed, display the error message.
     */
    @Override
    protected void onPostExecute(String address) {
        // Set activity indicator visibility to "gone"
        mActivityIndicator.setVisibility(View.GONE);
        // Display the results of the lookup.
        mAddress.setText(address);
    }
    ...
}
```

最后一步就是运行地址检索任务。

## 运行地址检索任务

为了获取地址信息，调用[execute\(\)](#)方法即可。例如，下面的代码片段展示了当用户点击"Get Address"按钮时应用就开始检索地址信息了：

```
public class MainActivity extends FragmentActivity {  
    ...  
    /**  
     * The "Get Address" button in the UI is defined with  
     * android:onClick="getAddress". The method is invoked whenever  
     * user clicks the button.  
     *  
     * @param v The view object associated with this method,  
     * in this case a Button.  
     */  
    public void getAddress(View v) {  
        // Ensure that a Geocoder services is available  
        if (Build.VERSION.SDK_INT >=  
            Build.VERSION_CODES.GINGERBREAD  
            &&  
            Geocoder.isPresent()) {  
            // Show the activity indicator  
            mActivityIndicator.setVisibility(View.VISIBLE);  
            /*  
             * Reverse geocoding is long-running and synchronous.  
             * Run it on a background thread.  
             * Pass the current location to the background task.  
             * When the task finishes,  
             * onPostExecute() displays the address.  
             */  
            (new GetAddressTask(this)).execute(mLocation);  
        }  
        ...  
    }  
    ...  
}
```

下一课，[创建和监视Geofences](#)将会教你如何定义地理围栏以及如何使用地理围栏来探测用户对一个兴趣位置的接近程度。

编写:[penkzhou](#)

校对:

# 创建并监视异常区域

地理围栏将用户当前位置感知和附件地点特征感知相结合，定义了用户对位置的接近程度。为了让一个位置有感知，你必须确定这个位置的经纬度。为了度量用户对位置的接近程度，你需要添加一个半径。综合经纬度和半径即可确定一个地理围栏。当然你可以一次性定义多个地理围栏。

Location Services将一个地理围栏看成是一片区域而不是一个点和一个接近程度。这样可以让它去探测用户是否进入或者正在某个地理围栏中。对于每个地理围栏，你可以让 Location Services给你发送进入或者退出地理围栏事件。你还可以通过设置一个毫秒级别的有效时间来限制地理围栏的生命周期。当地理围栏失效后，Location Services会自动移除这个地理围栏。

# 请求地理围栏监视

请求地理围栏监视的第一步就是设置必要的权限。在使用地理围栏时，你必须设置ACCESS\_FINE\_LOCATION权限。添加如下代码即可：

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
```

## 检查Google Play Services是否可用

位置服务是Google Play services 中的一部分。由于很难预料用户设备的状态，所以你在尝试连接位置服务之前应该要检测你的设备是否安装了Google Play services安装包。为了检测这个安装包是否被安装，你可以调

用[GooglePlayServicesUtil.isGooglePlayServicesAvailable\(\)](#)，这个方法将会返回一个结果代码。你可以通过查询[ConnectionResult](#)的参考文档中结果代码列表来理解对应的结果代码。如果你碰到了错误，你可以调用[GooglePlayServicesUtil.getErrorDialog\(\)](#)获取本地化的对话框来提示用户采取适当地行为，接着你需要将这个对话框置于一个[DialogFragment](#)中显示。这个对话框可以让用户去纠正这个问题，这个时候Google Services可以将结果返回给你的activity。为了处理这个结果，重写[onActivityResult\(\)](#)即可。

**注意:** 为了让你的应用能够兼容 Android 1.6 之后的版本，用来显示DialogFragment 的必须是FragmentActivity而不是之前的Activity。使用FragmentActivity同样可以调用 `getSupportFragmentManager()` 方法来显示 DialogFragment。

因为你的代码里通常会不止一次地检测Google Play services是否安装，为了方便，可以定义一个方法来封装这种检测行为。下面的代码片段包含了所有检测Google Play services是否安装需要用到的代码：

```
public class MainActivity extends FragmentActivity {
    ...
    //全局变量
    /*
     * 定义一个发送给Google Play services的请求代码
     * 这个代码将会在Activity.onActivityResult的方法中返回
     */
    private final static int
        CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;
    ...
    // 定义一个显示错误对话框的DialogFragment
    public static class ErrorDialogFragment extends DialogFragment
        // 表示错误对话框的全局属性
        private Dialog mDialog;
        // 默认的构造函数，将 dialog 属性设为空
        public ErrorDialogFragment() {
            super();
            mDialog = null;
        }
        // 设置要显示的dialog
        public void setDialog(Dialog dialog) {
            mDialog = dialog;
        }
}
```

```
        }
        // 返回一个 Dialog 给 DialogFragment.
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return mDialog;
    }
}

...
/*
 * 处理来自Google Play services 发给FragmentActivity的结果
 *
 */
@Override
protected void onActivityResult(
    int requestCode, int resultCode, Intent data) {
    // 根据请求代码来决定做什么
    switch (requestCode) {
        ...
        case CONNECTION_FAILURE_RESOLUTION_REQUEST :
        /*
         * 如果结果代码是 Activity.RESULT_OK, 尝试重新连接
         *
         */
        switch (resultCode) {
            case Activity.RESULT_OK :
            /*
             * 尝试重新请求
             */
            ...
            break;
        }
        ...
    }
}

...
private boolean servicesConnected() {
    // 检测Google Play services 是否可用
    int resultCode =
        GooglePlayServicesUtil.
            isGooglePlayServicesAvailable(this);
    // 如果 Google Play services 可用
    if (ConnectionResult.SUCCESS == resultCode) {
        // 在 debug 模式下, 记录程序日志
        Log.d("Location Updates",
            "Google Play services is available.");
        // Continue
        return true;
    }
    // 因为某些原因Google Play services 不可用
} else {
    // 获取error code
    int errorCode = connectionResult.getErrorCode();
    // 从Google Play services 获得 error dialog
    Dialog errorDialog = GooglePlayServicesUtil.getErrorDi
        errorCode,
        this,
```

```

        CONNECTION_FAILURE_RESOLUTION_REQUEST);

        // 如果 Google Play services可以提供一个error dialog
        if (errorDialog != null) {
            // 为这个error dialog 创建一个新的DialogFragment
            ErrorDialogFragment errorFragment =
                new ErrorDialogFragment();
            // 在DialogFragment中设置dialog
            errorFragment.setDialog(errorDialog);
            // 在DialogFragment中显示error dialog
            errorFragment.show(getSupportFragmentManager(),
                "Geofence Detection");
        }
    }
    ...
}

```

下面的代码片段使用了这个方法来检查Google Play services是否可用。

要使用地理围栏，你得先定义你要监控的地理围栏。通常你可以在本地保存地理围栏数据或者从互联网上获取地理围栏数据，然后你需要发送一个由[Geofence.Builder](#)创建的[Geofence](#)对象给Location Services。每一个[Geofence](#)对象都包括了以下数据：

### 精度, 纬度和半径

- 为地理围栏定义一个圆形区域。使用经纬度标记一个兴趣地点，然后使用半径定义地理围栏有效区域。半径越大，用户越有可能触发地理围栏的警报。例如，为一个家庭灯具地理围栏应用设置一个大的半径，这样当用户回家就可能触发地理围栏，然后灯就亮了。

### 有效时间

- 地理围栏保持激活状态的时间。一旦达到了有效时间，Location Services会删除这个地理围栏。大部分时候，你都应该为你的应用设置一个有效时间，但对于家居或者工作空间等类型的应用，可能设置需要永久的地理围栏。nces for the user's home or place of work.

### 触发事件类型

- Location Services 能探测到用户进入地理围栏的有效范围内或者走出有效范围的事件。

### 地理围栏 ID

- 一个与地理围栏一同保存的字符串。你必须保证这个字符串唯一，这样你就可以使用它从Location Services的记录里来移除特定地理围栏。

## 定义地理围栏存储

一个地理围栏应用需要将地理围栏数据做持久化的读写。但是你不能使用[Geofence](#)进行这样的操作；你可以使用数据库等方式来保存地理围栏的相关数据。

作为一个保存地理围栏数据的实例，下面的代码片段定义了两个使用[SharedPreferences](#) 的

类来进行地理围栏的数据持久化。SimpleGeofence类，类似于一条数据库记录，为一个[Geofence](#)对象存储数据。SimpleGeofenceStore类，类似于一个数据库，对SimpleGeofence的读写应用到[SharedPreferences](#)实例。

```
public class MainActivity extends FragmentActivity {  
    ...  
    /**  
     * A single Geofence object, defined by its center and radius.  
     */  
    public class SimpleGeofence {  
        // Instance variables  
        private final String mId;  
        private final double mLatitude;  
        private final double mLongitude;  
        private final float mRadius;  
        private long mExpirationDuration;  
        private int mTransitionType;  
  
        /**  
         * @param geofenceId The Geofence's request ID  
         * @param latitude Latitude of the Geofence's center.  
         * @param longitude Longitude of the Geofence's center.  
         * @param radius Radius of the geofence circle.  
         * @param expiration Geofence expiration duration  
         * @param transition Type of Geofence transition.  
         */  
        public SimpleGeofence(  
            String geofenceId,  
            double latitude,  
            double longitude,  
            float radius,  
            long expiration,  
            int transition) {  
            // Set the instance fields from the constructor  
            this.mId = geofenceId;  
            this.mLatitude = latitude;  
            this.mLongitude = longitude;  
            this.mRadius = radius;  
            this.mExpirationDuration = expiration;  
            this.mTransitionType = transition;  
        }  
        // Instance field getters  
        public String getId() {  
            return mId;  
        }  
        public double getLatitude() {  
            return mLatitude;  
        }  
        public double getLongitude() {  
            return mLongitude;  
        }  
        public float getRadius() {  
            return mRadius;  
        }  
    }  
}
```

```
    public long getExpirationDuration() {
        return mExpirationDuration;
    }
    public int getTransitionType() {
        return mTransitionType;
    }
    /**
     * Creates a Location Services Geofence object from a
     * SimpleGeofence.
     *
     * @return A Geofence object
     */
    public Geofence toGeofence() {
        // Build a new Geofence object
        return new Geofence.Builder()
            .setRequestId(getId())
            .setTransitionTypes(mTransitionType)
            .setCircularRegion(
                getLatitude(), getLongitude(), getRadius())
            .setExpirationDuration(mExpirationDuration)
            .build();
    }
}
...
/**
 * Storage for geofence values, implemented in SharedPreferences
 */
public class SimpleGeofenceStore {
    // Keys for flattened geofences stored in SharedPreferences
    public static final String KEY_LATITUDE =
        "com.example.android.geofence.KEY_LATITUDE";
    public static final String KEY_LONGITUDE =
        "com.example.android.geofence.KEY_LONGITUDE";
    public static final String KEY_RADIUS =
        "com.example.android.geofence.KEY_RADIUS";
    public static final String KEY_EXPIRATION_DURATION =
        "com.example.android.geofence.KEY_EXPIRATION_DURATION";
    public static final String KEY_TRANSITION_TYPE =
        "com.example.android.geofence.KEY_TRANSITION_TYPE";
    // The prefix for flattened geofence keys
    public static final String KEY_PREFIX =
        "com.example.android.geofence.KEY";
    /*
     * Invalid values, used to test geofence storage when
     * retrieving geofences
     */
    public static final long INVALID_LONG_VALUE = -999L;
    public static final float INVALID_FLOAT_VALUE = -999.0f;
    public static final int INVALID_INT_VALUE = -999;
    // The SharedPreferences object in which geofences are stored
    private final SharedPreferences mPrefs;
    // The name of the SharedPreferences
    private static final String SHARED_PREFERENCES =
        "SharedPreferences";
    // Create the SharedPreferences storage with private access
}
```

```
public SimpleGeofenceStore(Context context) {
    mPrefs =
        context.getSharedPreferences(
            SHARED_PREFERENCES,
            Context.MODE_PRIVATE);
}
/***
 * Returns a stored geofence by its id, or returns null
 * if it's not found.
 *
 * @param id The ID of a stored geofence
 * @return A geofence defined by its center and radius. See
 */
public SimpleGeofence getGeofence(String id) {
    /*
     * Get the latitude for the geofence identified by id,
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    double lat = mPrefs.getFloat(
        getGeofenceFieldKey(id, KEY_LATITUDE),
        INVALID_FLOAT_VALUE);
    /*
     * Get the longitude for the geofence identified by id,
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    double lng = mPrefs.getFloat(
        getGeofenceFieldKey(id, KEY_LONGITUDE),
        INVALID_FLOAT_VALUE);
    /*
     * Get the radius for the geofence identified by id,
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    float radius = mPrefs.getFloat(
        getGeofenceFieldKey(id, KEY_RADIUS),
        INVALID_FLOAT_VALUE);
    /*
     * Get the expiration duration for the geofence identified
     * by id, or INVALID_LONG_VALUE if it doesn't exist
     */
    long expirationDuration = mPrefs.getLong(
        getGeofenceFieldKey(id, KEY_EXPIRATION_DURATION),
        INVALID_LONG_VALUE);
    /*
     * Get the transition type for the geofence identified
     * by id, or INVALID_INT_VALUE if it doesn't exist
     */
    int transitionType = mPrefs.getInt(
        getGeofenceFieldKey(id, KEY_TRANSITION_TYPE),
        INVALID_INT_VALUE);
    // If none of the values is incorrect, return the object
    if (
        lat != GeofenceUtils.INVALID_FLOAT_VALUE &&
        lng != GeofenceUtils.INVALID_FLOAT_VALUE &&
        radius != GeofenceUtils.INVALID_FLOAT_VALUE &&
        expirationDuration !=
```

```
        GeofenceUtils.INVALID_LONG_VALUE &&
transitionType != GeofenceUtils.INVALID_INT_VALUE)

    // Return a true Geofence object
    return new SimpleGeofence(
        id, lat, lng, radius, expirationDuration,
        transitionType);
    // Otherwise, return null.
} else {
    return null;
}
}

/***
 * Save a geofence.
 * @param geofence The SimpleGeofence containing the
 * values you want to save in SharedPreferences
 */
public void setGeofence(String id, SimpleGeofence geofence
/*
 * Get a SharedPreferences editor instance. Among other
 * things, SharedPreferences ensures that updates are
 * and non-concurrent
 */
Editor editor = mPrefs.edit();
// Write the Geofence values to SharedPreferences
editor.putFloat(
    getGeofenceFieldKey(id, KEY_LATITUDE),
    (float) geofence.getLatitude());
editor.putFloat(
    getGeofenceFieldKey(id, KEY_LONGITUDE),
    (float) geofence.getLongitude());
editor.putFloat(
    getGeofenceFieldKey(id, KEY_RADIUS),
    geofence.getRadius());
editor.putLong(
    getGeofenceFieldKey(id, KEY_EXPIRATION_DURATION),
    geofence.getExpirationDuration());
editor.putInt(
    getGeofenceFieldKey(id, KEY_TRANSITION_TYPE),
    geofence.getTransitionType());
// Commit the changes
editor.commit();
}

public void clearGeofence(String id) {
/*
 * Remove a flattened geofence object from storage by
 * removing all of its keys
 */
Editor editor = mPrefs.edit();
editor.remove(getGeofenceFieldKey(id, KEY_LATITUDE));
editor.remove(getGeofenceFieldKey(id, KEY_LONGITUDE));
editor.remove(getGeofenceFieldKey(id, KEY_RADIUS));
editor.remove(getGeofenceFieldKey(id,
    KEY_EXPIRATION_DURATION));
editor.remove(getGeofenceFieldKey(id, KEY_TRANSITION_T
```

```

        editor.commit();
    }
    /**
     * Given a Geofence object's ID and the name of a field
     * (for example, KEY_LATITUDE), return the key name of the
     * object's values in SharedPreferences.
     *
     * @param id The ID of a Geofence object
     * @param fieldName The field represented by the key
     * @return The full key name of a value in SharedPreferences
     */
    private String getGeofenceFieldKey(String id,
                                       String fieldName) {
        return KEY_PREFIX + "_" + id + "_" + fieldName;
    }
}
...
}

```

## 创建地理围栏对象

下面的代码片段使用SimpleGeofence和SimpleGeofenceStore类从用户界面上获取地理围栏数据，然后将这些数据保存到`SimpleGeofence`对象里面，接着把这些SimpleGeofence对象保存到一个SimpleGeofenceStore里面，然后就可以创建[Geofence](#)对象了：

```

public class MainActivity extends FragmentActivity {
    ...
    /*
     * Use to set an expiration time for a geofence. After this amount
     * of time Location Services will stop tracking the geofence.
     */
    private static final long SECONDS_PER_HOUR = 60;
    private static final long MILLISECONDS_PER_SECOND = 1000;
    private static final long GEOFENCE_EXPIRATION_IN_HOURS = 12;
    private static final long GEOFENCE_EXPIRATION_TIME =
            GEOFENCE_EXPIRATION_IN_HOURS *
            SECONDS_PER_HOUR *
            MILLISECONDS_PER_SECOND;
    ...
    /*
     * Handles to UI views containing geofence data
     */
    // Handle to geofence 1 latitude in the UI
    private EditText mLatitude1;
    // Handle to geofence 1 longitude in the UI
    private EditText mLongitude1;
    // Handle to geofence 1 radius in the UI
    private EditText mRadius1;
    // Handle to geofence 2 latitude in the UI
    private EditText mLatitude2;
    // Handle to geofence 2 longitude in the UI
    private EditText mLongitude2;
}

```

```
// Handle to geofence 2 radius in the UI
private EditText mRadius2;
/*
 * Internal geofence objects for geofence 1 and 2
 */
private SimpleGeofence mUIGeofence1;
private SimpleGeofence mUIGeofence2;
...
// Internal List of Geofence objects
List<Geofence> mGeofenceList;
// Persistent storage for geofences
private SimpleGeofenceStore mGeofenceStorage;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    // Instantiate a new geofence storage area
    mGeofenceStorage = new SimpleGeofenceStore(this);

    // Instantiate the current List of geofences
    mCurrentGeofences = new ArrayList<Geofence>();
}
...
/**
 * Get the geofence parameters for each geofence from the UI
 * and add them to a List.
 */
public void createGeofences() {
    /*
     * Create an internal object to store the data. Set its
     * ID to "1". This is a "flattened" object that contains
     * a set of strings
     */
    mUIGeofence1 = new SimpleGeofence(
        "1",
        Double.valueOf(mLatitude1.getText().toString()),
        Double.valueOf(mLongitude1.getText().toString()),
        Float.valueOf(mRadius1.getText().toString()),
        GEOFENCE_EXPIRATION_TIME,
        // This geofence records only entry transitions
        Geofence.GEOFENCE_TRANSITION_ENTER);
    // Store this flat version
    mGeofenceStorage.setGeofence("1", mUIGeofence1);
    // Create another internal object. Set its ID to "2"
    mUIGeofence2 = new SimpleGeofence(
        "2",
        Double.valueOf(mLatitude2.getText().toString()),
        Double.valueOf(mLongitude2.getText().toString()),
        Float.valueOf(mRadius2.getText().toString()),
        GEOFENCE_EXPIRATION_TIME,
        // This geofence records both entry and exit trans
        Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT);
    // Store this flat version
```

```
mGeofenceStorage.setGeofence(2, mUIGeofence2);
mGeofenceList.add(mUIGeofence1.toGeofence());
mGeofenceList.add(mUIGeofence2.toGeofence());
}
...
}
```

除了这些你要监视的[Geofence](#)列表之外，你还需要为Location Services添加[Intent](#)，这个Intent在你的应用探测到地理围栏触发事件时会将这个事件发送给你的应用。

## 为地理围栏触发事件定义Intent

从Location Services发送来的Intent能够触发各种应用内的动作，但是不能用它来打开一个Activity或者Fragment，因为应用内的组件只能在响应用户动作时才能可见。大多数情况下，处理这一类的Intent最好使用[IntentService](#)。一个[IntentService](#)可以推送一个通知，可以进行长时的后台作业，可以将intent发送给其他的services，还可以广播intent。下面的代码展示了如何定义一个[PendingIntent](#)来启动一个IntentService：

```
public class MainActivity extends FragmentActivity {
    ...
    /*
     * Create a PendingIntent that triggers an IntentService in your
     * app when a geofence transition occurs.
     */
    private PendingIntent getTransitionPendingIntent() {
        // Create an explicit Intent
        Intent intent = new Intent(this,
            ReceiveTransitionsIntentService.class);
        /*
         * Return the PendingIntent
         */
        return PendingIntent.getService(
            this,
            0,
            intent,
            PendingIntent.FLAG_UPDATE_CURRENT);
    }
    ...
}
```

现在你已经拥有了所有发送监视地理围栏请求给Location Services的代码了。

## 发送监视请求

发送监视请求需要两个异步操作。第一个操作就是为这个请求获取一个location client，第二个操作就是使用这个client来生成请求。这两个操作里面，Location Services都会在操作结束的时候调用一个回调函数。处理这些操作最好的方法就是将这些方法调用连接起来。下面的代码展示了如何建立一个Activity，接着定义回调方法，然后以合适的顺序调用他们。首先，让Activity实现必要的回调接口。需要添加以下接口：[ConnectionCallbacks](#)

- 设置当location client已连接或者断开连接时Location Services调用的方法。

## [OnConnectionFailedListener](#)

- 设置当Location Services连接location client出错时Location Services调用的方法。

## [OnAddGeofencesResultListener](#)

- 设置当Location Services已经添加地理围栏的时候Location Services调用的方法。

例如：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
    ...
}
```

## 开始请求进程

接下啦，在连接Location Services的时候定义一个启动请求进程的方法。记得将这个请求设置为全局变量，这样就可以让你使用回调方法[ConnectionCallbacks.onConnected\(\)](#)来添加地理围栏，或者移除地理围栏。

为了防止当你的应用在第一个请求还没结束就开始第二个请求的时候不出现竞争状况，你可以定义一个boolean标志位来记录当前请求的状态：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
    ...
    // Holds the location client
    private LocationClient mLocationClient;
    // Stores the PendingIntent used to request geofence monitoring
    private PendingIntent mGeofenceRequestIntent;
    // Defines the allowable request types.
    public enum REQUEST_TYPE { ADD }
    private REQUEST_TYPE mRequestType;
    // Flag that indicates if a request is underway.
    private boolean mInProgress;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Start with the request flag set to false
        mInProgress = false;
        ...
    }
    ...
    /**
     * Start a request for geofence monitoring by calling
     * LocationClient.connect().
     */
}
```

```

public void addGeofences() {
    // Start a request to add geofences
    mRequestType = ADD;
    /*
     * Test for Google Play services after setting the request
     * If Google Play services isn't present, the proper request
     * can be restarted.
    */
    if (!servicesConnected()) {
        return;
    }
    /*
     * Create a new location client object. Since the current
     * activity class implements ConnectionCallbacks and
     * OnConnectionFailedListener, pass the current activity
     * as the listener for both parameters
    */
    mLocationClient = new LocationClient(this, this, this)
    // If a request is not already underway
    if (!mInProgress) {
        // Indicate that a request is underway
        mInProgress = true;
        // Request a connection from the client to Location Service
        mLocationClient.connect();
    } else {
        /*
         * A request is already underway. You can handle
         * this situation by disconnecting the client,
         * re-setting the flag, and then re-trying the
         * request.
        */
    }
}
...
}

```

## 发送添加地理围栏的请求

在你对回调方法[ConnectionCallbacks.onConnected\(\)](#)的实现里面，调用[LocationClient.addGeofences\(\)](#)。注意如果连接失败，[onConnected\(\)](#)方法不会被调用，这个请求也会停止。

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    /*
     * Provide the implementation of ConnectionCallbacks.onConnect
     * Once the connection is available, send a request to add the
     * Geofences
    */
    @Override

```

```

private void onConnected(Bundle dataBundle) {
    ...
    switch (mRequestType) {
        case ADD :
            // Get the PendingIntent for the request
            mTransitionPendingIntent =
                getTransitionPendingIntent();
            // Send a request to add the current geofences
            mLocationClient.addGeofences(
                mCurrentGeofences, pendingIntent, this);
            ...
    }
    ...
}

```

注意[addGeofences\(\)](#)方法会直接返回，但是请求的状态却不是直接返回的，只有等到Location Services调用[onAddGeofencesResult\(\)](#)方法。一旦这个方法被调用，你就能知道这个请求是否成功。

## 通过Location Services检测请求返回的结果

当 Location Services 你对回调函数[onAddGeofencesResult\(\)](#)的实现的时候，说明请求已经结束，你可以检测最终的结果状态码。如果请求成功，那么你轻轻的地理围栏是激活的，如果没有成功，那么你请求的地理围栏没有激活。如果没成功，你需要重试或者报告错误。例如：

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
    ...
    /*
     * Provide the implementation of
     * OnAddGeofencesResultListener.onAddGeofencesResult.
     * Handle the result of adding the geofences
     */
    @Override
    public void onAddGeofencesResult(
        int statusCode, String[] geofenceRequestIds) {
        // If adding the geofences was successful
        if (LocationStatusCodes.SUCCESS == statusCode) {
            /*
             * Handle successful addition of geofences here.
             * You can send out a broadcast intent or update the U
             * geofences into the Intent's extended data.
             */
        } else {
            // If adding the geofences failed
            /*
             * Report errors here.
             * You can log the error using Log.e() or update
            */
        }
    }
}

```

```
        * the UI.  
    */  
}  
// Turn off the in progress flag and disconnect the client  
mInProgress = false;  
mLocationClient.disconnect();  
}  
...  
}
```

## 处理断开连接

某些情况下，Location Services可能会在你调用disconnect()方法之前断开连接。为了处理这种情况，你需要实现onDisconnected()方法。在这个方法里面，设置请求状态标志位来表示这个请求已经不处于进程中，然后删除这个client：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks,  
    OnConnectionFailedListener,  
    OnAddGeofencesResultListener {  
    ...  
    /*  
     * Implement ConnectionCallbacks.onDisconnected()  
     * Called by Location Services once the location client is  
     * disconnected.  
     */  
    @Override  
    public void onDisconnected() {  
        // Turn off the request flag  
        mInProgress = false;  
        // Destroy the current location client  
        mLocationClient = null;  
    }  
    ...  
}
```

## 处理连接错误

在处理正常的回调函数之外，你还得提供一个回调函数来处理连接出现错误的情况。这个回调函数重用了前面在检查Google Play service的时候用到的DialogFragment类。它还可以重用之前在onActivityResult()方法里用来接收当用户和错误对话框交互时产生的结果用到的代码。下面的代码展示了如何实现回调函数：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks,  
    OnConnectionFailedListener,  
    OnAddGeofencesResultListener {  
    ...  
    // Implementation of OnConnectionFailedListener.onConnectionFa  
    @Override  
    public void onConnectionFailed(ConnectionString connectionResu  
        // Turn off the request flag
```

```
mInProgress = false;
/*
 * If the error has a resolution, start a Google Play service
 * activity to resolve it.
*/
if (connectionResult.hasResolution()) {
    try {
        connectionResult.startResolutionForResult(
            this,
            CONNECTION_FAILURE_RESOLUTION_REQUEST);
    } catch (SendIntentException e) {
        // Log the error
        e.printStackTrace();
    }
    // If no resolution is available, display an error dialog
} else {
    // Get the error code
    int errorCode = connectionResult.getErrorCode();
    // Get the error dialog from Google Play services
    Dialog errorDialog = GooglePlayServicesUtil.getErrorDialog(
        errorCode,
        this,
        CONNECTION_FAILURE_RESOLUTION_REQUEST);
    // If Google Play services can provide an error dialog
    if (errorDialog != null) {
        // Create a new DialogFragment for the error dialog
        ErrorDialogFragment errorFragment =
            new ErrorDialogFragment();
        // Set the dialog in the DialogFragment
        errorFragment.setDialog(errorDialog);
        // Show the error dialog in the DialogFragment
        errorFragment.show(
            getSupportFragmentManager(),
            "Geofence Detection");
    }
}
...
}
```

# 处理地理围栏触发事件

当Location Services探测到用户进入或者退出一个地理围栏，它会发送一个Intent，这个Intent就是

## 定义一个IntentService

下面的代码展示了如何定义一个当一个地理围栏触发事件出现的时候发送通知。当用户点击这个通知，这个应用的主界面出现：

```
public class ReceiveTransitionsIntentService extends IntentService
{
    ...
    /**
     * Sets an identifier for the service
     */
    public ReceiveTransitionsIntentService() {
        super("ReceiveTransitionsIntentService");
    }
    /**
     * Handles incoming intents
     * @param intent The Intent sent by Location Services. This
     * Intent is provided
     * to Location Services (inside a PendingIntent) when you call
     * addGeofences()
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // First check for errors
        if (LocationClient.hasError(intent)) {
            // Get the error code with a static method
            int errorCode = LocationClient.getErrorCode(intent);
            // Log the error
            Log.e("ReceiveTransitionsIntentService",
                  "Location Services error: " +
                  Integer.toString(errorCode));
        }
        /*
         * You can also send the error code to an Activity or
         * Fragment with a broadcast Intent
         */
        /*
         * If there's no error, get the transition type and the ID
         * of the geofence or geofences that triggered the transit
         */
        } else {
            // Get the type of transition (entry or exit)
            int transitionType =
                LocationClient.getGeofenceTransition(intent);
            // Test that a valid transition was reported
            if (
                (transitionType == Geofence.GEOFENCE_TRANSITION_ENTER
                 ||
                 transitionType == Geofence.GEOFENCE_TRANSITION_EXIT)
            ) {
```

```
        List <Geofence> triggerList =
            getTriggeringGeofences(intent);

        String[] triggerIds = new String[geofenceList.size]

        for (int i = 0; i < triggerIds.length; i++) {
            // Store the Id of each geofence
            triggerIds[i] = triggerList.get(i).getRequestId();
        }
        /*
         * At this point, you can store the IDs for further
         * display them, or display the details associated
         * with them.
        */
    }
    // An invalid transition was reported
} else {
    Log.e("ReceiveTransitionsIntentService",
        "Geofence transition error: " +
        Integer.toString()transitionType));
}
...
}
```

## 在manifest里面设置IntentService

为了在系统里面申明这个IntentService，在manifest里面添加一个<service>元素即可。例如：

```
<service
    android:name="com.example.android.location.ReceiveTransitionsI
    android:label="@string/app_name"
    android:exported="false">
</service>
```

注意你不必为这个service设置intent filters，因为它只接收特定的intent。这些地理围栏触发事件的intent是如何被创建的，请参看[发送监视请求](#)这一课。

## 停止地理围栏监视

要停止地理围栏监视，你要移除这些地理围栏。你可以移除特定的某个地理围栏集合或者移除与某个[PendingIntent](#)相关所有的地理围栏。这个过程与添加地理围栏类似。第一个操作就是获取一个移除请求的location client，然后使用这个client来生成请求。

Location Services在它完成移除地理围栏这个过程的时候调用的回调函数定义在[LocationClient.OnRemoveGeofencesResultListener](#)这个接口里面。在你的类里面申明这个接口，然后为它的两个方法添加定义：

### [onRemoveGeofencesByPendingIntentResult\(\)](#)

- 这个回调函数是Location Services完成移除所有地理围栏的请求时调用的，它是由[removeGeofences\(PendingIntent, LocationClient.OnRemoveGeofencesResultListener\)](#)方法生成的。

### [onRemoveGeofencesByRequestIdsResult\(List, LocationClient.OnRemoveGeofencesResultListener\)](#)

- 这个回调函数是Location Services完成移除特定地理围栏ID集合的地理围栏的请求时调用的，它由[removeGeofences\(List, LocationClient.OnRemoveGeofencesResultListener\)](#)方法生成的。

这些实现的代码将在下一个代码区域出现。

## 移除所有的地理围栏

因为移除地理围栏使用了添加地理围栏时的一些方法，你只需要添加另外几种请求类型即可：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    // Enum type for controlling the type of removal requested
    public enum REQUEST_TYPE { ADD, REMOVE_INTENT }
    ...
}
```

在连接上Location Services的时候启动移除的请求。如果连接失败，那么[onConnected\(\)](#)方法就不会被调用，请求也就停止了。下面的代码就展示了如何启动这个请求：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    /**
     * Start a request to remove geofences by calling
     * LocationClient.connect()
     */
}
```

```

public void removeGeofences(PendingIntent requestIntent) {
    // Record the type of removal request
    mRequestType = REMOVE_INTENT;
    /*
     * Test for Google Play services after setting the request
     * If Google Play services isn't present, the request can
     * restarted.
    */
    if (!servicesConnected()) {
        return;
    }
    // Store the PendingIntent
    mGeofenceRequestIntent = requestIntent;
    /*
     * Create a new location client object. Since the current
     * activity class implements ConnectionCallbacks and
     * OnConnectionFailedListener, pass the current activity o
     * as the listener for both parameters
    */
    mLocationClient = new LocationClient(this, this, this);
    // If a request is not already underway
    if (!mInProgress) {
        // Indicate that a request is underway
        mInProgress = true;
        // Request a connection from the client to Location Se
        mLocationClient.connect();
    } else {
        /*
         * A request is already underway. You can handle
         * this situation by disconnecting the client,
         * re-setting the flag, and then re-trying the
         * request.
        */
    }
}
...
}

```

当Location Services调用这个函数的时候就表明连接已经打开，可以进行移除所有地理围栏的请求了。在这个请求完成之后就可以断开连接了。例如：

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    /**
     * Once the connection is available, send a request to remove
     * Geofences. The method signature used depends on which type
     * remove request was originally received.
    */
    private void onConnected(Bundle dataBundle) {
        /*

```

```

        * Choose what to do based on the request type set in
        * removeGeofences
        */
    switch (mRequestType) {
        ...
        case REMOVE_INTENT :
            mLocationClient.removeGeofences(
                mGeofenceRequestIntent, this);
            break;
        ...
    }
    ...
}

```

对`removeGeofences(PendingIntent, LocationClient.OnRemoveGeofencesResultListener)`会直接返回，而移除地理围栏的请求的结果要等到Location Services 调用`onRemoveGeofencesByPendingIntentResult()`方法才会返回。下面的代码展示了如何定义这个方法：

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    /**
     * When the request to remove geofences by PendingIntent return
     * handle the result.
     *
     * @param statusCode the code returned by Location Services
     * @param requestIntent The Intent used to request the removal.
     */
    @Override
    public void onRemoveGeofencesByPendingIntentResult(int statusCode,
        PendingIntent requestIntent) {
        // If removing the geofences was successful
        if (statusCode == LocationStatusCodes.SUCCESS) {
            /*
             * Handle successful removal of geofences here.
             * You can send out a broadcast intent or update the UI
             * geofences into the Intent's extended data.
            */
        } else {
            // If adding the geocodes failed
            /*
             * Report errors here.
             * You can log the error using Log.e() or update
             * the UI.
            */
        }
    }
    /*
     * Disconnect the location client regardless of the
     * request status, and indicate that a request is no
    */
}

```

```
    * longer in progress
    */
    mInProgress = false;
    mLocationClient.disconnect();
}
...
}
```

## 移除特定的地理围栏

移除个别地理围栏和地理围栏集合与移除所有地理围栏的过程类似。为了确定你要移除的地理围栏，需要将他们的地理围栏ID存储到一个字符串列表里面。将这个列表传给removeGeofences方法。这个方法接下来就可以启动这个移除过程了。

开始的时候要添加一个请求类型来申明这是一个删除里一个列表里面的地理围栏请求。同时还要全局变量来保存地理围栏的ID列表：

```
...
// Enum type for controlling the type of removal requested
public enum REQUEST_TYPE {ADD, REMOVE_INTENT, REMOVE_LIST}
// Store the list of geofence Ids to remove
String<List> mGeofencesToRemove;
```

接下来，定义一个你要移除的地理围栏列表。例如，下面的代码就移除了地理围栏ID为1的地理围栏：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
...
    List<String> listOfGeofences =
        Collections.singletonList("1");
    removeGeofences(listOfGeofences);
...
}
```

下面的代码定义了removeGeofences()方法：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
...
/***
 * Start a request to remove monitoring by
 * calling LocationClient.connect()
 *
 */
public void removeGeofences(List<String> geofenceIds) {
    // If Google Play services is unavailable, exit
```

```

// Record the type of removal request
mRequestType = REMOVE_LIST;
/*
 * Test for Google Play services after setting the request
 * If Google Play services isn't present, the request can
 * restarted.
*/
if (!servicesConnected()) {
    return;
}
// Store the list of geofences to remove
mGeofencesToRemove = geofenceIds;
/*
 * Create a new location client object. Since the current
 * activity class implements ConnectionCallbacks and
 * OnConnectionFailedListener, pass the current activity o
 * as the listener for both parameters
*/
mLocationClient = new LocationClient(this, this, this);
// If a request is not already underway
if (!mInProgress) {
    // Indicate that a request is underway
    mInProgress = true;
    // Request a connection from the client to Location Se
    mLocationClient.connect();
} else {
    /*
     * A request is already underway. You can handle
     * this situation by disconnecting the client,
     * re-setting the flag, and then re-trying the
     * request.
    */
}
}
...
}

```

当Location Services 调用这个回调函数说明这个连接成功，可以进行移除一个列表的地理围栏请求了。完成请求后就可以断开连接。例如：

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    private void onConnected(Bundle dataBundle) {
        ...
        switch (mRequestType) {
        ...
        // If removeGeofencesById was called
        case REMOVE_LIST :
            mLocationClient.removeGeofences(
                mGeofencesToRemove, this);
    }
}

```

```
        break;
    ...
}
...
}
...
}
```

定义 [onRemoveGeofencesByRequestIdsResult\(\)](#)方法的实现。Location Services 调用这个方法的时候说明移除一个列表的地理围栏的请求已经完成了。在这个方法里面，检测得到的状态码并作出对应的动作：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {

    ...
    /**
     * When the request to remove geofences by IDs returns, handle
     * result.
     *
     * @param statusCode The code returned by Location Services
     * @param geofenceRequestIds The IDs removed
     */
    @Override
    public void onRemoveGeofencesByRequestIdsResult(
        int statusCode, String[] geofenceRequestIds) {
        // If removing the geocodes was successful
        if (LocationStatusCodes.SUCCESS == statusCode) {
            /*
             * Handle successful removal of geofences here.
             * You can send out a broadcast intent or update the UI
             * geofences into the Intent's extended data.
             */
        } else {
            // If removing the geofences failed
            /*
             * Report errors here.
             * You can log the error using Log.e() or update
             * the UI.
             */
        }
        // Indicate that a request is no longer in progress
        mInProgress = false;
        // Disconnect the location client
        mLocationClient.disconnect();
    }
    ...
}
```

你可以将地理围栏同其他位置感知的特性结合起来，比如周期性的位置更新或者用户的活动识别。

下一课，[识别用户的活动状态](#)将会告诉你如何请求和接受活动更新。Location Services 会以一个正常的频率向你发送当前用户的身体活动信息。基于这些信息，你可以改变你的应用的一些行为。；例如，当你探测到用户由驾驶改为步行的时候，你可以把应用的信息更新频率设置为更长的时间。

编写:[penkzhou](#)

校对:

# 识别用户的当下活动

活动识别会去探测用户当前的身体活动，比如步行，驾驶以及站立。通过一个不同于请求位置更新或者地理围栏的活动识别client来请求用户活动更新，但是请求方式是类似的。根据你设置的更新频率，Location Services会返回包含一个或者多个活动以及它们出现对应的概率的反馈信息。这一课将会向你展示如何从Location Services请求活动识别更新。

# 请求活动识别更新

从Location Services请求活动识别更新的过程与请求周期性的位置更新类似。你通过一个client发送请求，接着Location Services以[PendingIntent](#)的形式将更新数据返回。然而，你在开始之前必须设置好对应的权限。下面的课程将会教你如何设置权限，连接client以及请求更新。

## 设置接收更新数据的权限

一个应用想要获得活动识别数据就必须拥

有[com.google.android.gms.permission.ACTIVITY\\_RECOGNITION](#)权限。为了让你的应用有这个权限，在你的manifest文件里面将如下代码放到<manifest>标签的里面。

```
<uses-permission  
    android:name="com.google.android.gms.permission.ACTIVITY_RECOG
```

活动识别不需要[ACCESS\\_COARSE\\_LOCATION](#)权限和[ACCESS\\_FINE\\_LOCATION](#)权限。

## 检查Google Play Services是否可用

位置服务是Google Play services 中的一部分。由于很难预料用户设备的状态，所以你在尝试连接位置服务之前应该要检测你的设备是否安装了Google Play services安装包。为了检测这个安装包是否被安装，你可以调

用[GooglePlayServicesUtil.isGooglePlayServicesAvailable\(\)](#)，这个方法将会返回一个结果代码。你可以通过查询[ConnectionResult](#)的参考文档中结果代码列表来理解对应的结果代码。如果你碰到了错误，你可以调用[GooglePlayServicesUtil.getErrorDialog\(\)](#)获取本地化的对话框来提示用户采取适当地行为，接着你需要将这个对话框置于一个[DialogFragment](#)中显示。这个对话框可以让用户去纠正这个问题，这个时候Google Services可以将结果返回给你的activity。为了处理这个结果，重写[onActivityResult\(\)](#)即可。

**注意:** 为了让你的应用能够兼容 Android 1.6 之后的版本，用来显示DialogFragment 的必须是FragmentActivity而不是之前的Activity。使用FragmentActivity同样可以调用[getSupportFragmentManager\(\)](#)方法来显示 DialogFragment。

因为你的代码里通常会不止一次地检测Google Play services是否安装，为了方便，可以定义一个方法来封装这种检测行为。下面的代码片段包含了所有检测Google Play services是否安装需要用到的代码：

```
public class MainActivity extends FragmentActivity {  
    ...  
    //全局变量  
    /*  
     * 定义一个发送给Google Play services的请求代码  
     * 这个代码将会在Activity.onActivityResult的方法中返回  
     */  
    private final static int  
        CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;
```

```
...
// 定义一个显示错误对话框的DialogFragment
public static class ErrorDialogFragment extends DialogFragment
    // 表示错误对话框的全局属性
    private Dialog mDialog;
    // 默认的构造函数，将 dialog 属性设为空
    public ErrorDialogFragment() {
        super();
        mDialog = null;
    }
    // 设置要显示的dialog
    public void setDialog(Dialog dialog) {
        mDialog = dialog;
    }
    // 返回一个 Dialog 给 DialogFragment.
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return mDialog;
    }
}
...
/*
 * 处理来自Google Play services 发给FragmentActivity的结果
 *
 */
@Override
protected void onActivityResult(
    int requestCode, int resultCode, Intent data) {
    // 根据请求代码来决定做什么
    switch (requestCode) {
        ...
        case CONNECTION_FAILURE_RESOLUTION_REQUEST :
        /*
         * 如果结果代码是 Activity.RESULT_OK, 尝试重新连接
         *
         */
        switch (resultCode) {
            case Activity.RESULT_OK :
            /*
             * 尝试重新请求
             */
            ...
            break;
        }
        ...
    }
}
...
private boolean servicesConnected() {
    // 检测Google Play services 是否可用
    int resultCode =
        GooglePlayServicesUtil.
            isGooglePlayServicesAvailable(this);
    // 如果 Google Play services 可用
    if (ConnectionResult.SUCCESS == resultCode) {
```

```

        // 在 debug 模式下，记录程序日志
        Log.d("Location Updates",
              "Google Play services is available.");
        // Continue
        return true;
    // 因为某些原因Google Play services 不可用
} else {
    // 获取error code
    int errorCode = connectionResult.getErrorCode();
    // 从Google Play services 获取 error dialog
    Dialog errorDialog = GooglePlayServicesUtil.getErrorDialog(
        errorCode,
        this,
        CONNECTION_FAILURE_RESOLUTION_REQUEST);

    // 如果 Google Play services可以提供一个error dialog
    if (errorDialog != null) {
        // 为这个error dialog 创建一个新的DialogFragment
        ErrorDialogFragment errorFragment =
            new ErrorDialogFragment();
        // 在DialogFragment中设置dialog
        errorFragment.setDialog(errorDialog);
        // 在DialogFragment中显示error dialog
        errorFragment.show(getSupportFragmentManager(),
                           "Geofence Detection");
    }
}
...
}

```

下面的代码片段使用了这个方法来检查Google Play services是否可用。

## 发送活动更新数据请求

一般的更新数据请求都是从一个实现了Location Services回调函数的[Activity](#) 或者[Fragment](#)发出来的。生成这个请求的过程是一个异步过程，它是在你请求到活动识别client的连接的时候开始的。当这个client连接上的时候，Location Services对调用你对[onConnected\(\)](#)方法的实现。在这个方法里面，你可以发送更新数据的请求到Location Services；这个请求是异步的。一旦你生成这个请求，你就可以断开client的连接了。

这个过程会在下面的代码里面描述。

## 定义 Activity 和 Fragment

定义一个实现如下接口的[FragmentActivity](#) 或者[Fragment](#):

### [ConnectionCallbacks](#)

- 实现当client连接上或者断开连接时Location Services 调用的方法。

### [OnConnectionFailedListener](#)

- 实现当client连接出现错误时Location Services 调用的方法。

例如：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks, OnConnectionFailedListener {  
    ...  
}
```

接下来，定义全局变量。为更新频率定义一个常量，为活动识别client 定义一个变量，为 Location Services 用来发送更新的PendingIntent添加一个变量：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks, OnConnectionFailedListener {  
    ...  
    // Constants that define the activity detection interval  
    public static final int MILLISECONDS_PER_SECOND = 1000;  
    public static final int DETECTION_INTERVAL_SECONDS = 20;  
    public static final int DETECTION_INTERVAL_MILLISECONDS =  
        MILLISECONDS_PER_SECOND * DETECTION_INTERVAL_SECONDS;  
    ...  
    /*  
     * Store the PendingIntent used to send activity recognition e  
     * back to the app  
     */  
    private PendingIntent mActivityRecognitionPendingIntent;  
    // Store the current activity recognition client  
    private ActivityRecognitionClient mActivityRecognitionClient;  
    ...  
}
```

在 onCreate()方法里面，为活动识别client和PendingIntent赋值：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks, OnConnectionFailedListener {  
    ...  
    @Override  
    onCreate(Bundle savedInstanceState) {  
        ...  
        /*  
         * Instantiate a new activity recognition client. Since th  
         * parent Activity implements the connection listener and  
         * connection failure listener, the constructor uses "this"  
         * to specify the values of those parameters.  
         */  
        mActivityRecognitionClient =  
            new ActivityRecognitionClient(mContext, this, this  
        /*  
         * Create the PendingIntent that Location Services uses  
         * to send activity recognition updates back to this app.  
         */  
        Intent intent = new Intent(  
            mContext, ActivityRecognitionIntentService.class);  
        /*
```

```
    * Return a PendingIntent that starts the IntentService.  
    */  
    mActivityRecognitionPendingIntent =  
        PendingIntent.getService(mContext, 0, intent,  
        PendingIntent.FLAG_UPDATE_CURRENT);  
    ...  
}  
...  
}
```

## 开启请求进程

定义一个请求活动识别更新的方法。在这个方法里面，请求到Location Services的连接。你可以在activity的任何地方调用这个方法；这个方法是用来开启请求更新数据的方法链。

为了避免在你的第一个请求结束之前开启第二个请求时出现竞争的情况，你可以定义一个boolean标志位来记录当前请求的状态。在开始请求的时候设置标志位值为true，在请求结束的时候设置标志位为false。

下面的代码展示了如何开始一个更新请求：

```
public class MainActivity extends FragmentActivity implements  
    ConnectionCallbacks, OnConnectionFailedListener {  
    ...  
    // Global constants  
    ...  
    // Flag that indicates if a request is underway.  
    private boolean mInProgress;  
    ...  
    @Override  
    onCreate(Bundle savedInstanceState) {  
        ...  
        // Start with the request flag set to false  
        mInProgress = false;  
        ...  
    }  
    ...  
    /**  
     * Request activity recognition updates based on the current  
     * detection interval.  
     *  
     */  
    public void startUpdates() {  
        // Check for Google Play services  
  
        if (!servicesConnected()) {  
            return;  
        }  
        // If a request is not already underway  
        if (!mInProgress) {  
            // Indicate that a request is in progress  
            mInProgress = true;  
            // Request a connection to Location Services
```

```

        mActivityRecognitionClient.connect();
    //
    } else {
        /*
         * A request is already underway. You can handle
         * this situation by disconnecting the client,
         * re-setting the flag, and then re-trying the
         * request.
        */
    }
}

...
}

```

下面就实现了[onConnected\(\)](#)方法。在这个方法里面，从Location Services请求活动识别更新。当Location Services 结束对client的连接过程然后调用[onConnected\(\)](#)方法时，这个更新请求就会直接被调用：

```

public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
    /**
     * Called by Location Services once the location client is con-
     *
     * Continue by requesting activity updates.
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        /*
         * Request activity recognition updates using the preset
         * detection interval and PendingIntent. This call is
         * synchronous.
        */
        mActivityRecognitionClient.requestActivityUpdates(
            DETECTION_INTERVAL_MILLISECONDS,
            mActivityRecognitionPendingIntent);
        /*
         * Since the preceding call is synchronous, turn off the
         * in progress flag and disconnect the client
        */
        mInProgress = false;
        mActivityRecognitionClient.disconnect();
    }
    ...
}

```

## 处理断开连接

在某些情况下，Location Services可能会在你调用[disconnect\(\)](#)方法之前断开与活动识别client的连接。为了处理这种情况，实现[onDisconnected\(\)](#)方法即可。在这个方法里面，设置请求标志位来表示这个请求是否有效，并根据这个标志位来删除client：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
    /*
     * Called by Location Services once the activity recognition
     * client is disconnected.
     */
    @Override
    public void onDisconnected() {
        // Turn off the request flag
        mInProgress = false;
        // Delete the client
        mActivityRecognitionClient = null;
    }
    ...
}
```

## 处理连接错误

在处理正常的回调函数之外，你还得提供一个回调函数来处理连接出现错误的情况。这个回调函数重用了前面在检查Google Play service的时候用到的DialogFragment类。它还可以重用之前在onActivityResult()方法里用来接收当用户和错误对话框交互时产生的结果用到的代码。下面的代码展示了如何实现回调函数：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks,
    OnConnectionFailedListener,
    OnAddGeofencesResultListener {
    ...
    // Implementation of OnConnectionFailedListener.onConnectionFailed()
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        // Turn off the request flag
        mInProgress = false;
        /*
         * If the error has a resolution, start a Google Play service
         * activity to resolve it.
         */
        if (connectionResult.hasResolution()) {
            try {
                connectionResult.startResolutionForResult(
                    this,
                    CONNECTION_FAILURE_RESOLUTION_REQUEST);
            } catch (SendIntentException e) {
                // Log the error
                e.printStackTrace();
            }
        }
        // If no resolution is available, display an error dialog
    } else {
        // Get the error code
        int errorCode = connectionResult.getErrorCode();
        // Get the error dialog from Google Play services
    }
}
```

```
        Dialog errorDialog = GooglePlayServicesUtil.getErrorDi
            errorCode,
            this,
            CONNECTION_FAILURE_RESOLUTION_REQUEST);
        // If Google Play services can provide an error dialog
        if (errorDialog != null) {
            // Create a new DialogFragment for the error dialog
            ErrorDialogFragment errorFragment =
                new ErrorDialogFragment();
            // Set the dialog in the DialogFragment
            errorFragment.setDialog(errorDialog);
            // Show the error dialog in the DialogFragment
            errorFragment.show(
                getSupportFragmentManager(),
                "Activity Recognition");
        }
    }
    ...
}
```

# 处理活动更新数据

为了处理Location Services每一个周期发送的[Intent](#)，你可以定义一个[IntentService](#)以及它的[onHandleIntent\(\)](#)方法。 Location Services以[Intent](#)对象的形式返回活动识别更新数据，并使用了你在调用[requestActivityUpdates\(\)](#)方法时产生的[PendingIntent](#)。因为你为这个[PendingIntent](#)提供了一个单独的intent，那么接收这个intent的唯一组件就是[IntentService](#)了。

下面的代码展示了如何来检查活动识别更新数据。

## 定义一个IntentService

首先定义这个类以及它的[onHandleIntent\(\)](#)方法：

```
/**  
 * Service that receives ActivityRecognition updates. It receives  
 * updates in the background, even if the main Activity is not vis  
 */  
public class ActivityRecognitionIntentService extends IntentServic  
    ...  
    /**  
     * Called when a new activity detection update is available.  
     */  
    @Override  
    protected void onHandleIntent(Intent intent) {  
        ...  
    }  
    ...  
}
```

接下啦，在intent里面检查数据。你可以从这个数据里面获取到所有可能的活动列表以及它们对应的概率。下面的代码展示了如何获取可能性最大的活动，活动对应的概率以及它的类型：

```
public class ActivityRecognitionIntentService extends IntentServic  
    ...  
    @Override  
    protected void onHandleIntent(Intent intent) {  
        ...  
        // If the incoming intent contains an update  
        if (ActivityRecognitionResult.hasResult(intent)) {  
            // Get the update  
            ActivityRecognitionResult result =  
                ActivityRecognitionResult.extractResult(intent)  
            // Get the most probable activity  
            DetectedActivity mostProbableActivity =  
                result.getMostProbableActivity();  
            /*  
             * Get the probability that this activity is the  
             * the user's actual activity  
             */  
            int confidence = mostProbableActivity.getConfidence();  
        }  
    }
```

```
/*
 * Get an integer describing the type of activity
 */
int activityType = mostProbableActivity.getType();
String activityName = getNameFromType(activityType);
/*
 * At this point, you have retrieved all the information
 * for the current update. You can display this
 * information to the user in a notification, or
 * send it to an Activity or Service in a broadcast
 * Intent.
*/
...
} else {
/*
 * This implementation ignores intents that don't contain
 * an activity update. If you wish, you can report the
 * errors.
*/
}
...
}
...
}
```

`getNameFromType()` 方法将活动类型转化成了对应的描述性字符串。在一个正式的应用中，你应该从资源文件中去获取字符串而不是使用拥有固定值的变量：

```
public class ActivityRecognitionIntentService extends IntentService
...
/**
 * Map detected activity types to strings
 *@param activityType The detected activity type
 *@return A user-readable name for the type
 */
private String getNameFromType(int activityType) {
    switch(activityType) {
        case DetectedActivity.IN_VEHICLE:
            return "in_vehicle";
        case DetectedActivity.ON_BICYCLE:
            return "on_bicycle";
        case DetectedActivity.ON_FOOT:
            return "on_foot";
        case DetectedActivity.STILL:
            return "still";
        case DetectedActivity.UNKNOWN:
            return "unknown";
        case DetectedActivity.TILTING:
            return "tilting";
    }
    return "unknown";
}
...
```

```
}
```

## 在manifest文件里面添加IntentService

为了让系统识别这个IntentService，你需要在应用的manifest文件里面添加<service>标签：

```
<service
    android:name="com.example.android.location.ActivityRecognition"
    android:label="@string/app_name"
    android:exported="false">
</service>
```

注意你不必为这个服务去设置特定的intent filters，因为它只接受特定的intent。定义Activity和Fragment这一段已经描述了活动更新intent是如何被创建的。

## 停止活动识别更新

停止活动识别更新的过程与开启活动识别更新的过程类似，只要将调用的方法[removeActivityUpdates\(\)](#)换成[requestActivityUpdates\(\)](#)即可。

停止更新的过程使用了你在添加请求更新时使用过的几个方法，开始的时候要为两种操作定义请求类型：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {

    ...
    public enum REQUEST_TYPE { START, STOP }
    private REQUEST_TYPE mRequestType;
    ...
}
```

更改开始请求活动识别更新的代码，在里面使用 START 请求参数：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {

    ...
    public void startUpdates() {
        // Set the request type to START
        mRequestType = REQUEST_TYPE.START;
        /*
         * Test for Google Play services after setting the request
         * If Google Play services isn't present, the proper request
         * can be restarted.
        */
        if (!servicesConnected()) {
            return;
        }
        ...
    }
    ...

    public void onConnected(Bundle dataBundle) {
        switch (mRequestType) {
            case START :
                /*
                 * Request activity recognition updates using the
                 * preset detection interval and PendingIntent.
                 * This call is synchronous.
                */
                mActivityRecognitionClient.requestActivityUpdates(
                    DETECTION_INTERVAL_MILLISECONDS,
                    mActivityRecognitionPendingIntent);
                break;
                ...
            /*
             * An enum was added to the definition of REQUEST_
             * but it doesn't match a known case. Throw an exc
            */
        }
    }
}
```

```
        default :
            throw new Exception("Unknown request type in onCon
            break;
        }
        ...
    }
    ...
}
```

## 开始请求停止更新

定义一个方法来请求停止活动识别更新。在这个方法里面，设置号请求类型，然后向 Location Services发起连接。接着你就可以在activity里面的任何地方调用这个方法了。这样做的目的就是开启停止活动更新的方法链：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {

    ...
    /**
     * Turn off activity recognition updates
     *
     */
    public void stopUpdates() {
        // Set the request type to STOP
        mRequestType = REQUEST_TYPE.STOP;
        /*
         * Test for Google Play services after setting the request
         * If Google Play services isn't present, the request can
         * restarted.
        */
        if (!servicesConnected()) {
            return;
        }
        // If a request is not already underway
        if (!mInProgress) {
            // Indicate that a request is in progress
            mInProgress = true;
            // Request a connection to Location Services
            mActivityRecognitionClient.connect();
        }
        /*
        } else {
            /*
             * A request is already underway. You can handle
             * this situation by disconnecting the client,
             * re-setting the flag, and then re-trying the
             * request.
            */
        }
        ...
    }
    ...
}
```

在onConnected()方法里面，如果请求参数类型是 STOP，则调用 removeActivityUpdates() 方法。将你之前用来开启更新进程的PendingIntent作为一个参数传给removeActivityUpdates()方法：

```
public class MainActivity extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
    public void onConnected(Bundle dataBundle) {
        switch (mRequestType) {
            ...
            case STOP :
                mActivityRecognitionClient.removeActivityUpdates(
                    mActivityRecognitionPendingIntent);
                break;
            ...
        }
        ...
    }
    ...
}
```

你不需要改变你对onDisconnected()方法和onConnectionFailed()方法的实现，因为这些方法不依赖这些请求类型。

现在你已经拥有了一个实现了活动识别的应用的基本架构了。你还可以与其他几个基于地理位置的特征进行结合。

编写:[penkzhou](#)

校对:

# 使用模拟位置进行测试

当你在测试一个使用Location Services基于地理位置的应用时，你是不需要把你的设备从一个地方移动到另一个地方来产生位置数据的。你可以将Location Services设置成模拟模式。在这个模式里面，你可以发送模拟位置给Location Services，然后Location Services再将这些数据发送给位置client。在模拟模式里面，Location Services也可以使用模拟位置对象来触发地理围栏。

使用模拟位置有以下几个优点：

- 模拟位置可以让你创建特定的模拟数据，而不需要你移动你的设备到特定的地方来获取接近的数据。
- 因为模拟位置来源于Location Services，它们可以测试你处理地理位置代码的每一个部分。而且，因为你可以从你的正式版应用之外发送模拟数据，那么你就不必在发布你的应用之前禁用或者删掉测试代码。
- 因为你不必通过移动设备来产生测试位置，那你就可以使用模拟器来测试应用了。

使用模拟位置最好的方式就是从一个单独的模拟位置提供应用发送模拟位置数据。这一课就包括了一个位置提供应用，你可以下载下来测试你的软件。你也可以更改这个应用来满足你自己的需求。为应用提供测试数据的一些想法也列在[管理测试数据](#)这一块里面。

这个课程接下来的部分就是教你如何开启模拟模式以及如何使用一个位置client来发送模拟数据给Location Services。

注意： 模拟位置对Location Services的活动识别算法没有影响想要了解更多关于活动识别，请参看课程 [识别用户的当下活动](#)。

## 开启模拟模式

一个应用要想在模拟模式下面给Location Services发送模拟位置，那么它必须要设置[ACCESS\\_MOCK\\_LOCATION](#)权限。而且，你必须在测试设备上开启模拟位置选项。要了解如何开启设备的模拟位置选项，请参看开启设备的开发者模式。

为了在Location Services里面开启模拟模式，你需要先连接一个位置client到Location Services，就像之前的课程[接收当前位置信息](#)一样。接着，调用[LocationClient.setMockMode\(true\)](#)方法。一旦你调用了这个方法，Location Services就会关掉它内部的位置提供器，然后只转发你发给它的模拟位置。下面的代码教你如何调用[LocationClient.setMockMode\(true\)](#)方法：

```
// Define a LocationClient object
public LocationClient mLocationClient;
...
// Connect to Location Services
mLocationClient.connect();
...
// When the location client is connected, set mock mode
mLocationClient.setMockMode(true);
```

一旦这个位置client连接上了Location Services，你必须保持这个连接知道你结束发送模拟位置为止。一旦你调用[LocationClient.disconnect\(\)](#)这个方法，Location Services便会开始启用它的内部位置提供器。在位置client连接的时候调用[LocationClient.setMockMode\(false\)](#)方法就可以关掉模拟模式了。

## 发送模拟位置

一旦你设置好了模拟模式，你就可以创建模拟位置对象了，然后就可以将它们发送给 Location Services。接着，Location Services 又会把这些模拟位置发送给连接的位置 clients。 Location Services 还可以使用模拟位置来控制地理围栏的触发。

要创建一个新的模拟位置，你要用你的测试数据创建一个新的位置对象。你还需要将提供者的值设为 flp，接着 Location Services 把这些信息放到位置对象里面，然后发送出去。下面的代码展示了如何创建一个新的模拟位置：

```
private static final String PROVIDER = "flp";
private static final double LAT = 37.377166;
private static final double LNG = -122.086966;
private static final float ACCURACY = 3.0f;
...
/*
 * From input arguments, create a single Location with provider
 * "flp"
 */
public Location createLocation(double lat, double lng, float accuracy) {
    // Create a new Location
    Location newLocation = new Location(PROVIDER);
    newLocation.setLatitude(lat);
    newLocation.setLongitude(lng);
    newLocation.setAccuracy(accuracy);
    return newLocation;
}
...
// Example of creating a new Location from test data
Location testLocation = createLocation(LAT, LNG, ACCURACY);
```

在模拟模式里面，你需要使用[LocationClient.setMockLocation\(\)](#)方法来发送模拟位置给 Location Services。例如：

```
mLocationClient.setMockLocation(testLocation);
```

Location Services 将这个模拟位置设为当前位置，接着这个位置会在下一个位置更新来的时候被送出去。如何这个新的模拟位置进入了一个地理围栏，Location Services 会触发这个地理围栏的。

# 运行模拟位置提供应用

这一部分包含了这个模拟位置提供应用的总体概览，然后给你一些使用这个应用测试你自己的代码的一些指导。

## 总体概览

这个模拟位置提供应用从后台运行的已经启动的一个服务发送模拟位置对象给Location Services。通过使用一个已经启动服务，这个应用可以即使在主界面因为系统配置改变被销毁的前提下保持运行状态。通过使用一个后台线程，这个服务可以执行长时的测试而不会阻塞UI主线程。

这个应用启动的界面可以让你控制发送的模拟数据类型。你有以下可选项：

### Pause before test

- 这个参数可以设置应用在开始发送测试数据给Location Services之前要等待的秒数。这个间隔可以允许你在测试开始之前从模拟位置提供应用跳转至当前测试应用。

### Send interval

- 这个参数可以设置模拟位置发送周期。你可以参考下面的[测试小技巧](#)来了解更多发送周期的设置。

### Run once

- 从正常模式转换至模拟模式，运行完测试数据之后，又转换回正常模式，接着便终结服务。

### Run continuously

- 从正常模式转换至模拟模式，然后无期限的运行测试数据。后台线程和启动的服务会一直运行下去，即便主界面被销毁。

### Stop test

- 如果处于测试中，那么这个测试会被终止，否则会发回一个警告信息。启动的服务会从模拟模式转回正常模式，然后自己停止自己。这个操作也会停掉后台线程。

在这些选项之外，这个应用还提供了两种状态显示：

### App status

- 显示这个应用相关的生命周期信息。

### Connection status

- 显示这个连接的位置client相关的信息。会

在这个启动的服务运行的时候，它还会发送测试状态的通知。这些通知可以让你看到即便应用不在前台的时候也能知道它的状态更新。当你点击这些通知的时候，主界面会回到前台来。

## 使用模拟位置提供应用来测试

测试来自模拟位置提供应用的测试模拟位置数据：

1. 在已经安装好了Google Play Services的设备上安装模拟位置提供应用。Location Services是Google Play services的一部分。
2. 在设备上，开启模拟位置选项。要了解如何操作，请参看如何开启设备开发者模式。
3. 从桌面启动应用，然后选择你要设置的选项。
4. 除非你删掉这个pause interval这个特征，要不然应用会暂停几秒钟，然后开始发生模拟位置数据给Location Services。
5. 运行你要测试的应用。在模拟位置提供应用运行的时候，你测试的应用接收的时模拟位置而不是真实地位置。
6. 你可以在模拟应用测试到一半的时候点击停止测试将模式从模拟转换至真实位置。这个操作会强制启动的服务去停掉模拟模式，然后自己停掉自己。当服务自己停掉自己之后，后台线程也会被销毁。

# 测试小贴士

下面的教程包含了创建模拟位置数据以及使用模拟位置数据的一些小贴士。

## 选择一个发送周期

每一个位置提供者在为有Location Services发送的位置服务时都有自己的更新频率。例如，GPS最快的频率也是一秒钟一次更新，WiFi的更新频率最快是5秒钟一次。这些周期时间是真实位置里面的处理周期，但是你在使用模拟位置的时候你需要设置好这些。例如，你的频率不能超过一秒一次。如果你在室内测试，这说明你很依赖WiFi，那么你应该将频率设为5秒一次。

## 模拟速度

为了模拟一个真实设备的速度，缩短或者加长两个连续位置之间的距离。例如，通过每秒改变设备位置88英尺来模拟汽车驾驶，因为这样算出来的时速是60英里。作为比较，通过每秒改变设备位置1.5英尺来模拟跑步，因为换算成时速就是3英里。

## 计算位置数据

通过搜索，你可以找到很多计算指定距离的位置经纬度和两点之间的距离的小程序。事实上，Location类提供了两个计算位置距离的方法：

### [distanceBetween\(\)](#)

- 计算两个已知经纬度的地点之间的距离的静态方法。

### [distanceTo\(\)](#)

- 给定一个地点，返回到另一个地点的距离。

## 地理围栏测试

当你的测试一个使用地理围栏探测的应用时，使用反应不同运动模式的测试数据，这些模式包括步行，骑行，驾驶，火车。对于慢的运动模式，可以对位置做较小的更改；相反，对于快的运动模式，可以对位置做较大的更改。

## 管理测试数据

这一课里面的模拟位置提供应用以常量的形式包含了测试经纬度，数据精度。你可能想以其他形式来组织数据：

### XML

- 将位置数据保存到XML文件里面。这样将代码和数据分离开，你可以很容易更改数据了。

### Server download

- 将位置数据保存到服务器上面，然后使用应用下载下来。因为数据和应用已经完全分隔开来，你可以无需重建应用就可以更改数据了。你还可以直接在服务器上面更改数据然后影响你的模拟位置。

## Recorded data

- 除了生成测试数据，写一个工具应用来记录你的设备在移动的时候产生的地理位置信息。使用这些记录作为你的测试数据，或者使用这些数据来引导你开发测试数据。例如，记录你在步行的时候设备产生的位置信息，然后用它来创建模拟位置，因为这种数据随着时间有比较合适的改变。

编写:[kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/building-wearables.html>

# 为可穿戴设备创建应用

这些课程将教会你如何在手持应用上构建notification，并且使得这些notification能够自动同步到可穿戴设备上。同样也会教你如何创建直接运行在可穿戴设备上的应用。

有下面几个课程：

## [赋予Notification可穿戴的特性](#)

学习如何构建运行在手持设备的上得notification并且使得他们能够同步到可穿戴设备时有良好的体验。

## [创建可穿戴应用](#)

学习如何构建直接运行在可穿戴设备上的应用。

## [发送与同步数据](#)

学习如何在手持设备与可穿戴设备之间同步数据。

编写:[wangyachen](#) - 校对:

原文: <http://developer.android.com/training/wearables/notifications/index.html>

# 为Notification赋加可穿戴特性

当一部Android手持设备（手机或平板）与Android可穿戴设备连接起来，手持设备能够自动的与可穿戴设备共享Notification。在可穿戴设备上，每个Notification都是以一张新卡片的形式出现在[context stream](#)中的

与此同时，为了给予用户以最佳的体验，你应当为你的Notification增加一些具备可穿戴特性的功能。下面的课程将指导你如何实现同时支持手持设备和可穿戴设备的Notification。

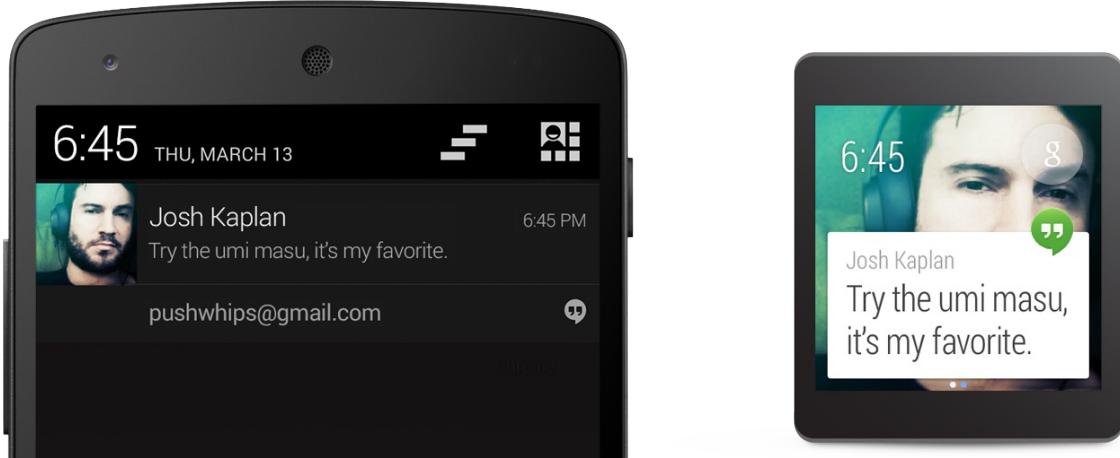


图1. 同时展示在手持设备和可穿戴设备的Notification

# 分集课程

- [创建一个Notification](#)

学习如何应用Android support library创建具备可穿戴特性的Notification。

- [在Notification中接收语音输入](#)

学习在可穿戴式Notification接收到来自用户的语音输入时添加一个action，并且将转录的消息传递给手持设备。

- [为Notification添加显示界面](#)

学习如何为Notification创建附加的页面，使得用户在向左滑动时能看到更多的信息。

- [以stack的方式显示Notification](#)

学习如何以stack的形式显示那些从app中发出的类似的Notification，使得用户能够看到每一个Notification是独立显示，而不是将各种卡片放入卡片流中。

编写:[wangyachen](#) - 校对: [kesenhoo](#)

原文: <http://developer.android.com/training/wearables/notifications/creating.html>

# 创建Notification

为了创建一个手持设备上的并且也能同时发送给可穿戴设备的Notification，需要使用[NotificationCompat.Builder](#)。当你使用这个类创建Notification之后，如何正确展示的工作就交由系统去完成，无论是在手持式设备上还是可穿戴设备上。

**Note:** Notification使用[RemoteViews](#)的话能够自定义layout，并且可穿戴设备上只能够显示文本和icon。但是，你能够通过创建一个运行在可穿戴式设备上的采用了自定义card layout的app去创建Notification。请参考[创建自定义的Notification](#)。

## Import必要的类

在你开始之前，从support library中import关键的类：

```
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.NotificationManagerCompat;
import android.support.v4.app.NotificationCompat.WearableExtender;
```

## 通过Notification Builder创建Notification

[v4 support library](#)能够让你采用最新的特性，诸如放置action button或采用large icon，去实现一个Notification，兼容Android1.6（API level4）及以上。

为了通过support library创建一个Notification，你需要创建一个[NotificationCompat.Builder](#)的实例，然后通过[notify\(\)](#)去激活。例如：

```
int notificationId = 001;
// Build intent for notification content
Intent viewIntent = new Intent(this, ViewEventActivity.class);
viewIntent.putExtra(EXTRA_EVENT_ID, eventId);
PendingIntent viewPendingIntent =
    PendingIntent.getActivity(this, 0, viewIntent, 0);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setContentTitle(eventTitle)
        .setContentText(eventLocation)
        .setContentIntent(viewPendingIntent);

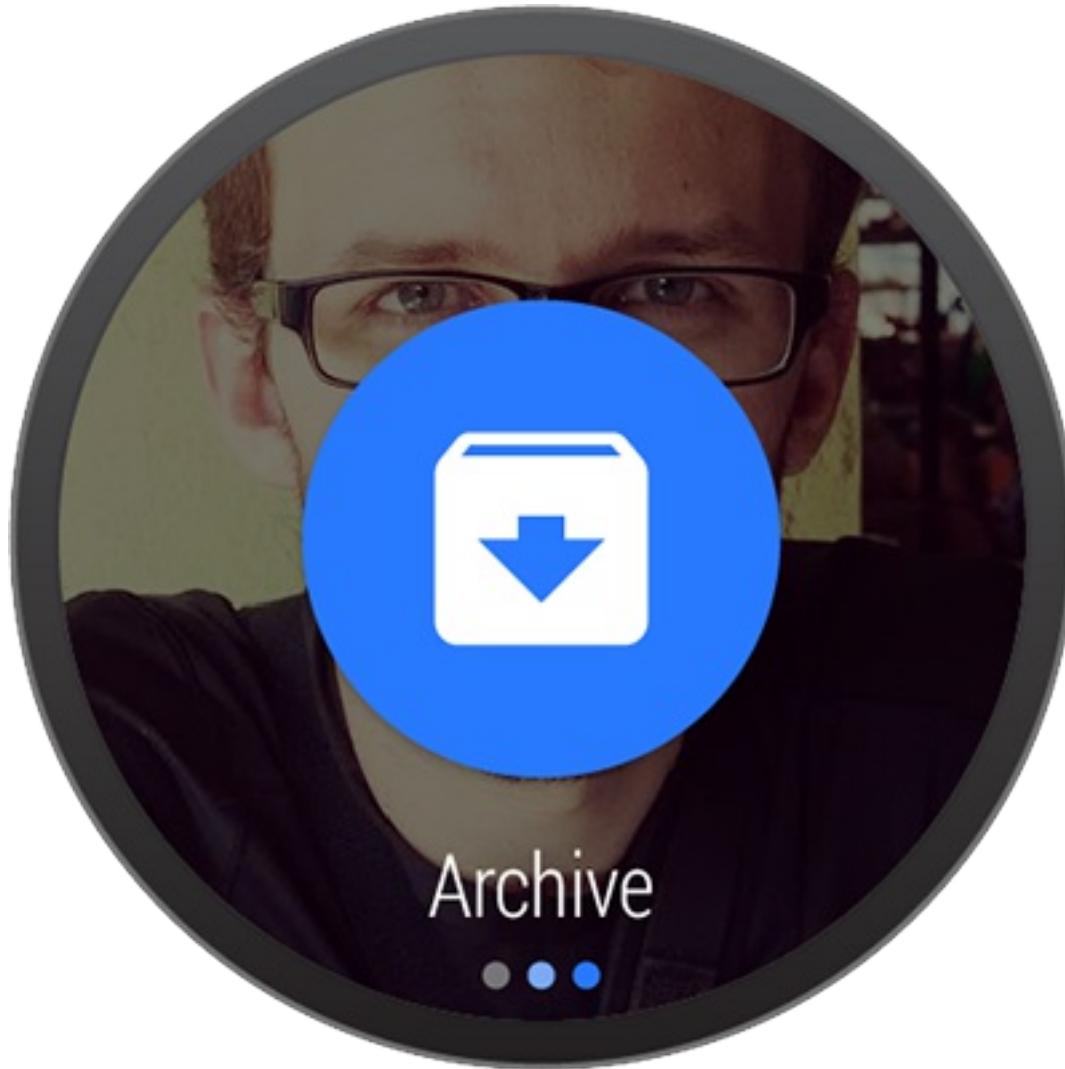
// Get an instance of the NotificationManager service
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);

// Build the notification and issues it with notification manager.
notificationManager.notify(notificationId, notificationBuilder.bui
```

当该Notification出现在手持设备上时，用户能够通过触摸Notification来触发之前通过[setContentIntent\(\)](#)设置的[PendingIntent](#)。当该Notification出现在可穿戴设备上时，用户能够通过向左滑动该Notification显示**Open**的action，点击这个action能够触发响应的Intent，作用在手持设备上。

## 添加Action按钮

除了通过[setContentIntent\(\)](#)定义的主要的action之外，你还可以通过传递一个[PendingIntent](#)给[addAction\(\)](#)的参数，从而添加更多的action。



例如，下面的代码展示了同上面一样的Notification，只不过添加了一个在地图上查看事件位置的action。

```
// Build an intent for an action to view a map
Intent mapIntent = new Intent(Intent.ACTION_VIEW);
Uri geoUri = Uri.parse("geo:0,0?q=" + Uri.encode(location));
mapIntent.setData(geoUri);
PendingIntent mapPendingIntent =
    PendingIntent.getActivity(this, 0, mapIntent, 0);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setContentTitle(eventTitle)
        .setContentText(eventLocation)
        .setContentIntent(viewPendingIntent)
        .addAction(R.drawable.ic_map,
            getString(R.string.map), mapPendingIntent);
```

在手持设备上，action表现为在Notification上附加的一个额外按钮。在可穿戴设备上，action表现为Notification左滑后出现的大按钮。当用户点击action时，能够触发手持设备上对应的intent。

**提示:** 如果你的Notification包含了一个"Reply"的action(例如短信类app)，你可以通过支持直接从Android可穿戴设备返回的语音输入，来加强该功能的使用。更多信息，详见[Receiving Voice Input from a Notification](#)。

## 指定可穿戴设备独有的Actions

如果你想要可穿戴式设备上的action与手持式设备不一样的话，可以使  
用[WearableExtender.addAction\(\)](#)，一旦你通过这种方式添加了action，可穿戴式设备便不会  
显示任何其他通过[NotificationCompat.Builder.addAction\(\)](#)添加的action。这是因为，只有通  
过[WearableExtender.addAction\(\)](#)添加的action才能只在可穿戴设备上显示且不在手持式设备  
上显示。

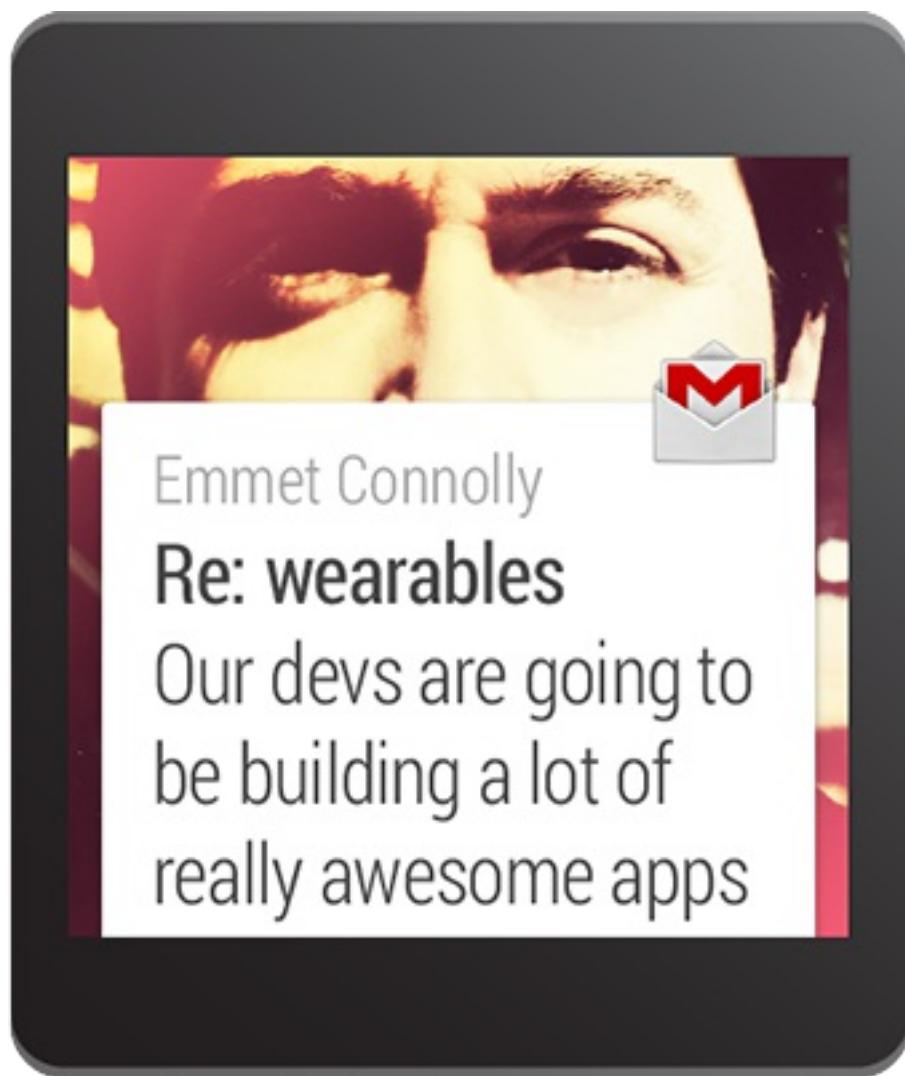
```
// Create an intent for the reply action
Intent actionIntent = new Intent(this, ActionActivity.class);
PendingIntent actionPendingIntent =
    PendingIntent.getActivity(this, 0, actionIntent,
        PendingIntent.FLAG_UPDATE_CURRENT);

// Create the action
NotificationCompat.Action action =
    new NotificationCompat.Action.Builder(R.drawable.ic_action
        getString(R.string.label, actionPendingIntent))
        .build();

// Build the notification and add the action via WearableExtender
Notification notification =
    new NotificationCompat.Builder(mContext)
        .setSmallIcon(R.drawable.ic_message)
        .setContentTitle(getString(R.string.title))
        .setContentText(getString(R.string.content))
        .extend(new WearableExtender().addAction(action))
        .build();
```

## 添加Big View

你可以在Notification中通过添加某种"big view"的style来插入扩展文本。在手持式设备上，用户能够通过扩展的Notification看见big view的内容。在可穿戴式设备上，big view内容是默认可见的。



可以通过对[NotificationCompat.Builder](#)的对象调用[setStyle\(\)](#)，设置参数为[BigTextStyle](#)或[InboxStyle](#)的实例。

比如，下面的代码为事件的Notification添加了一个[NotificationCompat.BigTextStyle](#)的实例，目的是为了包含完整的事件描述(这能够包含比[setContentText\(\)](#)所提供的空间所能容纳的字数更多的文字)。

```
// Specify the 'big view' content to display the long
// event description that may not fit the normal content text.
BigTextStyle bigStyle = new NotificationCompat.BigTextStyle();
bigStyle.bigText(eventDescription);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setLargeIcon(BitmapFactory.decodeResource(
            getResources(), R.drawable.notif_background))
        .setContentTitle(eventTitle)
```

```
.setContentText(eventLocation)
.setContentIntent(viewPendingIntent)
.addAction(R.drawable.ic_map,
        getString(R.string.map), mapPendingIntent)
.setStyle(bigStyle);
```

要注意的是，你可以通过[setLargeIcon\(\)](#)方法为任何Notification添加一张较大的背景图片。更多关于大图片在Notification上的设计，详见[Design Principles of Android Wear](#)。

## 为Notification添加可穿戴特性

如果你需要为Notification添加一些可穿戴的特性设置，比如制定额外的内容页，或者让用户通过语音输入一些文字，那么你可以使用 [NotificationCompat.WearableExtender](#) 来制定这些设置。请使用如下的API：

1. 创建一个[WearableExtender](#)的实例，设置可穿戴独有的Notification的选项。
2. 创建一个[NotificationCompat.Builder](#)的实例，就像本课程先前所说的，设置需要的Notification属性。
3. 执行Notification的[extend\(\)](#)方法，参数是WearableExtender实例。
4. 调用[build\(\)](#)去build一个Notification。

例如，以下代码调用[setHintHideIcon\(\)](#)把app的icon从Notification的卡片上remove掉。

```
// Create a WearableExtender to add functionality for wearables
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender()
    .setHintHideIcon(true);

// Create a NotificationCompat.Builder to build a standard notification
// then extend it with the WearableExtender
Notification notif = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender)
    .setContentText(subject)
    .setSmallIcon(R.drawable.new_mail);
    .extend(wearableExtender)
    .build();
```

这个[setHintHideIcon\(\)](#)方法只是使用[NotificationCompat.WearableExtender](#)构建notification新特性的一个例子。

如果你需要获取可穿戴特性的内容，可以使用相应的get方法，该例子通过调用[getHintHideIcon\(\)](#)去获取当前Notification是否隐藏了icon。

```
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender(notif);
boolean hintHideIcon = wearableExtender.getHintHideIcon();
```

## 交付显示Notification

如果你想要交付显示你的Notification, 请使用[NotificationManagerCompat](#)的API代替[NotificationManager](#):

```
// Get an instance of the NotificationManager service  
NotificationManagerCompat notificationManager =  
    NotificationManagerCompat.from(mContext);  
  
// Issue the notification with notification manager.  
notificationManager.notify(notificationId, notif);
```

如果你使用了framework中的[NotificationManager](#), 那么[NotificationCompat.WearableExtender](#)中的一些特性就会失效, 所以, 请确保使用[NotificationManagerCompat](#).

```
NotificationCompat.WearableExtender wearableExtender =  
    new NotificationCompat.WearableExtender(notif);  
boolean hintHideIcon = wearableExtender.getHintHideIcon();
```

[NotificationCompat.WearableExtender](#)的API允许你为Notification、stack Notification等添加额外的内容页。下面的课程将学习这些特性。

下一课: [在Notification中接收语音输入](#)

编写:[wangyachen](#) - 校对:[kesenhoo](#)

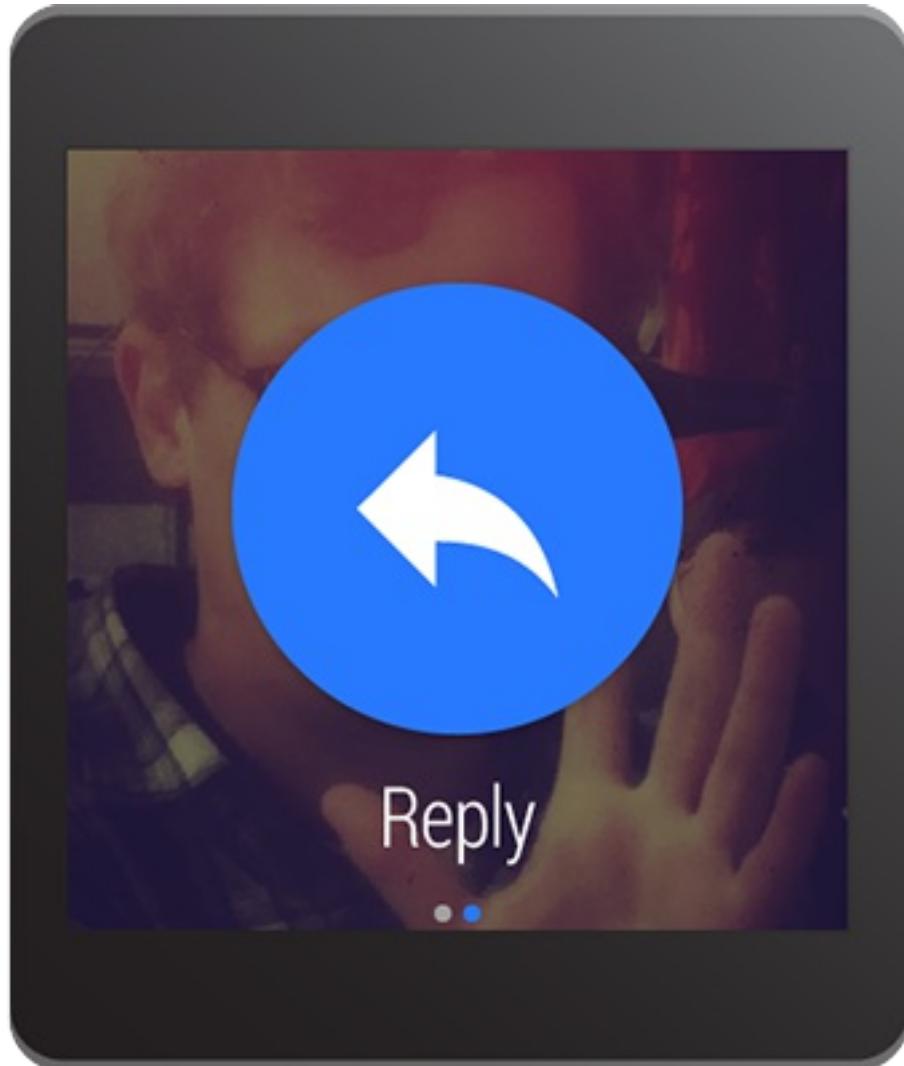
原文:<http://developer.android.com/training/wearables/notifications/voice-input.html>

# 在Notification中接收语音输入

如果你的手持式设备上的Notification包含了一个输入文本的action，比如回复邮件，那么这个action正常情况下应该会调起一个activity让用户进行输入。但是，当这个action出现在可穿戴式设备上时，是没有键盘可以让用户进行输入的，所以你应该让用户的口述一个回复或者通过[RemoteInput](#)预先设定好文本信息。

当用户通过语音或者选择已经存在可用的消息进行回复时，系统会将文本的反馈信息与你指定的Notification中的action中的[Intent](#)进行绑定，并且将该intent发送给你的手持设备中的app。

**Note:** Android模拟器并不支持语音输入。如果使用可穿戴式设备的模拟器的话，可以打开AVD设置中的**Hardware keyboard present**，实现用打字代替语音。



“

i'm biking over there  
right now :... :...:

## 定义语音输入

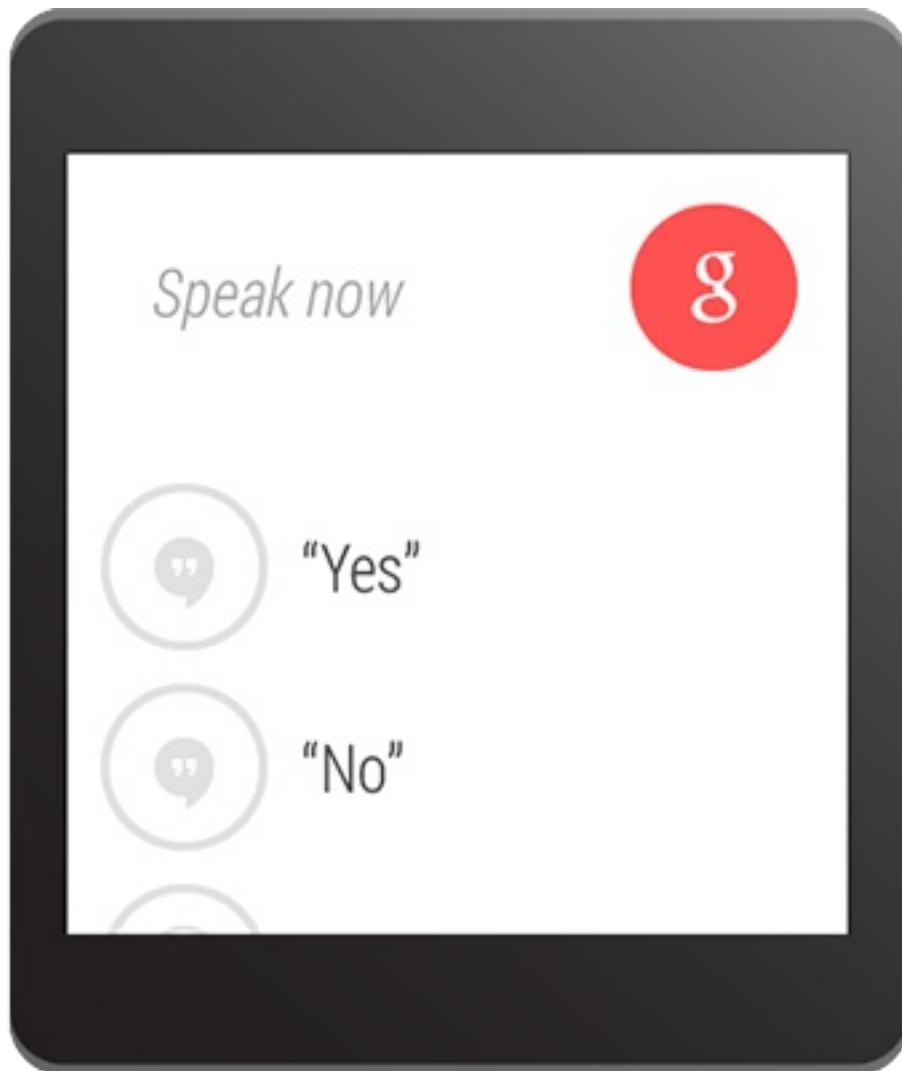
为了创建一个支持语音输入的action，需要创建一个[RemoteInput.Builder](#)的实例，将其加到你的Notification的action中。这个类的构造函数接受一个String类型的参数，该参数的含义是系统用来作为语音输入的key，这个key可以用来在手持设备中检索出所需要的那一次语音输入的内容。

举个例子，下面展示了如何创建一个[RemoteInput](#)对象，并且提供了一个自定义的label用于提示语音输入。

```
// Key for the string that's delivered in the action's intent  
private static final String EXTRA_VOICE_REPLY = "extra_voice_reply"  
  
String replyLabel = getResources().getString(R.string.reply_label)  
  
RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)  
    .setLabel(replyLabel)  
    .build();
```

## 添加预先设定的文本反馈

除了要打开语音输入支持之外，你还可以提供多达5条的文本反馈，这样用户可以直接进行选择实现快速回复。该功能可通过调用[setChoices\(\)](#)并传递一个String数组实现。



举个例子，你可以用resource数组的方式定义这些反馈。

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="reply_choices">
        <item>Yes</item>
        <item>No</item>
        <item>Maybe</item>
    </string-array>
</resources>
```

然后，获得该数组，并将其添加到[RemoteInput](#)中：

```
public static final EXTRA_VOICE_REPLY = "extra_voice_reply";
...
String replyLabel = getResources().getString(R.string.reply_label)
String[] replyChoices = getResources().getStringArray(R.array.reply_choices)

RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)
    .setLabel(replyLabel)
    .setChoices(replyChoices)
    .build();
```

## 添加语音输入作为Notification的action

为了实现设置语音输入，可以把你的[RemoteInput](#)对象通过[addRemoteInput\(\)](#)设置到一个action中。然后你可以将这个action应用到Notification中，例如：

```
// Create an intent for the reply action
Intent replyIntent = new Intent(this, ReplyActivity.class);
PendingIntent replyPendingIntent =
    PendingIntent.getActivity(this, 0, replyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT);

// Create the reply action and add the remote input
NotificationCompat.Action action =
    new NotificationCompat.Action.Builder(R.drawable.ic_reply_
        getString(R.string.label, replyPendingIntent))
        .addRemoteInput(remoteInput)
        .build();

// Build the notification and add the action via WearableExtender
Notification notification =
    new NotificationCompat.Builder(mContext)
        .setSmallIcon(R.drawable.ic_message)
        .setContentTitle(getString(R.string.title))
        .setContentText(getString(R.string.content))
        .extend(new WearableExtender().addAction(action))
        .build();

// Issue the notification
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(mContext);
notificationManager.notify(notificationId, notification);
```

当程序发出这个Notification的时候，用户在wear上左滑便可以看到reply的按钮。

## 接受语音输入作为String值

通过调用[getResultsFromIntent\(\)](#)方法，将返回的值放在"Reply"的action指定的intent中，你便可以在回复的action的intent中指定的activity里，接收到用户转录后的消息。

该方法返回的是一个[Bundle](#)，里面包含了文本反馈的内容，接下来你可以通过查询[Bundle](#)中的内容来获得这条反馈。

**Note:**请不要使用[Intent.getExtras\(\)](#)来获取语音输入的结果，因为语音输入的内容是保存成[ClipData](#)形式的。[getResultsFromIntent\(\)](#)提供了一条很方便的途径来接收字符数组类型的语音信息，并且不需要你自己来处理[ClipData](#)数据。

下面的代码展示了一个接收intent，并且返回语音反馈信息的方法，该方法是依据之前例子中的EXTRA\_VOICE\_REPLY作为key进行检索。

```
/**  
 * Obtain the intent that started this activity by calling  
 * Activity.getIntent() and pass it into this method to  
 * get the associated voice input string.  
 */  
  
private CharSequence getMessageText(Intent intent) {  
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);  
    if (remoteInput != null) {  
        return remoteInput.getCharSequence(EXTRA_VOICE_REPLY);  
    }  
    return null;  
}
```

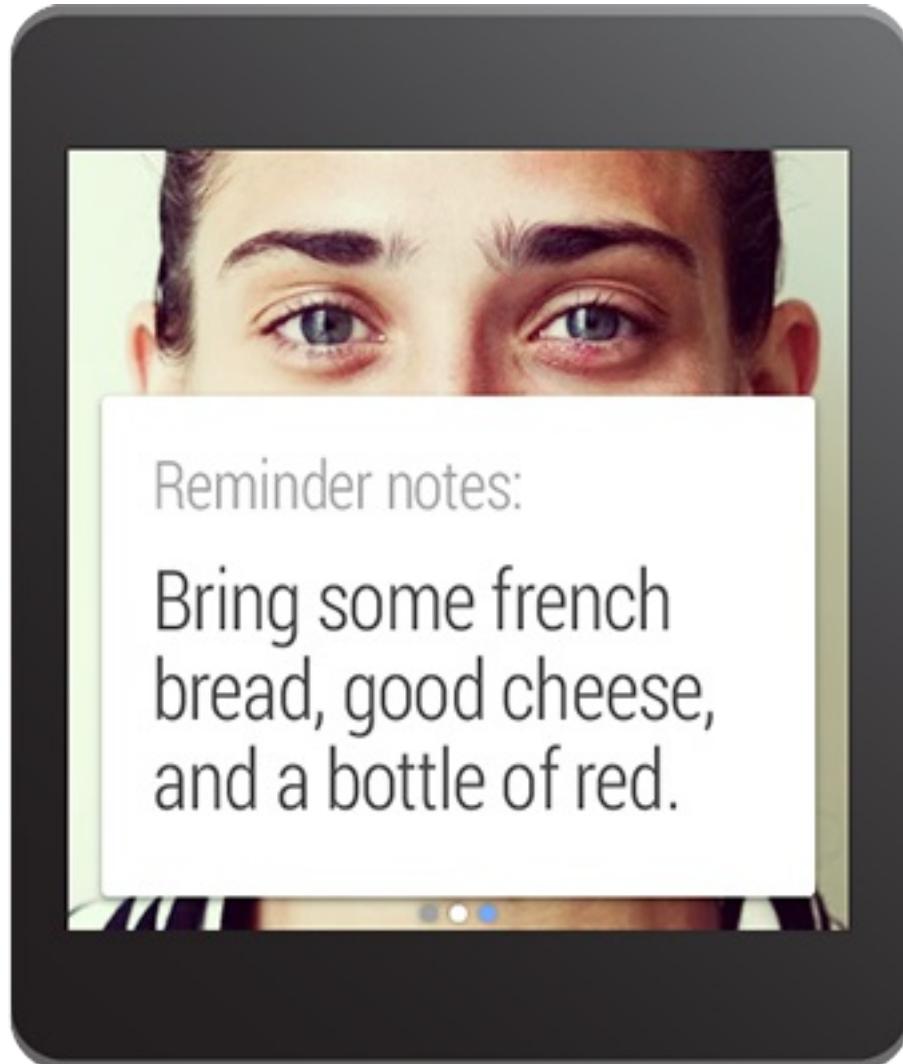
下一课：[为Notification添加显示页面](#)

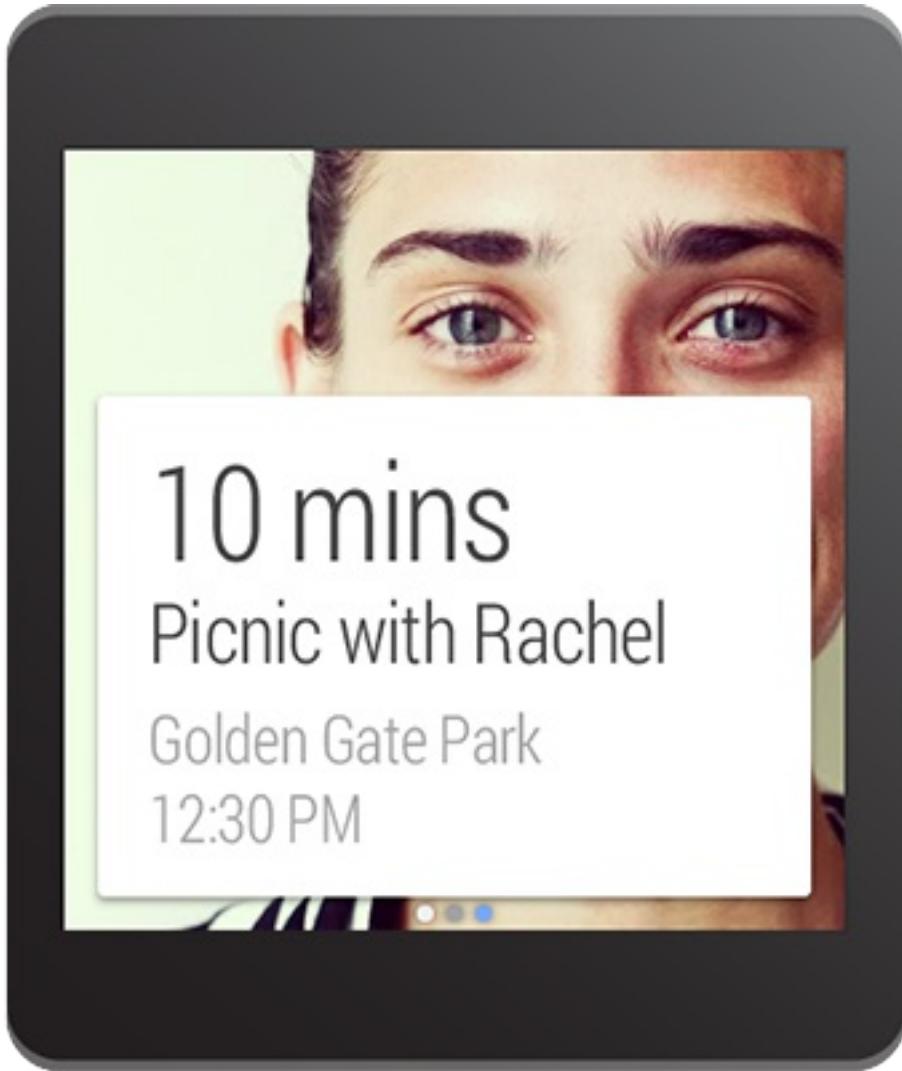
编写:[wangyachen](#) - 校对:

原文:<http://developer.android.com/training/wearables/notifications/pages.html>

# 为Notification添加显示页面

当你想要在不需要用户在他们的手机上打开你的app的情况下，还可以让你表达更多的信息，那么你可以在wear上的Notification中添加一个或更多的页面。





为了创建一个多页的Notification，你需要：

1. 通过[NotificationCompat.Builder](#)创建主Notification（首页），这个首页也是你想要呈现在手持设备上的。
2. 通过[NotificationCompat.Builder](#)为wear添加更多的页面。3.通过[addPage\(\)](#)方法为主Notification应用这些添加的页面，或者通过[addPage\(\)](#)添加一个[Collection](#)的多个页面。

举个例子，以下代码为Notification添加了第二个页面：

```
// Create builder for the main notification
NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.new_message)
        .setContentTitle("Page 1")
        .setContentText("Short message")
        .setContentIntent(viewPendingIntent);

// Create a big text style for the second page
BigTextStyle secondPageStyle = new NotificationCompat.BigTextStyle()
secondPageStyle.setBigContentTitle("Page 2")
    .bigText("A lot of text...");

// Create second page notification
```

```
Notification secondPageNotification =
    new NotificationCompat.Builder(this)
    .setStyle(secondPageStyle)
    .build();

// Add second page with wearable extender and extend the main noti
Notification twoPageNotification =
    new WearableExtender()
        .addPage(secondPageNotification)
        .extend(notificationBuilder)
        .build();

// Issue the notification
notificationManager =
    NotificationManagerCompat.from(this);
notificationManager.notify(notificationId, twoPageNotification);
```

下一课: [以Stack的方式显示Notifications](#)

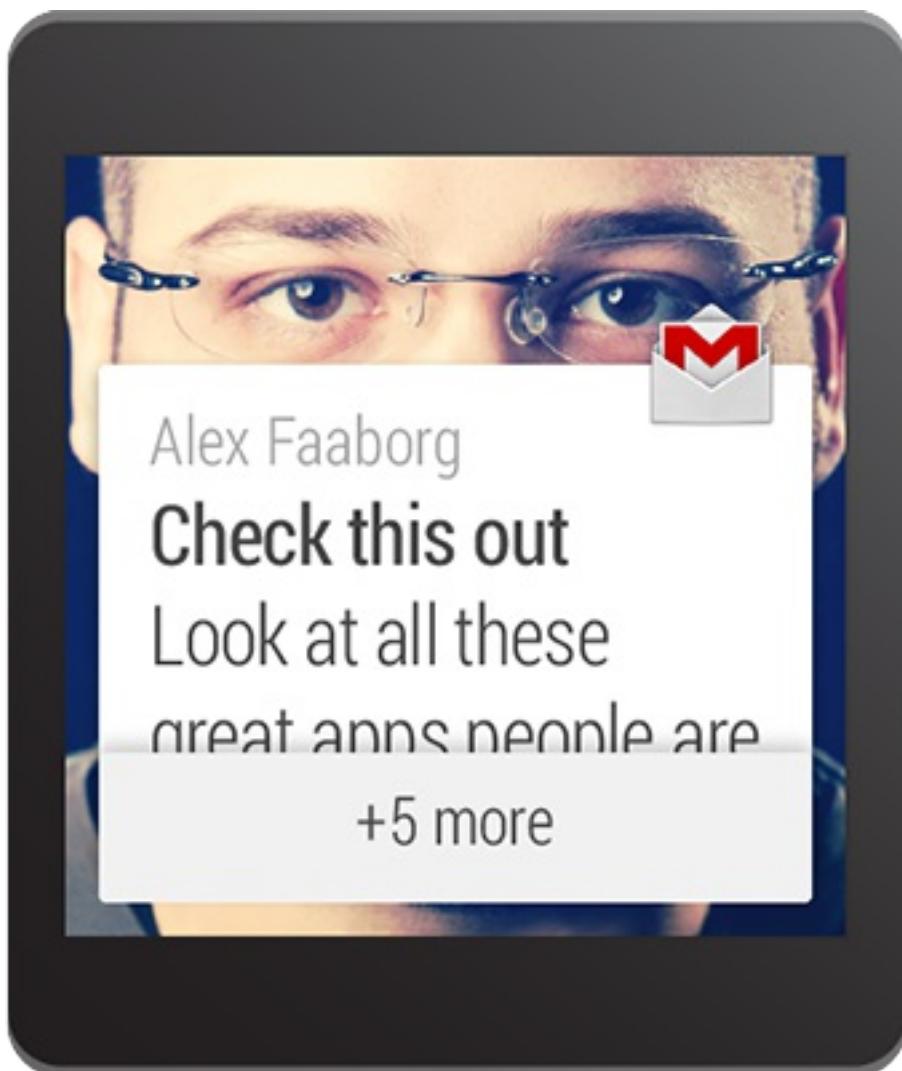
编写:[wangyachen](#) - 校对:[kesenhoo](#)

原文:

# 以Stack的方式显示Notifications

当为手持式设备创建Notification时，你应该将多个相似的Notification合并成一个概括式的Notification。例如，如果你的app创建了一系列接收短信的Notification，你不应该把它们都展示出来，当多于一条短信被接收的时候，用一条Notification提示总结性信息比如"2条新消息"。

尽管如此，概括式的Notification在wear上并不是很有用处，因为用户不可能在wear上还能够阅读每条消息的详细内容(他们必须在手持式设备上打开你的app才能看到更多信息)。所以对wear而言，你应该将所有的Notification都集中起来，以stack的形式进行展示。这个stack展示的时候就像一张卡片，用户可以在上面以扩展的方式分别看到其他的Notification。通过新方法[setGroup\(\)](#)能够实现该功能，同时，还能让你保持手持式设备上显示为一条概括式的Notification。





Alex Faaborg

**Check this out**

Look at all these  
great anns people are

Jeff Chang

**Launch party**

I'm thinking Sky Bar

## 将每个Notification添加到一个群组中

为了创建一个stack，可以对每个想要放入该stack的Notification调用[setGroup\(\)](#)，并且指定群组的key。然后调用[notify\(\)](#)将其发送至wear上。

```
final static String GROUP_KEY_EMAILS = "group_key_emails";

// Build the notification, setting the group appropriately
Notification notif = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender1)
    .setContentText(subject1)
    .setSmallIcon(R.drawable.new_mail);
    .setGroup(GROUP_KEY_EMAILS)
    .build();

// Issue the notification
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);
notificationManager.notify(notificationId1, notif);
```

稍后，当你创建另一个Notification的时候，指定同样的群组key。当你在调用[notify\(\)](#)的时候，这个Notification就会出现在之前那个Notification的同一个stack中，而非新建一张卡片。

```
Notification notif2 = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender2)
    .setContentText(subject2)
    .setSmallIcon(R.drawable.new_mail);
    .setGroup(GROUP_KEY_EMAILS)
    .build();

notificationManager.notify(notificationId2, notif2);
```

在默认的情况下，Notification的排列顺序由你添加的先后顺序决定，最近的Notification会被放置在最顶端。你可以通过[setSortKey\(\)](#)来修改Notification的排顺序。

## 添加概括式Notification

虽然上面介绍了最好将Notification都以stack的形式展示，但是，在手持设备上提供一个概括式的Notification还是很重要的。除了要将Notification放置在同一个stack中，还需要添加一个概括式的Notification，并对其调用[setGroupSummary\(\)](#)即可实现。

该Notification并不会出现在你wear设备上的stack中，只会出现在手持式设备上。



```
Bitmap largeIcon = BitmapFactory.decodeResource(getResources(),
    R.drawable.ic_large_icon);

// Create an InboxStyle notification
Notification summaryNotification = new NotificationCompat.Builder(
    .setContentTitle("2 new messages")
    .setSmallIcon(R.drawable.ic_small_icon)
    .setLargeIcon(largeIcon)
    ..setStyle(new NotificationCompat.InboxStyle()
        .addLine("Alex Faaborg Check this out")
        .addLine("Jeff Chang Launch Party")
        .setBigContentTitle("2 new messages")
        .setSummaryText("johndoe@gmail.com"))
    .setGroup(GROUP_KEY_EMAILS)
    .setGroupSummary(true)
    .build());

notificationManager.notify(notificationId3, summaryNotification);
```

该Notification使用了[NotificationCompat.InboxStyle](#)，这个style能够让你很轻松的创建email或者短信类型的app。你可以对概括式Notification使用这个style，或者[NotificationCompat](#)中定义的其他style，或者不使用任何style也可以。

提示:如果想要和上面截图中一样的设计文本，请参考[Styling with HTML markup](#)和[Styling with Spannables](#)。

概括式Notification能够在不显示在wear的前提下做到影响其他的Notification。当你创建一个概括式Notification时，你可以利用[NotificationCompat.WearableExtender](#)，调用[setBackground\(\)](#)或者[addAction\(\)](#)为wear上的整个stack设置一个背景图片或者一个action。以下代码展示了如何为整个stack设置背景：

```
Bitmap background = BitmapFactory.decodeResource(getResources(),  
        R.drawable.ic_background);  
  
NotificationCompat.WearableExtender wearableExtender =  
    new NotificationCompat.WearableExtender()  
    .setBackground(background);  
  
// Create an InboxStyle notification  
Notification summaryNotificationWithBackground =  
    new NotificationCompat.Builder(mContext)  
    .setContentTitle("2 new messages")  
    ...  
    .extend(wearableExtender)  
    .setGroup(GROUP_KEY_EMAILS)  
    .setGroupSummary(true)  
    .build();
```

下一课：[创建可穿戴的应用](#)

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/index.html>

# 创建可穿戴的应用

可穿戴应用直接运行在穿戴设备上，应用可以直接访问例如传感器与GPU这样的硬件。这些应用和一般的Android应用的基础部分是一致的，只是在设计与可用性还有一些特殊功能上有比较大差异。手持设备与可穿戴设备上的应用主要有下面的一些差异：

- 系统会强制执行超时机制。如果你显示了一个Activity，用户并没有进行操作，设备会进入睡眠状态。当设备唤醒时，穿戴设备会显示主界面而不是你刚才的activity。如果你想要持续的显示一些东西，请使用notification来替代。
- 相比起手持设备的应用，可穿戴应用的界面相对更小，功能也相对更少。他仅仅包含了那些对于可穿戴有意义的功能，这些功能通常是手持设备的一个子集。通常来说，你应该尽可能的把运行操作搬到手持设备上，然后发送操作结果到可穿戴设备。
- 用户不能直接给可穿戴设备安装应用。你需要给手持设备的应用绑定一个可穿戴设备的应用。当用户安装手持设备的应用时，系统会自动安装可穿戴应用。然而，为了开发便利，你还是可以直接安装应用到可穿戴设备。
- 可穿戴应用可以使用大多数的标准Android APIs，除了下面的以外：
  - [android.webkit](#)
  - [android.print](#)
  - [android.app.backup](#)
  - [android.appwidget](#)
  - [android.hardware.usb](#)

在使用某个API之前，你可以通过执行[hasSystemFeature\(\)](#) 来判断功能是否可用。

**Note:** 我们推荐使用Android Studio来开发Android Wear的应用，因为它提供了建立工程，添加库依赖，打包程序等等在ADT上没有的功能。下面的培训课程的前提是假设你已经在使用Android Studio了。

# Lessons

- [创建并执行可穿戴应用\(Creating and Running a Wearable App\)](#)

学习如何创建一个包含了可穿戴与手持应用的Android Studio工程。学习如何在设备或者模拟器上执行程序。

- [创建自定义的布局\(Creating Custom Layouts\)](#)

学习如何为notification与activiyt，创建并显示一个自定义的布局

- [添加语言能力\(Adding Voice Capabilities\)](#)

学习如何使用语音指令启动一个activity，学习如何启动系统语音识别应用来获取用户的语音输入。

- [打包可穿戴应用\(Packaging Wearable Apps\)](#)

学习如何把可穿戴应用打包到手持应用上。这使得系统能够在安装Google Play上的手持应用时自动安装可穿戴应用。

- [通过蓝牙进行调试\(Debugging over Bluetooth\)](#)

学习如何通过蓝牙而不是USB来调试你的可穿戴应用。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/creating.html>

# 创建并执行可穿戴应用

可穿戴应用可以直接运行在可穿戴的设备上。拥有访问类似传感器的硬件权限，还有操作activity，services等权限。

你无法直接发布可穿戴应用到Google Play商城，需要利用手持应用来达到目的。因为可穿戴的设备不支持Google Play商城，所以当用户下载手持设备应用的时候，，会自动安装可穿戴应用到可穿戴设备上。手持应用还可以用来处理一些复杂繁重的任务，网络指令，或者其他的任务，最好发送操作结果返回给可穿戴设备。

这节课会介绍如何创建一个包含了手持应用与可穿戴应用的工程。

## 搭建Android Wear模拟器或者真机设备。

我们推荐在真机上进行开发，这样可以更好的评估用户体验。然而，模拟器可以使得你在不同类型的设备屏幕上进行模拟，这对测试来说更加有用。

### 搭建Android Wear虚拟设备

建立Android Wear虚拟设备需要下面几个步骤：

1. 点击**Tools > Android > AVD Manager.**
2. 点击**Create....**
3. 填写下面几项详细的设置，其余选项保留默认：
  - **AVD Name** - AVD的名字
  - **Device** - Android Wear圆形还是方形
  - **Target** - Android 4.4W - API Level 20
  - **CPU/ABI** - Android Wear ARM (armeabi-v7a)
  - **Keyboard** - 选择Hardware keyboard present
  - **Skin** - 圆形还是方形取决于选择的设备类型
  - **Snapshot** - 不勾选 selected
  - **Use Host GPU** - 勾选，为了支持自定义的activity能够显示可穿戴的 notification。
4. 点击**OK.**
5. 启动模拟器：
  - 选择你刚才创建的虚拟设备
  - 点击**Start...**，然后选择**Launch.**
  - 等待模拟器初始化直到显示Android Wear的主界面。
6. 匹配你的手持和模拟器：
  - 在你的手持设备上，从Google Play安装Android Wear应用(这是一个由 Google公司写的用来匹配的应用)
  - 通过USB连接你的手持设备到你的电脑。
  - 切换AVD的接口到手持设备(这个步骤需要每次连接都执行)

```
adb -d forward tcp:5601 tcp:5601
```

- 启动手持设备上的Android Wear应用，并连接到模拟器。
- 点击右上角的菜单，选择Demo Cards。
- 你选择的卡片呈现在模拟器上会类似一个Notification。

### 搭建Android Wear真机

建立Android Wear真机，需要下面几个步骤：

- 在你的手持设备的Google Play上安装Android Wear应用。
- 按照应用的命令指示与你的可穿戴设备进行匹对。如果你有做建立notification的操作，这个步骤刚好可以测试这一功能。
- 保持Android Wear应用在手机上的打开状态。
- 通过USB连接可穿戴设备到电脑上，这样你能够直接安装应用到可穿戴设备上。在可穿戴设备与Android Wear应用上会显示一个消息提示，是否允许进行调试。
- 在Android Wear应用上，总是选择允许连接。

Android Studio上的Tool的窗口可以显示可穿戴设备的日志。当你执行adb devices命令的时候，也可以看到wearable的存在。

## 创建Wear项目

在开始开发之前，需要创建一个项目包含可穿戴应用与手持应用这两个模块。在Android Studio中，点击File > New Project 然后按照[创建项目的指引](#)进行操作。如果你按照安装向导操作，需要输入下面的信息：

1. 在确认项目的窗口，输入你的应用的名称与包名。
2. 在应用参数选择窗口：
  - 勾选Phone 与 Tablet 并选择API 8: Android 2.2 (Froyo) 作为Minimum SDK.
  - 勾选可穿戴并选择API 20: Android 4.4 (KitKat Wear) 作为Minimum SDK.
3. 在第一个添加activity的窗口，选择为Mobile模块添加一个空白的activity。
4. 在第二个添加activity的窗口，选择为Wear模块添加一个空白的activity。

当安装向导完成后，Andorid Studio创建了一个包含Mobile与Wear两个模块的项目。你可以在这2个模块中各自创建activity，service，layout等等。在手持应用里面，需要承担大部分繁重的任务，例如网络请求，密集计算任务或者是需要大量用户交互的任务。待这些任务完成之后，再通常把任务结果通过notification发送给可穿戴设备上，或者是通过同步机制发送数据给可穿戴设备。

**Note:** 可穿戴模块包含了一个"Hello World"的activity，它是使用WatchViewStub的布局。WatchViewStub是可穿戴support library中的一个UI组件。

## 安装可穿戴应用

在开发过程中，你可以像安装手持应用一样直接安装可穿戴应用。可以使用adb install命令也可以使用Android Studio上面的Play按钮。

当需要把应用发布给用户的时候，你需要把可穿戴应用打包到手持应用中。当用户从Google Play安装手持应用时，连接上得可穿戴设备会自动收到可穿戴应用。

**Note:** 如果你给应用签名是Debug Key，是无法完成自动安装可穿戴应用的。请参考[打包可穿戴应用](#)获取更多信息，学习如何正确的打包。

为了安装"Hello World"应用到可穿戴设备，在Android Studiod的Run/Debug的下拉选项中选中Wear模块，点击Play按钮即可。在可穿戴设备上会显示activity并打印"Hello world!"

## include需要的libraries

项目安装向导会自动把合适的模块依赖添加到对应的build.gradle文件中。然而，这些依赖并不是必须得，请阅读下面描述判断你是否需要这些依赖。

- **Notifications**

[The Android v4 support library](#) (or v13)能够支持运行在手持应用的notification也能够在可穿戴设备上显示。

对于只显示在可穿戴设备上得notification(这意味着，他们是由直接执行在可穿戴设备上得app进行处理的)，你可以在Wear模块仅仅使用标准APIs (API Level 20) 并且把Mobile模块的依赖support library移除。

- **Wearable Data Layer**

可穿戴与手持设备之间进行同步与发送数据需要使用Wearable Data Layer APIs, 你需要最新版本的[Google Play Services](#)。如果你不需要这些APIs，可以从这两个模块中把这部分的依赖移除。

- **Wearable UI support library**

这是一个非官方正式的library，它包含了为可穿戴设备设计的UI组件。我们鼓励你在你的应用中使用他们。因为这些组件是最佳实践的例证。但是他们可能随时发生变化。然而，如果library有更新，你的应用并不会发送崩溃，因为那些代码已经编译到你的应用中了。为了获取更新包中新的功能，你只需要更新链接到新的版本并相应的更新你的应用就好了。这个library只是在你需要创建可穿戴应用时才会使用到。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/layouts.html>

# 创建自定义的布局

为可穿戴设备创建布局是和手持设备是一样的。但是 不要期望通过搬迁手持应用的功能与 UI 到可穿戴上会有一个好的用户体验。仅仅在有需要的时候，你才应该创建自定义的布局。请参考可穿戴设备的[design guidelines](#) 学习如何设计一个优秀的可穿戴应用。

# 创建自定义Notification

通常来说，你应该在手持应用上创建好notification，然后让它自动同步到可穿戴设备上。这使得你只需要创建一次notification，然后可以在不同类型的设备(不仅仅是可穿戴设备，也包含车载设备与电视)上进行显示，免去为不同设备进行重新设计。

如果标准的notification风格无法满足你的需求(例如[NotificationCompat.BigTextStyle](#) 或者 [NotificationCompat.InboxStyle](#))，你可以使用activity，显示一个自定义的布局来达到目的。在可穿戴设备上你只可以创建并处理自定义的notification，同时系统无法为这些notification同步到手持设备上。

**Note:**当在可穿戴设备上创建自定义的notification时，你可以使用API Level 20上标准的APIs，不需要使用Support Library。

为了创建自定义的notification，步骤如下：

1. 创建布局并设置这个布局为需要显示的activity的content view:

```
public void onCreate(Bundle bundle) {  
    ...  
    setContentView(R.layout.notification_activity);  
}
```

2. 为了使得activity能够显示在可穿戴设备上，需要在manifest文件中为activity定义必须的属性。你需要把activity声明为exportable，embeddable以及拥有一个空的task affinity。我们也推荐把activity的主题设置为Theme.DeviceDefault.Light。例如：

```
<activity android:name="com.example.MyDisplayActivity"  
    android:exported="true"  
    android:allowEmbedded="true"  
    android:taskAffinity=""  
    android:theme="@android:style/Theme.DeviceDefault.Light" />
```

3. 为activity创建[PendingIntent](#)，例如：：

```
Intent notificationIntent = new Intent(this, NotificationActivity.class);  
PendingIntent notificationPendingIntent = PendingIntent.getActivity(this, 0,  
    notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

4. 创建[Notification](#)并执行[setDisplayIntent\(\)](#)方法，参数是前面创建的PendingIntent。当用户查看这个notification时，系统使用这个PendingIntent来启动activity。
5. 触发notification使用[notify\(\)](#)的方法。

**Note:** 当notification呈现在主页时，系统会根据notification的语义，使用一个标准的模板来呈现它。这个模板可以在所有的表盘上进行显示。当用户往上滑动notification时，将会看到为这个notification准备的自定义的activity。

## 使用Wearable UI库创建布局

当你使用Android Studio的引导功能创建一个Wearable应用的时候，会自动包含一个非官方的UI库文件。你也可以通过给build.gradle文件添加下面的依赖声明把库文件添加到项目：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.google.android.support:wearable:+'
    compile 'com.google.android.gms:play-services-wearable:+'
}
```

这个库文件帮助你建立你设计的UI。下面是一些主要的类：

- **BoxInsetLayout** - 一个能够感知屏幕的形状并把子控件居中摆放的FrameLayout,。
- **CardFragment** - 一个能够可拉伸，垂直可滑动卡片的fragment。
- **CircledImageView** - 一个圆形的image view。
- **ConfirmationActivity** - 一个在用户完成一个操作之后用来显示确认动画的activity。  
\**DismissOverlayView* - 一个用来实现长按消失的View。
- **GridViewPager** - 一个可以横向与纵向滑动的局部控制器。你需要提供一个GridPagerAdapter用来生成显示页面的数据。
- **GridPagerAdapter** - 一个提供给GridViewPager显示页面的适配器。
- **FragmentGridPagerAdapter** - 一个为每个页面提供单独的fragment的适配器。
- **WatchViewStub** - 一个可以根据屏幕的形状生成特定布局的类。
- **WearableListView** - 一个针对可穿戴设备优化过后的ListView。它会垂直的显示列表内容，并在用户停止滑动时自动显示最靠近的Item。

[点击下载完整的API说明文档](#) 这个文档会详细的介绍每一个UI组件。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/voice.html>

# 添加语音能力

语音指令是可穿戴体验的一个重要的部分。这使得用户可以释放双手，快速发出指令。穿戴提供了2种类型的语音操作：

- 系统提供的

这些语音指令都是基于任务的，并且内置在Wear的平台内。你在activity中过滤你想接收的指令。例如包含"Take a note" 或者 "Set an alarm"的指令。

- 应用提供的

这些语音指令都是基于应用的，你需要像声明一个Launcher Icon一样定义这些指令。用户通过说"Start XXX"来使用那些语音指令，然后会启动你指定启动的activity。

## 声明系统提供的语音指令

Android Wear平台基于用户的操作提供了一些语音指令，例如"Take a note" 或者 "Set an alarm"。用户发出想要做的操作指令，让系统寻找应该启动最合适的activity。

当用户说出语音指令时，你的应用能够被过滤出来启动一个activity。如果你想要启动一个service在后台执行任务，需要显示一个activity呈现作为线索。当你想要废弃这个可见的线索时，需要确保执行了finish()。

例如，对于"Take a note"的指令，定义一个MyNoteActivity来接收这个指令：

```
<activity android:name="MyNoteActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="com.google.android.voicesearch.S
    </intent-filter>
</activity>
```

下面列出了Wear平台支持的语音指令：

| Name            | Example Phrases                       | Intent   |
|-----------------|---------------------------------------|--|
| Call a car/taxi | "OK Google, get me a taxi"            | Action<br>com.google.android.gms.actions.RESERVE_TAXI_RESERVATION  |
|                 | "OK Google, call me a car"            |  |
| Take a note     | "OK Google, take a note"              | Action<br>android.intent.action.SEND<br>Category<br>com.google.android.voicesearch.SELF_NOTE   |
|                 | "OK Google, note to self"             | Extras<br>android.content.Intent.EXTRA_TEXT - a string with note body  |
| Set alarm       | "OK Google, set an alarm for 8 AM"    | Action<br>android.intent.action.SET_ALARM<br>Extras<br>android.provider.AlarmClock.EXTRA_HOUR - an integer with the hour of the alarm.                     |
|                 | "OK Google, wake me up at 6 tomorrow" | android.provider.AlarmClock.EXTRA_MINUTES - an integer with the minute of the alarm<br><br>(these 2 extras are optional, either none or both are provided) |

|                        |  |   |
|------------------------|--|---|
| Set timer              | "Ok Google, set a timer for 10 minutes"  | Action<br><code>android.provider.AlarmClock.ACTION_SET_TIMER</code><br><br>Extras<br><code>android.provider.AlarmClock.EXTRA_LENGTH</code> - an integer in the range of 1 to 86400 (number of seconds in 24 hours) representing the length of the timer                                     |
| Start/Stop a bike ride | "OK Google, start cycling"<br><br>"OK Google, start my bike ride"<br><br>"OK Google, stop cycling" | Action<br><code>vnd.google.fitness.TRACK</code><br><br>MimeType<br><code>vnd.google.fitness.activity/biking</code><br><br>Extras<br><code>actionStatus</code> - a string with the value <code>ActiveActionStatus</code> when starting and <code>CompletedActionStatus</code> when stopping. |
| Start/Stop a run       | "OK Google, track my run"<br><br>"OK Google, start running"<br><br>"OK Google, stop running"       | Action<br><code>vnd.google.fitness.TRACK</code><br><br>MimeType<br><code>vnd.google.fitness.activity/running</code><br><br>Extras<br><code>actionStatus</code> - a string with the value <code>ActiveActionStatus</code> when starting and <code>CompletedActionStatus</code> when stopping |

|                      |  |   |
|----------------------|--|---|
| Start/Stop a workout | "OK Google, start a workout"<br><br>"OK Google, track my workout"<br><br>"OK Google, stop workout" | Action<br><br><code>vnd.google.fitness.TRACK</code><br><br>MimeType<br><br><code>vnd.google.fitness.activity/other</code><br><br>Extras<br><br><code>actionStatus</code> - a string with the value <code>ActiveActionStatus</code> when starting and <code>CompletedActionStatus</code> when stopping |
| Show heart rate      | "OK Google, what's my heart rate?"<br><br>"OK Google, what's my bpm?"                              | Action<br><br><code>vnd.google.fitness.VIEW</code><br><br>MimeType<br><br><code>vnd.google.fitness.data_type/com.google.heart_rate.bpm</code>   |
| Show step count      | "OK Google, how many steps have I taken?"<br><br>"OK Google, what's my step count?"                | Action<br><br><code>vnd.google.fitness.VIEW</code><br><br>MimeType<br><br><code>vnd.google.fitness.data_type/com.google.step_count.cumulative</code>  |

关于注册intent与获取intent extra的信息，请参考[Common intents](#).

## 声明应用提供的语音指令

如果系统提供的语音指令无法满足你的需求，你可以使用"Start MyActivityName"的指令来直接启动你的应用。

注册一个"Start"指令和注册手持应用上得Launcher是一样的。

在"Start"指令的后面需要指定的文字, 这个文字需要注册在activity的label属性上。例如，下面的设置能够识别"Start MyRunningApp"的语音指令并启动StartRunActivity.

```
<application>
    <activity android:name="StartRunActivity" android:label="MyRunni
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER
        </intent-filter>
    </activity>
</application>
```

## 获取输入的自由语音

除了使用语音指令来启动activity之外，你也可以执行系统内置的语言识别activity来获取用户的语音输入。这对于获取用户的输入信息非常有帮助，例如执行搜索或者发送一个消息。

在你的应用中，`startActivityForResult()`使用`ACTION_RECOGNIZE_SPEECH`启动系统语音识别应用。在`onActivityResult()`中处理返回的结果：

```
private static final int SPEECH_REQUEST_CODE = 0;

// Create an intent that can start the Speech Recognizer activity
private void displaySpeechRecognizer() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                   RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    // Start the activity, the intent will be populated with the speech
    startActivityForResult(intent, SPEECH_REQUEST_CODE);
}

// This callback is invoked when the Speech Recognizer returns.
// This is where you process the intent and extract the speech text
@Override
protected void onActivityResult(int requestCode, int resultCode,
                                Intent data) {
    if (requestCode == SPEECH_REQUEST && resultCode == RESULT_OK)
        List<String> results = data.getStringArrayListExtra(
            RecognizerIntent.EXTRA_RESULTS);
        String spokenText = results.get(0);
        // Do something with spokenText
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/packaging.html>

# 打包可穿戴应用

当发布应用给用户之前，你必须把可穿戴应用打包到手持应用内。因为不能直接在可穿戴设备上浏览并安装应用。如果打包正确，当用户下载手持应用时，系统会自动下发可穿戴应用到匹配的可穿戴设备上。

**Note:** 如果开发时签名用的是debug key，这个特性是无法正常工作的。在开发时，需要使用adb install的命令或者Android Studio来安装可穿戴应用。

# 使用Android Studio打包

在Android Studio中打包可穿戴应用有下面几个步骤：

1. 在手持应用的build.gradle文件中把可穿戴应用声明为依赖：

```
dependencies {
    compile 'com.google.android.gms:play-services:5.0.+@aar'
    compile 'com.android.support:support-v4:20.0.+'
    wearApp project(':wearable')
}
```

2. 点击**Build > Generate Signed APK...** 安装屏幕上的指示来制定你的release key并为你的app进行签名。Android Studio导出签名好的手持应用，他内置了可穿戴应用。或者，你可以在可穿戴应用与手持应用的build.gradle文件里面建立一个签名规则。为了能够正常自动推送可穿戴应用，这两个应用都必须签名。

```
android {
    ...
    signingConfigs {
        release {
            keyAlias 'myAlias'
            keyPassword 'myPw'
            storeFile file('path/to/release.keystore')
            storePassword 'myPw'
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
        ...
    }
}
```

通过点击Android Studio右边的Gradle按钮来建立手持应用，并执行**assembleRelease**任务。这个任务放在**Project name > Handheld module name > assembleRelease**。

**Note:**这个例子中把密码写在了Gradle文件中，这应该不是期待的写法。请参考[Configure signing settings](#)学习如何为密码创建环境变量。

## 分别为可穿戴应用与手持应用进行签名

如果你的Build任务需要为可穿戴应用与手持应用签不同的Key，你可以像下面一样在手持应用的build.gradle文件中声明规则。

```
dependencies {
    ...
    wearApp files('/path/to/wearable_app.apk')
}
```

你可以为手持应用手动进行签任何形式的Key (可以是Android Studio Build > Generate Signed APK...的方式，也可以是Gradle signingConfig规则的方式)

## 手动打包

如果你使用的是其他IDE，你仍然可以把可穿戴应用打包到手持应用中。

1. 把签好名的可穿戴应用放到手持应用的res/raw目录下。我们把这个应用作为wearable\_app.apk。
2. 创建res/xml/wearable\_app\_desc.xml文件，里面包含可穿戴设备的版本信息与路径。例如：

```
<wearableApp package="wearable.app.package.name">
<versionCode>1</versionCode>
<versionName>1.0</versionName>
<rawPathResId>wearable_app</rawPathResId>
</wearableApp>
```

package, versionCode, 与 versionName需要和可穿戴应用的AndroidManifest.xml里面的信息一致。rawPathResId是一个static的变量表示APK的名称。。

3. 添加meta-data标签到你的手持应用的<application>标签下，指明引用wearable\_app\_desc.xml文件

```
<meta-data android:name="com.google.android.wearable.beta.apk"
          android:resource="@xml/wearable_app_desc"/>
```

4. 构建并签名手持应用。

## 关闭资源压缩

许多构建工具会自动压缩放在res/raw目录下的文件。因为可穿戴APK已经被压缩过了，那些压缩工作再次压缩会导致应用无法正常安装。

这样的话，安装失败。在手持应用上，`PackageUpdateService`会输出如下的错误日志：“this file cannot be opened as a file descriptor; it is probably compressed.”

Android Studio 默认不会压缩你的APK，如果你使用另外一个构建流程，需要确保不会发生重复压缩可穿戴应用的事情。

编写: [kesenhoo](#) - 校对:

原文: <http://developer.android.com/training/wearables/apps/bt-debugging.html>

# 通过蓝牙进行调试

你可以通过蓝牙来调试你的可穿戴应用，通过蓝牙把调试数据输出到手持设备上，手持设备是有连接到开发电脑上的。

## 搭建好设备用来调试

- 开启手持设备的USB调试：
  - 打开设置应用并滑动到底部。
  - 如果在设置里面没有开发者选项，点击关于手机，滑动到底部，点击build number 7次。
  - 返回并点击开发者选项。
  - 开启USB调试。
- 开启可穿戴设备的蓝牙调试：
  - 点击主界面2次，来到Wear菜单界面。
  - 滑动到底部，点击设置。
  - 滑动到底部，如果没有开发者选项，点击Build Number 7次。
  - 点击开发者选项。
  - 开启蓝牙调试。

## 建立调试会话

1. 在手持设备上，打开Android Wear这个伴侣应用。
2. 点击右上角的菜单，选择设置。
3. 开启蓝牙调试。你将会在选项下面看到一个小的状态信息：

```
Host: disconnected  
Target: connected
```

4. 通过USB连接手持设备到你的电脑上，并执行下面的命令：

```
adb forward tcp:4444 localabstract:/adb-hub; adb connect localhost:4444
```

**Note:** 你可以使用任何可用的端口。

在Android Wear伴侣应用上，你将会看到状态变为：

```
Host: connected  
Target: connected
```

## 调试你的应用

当运行adb devices的命令时，你的可穿戴设备是作为localhost:4444的。执行任何的adb命令，需要使用下面的格式：

```
adb -s localhost:4444 <command>
```

如果没有任何其他的设备通过TCP/IP连接到手持设备，你可以使用下面的简短命令：

```
adb -e <command>
```

例如：

```
adb -e logcat  
adb -e shell  
adb -e bugreport
```

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/index.html>

# 发送并同步数据

可穿戴数据层API(The Wearable Data Layer API)，Google Play服务的一部分，为你的手持与可穿戴应用提供了一个交流通道。此API包括一套的，可由系统通过网络和通告你应用数据层重要事件的监听器发送和同步的数据对象：

## Data Items

数据元提供了手持设备与可穿戴设备间的可自动同步的数据储存。

## Messages

MessageApi类可以发送“自动跟踪”命令消息，比如，从可穿戴设备上控制手持设备的媒体播放器，或在可穿戴设备上开启一个来自手持设备的intent。当手持设备与可穿戴设备成功连接时，系统会发送这个消息，否则，会发送一个错误。

## Asset

资源对象是为了发送如图像这样的二进制数据。将资源附加到数据项，系统会自动负责传递，并通过缓存大资源来避免重复传送以保护蓝牙带宽。

## WearableListenerService (for services)

拓展的WearableListenerService能够监听一个service中重要的数据层事件。系统控制 WearableListenerService的生命周期，并当需要发送数据项或消息时，将其与service绑定，否则解除绑定。

## DataListener (for foreground activities)

在一个前台activity中实现DataListener能够监听重要的数据通道事件。只有当用户活跃地使用应用时，用此代替WearableListenerService来监听改变。

**Warning:** 因为这些Api是为手持设备与可穿戴设备间通信设计，所以你只能使用这些Api来建立这些设备间的通信。例如，不能试着打开底层sockets来创建通信通道。

# Lessons

- [Accessing the Wearable Data Layer\(访问可穿戴数据层\)](#)

这节课向你展示了如何创建一个客户端访问数据层Api。

- [Syncing Data Items\(同步数据单元\)](#)

数据元是存储在一个可自动由手持设备同步到可穿戴设备上复制来的数据仓库中的对象。

- [Transferring Assets\(传输资源\)](#)

资源是典型地用来传输图像和媒体二进制数据。

- [Sending and Receiving Messages\(发送与接收消息\)](#)

消息被设计为自动跟踪的消息，可以在手持与可穿戴设备间来回传送。

- [Handling Data Layer Events\(处理数据层的事件\)](#)

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/accessing.html>

# 访问可穿戴数据层

调用数据层API，需创建一个[GoogleApiClient](#)实例，所有 Google Play services APIs的主要入口点。

[GoogleApiClient](#) 提供了一个易于创建客户端实例的builder。最简单的[GoogleApiClient](#)如下：

**Note:** 目前，此client可以启动。但是，更多创建GoogleApiClient，实现回调方法和处理错误等内容，详见 [Accessing Google Play services APIs](#)。

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(new ConnectionCallbacks() {
        @Override
        public void onConnected(Bundle connectionHint) {
            Log.d(TAG, "onConnected: " + connectionHint);
            // Now you can use the data layer API
        }
        @Override
        public void onConnectionSuspended(int cause) {
            Log.d(TAG, "onConnectionSuspended: " + cause);
        }
    })
    .addOnConnectionFailedListener(new OnConnectionFailedListener() {
        @Override
        public void onConnectionFailed(ConnectionResult result) {
            Log.d(TAG, "onConnectionFailed: " + result);
        }
    })
    .addApi(Wearable.API)
    .build();
```

在你使用数据层API之前，通过调用[connect\(\)](#)方法进行连接，如 Accessing Google Play services APIs中所述。当系统为你的客户端调用了onConnected()方法，你就可以使用数据层API了。

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/data-items.html>

# 同步数据单元

[DataItem](#)是指系统用于同步手持设备与可穿戴设备间数据的接口。一个[DataItem](#)通常包括以下几点：

- **Payload**- 一个字节数组，你可以用来设置任何数据，让你的object序列化和反序列化。Payload的大小限制在100k之内。
- **Path**- 唯一的必须以斜线开头(如： "/path/to/data")的string。

通常不直接实现[DataItem](#)，而是：

1. 创建一个[PutdataRequest](#)对象，指明一个string path。
2. 调用[setData\(\)](#)方法。
3. 调用[DataApi.putDataItem\(\)](#)方法，请求系统创建数据元。
4. 当请求的时候，系统会返回正确实现DataItem接口的对象。

然而，我们建议使用[Data Map](#)来显示装在一个易用的类似[Bundle](#)接口中的数据元，而用不是[setData\(\)](#)来处理原始字节。

# 用 Data Map 同步数据

使用[DataMap](#)类，将数据元处理为 Android [Bundle](#)的形式，因此对象的序列化和反序列化就会完成，你就可以以 key-value 对的形式操纵数据。

如何使用：

1. 创建一个 [PutDataMapRequest](#)对象，设置数据元的path。

**Note:** path 字符串对数据元是唯一确定的，这样能够使你从另一连接端访问。Path须以斜线开始。如果你想在应用中使用分层数据，就要创建一个适合数据结构的路径方案。

2. 调用[PutDataMapRequest.getDataMap\(\)](#)获取一个你可以使用的data map 对象。
3. 使用put...()方法，如：[putString\(\)](#),为data map设置数据。
4. 调用[PutDataMapRequest.asPutDataRequest\(\)](#)获得[PutDataRequest](#)对象。
5. 调用 [DataApi.putDataItem\(\)](#) 请求系统创建数据元

**Note:** 如果手机和可穿戴设备没有连接，数据会缓冲并在重新建立连接时同步。

接下来的例子展示如何创建一个data map，并设置数据：

```
PutDataMapRequest dataMap = PutDataMapRequest.create("/count");
dataMap.getDataMap().putInt(COUNT_KEY, count++);
PutDataRequest request = dataMap.asPutDataRequest();
PendingResult<DataApi.DataItemResult> pendingResult = Wearable.Data
    .putDataItem(mGoogleApiClient, request);
```

## 监听数据元事件

如果一端的数据层的数据发生改变，想要在另一端被提醒此改变，你可以通过实现一个数据元事件的监听器来完成。

例如，这是一个典型的用回调来实现特定的action,当数据发生改变时。

```
@Override  
public void onDataChanged(DataEventBuffer dataEvents) {  
    for (DataEvent event : dataEvents) {  
        if (event.getType() == DataEvent.TYPE_DELETED) {  
            Log.d(TAG, "DataItem deleted: " + event.getDataItem());  
        } else if (event.getType() == DataEvent.TYPE_CHANGED) {  
            Log.d(TAG, "DataItem changed: " + event.getDataItem());  
        }  
    }  
}
```

这仅是一小片段，如何实现完整监听和activity 请见 [Listening for Data Layer Events](#)。

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/assets.html>

# 传输资源

将一个[Asset](#)附加到数据元上，并放入复制而来的数据库中，通过蓝牙来传送大量的二进制数据。

Assets 能够自动地处理数据缓存以避免重复发送，保护蓝牙带宽。一般的模式是：手持设备下载图像，并将它压缩到适合在可穿戴设备上显示的大小，并以Asset传给可穿戴设备。下面的例子演示此模式。

**Note:** 尽管数据元的大小限制在100KB,但资源可以任意大。然而，传输大量资源会多方面地影响用户体验，因此，要测试你的应用以保证当你传输大量资源时，它会表现良好。

## 传输资源

在Asset类中使用creat..()方法创建资源。下面，我们将一个bitmap转化为字节流，然后调用creatFromBytes()方法创建资源。

```
private static Asset createAssetFromBitmap(Bitmap bitmap) {  
    final ByteArrayOutputStream byteStream = new ByteArrayOutputStream();  
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, byteStream);  
    return Asset.createFromBytes(byteStream.toByteArray());  
}
```

创建资源后，使用[DataMap](#)或者[PutDataRepuest](#)类中的putAsset()方法将其附加到数据元上，然后用[putDataItem\(\)](#)方法将数据元放入数据库。

### 使用 PutDataRequest

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.  
Asset asset = createAssetFromBitmap(bitmap);  
PutDataRequest request = PutDataRequest.create("/image");  
request.putAsset("profileImage", asset);  
Wearable.DataApi.putDataItem(mGoogleApiClient, request);
```

### 使用 PutDataMapRequest

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.  
Asset asset = createAssetFromBitmap(bitmap);  
PutDataMapRequest dataMap = PutDataMapRequest.create("/image");  
dataMap.getDataMap().putAsset("profileImage", asset)  
PutDataRequest request = dataMap.asPutDataRequest();  
PendingResult<DataApi.DataItemResult> pendingResult = Wearable.DataApi  
    .putDataItem(mGoogleApiClient, request);
```

## 接收资源

创建资源后，如何在另一连接端读取。以下是如何实现回调以发现资源变化和提取Asset对象。

```
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
    for (DataEvent event : dataEvents) {
        if (event.getType() == DataEvent.TYPE_CHANGED &&
            event.getDataItem().getUri().getPath().equals("/image")) {
            DataMapItem dataMapItem = DataMapItem.fromDataItem(dataItem)
            Asset profileAsset = dataMapItem.getDataMap().getAsset("profile");
            Bitmap bitmap = loadBitmapFromAsset(profileAsset);
            // Do something with the bitmap
        }
    }
}

public Bitmap loadBitmapFromAsset(Asset asset) {
    if (asset == null) {
        throw new IllegalArgumentException("Asset must be non-null");
    }
    ConnectionResult result =
        mGoogleApiClient.blockingConnect(TIMEOUT_MS, TimeUnit.MILLISECONDS);
    if (!result.isSuccess()) {
        return null;
    }
    // convert asset into a file descriptor and block until it's ready
    InputStream assetInputStream = Wearable.DataApi.getFdForAsset(
        mGoogleApiClient, asset).await().getInputStream();
    mGoogleApiClient.disconnect();

    if (assetInputStream == null) {
        Log.w(TAG, "Requested an unknown Asset.");
        return null;
    }
    // decode the stream into a bitmap
    return BitmapFactory.decodeStream(assetInputStream);
}
```

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/messages.html>

# 发送与接收消息

使用[MessageApi](#)发送消息，要附加以下几项：

- 任一payload(可选);
- 唯一确定消息action的path。

不像数据元，消息在手持设备和可穿戴应用之间没有同步Messages是单向交流机制，意味着 "fire-and-forget" tasks，比如：发送消息到可穿戴设备以开启activity。也可以用请求/回应的模式，一连接端发送消息，完成任务，传回响应消息。

## 发送消息

下面的例子展示如何发送消息到另一连接端开启一个activity。调用是同步的，当收到消息或请求超时时发生阻塞。

**Note:** 阅读 [Communicate with Google Play Services](#) 了解更多关于异步和同步调用，以及何时使用哪个。

```
Node node; // the connected device to send the message to
GoogleApiClient mGoogleApiClient;
public static final START_ACTIVITY_PATH = "/start/MainActivity";
...

SendMessageResult result = Wearable.MessageApi.sendMessage(
    mGoogleApiClient, node, START_ACTIVITY_PATH, null).await();
if (!result.getStatus().isSuccess()) {
    Log.e(TAG, "ERROR: failed to send Message: " + result.getStatus());
}
```

这是一个简单的方法获得一列你可能发送消息的节点：

```
private Collection<String> getNodes() {
    HashSet <String> results= new HashSet<String>();
    NodeApi.GetConnectedNodesResult nodes =
        Wearable.NodeApi.getConnectedNodes(mGoogleApiClient).await();
    for (Node node : nodes.getNodes()) {
        results.add(node.getId());
    }
    return results;
}
```

## 接收消息

如要在收到消息时被提醒，需要实现消息事件的监听。这个例子展示你可以通过检查上例中发送消息时使用到的START\_ACTIVITY\_PATH的状态，若是true,特定的activity就会启动。

```
@Override  
public void onMessageReceived(MessageEvent messageEvent) {  
    if (messageEvent.getPath().equals(START_ACTIVITY_PATH)) {  
        Intent startIntent = new Intent(this, MainActivity.class);  
        startIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
        startActivity(startIntent);  
    }  
}
```

编写:[wly2014](#)- 校对:

原文: <http://developer.android.com/training/wearables/data-layer/events.html>

# 处理数据层的事件

当做出数据层上的调用时，你可以得到它完成后的调用状态，也可以用监听器监听到调用最终实现的改变。

## 等待数据层调用状态

注意到，调用数据层API，有时会返回[PendingResult](#)，如[putDataItem\(\)](#)。PendingResult一被创建，操作就会在后台排列等候。之后你若无动作，这些操作最终会默默完成。然而，通常要处理操作完成后的结果，PendingResult能够让你同步或异步地等待结果。

### 异步等待

若你的代码运行在主UI线程上，不使阻塞调用数据层API。你可以增加一个PendingResult对象回调来运行异步调用，将在操作完成时触发。

```
pendingResult.setResultCallback(new ResultCallback<DataItemResult>
    @Override
    public void onResult(final DataItemResult result) {
        if(result.getStatus().isSuccess()) {
            Log.d(TAG, "Data item set: " + result.getDataItem().getUri());
        }
    });
});
```

### 同步等待

如果你的代码是运行在后台服务的一个单独的处理线程上（[WearableListenerService](#)的情况），则适合调用阻塞。在这种情况下，你可以用PendingResult对象调用[await\(\)](#)，它将阻塞至请求完成，并返回一个Result对象。

```
DataItemResult result = pendingResult.await();
if(result.getStatus().isSuccess()) {
    Log.d(TAG, "Data item set: " + result.getDataItem().getUri());
}
```

# 监听数据层事件

因为数据层在手持和可穿戴设备间同步并发送数据,所以通常要监听重要事件,例如创建数据元,接受消息,或当可穿戴设备和手机连接时。

对于监听数据层事件,有两种选择:

- 创建一个继承自WearableListenerService的service。
- 创建一个实现[DataApi.DataListener](#)接口的activity。

通过这两种选择,你覆盖任何你关心的数据事件回调方法来处理您的实现。

## 使用 WearableListenerService

典型地,在你的手持设备和可穿戴设备上都创建service实例。如果你不关心其中一个应用中的数据事件,就不需要在相应的应用中实现此service。

例如,您可以在一个手持设备应用程序上操作数据元对象,可穿戴设备的应用监听这些更新并更新UI。而可穿戴不更新任何数据元,所以手持设备的应用不监听任何可穿戴式设备应用的数据事件。

你可以用 WearableListenerService 监听如下事件:

- [onDataChanged\(\)](#) - 当数据元对象创建,更改,删除时调用。一连接端的事件触发两端的回调方法。
- [onMessageReceived\(\)](#) - 消息从一连接端发出时在另一连接端触发此回调方法。
- [onPeerConnected\(\)](#) 和 [onPeerDisconnected\(\)](#) - 当与手持或可穿戴设备连接或断开时调用。一连接端连接状态的改变会在两端触发此回调方法。

创建WearableListenerService:

1. 创建一个继承自WearableListenerService的类。
2. 监听你关心的事件,比如onDataChanged()。
3. 在Android manifest中声明一个intent filter,通知系统你的WearableListenerService。这样允许系统需要时绑定你的service。

下例展示如何实现一个简单的WearableListenerService:

```
public class DataLayerListenerService extends WearableListenerService {

    private static final String TAG = "DataLayerSample";
    private static final String START_ACTIVITY_PATH = "/start-activity";
    private static final String DATA_ITEM_RECEIVED_PATH = "/data-item-received";

    @Override
    public void onDataChanged(DataEventBuffer dataEvents) {
        if (Log.isLoggable(TAG, Log.DEBUG)) {
            Log.d(TAG, "onDataChanged: " + dataEvents);
        }
        final List events = FreezableUtils
            .freezeIterable(dataEvents);

        GoogleApiClient googleApiClient = new GoogleApiClient.Bui
```

```

        .addApi(Wearable.API)
        .build();

    ConnectionResult connectionResult =
        googleApiClient.blockingConnect(30, TimeUnit.SECONDS);

    if (!connectionResult.isSuccess()) {
        Log.e(TAG, "Failed to connect to GoogleApiClient.");
        return;
    }

    // Loop through the events and send a message
    // to the node that created the data item.
    for (DataEvent event : events) {
        Uri uri = event.getDataItem().getUri();

        // Get the node id from the host value of the URI
        String nodeId = uri.getHost();
        // Set the data of the message to be the bytes of the
        byte[] payload = uri.toString().getBytes();

        // Send the RPC
        Wearable.MessageApi.sendMessage(googleApiClient, nodeId,
            DATA_ITEM_RECEIVED_PATH, payload);
    }
}
}

```

这是Android manifest中相应的intent filter:

```

<service android:name=".DataLayerListenerService">
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.BIND_LISTENER"
    </intent-filter>
</service>

```

## 数据层回调权限

为了在数据层事件上向你的程序提供回调方法, Google Play服务绑定到你的 WearableListenerService, 通过IPC调用回调方法。这样的结果是,你的回调方法继承了调用进程的权限。

如果你想在一个回调中执行权限操作,安全检查会失败,因为你的回调是以调用进程的身份运行,而不是应用程序进程的身份运行。

为了解决这个问题,在进入IPC后使用[clearCallingIdentity\(\)](#)重置身份,当你完成权限操作后, 使用[restoreCallingIdentity\(\)](#)恢复身份:

```

long token = Binder.clearCallingIdentity();
try {
    performOperationRequiringPermissions();
} finally {

```

```
        Binder.restoreCallingIdentity(token);  
    }  
}
```

## 使用监听 Activity

如果你的应用只关心数据事件，当用户正与应用交互并且不需要长时间运行的service来处理数据改变时，你可以在一个activity中通过实现如下一个和多个接口监听事件：

- DataApi.DataListener
- MessageApi.MessageListener
- NodeApi.NodeListener

创建一个activity监听数据事件：

- 实现所需的接口。
- 在onCreate(Bundle)中创建 GoogleApiClient实例。
- 在onStart()中调用connect() 将客户端连接到 Google Play 服务。
- 当连接到Google Play 服务时，系统调用 onConnected()。这里你调用以提醒Google Play 服务你的activity要监听数据层事件。
- 在 onStop()中，用 DataApi.removeListener(), MessageApi.removeListener() 或 NodeApi.removeListener() 注销监听。
- 基于你实现的接口实现 onDataChanged(), onMessageReceived(), onPeerConnected()和 onPeerDisconnected()。

这是实现DataApi.DataListener的例子：

```
public class MainActivity extends Activity implements  
    DataApi.DataListener, ConnectionCallbacks, OnConnectionFai  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.main);  
        mGoogleApiClient = new GoogleApiClient.Builder(this)  
            .addApi(Wearable.API)  
            .addConnectionCallbacks(this)  
            .addOnConnectionFailedListener(this)  
            .build();  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        if (!mResolvingError) {  
            mGoogleApiClient.connect();  
        }  
    }  
  
    @Override  
    public void onConnected(Bundle connectionHint) {  
        if (Log.isLoggable(TAG, Log.DEBUG)) {  
            Log.d(TAG, "Connected to Google Api Service");  
        }  
    }  
}
```

```
        }
        Wearable.DataApi.addListener(mGoogleApiClient, this);
    }

@Override
protected void onStop() {
    if (null != mGoogleApiClient && mGoogleApiClient.isConnected())
        Wearable.DataApi.removeListener(mGoogleApiClient, this);
    mGoogleApiClient.disconnect();
}
super.onStop();
}

@Override
public void onDataChanged(DataEventBuffer dataEvents) {
    for (DataEvent event : dataEvents) {
        if (event.getType() == DataEvent.TYPE_DELETED) {
            Log.d(TAG, "DataItem deleted: " + event.getDataItem());
        } else if (event.getType() == DataEvent.TYPE_CHANGED) {
            Log.d(TAG, "DataItem changed: " + event.getDataItem());
        }
    }
}
```

编写:[kesenhoo](#)

校对:

# 交互与界面

These classes focus on the best Android user experience for your app. In some cases, the success of your app on Android is heavily affected by whether your app conforms to the user's expectations for UI and navigation on an Android device. Follow these recommendations to ensure that your app looks and behaves in a way that satisfies Android users.

## [Designing Effective Navigation](#)

How to plan your app's screen hierarchy and forms of navigation so users can effectively and intuitively traverse your app content using various navigation patterns.

## [Implementing Effective Navigation](#)

How to implement various navigation patterns such as swipe views, a navigation drawer, and up navigation.

## [Notifying the User](#)

How to display messages called notifications outside of your application's UI.

## [Adding Search Functionality](#)

How to properly add a search interface to your app and create a searchable database.

## [Designing for Multiple Screens](#)

How to build a user interface that's flexible enough to fit perfectly on any screen and how to create different interaction patterns that are optimized for different screen sizes.

## [Designing for TV](#)

How to optimize your app's user interface and user input for the "ten foot experience" of a TV screen.

## [Creating Custom Views](#)

How to build custom UI widgets that are interactive and smooth.

## [Creating Backward-Compatible UIs](#)

How to use UI components and other APIs from the more recent versions of Android while remaining compatible with older versions of the platform.

## [Implementing Accessibility](#)

How to make your app accessible to users with vision impairment or other physical disabilities.

## [Managing the System UI](#)

How to hide and show status and navigation bars across different versions of Android, while managing the display of other screen components.

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/index.html>

# 设计高效的导航

设计开发 App 的起初步骤之一就是决定用户能够在App上看到什么和做什么。一旦你知道用户在App上和哪种内容互动，下一步就是去设计容许用户在 App 的不同内容块间切换，进入，回退的交互。

本课程演示如何为你的应用规划出高标准的屏幕层次，然后为它选择适宜的导航形式来允许用户高效而直观的浏览内容。按粗略的先后顺序,每堂课涵盖Android应用导航交互设计过程中的不同阶段。学过这些课之后，你应该可以应用这些列出的方法和设计范例到你自己的应用中，为你的用户提供一致的导航体验了。

# 课程

- 规划界面和他们之间的关系

学习如何选择你应用应该包含的画面。并且学习如何决定某个画面对其他画面可直达。这节课介绍了一个假想的新闻应用为以后课程作例子。

- 为多种大小的屏幕进行规划

学习如何在大屏设备上组合相关画面来优化用户可视画面空间。

- 提供向下和横向导航

学习容许用户深入某一层或者在内容层次间横跨的技巧。而且学习一些特定导航 UI 元件在不同情景下的优缺点和最佳用法。

- 提供向上和时间导航

学习如何容许用户在内容层级向上导航。并且学习 Back 键和时间导航的最佳做法，也即导航到和层次无关的之前的画面。

- 综合：设计样例 App

学习如何创建画面的 Wireframe（线框图，模糊的图形模型），它代表新闻应用基于设想信息模型的界面。这些 Wireframe 利用上述课程讨论的导航元件来展示直观高效导航。

[开始吧](#)

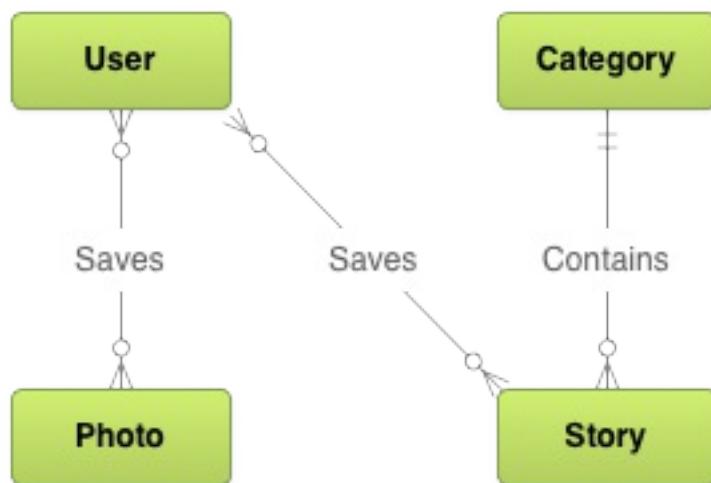
编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/screen-planning.html>

# 规划界面和他们之间的关系

多数 App 都有一种内在的信息模型，它能被表示成一个对象类型的树或图。更浅显的说，你可以画一个有不同类型信息的图，这些信息代表用户在你 App 里与之互动的各种东西。软件工程师和数据架构师经常使用实例-关系图（Entity-Relationship Diagram, ERD）描述一个应用的信息模型。

让我们考虑一个让用户浏览一群分类好的新闻事件和图片的应用例子。这种 App 一个可能的模型如下 ERD 图。



## 创建一个界面列表

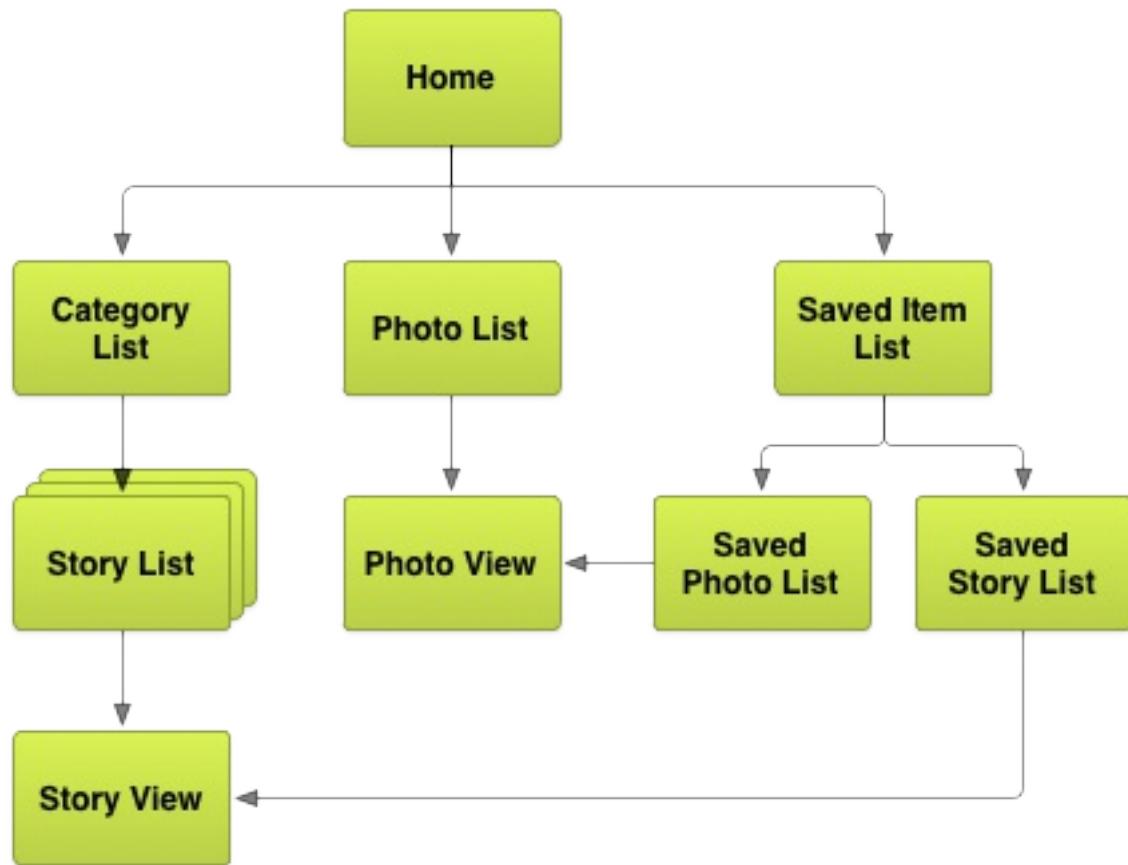
一旦你定义了信息模型，你就可以开始定义那些使用户在你的 App 中有效地发掘，查看和操作数据的上下文环境了。实际上，一种方法就是确定供用户导航和交互数据所需的 界面完备集（归纳了所有界面的集合）。但我们实际发现的画面集合应该根据目标设备变化。在设计过程中早点考虑到这点很重要，这样可以保证程序可以适应运行环境。

在我们的例子中，我们想让用户查看，保存和分享分类好了的新闻和图片。下面是涵盖了这些用例的画面完备列表。

- 用来接触新闻和图片的 Home 或者 Launchpad 画面
- 类别列表
- 某个分类下新新闻列表
- 新闻详细信息 View （在这里面我们可以保存和分享）
- 图片列表，不分类
- 图片详细信息 View （在这里面我们可以保存和分享）
- 所有保存项列表
- 图片保存列表
- 新闻保存列表

## 图示界面关系

现在我们可以定义界面间的有向关系了。一个从界面 *A* 指向另一个界面 *B* 的箭头表示通过用户在画面 *A* 的某个交互动作可直达画面 *B*。一旦我们定义了界面集和他们之间的关系，我们可以将他们一起全部表示在一张界面图中了：



如果之后我们想允许用户提交新闻事件或者上传图片，我们可以在图中加额外的界面。

## 脱离简陋设计

就此，我们有希望根据这张完备的界面图设计一个功能完备应用。一个简陋的 UI 可以由列表和导向子画面的按钮组成。

- 导向不同部分的按钮（例如，事件，图片，保存的项目）
- 纵向列表表示集合（例如，事件列表，图片列表，等等）
- 详细信息（例如，事件 View，图片 View，等等）

但是，你可以利用屏幕组合技术和更高深导航元件以一种更直观，设备更理解的方式呈现内容。下节课，我们探索屏幕组合技术，比如为平板而生的多视窗（Multi-pane）布局。之后，我将深入讲解更多不同的 Android 常见导航模式。

[下节课：规划多种触屏大小](#)

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/multiple-sizes.html>

# 为多种大小的屏幕进行规划

虽然上节中的界面完备图在手持设备和相似大小设备上可行，但并不是和某个设备因素绑死的。Android应用需要适配一大把不同类型的设备，从3"的手机到10"的平板到42"的电视。这节课中我们探讨把完备图中不同界面组合起来的策略和原因。

**注意:** 为电视设计应用程序还需要注意其他的因素，包括互动方式（就是说，它没触屏），长距离情况下文本的可读性，还有其他。虽然这种讨论在本课范畴之外，你仍然可以在 [Google TV 文档的设计模式](#) 中找到有关为电视设计的信息。

# 用多视窗布局（Multi-pane Layout）组合界面

## 多视窗布局（Multi-pane Layout）设计

设计指南请阅读 Android 设计部分的[多视窗布局模式指导](#)。

3 到 4" 的屏幕通常只适合每次展示单个纵向内容视窗，一个列表，或某列表项的具体信息，等等。所以在这些设备上，界面通常对映于信息层次上的某一级（类别 → 列表 → 详情）。

更大的诸如平板和电视上的屏幕通常会有更多的可用界面空间，并且他们能够展示多个内容视窗。横屏中，视窗从左到右以细节程度递增的顺序排列。因常年使用桌面应用和网站，用户变得特别适应大屏上的多视窗。很多桌面应用和网站提供左侧导航视窗，或者使用总/分（master/detail）俩视窗布局。

为了符合这些用户期望，通常很有必要为平板提供多个信息视窗来避免留下过多空白或无意间引入尴尬的交互，比如 10 x 0.5" 按钮。

下面图例示范了当把 UI 设计迁移到更大的布局时出现的一些问题，并且展示了如何用多视窗布局来处理这些问题：

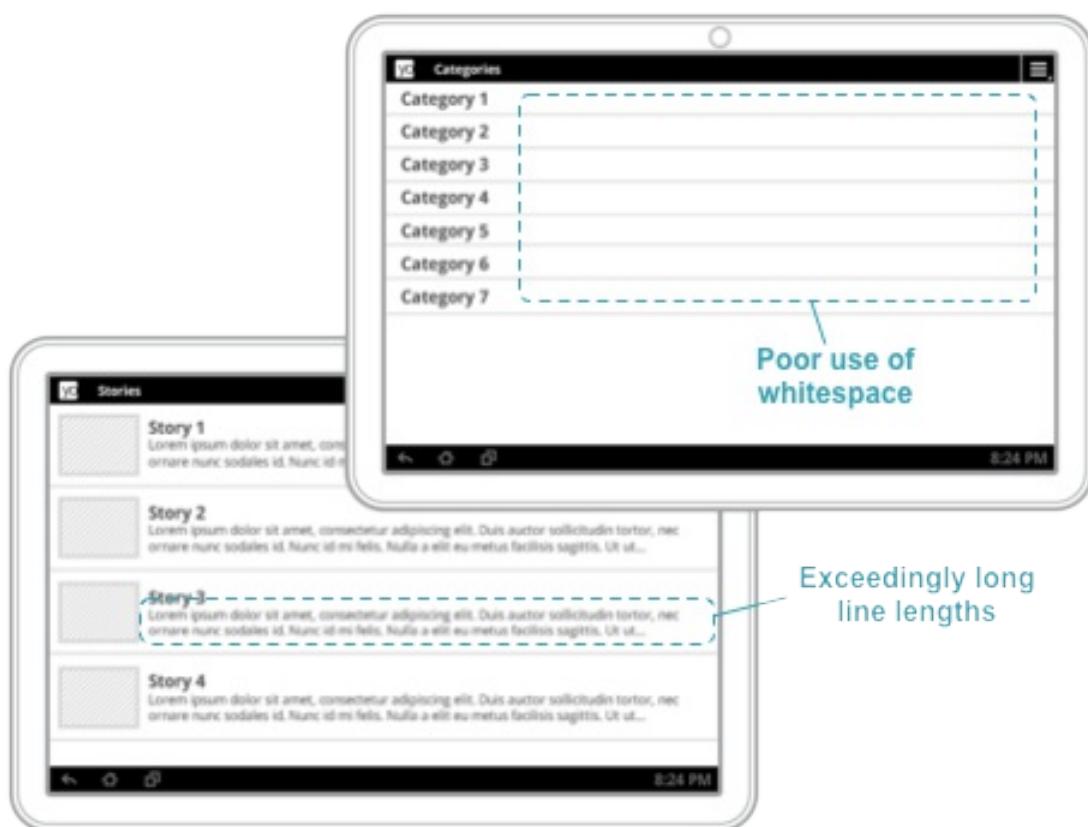


图 1. 大横屏使用单视窗导致尴尬的空白和过长行。



图 2. 横屏多视窗布局产生更好的视觉平衡，更大的效用和可读性。

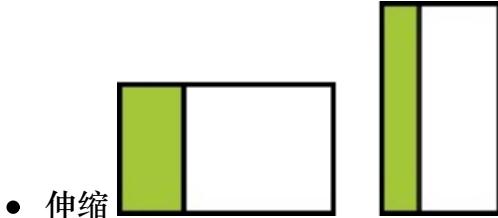
**实现提醒：**当决定好了区分使用单视窗布局和多视窗布局的屏幕大小基准线后，你就可以为不同屏幕大小区间（例如 large/xlarge）或最低屏幕宽度（例如 sw600dp）提供不同的布局了。

**实现提醒：**单一界面被实现为 [Activity](#) 的子类，单独的内容视窗则可实现为 [Fragment](#) 的子类。这样最大化了跨越不同结构因素和不同屏幕内容的代码复用。

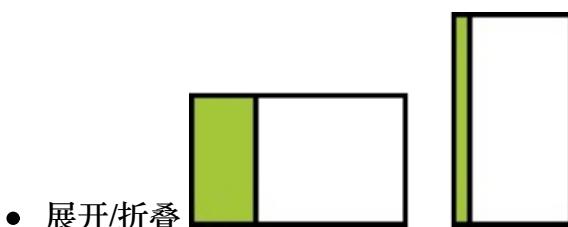
## 为不同平板方向设计

虽然现在我们还没有开始在我们的屏幕上排布 UI 元件，这是个很好的时间来考虑下我们的多视窗界面如何适配不同的设备方向。多视窗布局在横屏时表现的非常棒，因为有大量可用的横向空间。然而，在竖屏时，你的横向空间被限制了，所以你需要为这个方向设计一个单独的布局。

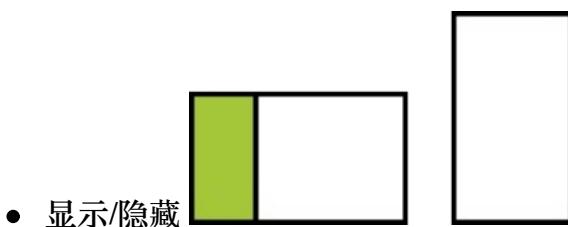
下面是一些创建竖屏布局的常见策略：



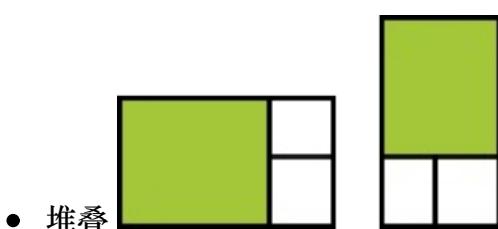
最直接的策略就是简单地伸缩每个视窗的宽度来最好地在竖屏下的呈现内容。视窗可设置固定宽度或占可用界面宽度的一定比例。



伸缩策略的一个变种就是在竖屏中折叠左侧视窗的内容。当遇到总/分 (master/detail) 视窗中左侧 (master) 视窗包含易折叠列表项时，这个策略很有效。以一个实时聊天应用为例。横屏中，左侧列表包含聊天联系人的照片，名称和在线状态。在竖屏中，将以一个隐藏名称，只显示照片和在线状态的提示图标的方式来折叠横向空间。也可以选择性的提供展开控制，它允许用户展开左侧视窗或相反的操作。



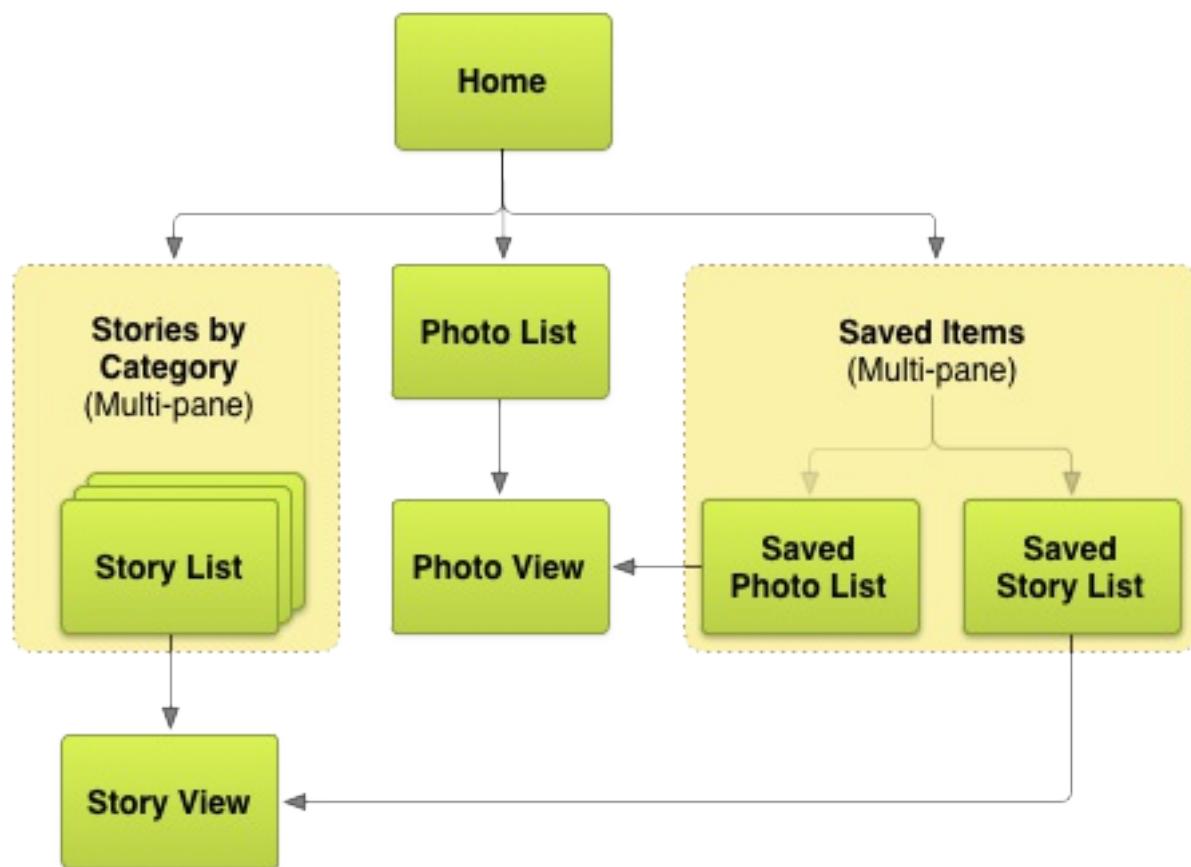
这个方案中，左侧视窗在竖屏模式下完全隐藏。然而，为了保证你界面的功能等价性，左侧视窗必须功能可见（比如，添一个按钮）。通常适合在 Action Bar 使用 Up 按钮（详询Android设计的[模式](#)文档）来展示左侧视窗，这将在[之后](#)讨论。



最后的策略就是在竖屏时垂直地堆放你一般横向排布的视窗。当你的视窗不是简单的文本列表或者当有多个内容模块与基本内容视窗同时运行时，这个策略很奏效。但是当心使用这个策略时出现上文提到的尴尬的空白问题。

## 组合界面图中的界面

既然现在我们能够通过提供大屏设备上的多视窗布局来组合单独的界面，那么就让我们把这个技术应用到我们[上节课](#)界面完备图上吧，这样我们应用的层次在这类设备上变得更有意义了：



下节课我们将讨论 向下 和 横向 导航，并且探讨更多方法来组合界面使能最大化应用 UI 的直观性和内容获取速度。

[下一节：提供向下和横向导航](#)

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/descendant-lateral.html>

# 提供向下与横向导航

一种提供查看应用整体画面的方式就是显示层级导航。这节课我们讨论 向下导航，它允许用户进入子界面。我们还讨论 横向 导航，它允许用户访问同级界面。

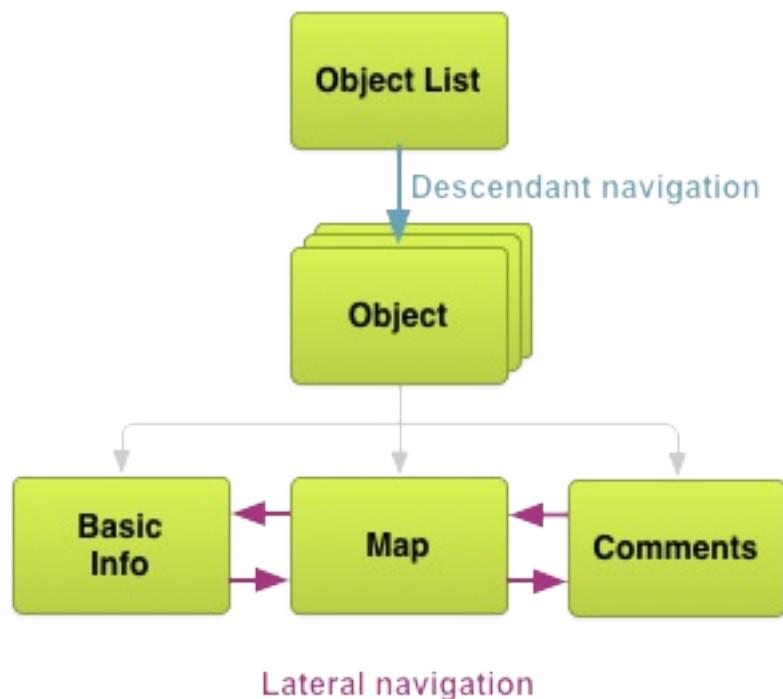


图 1. 向下和横向导航

有两种同级界面：容器关联和区块关联界面。容器关联 (*Collection-related*) 界面展示由父界面放入同一个容器里地那些条目。区块关联(*Section-related*) 界面展示父界面不同部分的信息，例如：一个部分可能展示某对象的文字信息，可是另一个部分则提供对象地理位置的地图。一个父界面的区块关联界面数量通常较少。

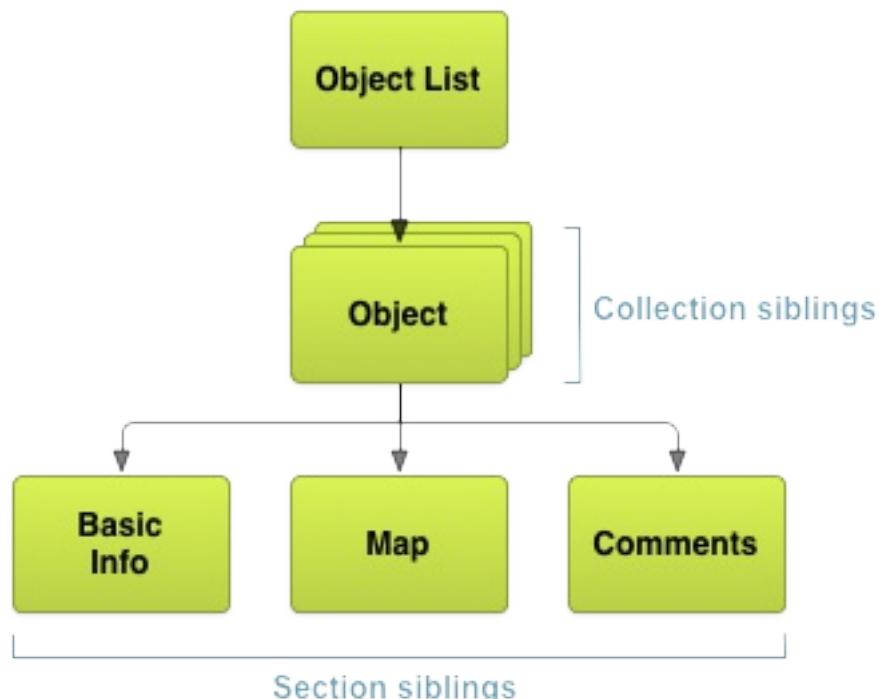


图 2. 容器关联子界面和区块关联子界面。

向下和横向导航可用List（列表），Tab（标签）或者其他UI模式来实现。UI模式，与软件设计模式很类似，是重复交互设计问题的一般化解决方案。下几章，我们将探究一些常用的横向导航模式。

## Button（按钮）和简单的控件

### Button设计

设计指南请阅读 Android 设计文档的[Button](#)指导

对于区块关联的界面，最直接和熟悉的导航界面就是提供可触或键盘可得焦点的控件。例如，Button，固定大小的 List View 或文本链接，虽然后者不是一个触屏导航的理想 UI 原件。一旦点选了这些控件，子界面被打开，完全替代当前上下文环境。Button或其他简单地控件很少被用来呈现容器中的项目。



图 3. Button导航模式例子和对应界面图。Dashboard 模式见下文。

Dashboard（操作面板）模式时一种一般以Button为主来获取不同应用划分模块的模式。一个dashboard就是个大图标Button表格，它表示了父界面绝大部分内容。这个表格通常是2、3行或列，取决于 App 的顶层划分。此模式展示全部区块的视觉效果非常丰富。巨大的触摸控件也让 UI 特别好使。当每个区块都同等重要时，Dashboard模式最好用。然而，这个模式在大屏上效果不佳，他让用户直接获取 App 内容时多走了一步弯路。

还有更多高级 UI 模式套用了各种其他 UI 模式来提升内容即得性和独特的展示效果。但他们仍保持着直观的特点。

# Lists (列表) , Grids (表格) , Carousels (内容循环) , and Stacks (栈)

## List 和 Grid List 设计

设计指南请阅读 Android 设计文档的[Lists](#)和[Grid Lists](#)指导。

对于容器关联的界面，特别是文字信息，竖直滑动列表通常是最直接和熟悉的做法。对于视觉更丰富的内容（例如，图片，视屏），可用竖直滑动的 Grid，水平滚动的 List（有时被叫做 Carousel）, 或 Stack（有时叫做卡片（Card））来代替。这些 UI 元件通常用在呈现容器内的条目，或大量子界面最好，而不是零星的毫无关联的同级子界面。

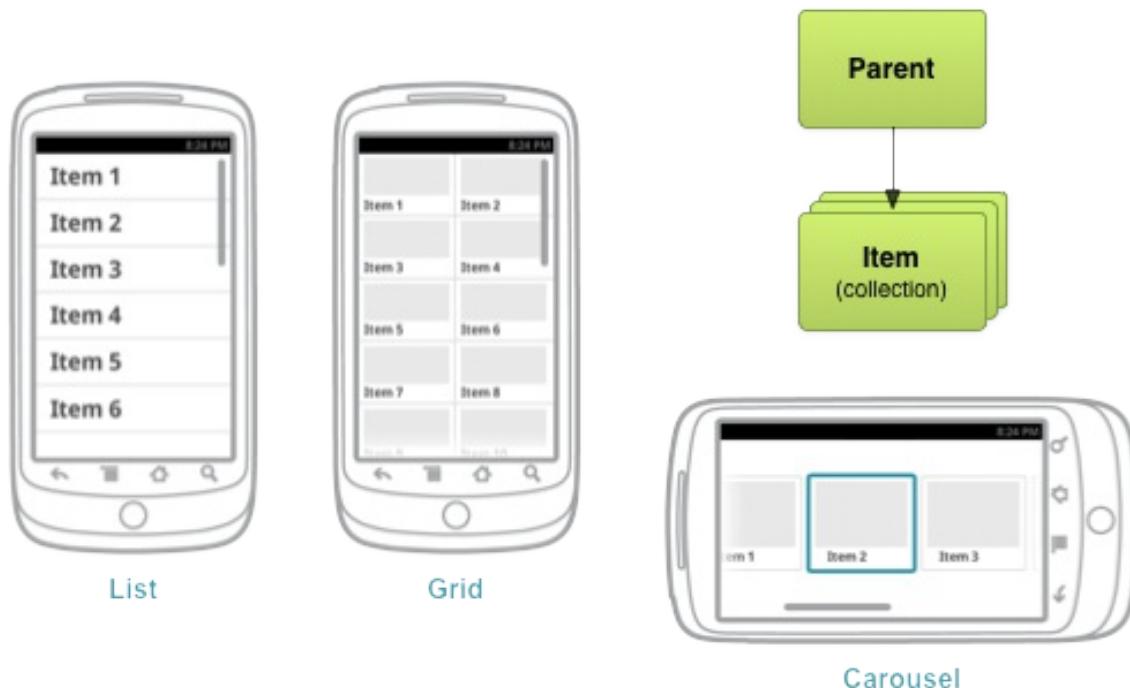


图 4. 控件例子和对应界面图

这个模式还有些问题。深层列表导航常常叫 drill-down (钻井) 列表导航，它的list层层嵌套。这种导航笨拙低效。获得某块内容需要点击多次，带给用户很差的体验，特别是活跃用户。

使用纵向list也可能带来尴尬的用户交互，并且如果list条目简单地的拉伸话也可能用不好大屏空白。解决方法就是提供额外的信息，例如用文字汇总填充那些可用的水平空间。或者在左右添加个视窗。

## Tabs (标签)

### Tab 设计

设计指南请阅读 Android 设计文档的[Tab 指导](#)

Tab是非常流行的横向导航。这个模式允许组合同级界面，就是说tab可嵌入原本可能成为另一个界面的子界面内容。Tab适合用在小量的区块关联界面。

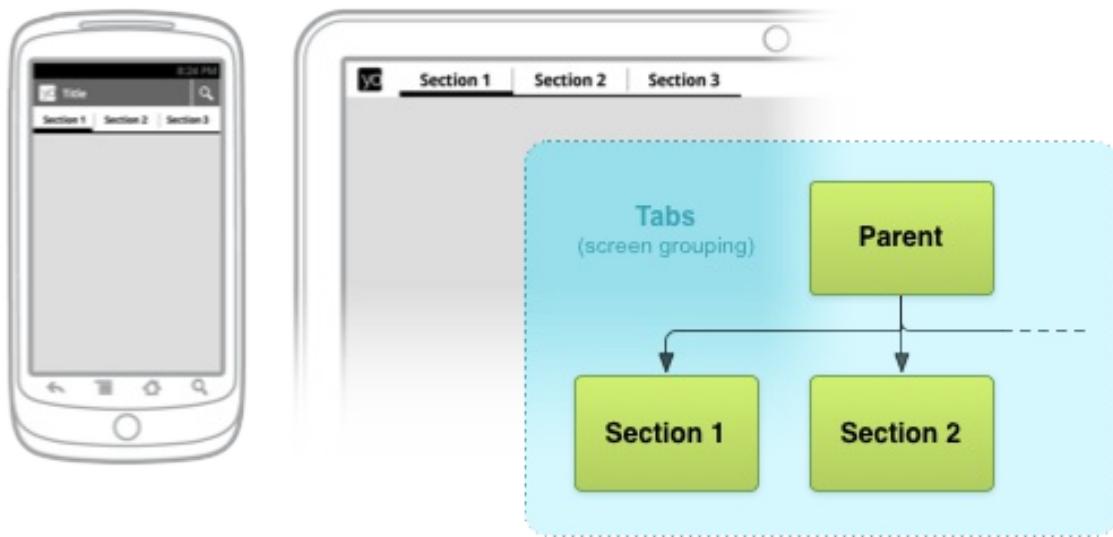


图 5. 手机和平板导航例子和对应界面图

几个使用Tab时的最佳做法。当选择时Tab被跳过，Tab应该保持原状，只有指定内容区域发生改变，并且tab任何时候都可用。此外，tab切换不能算作历史。例如，如果用户从 Tab A 切换到 Tab B，按 Back 按钮（详情看[下节](#)）不该重选 Tab A。Tab通常水平排布，可是有时其他tab展现形式，例如Action bar（详询Android 设计的[模式](#)章节）的下拉菜单，也是可以的。最后，最重要的是，tab应该在界面顶端和内容对应。

tab导航相对于list和button导航，有很多即得的优点：

- 既然只有一个既选的活动tab，用户能立即从界面获取tab的内容。
- 用户可在相关界面内快速导航，不用重新访问父界面。

**注意：**当切换Tab时，保证立即切换很重要。不要加载时弹个确认对话框来阻塞tab的访问。

导致这个模式被批评常见的原因就是必须从内容空间分一些给tab。但是结果还能接受，权衡的天平一般都向使用此模式的方向倾斜。请随意自定义你的tab，加点文字或图标什么的让纵向空间合理利用。但是调整tab宽度时，请确保tab够大到能让人无误点击。

# 水平分页 (Swipe View)

## Swipe View 设计

设计指南请阅读 [Android 设计文档的Swipe View指导](#)

另一种横向导航的模式就是水平分页，也叫做 Swipe View。这个模式在容器关联的同级界面上最好用，例如类别列表（世界，金融，技术和健康新闻）。就像Tab，这个模式也允许组合界面，这样父界面就能在布局内嵌入子界面的内容了。

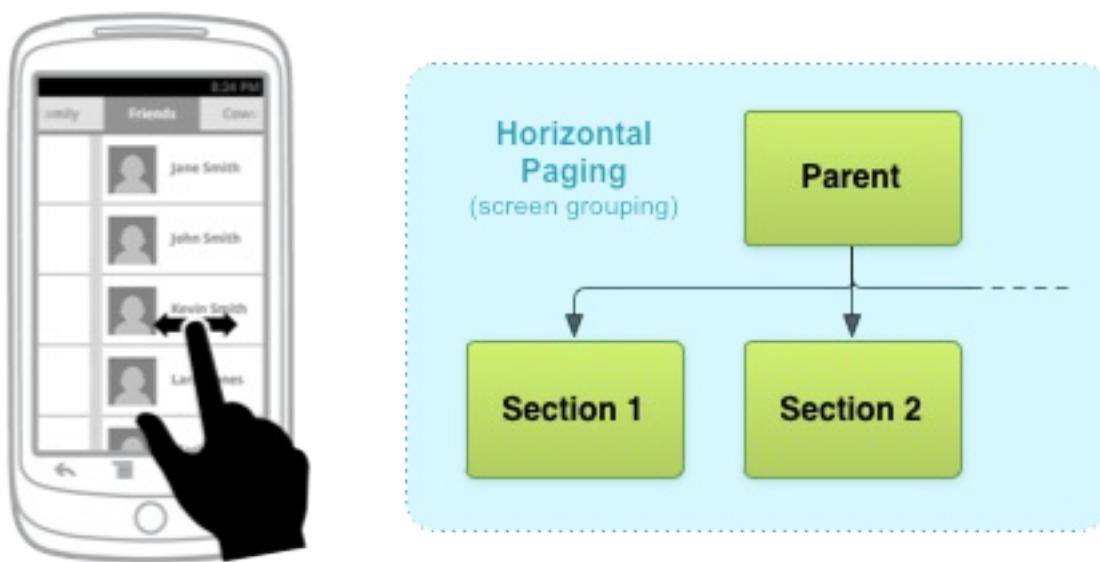


图 6. 水平分页导航例子和对应界面图

在水平分页 UI 中，一次只展示一个子界面（这儿叫页（page））。用户能通过触摸屏幕然后按想要访问相邻页面的方向拖拽导航到同级界面。为补充这种手势交互通常由另一种 UI 元件提示当前页和可访问页。这样能帮助用户发觉内容并且也提供了更多的上下文环境信息给用户。当为区块关联的同级模块使用这种模式的水平导航时，这个做法很有必要。这些提示元件的例子包括mark（标记），滑动标签（scrolling label）和tab：



图 7. 搭配分页的 UI 元件。

当子界面包含水平平移视图时（例如地图）也最好避免使用这种模式，因为这些冲突的交互会威胁你屏幕的易用性。

此外，对于同级关联界面，如果内容类型具有一定相似性而且同级界面数量较少时，水平分页再适合不过了。就这一点，这个模式可以和tab一起用。tab放在内容上方来最大化界面直观性。对于容器关联界面，当界面间有天然的顺序时，水平分页是最符合直觉的，例如页面代表连续的日历日。对于无穷无尽的数据，特别是双向都有内容数据，分页机制效果非常棒。

下节课，我们讨论在内容层级中允许用户往上和回退到之前访问界面的导航的机制。

[下节课：提供向上和时间导航](#)

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/ancestral-temporal.html>

# 提供向上导航与时间导航

既然现在我们能进入应用界面某个层级，我需要提供一个方法来在层级里向上导航到父亲或祖先界面中。此外，我们应该保证通过 *Back* 按钮来操作的历史导航遵循 Android 惯例。

回退/向上导航设计

设计指南请阅读 Android 设计文档的[Navigation](#)模式指导

## 支持时间导航：*Back*

时间导航，或者说在历史的界面间导航，在Android系统中由来已久。不论其他状态如何，所有Android用户都期望*Back*按钮能带他们回到之前的界面。历史界面集全都以用户的Launcher应用为基础（电话的“Home”键）。也就是说，按下*Back*键足够多次数后你应该回到Launcher，之后*Back*键不做任何事情。



图 1. 从 Contacts（联系人）app 中进入电子邮件 app 然后按 Back 键的行为

应用自身通常不必考虑去管理*Back*按钮。系统自己自动处理[task 和 back stack（回退栈）](#)，或者叫历史界面列表。*Back*按钮默认反向访问界面列表，然后当按钮被按下时从列表中移除当前界面。

但是总是有一些你可能需要重写*Back*行为的例子。比如，你屏幕包含一个嵌入的网页浏览器，在这个浏览器中你的用户可和页面元件进行交互来在网页间导航。你可能希望当用户按下设备的*Back*键时触发嵌入浏览器的默认*back*操作。当到达了浏览器内部历史的起始点，你就应该遵从系统*Back*按钮的默认行为了。

## 提供向上导航： *Up* 和 *Home*

Android 3.0 之前，最常见的向上导航的形式以 *Home* 表示。大体上是以在设备 *Menu* 按钮里提供一个 *Home* 的可选项这样的方法来实现，或者 *Home* 按钮出现在屏幕的左上角作为 Action Bar (详询Android 设计的[模式](#)章节) 的一个组件。当选中 *Home* 后，用户被带到界面层级的顶层，通常被叫做应用的主界面。

提供对程序主界面的直接访问能带给用户一种舒适感和安全感。无论位于应用程序何处，如果你在 App 中迷路了，你可以点选 *Home* 然后回到那熟悉的主界面。

Android 3.0 引入了 *Up* 记号，它被展示在了 Action Bar 上代替了上述的 *Home* 按钮。点击 *Up*，用户将被带入到结构中的父界面。这个导航操作通常就是进入前一个界面（就像之前 *Back* 按钮讨论中描述的一样），但是并不是永远都这样。因此，开发者必须保证 *Up* 对于每个界面都会导航到某个既定的父亲界面。

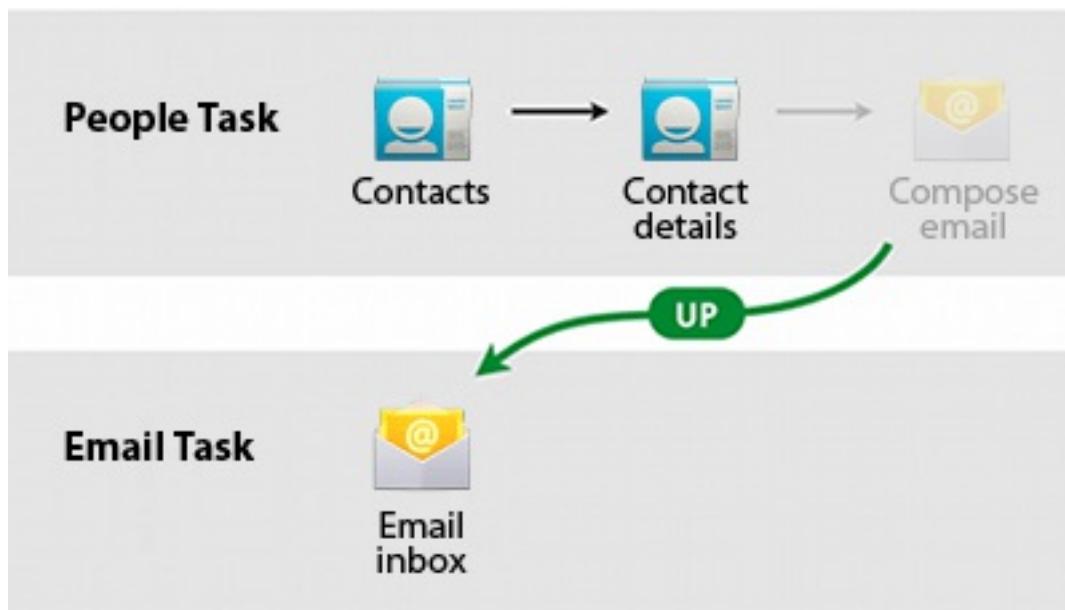


图 2. 从联系人 App 中进入电子邮件 App 然后按 *Up* 导航的行为

某些情况下，*Up* 适合执行某个行为而非导航到一个父亲节点。以 Android 3.0 平板上的 Gmail 应用为例。当查看一封邮件的对话时把设备平放，对话列表和对话详情将并排显示。这是一种[之前课程](#)中的父、子界面组合。然而，当竖屏查看邮件对话时，只有对话详情被显示。*Up* 按钮被用来使父视窗滑入屏幕显示。当左侧视窗可见时再按一次 *Up* 按钮，单个对话便回到全屏的对话列表中。

**实现提醒：** 实现 *Home* 或 *Up* 的最佳做法就是保证清除back stack中的子界面。对于 *Home*，主界面是唯一留在back stack中的界面。对于 *Up* 导航，当前界面也应该从back stack中移除，除非 *Back* 在不同界面层级间导航。你可以将 [FLAG\\_ACTIVITY\\_CLEAR\\_TOP](#) 和 [FLAG\\_ACTIVITY\\_NEW\\_TASK](#) 这两个 Intent 标记一起使用来实现它。

最后一节课中，我们应用现在为止所有课程中讨论的概念来为我们新闻应用例子创建交互设计 Wireframe (线框图)。

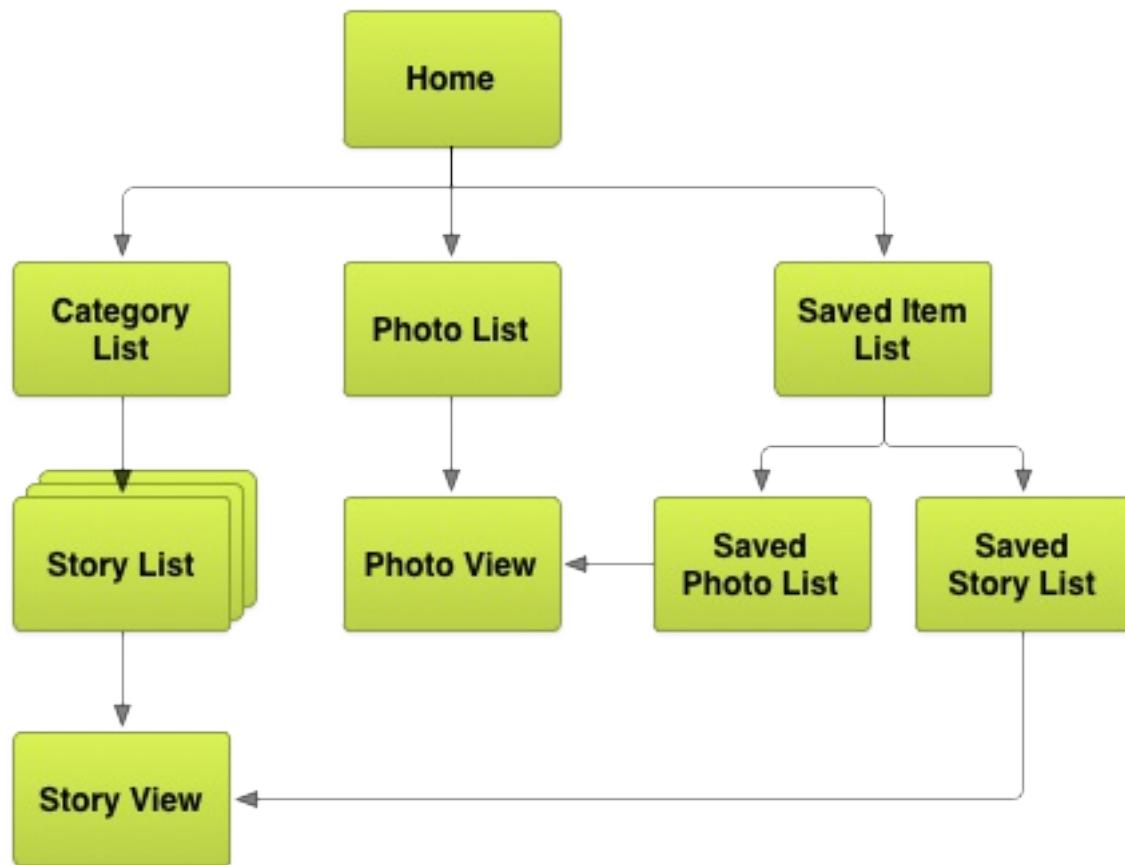
[下节课：综合：设计我们的样例 App](#)

编写: [XizhiXu](#) - 校对:

原文: <http://developer.android.com/training/design-navigation/wireframing.html>

# 综合：设计我们的样例 App

现在我们对导航模式和界面组合技术有了坚实的理解，是时候应用到我们的界面上了。让我再看看我们第一节课上提到的新闻应用的界面完备图：



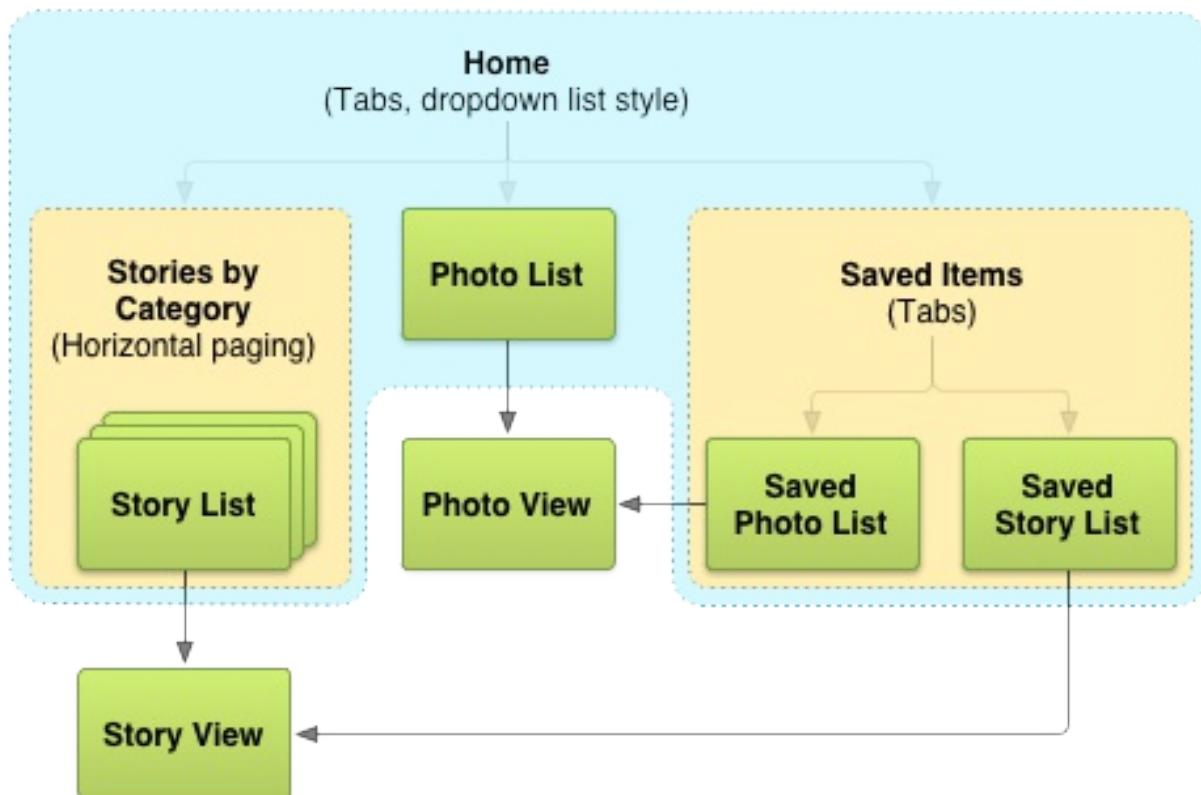
我们下一步得去我们前几节讨论的导航模式选择，然后应用到这个界面图中。这样就能最大化导航速度并且最少化获取内容的点击次数，但又能参考 Android 做法来保证界面的直观性和一致性。此外，我们也需要根据我们不同目标设备的参数做出不同的决定。为方便，我们集中讨论平板和手持设备。

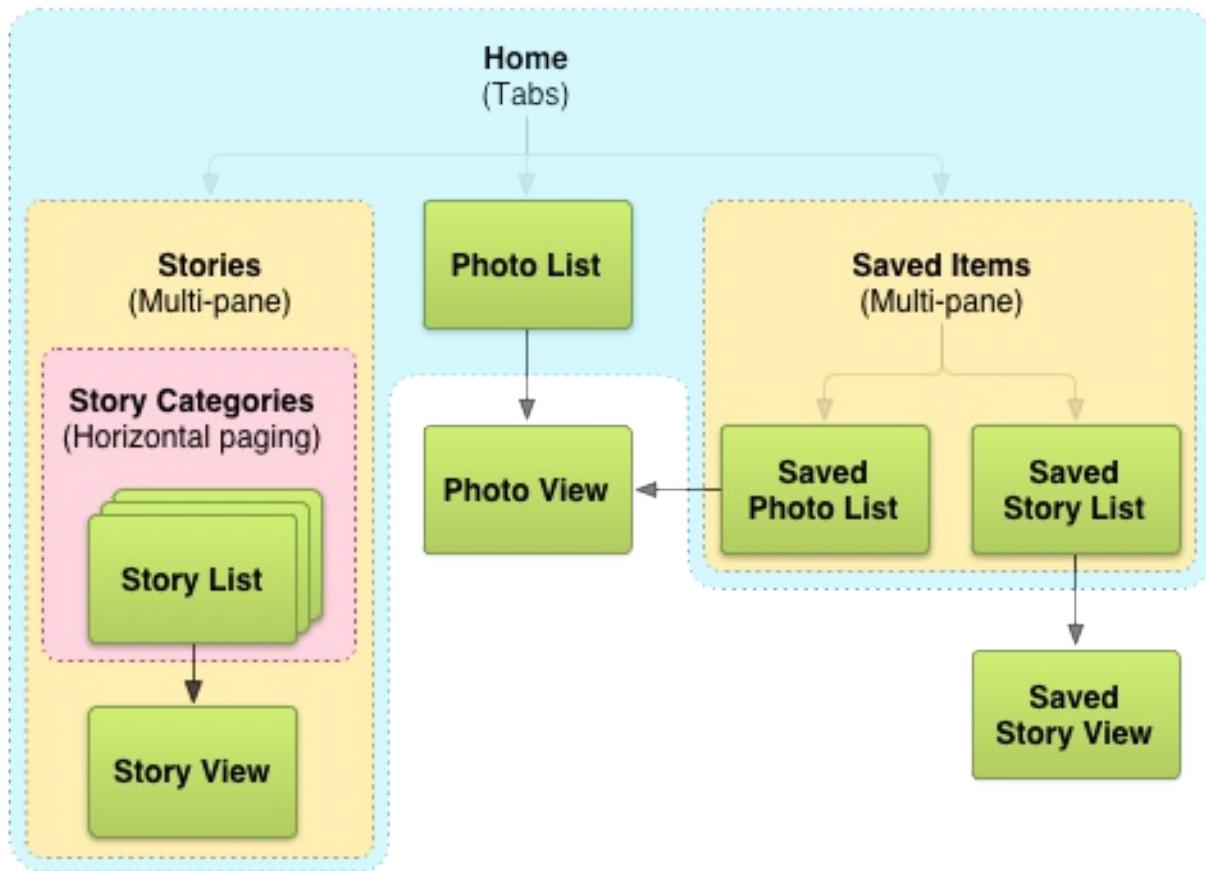
## 选择模式

首先，我们二级界面（新闻类别列表，图片列表和保存列表）可用 Tab 组合在一起。注意到我们不必使用水平排列的 Tab；某些情况下下拉菜单可作为合适的替代品，特别在手机这种窄屏设备上。在手机上，我们能用 Tab 把图片保存列表和新闻保存列表组合到一起，或在平板上用多个纵向排列的内容视窗。

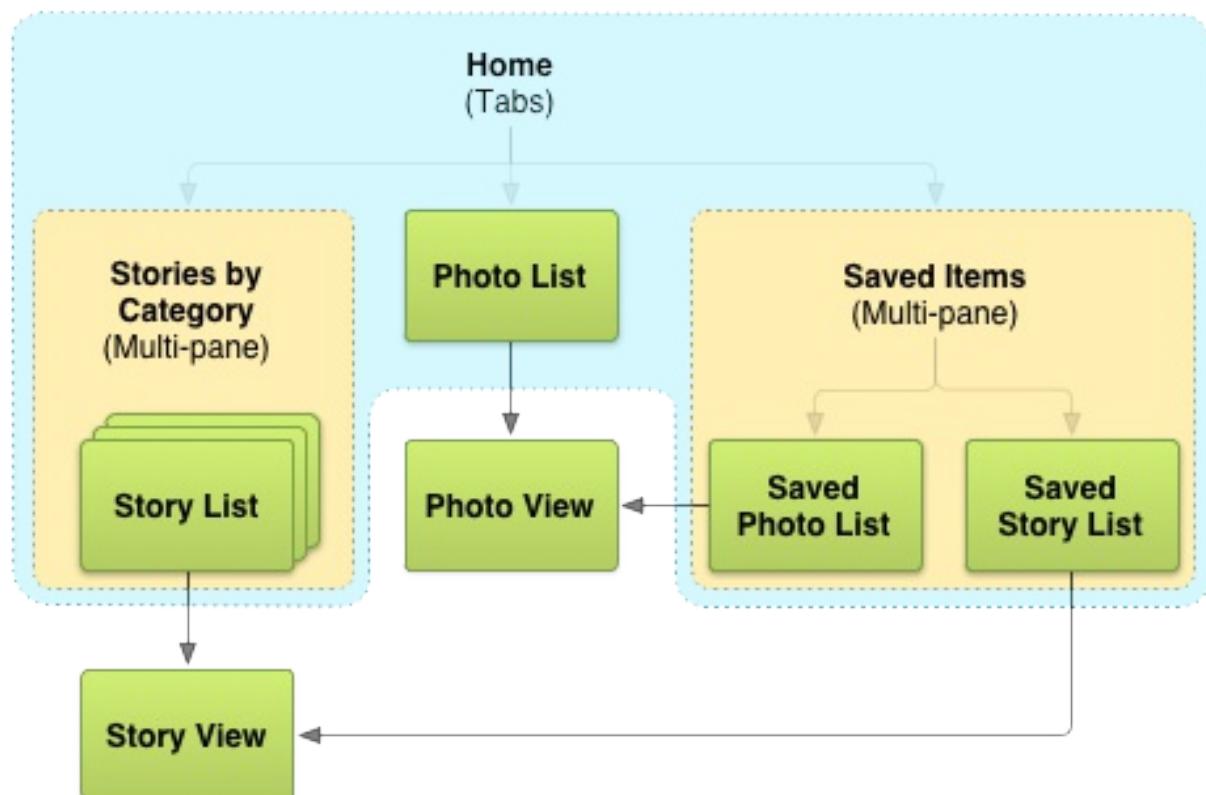
最后，让我们看看如何展示新闻。第一个简化不同新闻类别间导航的选项：使用水平分页，然后再在滑动区域上添加一组标签来提示当前可见和临近的新闻类别。对于平板横屏，我们可以进一步地展示能水平分页的新闻列表界面作为左边的视窗，并且把新闻详情 View 界面作为基础内容视窗放在右边。

下图分别表示在手持设备和平板上应用了这些导航模式后的新界面图。





至此，得好好考虑下界面图的衍化了，以免我们选择的模式实际上用不了（比如当你画应用界面布局的草图时）。下面有个为平板衍化的界面图样例，它并排展示不同类别的新闻列表，但是新闻详情View保持独立。



## 打草稿

[Wireframing](#)就是设计过程中你开始排布界面的那步。发挥你的创造性，想想怎么排列这些UI元件来帮助你的用户在你的App中导航。这时你要记住细枝末节是不重要的（别去想着做个实物）。

最简单快速的起步方法就是用纸笔手画你界面。一旦你开始画，你会发现在你原本的界面图或在你决定使用的模式中有很多实际的问题。某些情况下，模式理论上能很好的解决特定设计问题，但实际上他们可能失效并且给视觉交互添乱（例如，界面上出现了两行Tab）。如果那样，探索下其他的导航模式，或在选择的模式上做点变化，来让你的草稿更优。

当你对初稿满意后，继续用一些软件画你的数字wireframe吧，例如：Adobe® Illustrator，Adobe® Fireworks，OmniGraffle或者向量图工具。选择画图工具时，考虑以下特性：

- 能画体现交互的 wireframe 么？像Adobe® Fireworks就能提供这个功能。
- 有界面“大师”功能（允许不同界面的视觉元素重用）？例如，ActionBar必须在你应用的每个界面都出现。
- 学习曲线怎样？专业向量图工具可能有个陡峭的学习曲线（越学越难），但有些功能小巧的 wireframing 设计工具可能更适合这个任务。

最后，XML布局编辑器，[Android 开发工具包 \(ADT\)](#) 里面的一个Eclipse插件，经常被用来画草图原型。但是，你应当贯注于高质量的布局而非细节视觉设计。

## 创建数字草图

在纸上画完草图并且选择好一款心仪的数字wireframing工具后，你可以创建一个数字wireframe作为你应用视觉设计的起点。下面就是一些我们新闻客户端wireframe例子，他们和我们之前的界面图一一对应。



图 5. 新闻客户端手机竖屏Wireframe样例（下载 [SVG](#) 图）



图 6. 新闻客户端平板横屏Wireframe样例（下载 [SVG 图](#)）

（[下载表示设备的 Wireframe 的 SVG 图](#)）

## 下一步

现在你已经为你的应用设计出了高效直观的 App 内部导航，你可用开始花时间来为单个界面改善 UI 了。例如，展示交互内容时，你可以选择使用更花哨的控件来代替简单的文本标签，图像和按钮。你也可以开始定义你应用的视觉风格。在这过程中把你品牌的元素作为视觉语言融入其中吧。

最后，也适时实现你的设计吧，使用 Android SDK 为你的应用写写代码。想开始？看看下面的这些资源吧：

- [开发者指导：UI](#) :学习如何用 Android SDK 实现你的 UI 设计。
- [ActionBar](#) :实现tab，向上导航，屏幕上动作，等等。
- [Fragment](#) :实现可重用，多视窗布局
- [支持库](#) :用ViewPager实现水平分页（Swipe View）

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/implementing-navigation/index.html>

# 实现高效的导航

这节课将会演示如何实现在[Designing Effective Navigation](#)中所详述的关键导航设计模式。

在阅读这节课程内容之后，你会对如何使用tabs, swipe views, 和navigation drawer实现导航模式有一个深刻的理解。也会明白如何提供合适的向前向后导航(Up and Back navigation)。

**Note:**本节课中的几个元素需要使用[Support Library](#) API。如果你之前没有使用过Support Library，可以按照[Support Library Setup](#)文档说明来使用。

# Lessons

- [使用Tabs创建Swipe View](#)

学习如何在action bar中实现tab，并提供横向分页(swipe views)在tab之间导航切换。

- [创建抽屉导航\(Navigation Drawer\)](#)

学习如何建立隐藏于屏幕边上的界面，通过划屏(swipe)或点击action bar中的app图标来显示这个界面。

- [提供向上导航](#)

学习如何使用action bar中的app图标实现向上导航

- [提供适当的向后导航](#)

学习如何正确处理特殊情况下的向后按钮(Back button)，包括在通知或app widget中的深度链接，如何将activity插入后退栈(back stack)中。

- [实现Descendant Navigation](#)

学习更精细地导航进入你的应用信息层。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/implementing-navigation/lateral.html>

# 使用Tabs创建Swipe视图

Swipe View提供在同级屏幕中的横向导航，例如通过横向划屏手势切换的tab(一种称作横向分页的模式)。这节课会教你如何使用swipe view创建一个tab layout实现在tab之间切换，或显示一个标题条替代tab。

## Swipe View Design

在实现这些功能之前，你要先明白在[Designing Effective Navigation, Swipe Views](#) design guide中的概念和建议

# 实现Swipe View

你可以使用[Support Library](#)中的[ViewPager](#)控件在你的app中创建swipe view。[ViewPager](#)是一个子视图在layout上相互独立的布局控件(layout widget)。

使用[ViewPager](#)来设置你的layout，要添加一个<ViewPager>元素到你的XML layout中。例如，在你的swipe view中如果每一个页面都会占用整个layout，那么你的layout应该是这样：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

要插入每一个页面的子视图，你需要把这个layout与[PagerAdapter](#)挂钩。有两种adapter(适配器)你可以用：

## [FragmentPagerAdapter](#)

在同级屏幕(sibling screen)只有少量的几个固定页面时，使用这个最好。

## [FragmentStatePagerAdapter](#)

当根据对象集的数量来划分页面，即一开始页面的数量未确定时，使用这个最好。当用户切换到其他页面时，fragment会被销毁来降低内存消耗。

例如，这里的代码是当你使用[FragmentStatePagerAdapter](#)来在[Fragment](#)对象集合中进行横屏切换：

```
public class CollectionDemoActivity extends FragmentActivity {
    // 当被请求时，这个adapter会返回一个DemoObjectFragment,
    // 代表在对象集中的一个对象。
    DemoCollectionPagerAdapter mDemoCollectionPagerAdapter;
    ViewPager mViewPager;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_collection_demo);

        // ViewPager和他的adapter使用了support library
        // fragments,所以要用getSupportFragmentManager.
        mDemoCollectionPagerAdapter =
            new DemoCollectionPagerAdapter(
                getSupportFragmentManager());
        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mDemoCollectionPagerAdapter);
    }
}

// 因为这是一个对象集所以使用FragmentStatePagerAdapter,
// 而不是FragmentPagerAdapter.
public class DemoCollectionPagerAdapter extends FragmentStatePagerAdapter
```

```
public DemoCollectionPagerAdapter(FragmentManager fm) {
    super(fm);
}

@Override
public Fragment getItem(int i) {
    Fragment fragment = new DemoObjectFragment();
    Bundle args = new Bundle();
    // 我们的对象只是一个整数 :-P
    args.putInt(DemoObjectFragment.ARG_OBJECT, i + 1);
    fragment.setArguments(args);
    return fragment;
}

@Override
public int getCount() {
    return 100;
}

@Override
public CharSequence getPageTitle(int position) {
    return "OBJECT " + (position + 1);
}

// 这个类的实例是一个代表了数据集中一个对象的fragment
public static class DemoObjectFragment extends Fragment {
    public static final String ARG_OBJECT = "object";

    @Override
    public View onCreateView(LayoutInflater inflater,
                           ViewGroup container, Bundle savedInstanceState) {
        // 最后两个参数保证LayoutParams能被正确填充
        View rootView = inflater.inflate(
            R.layout.fragment_collection_object, container, false);
        Bundle args = getArguments();
        ((TextView) rootView.findViewById(android.R.id.text1)).set
            Integer.toString(args.getInt(ARG_OBJECT)));
        return rootView;
    }
}
```

这个例子只显示了创建swipe view的必要代码。下面一节向你说明如何通过添加tab使导航更方便在页面间切换。

## 添加Tab到Action Bar

Action bar [tab](#)能给用户提供更熟悉的界面来在app的同级屏幕中切换和分辨。

使用[ActionBar](#)来创建tab，你需要启用[NAVIGATION\\_MODE\\_TABS](#)，然后创建几个[ActionBar.Tab](#)的实例，并对每个实例实现[ActionBar.TabListener](#)接口。例如在你的activity的[onCreate\(\)](#)方法中，你可以使用与下面相似的代码：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    final ActionBar actionBar = getActionBar();  
    ...  
  
    // 指定在action bar中显示tab.  
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);  
  
    // 创建一个tab listener，在用户切换tab时调用。  
    ActionBar.TabListener tabListener = new ActionBar.TabListener()  
        public void onTabSelected(ActionBar.Tab tab, FragmentTrans  
            // 显示指定的tab  
        }  
  
        public void onTabUnselected(ActionBar.Tab tab, FragmentTra  
            // 隐藏指定的tab  
        }  
  
        public void onTabReselected(ActionBar.Tab tab, FragmentTra  
            // 可以忽略这个事件  
        }  
    };  
  
    // 添加3个tab，并指定tab的文字和TabListener  
    for (int i = 0; i < 3; i++) {  
        actionBar.addTab(  
            actionBar.newTab()  
                .setText("Tab " + (i + 1))  
                .setTabListener(tabListener));  
    }  
}
```

根据你如何创建你的内容来处理[ActionBar.TabListener](#)回调改变tab。但是如果你是像上面那样，通过[ViewPager](#)对每个tab使用fragment，下面这节就会说明当用户选择一个tab时如何切换页面，当用户划屏切换页面时如何更新相应页面的tab。

## 使用Swipe View切换Tab

当用户选择tab时，在ViewPager中切换页面，需要实现ActionBar.TabListener来调用在ViewPager中的setCurrentItem()来选择相应的页面：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
  
    // Create a tab listener that is called when the user changes  
    ActionBar.TabListener tabListener = new ActionBar.TabListener(  
        public void onTabSelected(ActionBar.Tab tab, FragmentTrans  
            // 当tab被选中时，切换到ViewPager中相应的页面。  
            mViewPager.setCurrentItem(tab.getPosition());  
        }  
        ...  
    );  
}
```

同样的，当用户通过触屏手势(touch gesture)切换页面时，你也应该选择相应的tab。你可以通过实现ViewPager.OnPageChangeListener接口来设置这个操作，当页面变化时当前的tab也相应变化。例如：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
  
    mViewPager = (ViewPager) findViewById(R.id.pager);  
    mViewPager.setOnPageChangeListener(  
        new ViewPager.SimpleOnPageChangeListener() {  
            @Override  
            public void onPageSelected(int position) {  
                // 当划屏切换页面时，选择相应的tab。  
                getActionBar().setSelectedNavigationItem(position);  
            }  
        } );  
    ...  
}
```

## 使用标题条替代Tab

如果你不想使用action bar tab，而想使用[scrollable tabs](#)来提供一个更简短的可视化配置，你可以在swipe view中使用[PagerTitleStrip](#)。

下面是一个内容为[ViewPager](#)，有一个[PagerTitleStrip](#)顶端对齐的activity的layout XML文件示例。单个页面(adapter提供)占据[ViewPager](#)中的剩余空间。

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTitleStrip
        android:id="@+id/pager_title_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#33b5e5"
        android:textColor="#fff"
        android:paddingTop="4dp"
        android:paddingBottom="4dp" />

</android.support.v4.view.ViewPager>
```

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/implementing-navigation/nav-drawer.html>

# 创建抽屉式导航(navigation drawer)

navigation drawer是一个在屏幕左侧边缘显示导航选项的面板。大部分时候是隐藏的，当用户从屏幕左侧划屏，或在top level模式的app中点击action bar中的app图标时，才会显示。

这节课叙述如何使用[Support Library](#)中的[DrawerLayout API](#)，来实现navigation drawer。

## Navigation Drawer Design

在你决定在你的app中使用Navigation Drawer之前，你应该先理解在[Navigation Drawer design guide](#)中定义的使用情况和设计准则。

# 创建一个Drawer Layout

要添加一个navigation drawer，在你的用户界面layout中声明一个用作root view(根视图)的[DrawerLayout](#)对象。在[DrawerLayout](#)中为屏幕添加一个包含主要内容的view(当drawer隐藏时的主layout)，和其他一些包含navigation drawer内容的view。

例如，下面的layout使用了有两个子视图(child view)的[DrawerLayout](#):一个[FrameLayout](#)用来包含主要内容(在运行时被[Fragment](#)填入)，和一个navigation drawer使用的[ListView](#)。

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- 包含主要内容的 view -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!-- navigation drawer(抽屉式导航) -->
    <ListView android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

这个layout展示了一些layout的重要特点:

- 主内容view(上面的[FrameLayout](#))，在[DrawerLayout](#)中必须是第一个子视图，因为XML的顺序代表着Z轴(垂直于手机屏幕)的顺序，并且drawer必须在内容的前端。
- 主内容view被设置为匹配父视图的宽和高，因为当navigation drawer隐藏时，主内容表示整个UI部分。
- drawer视图([ListView](#))必须使用`android:layout_gravity`属性指定它的**horizontal gravity**。为了支持从右边阅读的语言(right-to-left(RTL) language)，指定它的值为"start"而不是"left"(当layout是RTL时drawer在右边显示)。
- drawer视图以dp为单位指定它的宽和高来匹配父视图。drawer的宽度不能大于320dp，这样用户总能看到部分主要内容。

## 初始化Drawer List

在你的activity中，首先你要做的事就是要初始化drawer的item列表。这要根据你的app内容来处理，但是一个navigation drawer通常由一个[ListView](#)组成，所以列表应该通过一个[Adapter](#)(例如[ArrayAdapter](#)或[SimpleCursorAdapter](#))填入。

例如，如何使用一个字符串数组([string array](#))来初始化导航列表(navigation list):

```
public class MainActivity extends Activity {  
    private String[] mPlanetTitles;  
    private DrawerLayout mDrawerLayout;  
    private ListView mDrawerList;  
    ...  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mPlanetTitles = getResources().getStringArray(R.array.planet_titles);  
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);  
        mDrawerList = (ListView) findViewById(R.id.left_drawer);  
  
        // 为list view设置adapter  
        mDrawerList.setAdapter(new ArrayAdapter<String>(this,  
            R.layout.drawer_list_item, mPlanetTitles));  
        // 为list设置click listener  
        mDrawerList.setOnItemClickListener(new DrawerItemClickListener());  
  
        ...  
    }  
}
```

这段代码也调用了[setOnItemClickListener\(\)](#)来接收navigation drawer列表的点击事件。下一节会说明如何实现这个接口，并且当用户选择一个item时如何改变内容视图(content view)。

## 处理导航的点击事件

当用户选择drawer列表中的item，系统会调用在[setOnItemClickListener\(\)](#)中所设置的[OnItemClickListener](#)的[onItemClick\(\)](#)。

在[onItemClick\(\)](#)方法中做什么，取决于你如何实现你的app结构([app structure](#))。在下面的例子中，每选择一个列表中的item，就插入一个不同的[Fragment](#)到主内容视图中([FrameLayout](#)元素通过R.id.content\_frame ID辨识)：

```
private class DrawerItemClickListener implements ListView.OnItemCl
    @Override
    public void onItemClick(AdapterView parent, View view, int pos
        selectItem(position);
    }
}

/** 在主内容视图中交换fragment */
private void selectItem(int position) {
    // 创建一个新的fragment并且根据行星的位置来显示
    Fragment fragment = new PlanetFragment();
    Bundle args = new Bundle();
    args.putInt(PlanetFragment.ARG_PLANET_NUMBER, position);
    fragment.setArguments(args);

    // 通过替换已存在的fragment来插入新的fragment
    FragmentManager fragmentManager = getFragmentManager();
    fragmentManager.beginTransaction()
        .replace(R.id.content_frame, fragment)
        .commit();

    // 高亮被选择的item，更新标题，并关闭drawer
    mDrawerList.setItemChecked(position, true);
    setTitle(mPlanetTitles[position]);
    mDrawerLayout.closeDrawer(mDrawerList);
}

@Override
public void setTitle(CharSequence title) {
    mTitle = title;
    getSupportActionBar().setTitle(mTitle);
}
```

## 监听打开和关闭事件

要监听drawer的打开和关闭事件，在你的`DrawerLayout`中调用`setDrawerListener()`，并传入一个`DrawerLayout.DrawerListener`的实现。这个接口提供drawer事件的回调例如`onDrawerOpened()`和`onDrawerClosed()`。

但是，如果你的activity包含有`action bar`可以不用实现`DrawerLayout.DrawerListener`，你可以继承`ActionBarDrawerToggle`来替代。`ActionBarDrawerToggle`实现了`DrawerLayout.DrawerListener`，所以你仍然可以重写这些回调。这么做也能使`action bar`图标和 navigation drawer的交互操作变得更容易(在下节详述)。

如`Navigation Drawer` design guide中所述,当drawer可见时，你应该修改`action bar`的内容，比如改变标题和移除与主文字内容相关的action item。下面的代码向你说明如何通过`ActionBarDrawerToggle`类的实例，重写`DrawerLayout.DrawerListener`的回调方法来实现这个目的:

```
public class MainActivity extends Activity {
    private DrawerLayout mDrawerLayout;
    private ActionBarDrawerToggle mDrawerToggle;
    private CharSequence mDrawerTitle;
    private CharSequence mTitle;
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...

        mTitle = mDrawerTitle = getTitle();
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
                R.drawable.ic_drawer, R.string.drawer_open, R.string.drawer_close);
        mDrawerToggle.setDrawerIndicatorEnabled(true);
        mDrawerToggle.syncState();

        // 当drawer处于完全关闭的状态时调用 */
        public void onDrawerClosed(View view) {
            super.onDrawerClosed(view);
            getSupportActionBar().setTitle(mTitle);
            invalidateOptionsMenu(); // 创建对onPrepareOptionsMenu的调用
        }

        // 当drawer处于完全打开的状态时调用 */
        public void onDrawerOpened(View drawerView) {
            super.onDrawerOpened(drawerView);
            getSupportActionBar().setTitle(mDrawerTitle);
            invalidateOptionsMenu(); // 创建对onPrepareOptionsMenu的调用
        }
    };

    // 设置drawer触发器为DrawerListener
    mDrawerLayout.setDrawerListener(mDrawerToggle);
}
```

```
/* 当invalidateOptionsMenu() 调用时调用 */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // 如果nav drawer是打开的，隐藏与内容视图相关联的action items
    boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerList);
    menu.findItem(R.id.action_websearch).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}
```

下一节会描述ActionBarDrawerToggle的构造参数，和处理与action bar交互所需的其他步骤。

# 使用App图标来打开和关闭

用户可以在屏幕左侧使用划屏手势来打开和关闭navigation drawer，但是如果你使用[action bar](#)你也应该允许用户通过点击app图标来打开或关闭。并且app图标也应该使用一个特殊的图标来指明navigation drawer的存在。你可以通过使用上一节所说的[ActionBarDrawerToggle](#)来实现所有的这些操作。

要使[ActionBarDrawerToggle](#)起作用，通过它的构造函数创建一个实例，需要用到以下参数：

- [Activity](#)用来容纳drawer。
- [DrawerLayout](#)。
- 一个drawable资源用作drawer指示器。标准的navigation drawer可以在[Download the Action Bar Icon Pack](#)获的
- 一个字符串资源描述"打开抽屉"操作(便于访问)
- 一个字符串资源描述"关闭抽屉"操作(便于访问)

那么，不论你是否创建了用作drawer监听器的[ActionBarDrawerToggle](#)的子类，你都需要在activity生命周期中的某些地方根据你的[ActionBarDrawerToggle](#)来调用。

```
public class MainActivity extends Activity {  
    private DrawerLayout mDrawerLayout;  
    private ActionBarDrawerToggle mDrawerToggle;  
    ...  
  
    public void onCreate(Bundle savedInstanceState) {  
        ...  
  
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_la  
        mDrawerToggle = new ActionBarDrawerToggle(  
            this, /* 承载 Activity */  
            mDrawerLayout, /* DrawerLayout 对象 */  
            R.drawable.ic_drawer, /* nav drawer 图标用来替换'Up'  
            R.string.drawer_open, /* "打开 drawer" 描述 */  
            R.string.drawer_close /* "关闭 drawer" 描述 */  
        ) {  
  
            /** 当drawer处于完全关闭的状态时调用 */  
            public void onDrawerClosed(View view) {  
                super.onDrawerClosed(view);  
                getActionBar().setTitle(mTitle);  
            }  
  
            /** 当drawer处于完全打开的状态时调用 */  
            public void onDrawerOpened(View drawerView) {  
                super.onDrawerOpened(drawerView);  
                getActionBar().setTitle(mDrawerTitle);  
            }  
        };  
  
        // 设置drawer触发器为DrawerListener
```

```
mDrawerLayout.setDrawerListener(mDrawerToggle);

getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 在onRestoreInstanceState发生后，同步触发器状态。
    mDrawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mDrawerToggle.onConfigurationChanged(newConfig);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // 将事件传递给ActionBarDrawerToggle，如果返回true，表示app 图
    if (mDrawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
    // 处理你的其他action bar items...

    return super.onOptionsItemSelected(item);
}

...
}
```

一个完整的navigation drawer例子,可以在原文页面顶端的sample下载

编写: [Lin-H](#) - 校对:

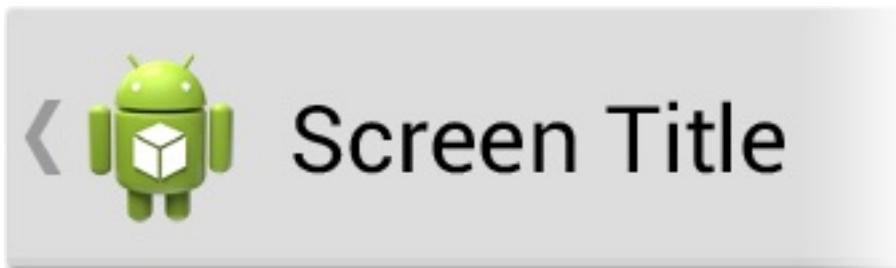
原文: <http://developer.android.com/training/implementing-navigation/ancestral.html>

# 提供向上的导航

所有不是从主屏幕("home"屏幕)进入app的，都应该给用户提供一种方法，通过点击[action bar](#)中的Up按钮。可以回到app的结构层次中逻辑父屏幕。本课程向你说明如何正确地实现这一操作。

## Up Navigation Design

[Designing Effective Navigation](#)和the [Navigation](#) design guide中描述了向上导航的概念和设计准则。



**Figure 1.** action bar中的Up按钮.

## 指定父Activity

要实现向上导航，第一步就是为每一个activity声明合适的父activity。这么做可以使系统简化导航模式，例如向上导航，因为系统可以从manifest文件中判断它的逻辑父(logical parent)activity。

从Android 4.1 (API level 16)开始，你可以通过指定[`<activity>`](#)元素中的[`android:parentActivityName`](#)属性来声明每一个activity的逻辑父activity。

如果你的app需要支持Android 4.0以下版本，在你的app中包含[`Support Library`](#)并添加[`<meta-data>`](#)元素到[`<activity>`](#)中。然后指定父activity的值为[`android.support.PARENT\_ACTIVITY`](#)，并匹配[`android:parentActivityName`](#)的值。

例如：

```
<application ... >
    ...
    <!-- main/home activity (没有父activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- 主activity的一个子activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivit
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity"
        <!-- 父activity的meta-data，用来支持4.0以下版本 -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

在父activity这样声明后，你可以使用[`NavUtils`](#) API进行向上导航操作，就像下一面这节。

## 添加向上操作(Up Action)

要使用action bar的app图标来完成向上导航，需要调用[setDisplayHomeAsUpEnabled\(\)](#):

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
    getActionBar().setDisplayHomeAsUpEnabled(true);  
}
```

这样，在app旁添加了一个左向符号，并用作操作按钮。当用户点击它时，你的activity会接收一个对[onOptionsItemSelected\(\)](#)的调用。操作的ID是android.R.id.home。

## 向上导航至父activity

要在用户点击app图标时向上导航，你可以使用[NavUtils](#)类中的静态方法[navigateUpFromSameTask\(\)](#)。当你调用这一方法时，系统会结束当前的activity并启动(或恢复)相应的父activity。如果目标activity在任务的后退栈中(back stack)，则目标activity会像[FLAG\\_ACTIVITY\\_CLEAR\\_TOP](#)定义的那样，提到栈顶。

例如：

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // 对action bar的Up/Home按钮做出反应  
        case android.R.id.home:  
            NavUtils.navigateUpFromSameTask(this);  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

但是，只能是当你的app拥有当前任务(**current task**)(用户从你的app中发起这一任务)时[navigateUpFromSameTask\(\)](#)才有用。如果你的activity是从别的app的任务中启动的话，向上导航操作就应该创建一个属于你的app的新任务，并需要你创建一个新的后退栈。

## 用新的后退栈来向上导航

如果你的activity提供了任何允许被别的app启动的[intent filters](#)，那么你应该实现[onOptionsItemSelected\(\)](#)回调，在用户从别的app任务进入你的activity后，点击Up按钮，在向上导航之前你的app用相应的后退栈开启一个新的任务。

在这么做之前，你可以先调用[shouldUpRecreateTask\(\)](#)来检查当前的activity实例是否在另一个不同的app任务中。如果返回true，就使用[TaskStackBuilder](#)创建一个新任务。或者，你可以向上面那样使用[navigateUpFromSameTask\(\)](#)方法。

例如：

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // 对action bar的Up/Home按钮做出反应  
        case android.R.id.home:  
            Intent upIntent = NavUtils.getParentActivityIntent(this);  
            if (NavUtils.shouldUpRecreateTask(this, upIntent)) {  
                // 这个activity不是这个app任务的一部分，所以当向上导航时创建  
                // 用合成后退栈(synthesized back stack)创建一个新任务。  
                TaskStackBuilder.create(this)  
                    // 添加这个activity的所有父activity到后退栈中  
                    .addNextIntentWithParentStack(upIntent)  
                    // 向上导航到最近的一个父activity  
                    .startActivities();  
            } else {  
                // 这个activity是这个app任务的一部分，所以
```

```
// 向上导航至逻辑父activity.  
NavUtils.navigateUpTo(this, upIntent);  
}  
return true;  
}  
return super.onOptionsItemSelected(item);  
}
```

**Note:**为了能使[addNextIntentWithParentStack\(\)](#)方法起作用，你必须像上面说的那样，在你的manifest文件中使用[android:parentActivityName](#)(和相应的[`<meta-data>`](#)元素)属性声明所有的activity的逻辑父activity。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/implementing-navigation/temporal.html>

# 提供向后的导航

向后导航(Back navigation)是用户根据屏幕历史记录返回之前所查看的界面。所有Android设备都可以为这种导航提供后退按钮，所以你的app不需要在UI中添加后退按钮。

在几乎所有情况下，当用户在应用中进行导航时，系统会保存activity的后退栈。这样当用户点击后退按钮时，系统可以正确地向后导航。但是，有少数几种情况需要手动指定app的后退操作，来提供更好的用户体验。

## Back Navigation Design

在继续阅读篇文章之前，你应该先在[Navigation design guide](#)中对后退导航的概念和设计准则有个了解。

手动指定后退操作需要的导航模式：

- 当用户从[notification](#)(通知), [app widget](#), [navigation drawer](#)直接进入深层次activity。
- 用户在[fragment](#)之间切换的某些情况。
- 当用户在[WebView](#)中对网页进行导航。

下面说明如何在这几种情况下实现恰当的向后导航。

## 为深度链接合并新的后退栈

一般而言，当用户从一个activity导航到下一个时，系统会递增地创建后退栈。但是当用户从一个在自己的任务中启动activity的深度链接进入app，你就有必要去同步新的后退栈，因为新的activity是运行在一个没有任何后退栈的任务中。

例如，当用户从通知进入你的app中的深层activity时，你应该添加别的activity到你的任务的后退栈中，这样当点击后退(Back)时向上导航，而不是退出app。这个模式在[Navigation design guide](#)中有更详细的介绍。

### 在manifest中指定父activity

从Android 4.1 (API level 16)开始，你可以通过指定[`<activity>`](#)元素中的[`android:parentActivityName`](#)属性来声明每一个activity的逻辑父activity。这样系统可以使导航模式变得更容易，因为系统可以根据这些信息判断Back Up navigation的路径。

如果你的app需要支持Android 4.0以下版本，在你的app中包含[Support Library](#)并添加[`<meta-data>`](#)元素到[`<activity>`](#)中。然后指定父activity的值为`android.support.PARENT_ACTIVITY`，并匹配[`android:parentActivityName`](#)的值。

例如：

```
<application ... >
    ...
    <!-- main/home activity (没有父activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- 主activity的一个子activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivit"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity"
        <!-- 4.1 以下的版本需要使用meta-data元素 -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

当父activity用这种方式声明，你就可以使用[NavUtils API](#)，通过确定每个activity相应的父activity来同步新的后退栈。

### 在启动activity时创建后退栈

在发生用户进入app的事件时，开始添加activity到后退栈中。就是说，使用[TaskStackBuilder API](#)定义每个被放到新后退栈的activity，不使用[`startActivity\(\)`](#)。然后调用[`startActivities\(\)`](#)来启动目标activity，或调用[`getPendingIntent\(\)`](#)来创建相应的[PendingIntent](#)。

例如，当用户从通知进入你的app中的深层activity时，你可以使用这段代码来创建一个启动activity并把新后退栈插入目标任务的[PendingIntent](#)。

```
// 当用户选择通知时，启动activity的intent
Intent detailsIntent = new Intent(this, DetailsActivity.class);

// 使用TaskStackBuilder创建后退栈，并获取PendingIntent
PendingIntent pendingIntent =
    TaskStackBuilder.create(this)
        // 添加所有DetailsActivity的父activity到栈中
        // 然后再添加DetailsActivity自己
        .addNextIntentWithParentStack(upIntent)
        .getPendingIntent(0, PendingIntent.FLAG_UP

NotificationCompat.Builder builder = new NotificationCompat.Bui
builder.setContentIntent(pendingIntent);
...
```

产生的PendingIntent不仅指定了启动哪个activity(被detailsIntent所定义)还指定了要插入任务(所有被detailsIntent定义的DetailsActivity)的后退栈。所以当DetailsActivity启动时，点击Back向后导航至每一个DetailsActivity类的父activity。

**Note:**为了使[addNextIntentWithParentStack\(\)](#)方法起作用，像上面所说那样，你必须在你的manifest文件中使用[android:parentActivityName](#)(和相应的元素[`<meta-data>`](#))属性声明每个activity的逻辑父activity。

## 为Fragment实现向后导航

当在app中使用fragment时，个别的[FragmentTransaction](#)对象可以代表要加入后退栈中变化的内容。例如，如果你要在手机上通过交换fragment实现一个[master/detail flow](#)(主/详细流程)，你就要保证点击Back按钮可以从detail screen返回到master screen。要这么做，你可以在提交事务(transaction)之前调用[addBackStack\(\)](#):

```
// 使用framework FragmentManager  
// 或support package FragmentManager (getSupportFragmentManager).  
getSupportFragmentManager().beginTransaction()  
    .add(detailFragment, "detail")  
    // 提交这一事务到后退栈中  
    .addToBackStack()  
    .commit();
```

当后退栈中有[FragmentTransaction](#)对象并且用户点击Back按钮时,[FragmentManager](#)会从后退栈中弹出最近的事务，然后执行反向操作(例如如果事务添加了一个fragment，那么就删除一个fragment)。

**Note:**当事务用作水平导航(例如切换tab)或者修改内容外观(例如在调整filter时)时，不要将这个事务添加到后退栈中。更多关于向后导航的恰当时长的信息，详见[Navigation design guide](#)。

如果你的应用更新了别的UI元素来反应当前的fragment状态，例如action bar，记得当你提交事务时更新UI。除了在提交事务的时候，在后退栈发生变化时也要更新你的UI。你可以设置一个[FragmentManager.OnBackStackChangedListener](#)来监听[FragmentTransaction](#)什么时候复原:

```
getSupportFragmentManager().addOnBackStackChangedListener(  
    new FragmentManager.OnBackStackChangedListener() {  
        public void onBackStackChanged() {  
            // 在这里更新你的UI  
        }  
    } );
```

## 为WebView实现向后导航

如果你的应用的某部分中包含有[WebView](#), 可以通过浏览器历史使用Back。要这么做, 如果[WebView](#)有历史记录, 你可以重写onBackPressed()并代理给[WebView](#):

```
@Override  
public void onBackPressed() {  
    if (mWebView.canGoBack()) {  
        mWebView.goBack();  
        return;  
    }  
  
    // 否则遵从系统的默认操作.  
    super.onBackPressed();  
}
```

要注意当使用这一机制时, 高动态化的页面会产生大量历史。会生成大量历史的页面, 例如经常改变文件散列(document hash)的页面, 当要退出你的activity时, 这会使你的用户感到繁琐。

更多关于使用[WebView](#)的信息, 详见[Building Web Apps in WebView](#)。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/implementing-navigation/descendant.html>

# 实现Descendant Navigation

Descendant Navigation是用来向下导航至应用的信息层次。在[Designing Effective Navigation](#)和[Android Design: Application Structure](#)中说明。

Descendant navigation通常使用[Intent](#)和[startActivity\(\)](#)实现，或使用[FragmentTransaction](#)对象添加fragment到一个activity中。这节课程涵盖了在实现Descendant navigation时遇到的其他有趣的情况。

## 在手机和平板(Tablet)上实现Master/Detail Flow

在master/detail导航流程(navigation flow)中， master screen(主屏幕)包含一个集合中item的列表， detail screen(详细屏幕)显示集合中特定item的详细信息。实现从master screen到detail screen的导航是Descendant Navigation的一种形式。

手机触摸屏非常适合一次显示一种屏幕(master screen或detail screen)；这一想法在[Planning for Multiple Touchscreen Sizes](#)中进一步说明。在这种情况下，一般使用[Intent](#)启动detail screen来实现activity Descendant navigation。另一方面，平板的显示，特别是用横屏来浏览时，最适合一次显示多个内容窗格， master内容在左边， detail在右边。在这里一般就使用[FragmentTransaction](#)实现descendant navigation。[FragmentTransaction](#)用来添加、删除或用新内容替换detail窗格(pane)。

实现这一模式的基础内容在Designing for Multiple Screens的[Implementing Adaptive UI Flows](#)课程中说明。课程中说明了如何在手机上使用两个activity，在平板上使用一个activity来实现master/detail flow。

## 导航至外部Activities

有很多情况，是从别的应用下降(descend)至你的应用信息层次(application's information hierarchy)再到activity。例如，当正在浏览手机通讯录中联系信息的details screen，子屏幕详细显示由社交网络联系提供的最近文章，子屏幕可就可以属于一个社交网络应用。

当启动另一个应用的activity来允许用户说话，发邮件或选择一个照片附件，如果用户是从启动器(设备的home屏幕)重启你的应用，你一般不会希望用户返回到别的activity。如果点击你的应用图标又回到“发邮件”的屏幕，这会使用户感到很迷惑。

为防止这种情况的发生，只需要添加[FLAG\\_ACTIVITY\\_CLEAR\\_WHEN\\_TASK\\_RESET](#)标记到用来启动外部activity的intent中，就像：

```
Intent externalActivityIntent = new Intent(Intent.ACTION_PICK);
externalActivityIntent.setType("image/*");
externalActivityIntent.addFlags(
    Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
startActivity(externalActivityIntent);
```

编写:[fastcome1985](#), 校对:

原文:<http://developer.android.com/training/notify-user/index.html>

# 通知提示用户

- Notification是一种在你APP常规UI外展示、用来指示某个事件发生的用户交互元素。用户可以在使用其它apps时查看notification，并在方便的时候做出回应。
- [Notification设计指导](#)向你展示如何设计实用的notifications以及何时使用它们。这节课将会教你实现大多数常用的notification设计。
- 完整的Demo示例：[NotifyUser.zip](#)

# Lessons

- [建立一个Notification](#)

学习如何创建一个notification [Builder](#), 设置需要的特征，以及发布notification。

- [当Activity启动时保留导航](#)

学习如何为一个从notification启动的[Activity](#)执行适当的导航。

- [更新notifications](#)

学习如何更新与移除notifications

- [使用BigView风格](#)

学习用扩展的notification来创建一个BigView，并且维持老版本的兼容性。

- [在notification中展示进度](#)

学习在notification中显示某个操作的进度，既可以用于那些你可以估算已经完成多少（确定进度， determinate）的操作，也可以用于那些你无法知道完成了多少（不确定进度， indefinite ）的操作

编写:[fastcome1985](#)

校对:

# 建立一个Notification

- 这节课向你说明如何创建与发布一个Notification。
- 这节课的例子是基于[NotificationCompat.Builder](#)类的，[NotificationCompat.Builder](#)在[Support Library](#)中。为了给许多各种不同的平台提供最好的notification支持，你应该使用[NotificationCompat](#)以及它的子类，特别是[NotificationCompat.Builder](#)。

## 创建Notification Buider

- 创建Notification时，可以用[NotificationCompat.Builder](#)对象指定Notification的UI内容与行为。一个[Builder](#)至少包含以下内容：

- 一个小的icon，用[setSmallIcon\(\)](#)方法设置
- 一个标题，用[setContentTitle\(\)](#)方法设置。
- 详细的文本，用[setContentText\(\)](#)方法设置

比如：

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```

## 定义Notification的Action（行为）

- 尽管在Notification中Actions是可选的，但是你应该至少添加一种Action。一种Action可以让用户从Notification直接进入你应用内的[Activity](#)，在这个activity中他们可以查看引起Notification的事件或者做下一步的处理。在Notification中，action本身是由[PendingIntent](#)定义的，PendingIntent包含了一个启动你应用内[Activity](#)的[Intent](#)。
- 如何构建一个[PendingIntent](#)取决于你要启动的[activity](#)的类型。当从Notification中启动一个[activity](#)时，你必须保存用户的导航体验。在下面的代码片段中，点击Notification启动一个新的activity，这个activity有效地扩展了Notification的行为。在这种情形下，就没必要人为地去创建一个返回栈（更多关于这方面的信息，请查看[Preserving Navigation when Starting an Activity](#)）

```
Intent resultIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity
// no need to create an artificial back stack.
PendingIntent resultPendingIntent =
    PendingIntent.getActivity(
        this,
        0,
        resultIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
```

## 设置Notification的点击行为

- 可以通过调用[NotificationCompat.Builder](#)中合适的方法，将上一步创建的[PendingIntent](#)与一个手势产生关联。比方说，当点击Notification抽屉里的Notification文本时，启动一个activity，可以通过调用[setContentIntent\(\)](#)方法把[PendingIntent](#)添加进去。

比如：

```
PendingIntent resultPendingIntent;  
...  
mBuilder.setContentIntent(resultPendingIntent);
```

# 发布Notification

- 为了发布notification:
  - 获取一个[NotificationManager](#)实例
  - 使用[notify\(\)](#)方法发布Notification。当你调用[notify\(\)](#)方法时，指定一个notification ID。你可以在以后使用这个ID来更新你的notification。这在[Managing Notifications](#)中有更详细的描述。
  - 调用[build\(\)](#)方法，会返回一个包含你的特征的[Notification](#)对象。

举个例子：

```
NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

编写:[fastcome1985](#)

校对:

# 启动Activity时保留导航

- 部分设计一个notification的目的是为了保持用户的导航体验。为了详细讨论这个课题, 请看 [Notifications API](#) 引导, 分为下列两种主要情况:

- 常规的activity

你启动的是你application工作流中的一部分[Activity](#)。

- 特定的activity

用户只能从notification中启动, 才能看到这个[Activity](#), 在某种意义上, 这个[Activity](#)是notification的扩展, 额外展示了一些notification本身难以展示的信息。

# 设置一个常规的Activity PendingIntent

- 设置一个直接启动的入口Activity的PendingIntent，遵循以下步骤：

1 在manifest中定义你application的[Activity](#)层次，最终的manifest文件应该像这个：

```
<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>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

2 在基于启动[Activity](#)的[Intent](#)中创建一个返回栈，比如：

```
int id = 1;
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

# 设置一个特定的Activity PendingIntent

- 一个特定的Activity不需要一个返回栈，所以你不需要在manifest中定义Activity的层次，以及你不需要调用[addParentStack\(\)](#)方法去构建一个返回栈。作为代替，你需要用manifest设置Activity任务选项，以及调用[getActivity\(\)](#)创建PendingIntent

1 manifest中，在Activity的标签中增加下列属性：

`android:name="activityclass"`

activity的完整的类名。

`android:taskAffinity=""`

结合你在代码里设置的FLAG\_ACTIVITY\_NEW\_TASK标识，确保这个Activity不会进入application的默认任务。任何与application的默认任务有密切关系的任务都不会受到影响。

`android:excludeFromRecents="true"`

将新任务从最近列表中排除，目的是为了防止用户不小心返回到它。

2 建立以及发布notification：

a. 创建一个启动Activity的Intent。

b. 通过调用[setFlags\(\)](#)方法并设置标识FLAG\_ACTIVITY\_NEW\_TASK与

FLAG\_ACTIVITY\_CLEAR\_TASK，来设置Activity在一个新的，空的任务中启动。

c. 在Intent中设置其他你需要的选项。d. 通过调用[getActivity\(\)](#)方法从Intent中创建一个PendingIntent，你可以把这个PendingIntent当做[setContentIntent\(\)](#)的参数来使用。

- 下面的代码片段演示了这个过程：

```
// Instantiate a Builder object.
NotificationCompat.Builder builder = new NotificationCompat.Bui
// Creates an Intent for the Activity
Intent notifyIntent =
    new Intent(new ComponentName(this, ResultActivity.class));
// Sets the Activity to start in a new, empty task
notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
    Intent.FLAG_ACTIVITY_CLEAR_TASK);
// Creates the PendingIntent
PendingIntent notifyIntent =
    PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SE
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```

编写:[fastcome1985](#)

校对:

# 更新Notification

- 当你需要对同一事件发布多次Notification时，你应该避免每次都生成一个全新的Notification。相反，你应该考虑去更新先前的Notification，或者改变它的值，或者增加一些值，或者两者同时进行。
- 下面的章节描述了如何更新Notifications，以及如何移除它们。

## 改变一个Notification

- 想要设置一个可以被更新的Notification，需要在发布它的时候调用[NotificationManager.notify\(ID, notification\)](#)方法为它指定一个notification ID。更新一个已经发布的Notification，需要更新或者创建一个[NotificationCompat.Builder](#)对象，并从这个对象创建一个[Notification](#)对象，然后用与先前一样的ID去发布这个[Notification](#)。
- 下面的代码片段演示了更新一个notification来反映事件发生的次数，它把notification堆积起来，显示一个总数。

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status);
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
// Because the ID remains unchanged, the existing notification
// is updated.
mNotificationManager.notify(
    notifyID,
    mNotifyBuilder.build());
...
```

## 移除Notification

- Notifications 将持续可见，除非下面任何一种情况发生。

|   |
|---|
| * 用户清除Notification单独地或者使用“清除所有”（如果Notification能被清除）   |
| * 你在创建notification时调用了 <code>setAutoCancel(developer.android.com/reference/android/app/Notification.html#setAutoCancel(boolean))</code>         |
| * 你为一个指定的 notification ID调用了 <code>[cancel()]</code> ( <code>developer.android.com/reference/android/app/Notification.html#cancel(int)</code> ) |
| * 你调用了 <code>[cancelAll()]</code> ( <code>developer.android.com/reference/android/app/Notification.html#cancelAll()</code> )                    |

编写:[fastcome1985](#)

校对:

# 使用BigView样式

- Notification抽屉中的Notification主要有两种视觉展示形式，normal view（平常的视图，下同）与 big view（大视图，下同）。Notification的 big view样式只有当Notification被扩展时才能出现。当Notification在Notification抽屉的最上方或者用户点击Notification时才会展现大视图。
- Big views在Android4.1被引进的，它不支持老版本设备。这节课叫你如何让把big view notifications合并进你的APP，同时提供normal view的全部功能。更多信息请见[Notifications API guide](#)。
- 这是一个 normal view的例子

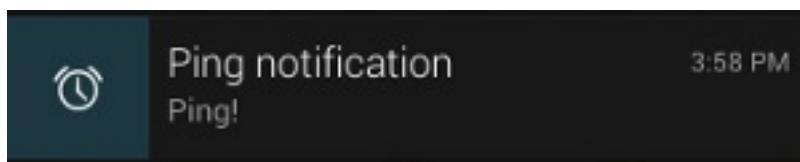


图1 Normal view notification.

- 这是一个 big view的例子

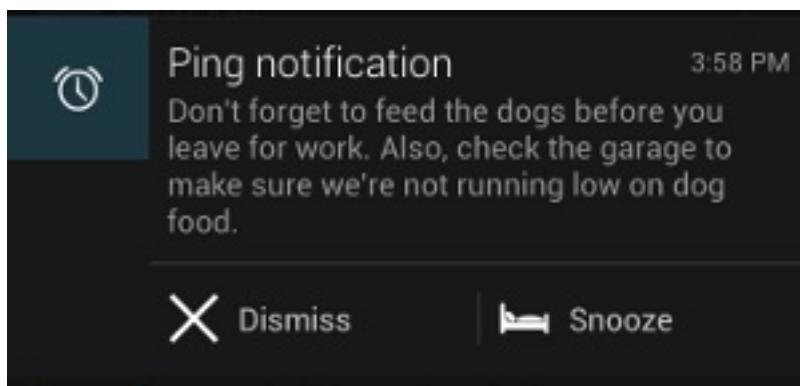


图2 Big view notification.

- 在这节课的例子应用中，normal view 与 big view给用户相同的功能：
  - 继续小睡或者消除Notification
  - 一个查看用户设置的类似计时器的提醒文字的方法，
- normal view 通过当用户点击Notification来启动一个新的activity的方式提供这些特性，记住当你设计你的notifications时，首先在normal view 中提供这些功能，因为很多用户会与notification交互。

## 设置Notification用来登陆一个新的Activity

- 这个例子应用用[IntentService](#)的子类（PingService）来构造以及发布notification。
- 在这个代码片段中，[IntentService](#)中的方法[onHandleIntent\(\)](#)指定了当用户点击notification时启动一个新的activity。方法[setContentIntent\(\)](#)定义了pending intent在用户点击notification时被激发，因此登陆这个activity。

```
Intent resultIntent = new Intent(this, ResultActivity.class);
resultIntent.putExtra(CommonConstants.EXTRA_MESSAGE, msg);
resultIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
    Intent.FLAG_ACTIVITY_CLEAR_TASK);

// Because clicking the notification launches a new ("special") ac
// there's no need to create an artificial back stack.
PendingIntent resultPendingIntent =
    PendingIntent.getActivity(
        this,
        0,
        resultIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// This sets the pending intent that should be fired when the user
// notification. Clicking the notification launches a new activity
builder.setContentIntent(resultPendingIntent);
```

## 构造big view

- 这个代码片段展示了如何在big view中设置buttons

```
// Sets up the Snooze and Dismiss action buttons that will appear
// big view of the notification.
Intent dismissIntent = new Intent(this, PingService.class);
dismissIntent.setAction(CommonConstants.ACTION_DISMISS);
PendingIntent piDismiss = PendingIntent.getService(this, 0, dismissI

Intent snoozeIntent = new Intent(this, PingService.class);
snoozeIntent.setAction(CommonConstants.ACTION_SNOOZE);
PendingIntent piSnooze = PendingIntent.getService(this, 0, snoozeI
```

- 这个代码片段展示了如何构造一个[Builder](#)对象，它设置了big view 的样式为"big text",同时设置了它的内容为提醒文字。它使用[addAction\(\)](#)方法来添加将要在big view中出现的Snooze与Dismiss按钮（以及它们相关联的pending intents）。

```
// Constructs the Builder object.
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_stat_notification)
        .setContentTitle(getString(R.string.notification))
        .setContentText(getString(R.string.ping))
        .setDefaults(Notification.DEFAULT_ALL) // requires VIBRATE
    /*
     * Sets the big view "big text" style and supplies the
     * text (the user's reminder message) that will be displayed
     * in the detail area of the expanded notification.
     * These calls are ignored by the support library for
     * pre-4.1 devices.
     */
    .setStyle(new NotificationCompat.BigTextStyle()
        .bigText(msg))
    .addAction (R.drawable.ic_stat_dismiss,
        getString(R.string.dismiss), piDismiss)
    .addAction (R.drawable.ic_stat_snooze,
        getString(R.string.snooze), piSnooze);
```

编写:[fastcome1985](#)

校对:

# 显示Notification进度

- Notifications可以包含一个展示用户正在进行的操作状态的动画进度指示器。如果你可以在任何时候估算这个操作得花多少时间以及当前已经完成多少，你可以用“determinate（确定的，下同）”形式的指示器（一个进度条）。如果你不能估算这个操作的长度，使用“indeterminate（不确定，下同）”形式的指示器（一个活动的指示器）。
- 进度指示器用[ProgressBar](#)平台实现类来显示。
- 使用进度指示器，可以调用[setProgress\(\)](#)方法。determinate 与 indeterminate形式将在下面的章节中介绍。

## 展示固定长度的进度指示器

- 为了展示一个确定长度的进度条，调用 [setProgress\(max, progress, false\)](#)方法将进度条添加进notification，然后发布这个notification，第三个参数是个boolean类型，决定进度条是 indeterminate (true) 还是 determinate (false)。在你操作进行时，增加progress，更新notification。在操作结束时，progress应该等于max。一个常用的调用[setProgress\(\)](#)的方法是设置max为100，然后增加progress就像操作的“完成百分比”。
- 当操作完成的时候，你可以选择或者让进度条继续展示，或者移除它。无论哪种情况下，记得更新notification的文字来显示操作完成。移除进度条，调用[setProgress\(0, 0, false\)](#)方法.比如：

```
int id = 1;
...
mNotifyManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value,
                // current completion percentage, and "determinate"
                // state
                mBuilder.setProgress(100, incr, false);
                // Displays the progress bar for the first time
                mNotifyManager.notify(id, mBuilder.build());
                // Sleeps the thread, simulating an operation
                // that takes time
                try {
                    // Sleep for 5 seconds
                    Thread.sleep(5*1000);
                } catch (InterruptedException e) {
                    Log.d(TAG, "sleep failure");
                }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
            .setProgress(0,0,false);
            mNotifyManager.notify(id, mBuilder.build());
        }
    }
)
// Starts the thread by calling the run() method in its Runnable
```

```
) .start();
```

- 结果notifications显示在图1中，左边是操作正在进行中的notification的快照，右边是操作已经完成的notification的快照。

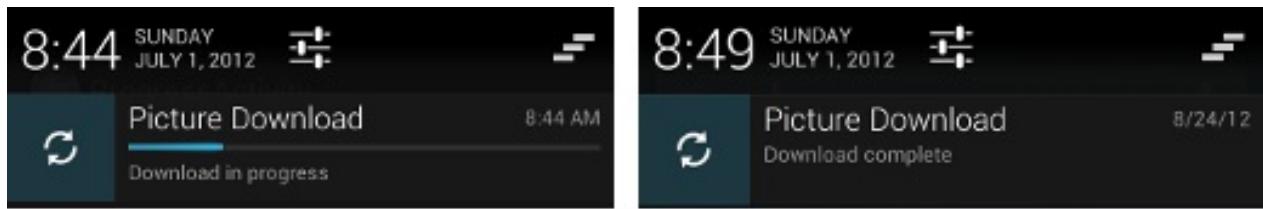


图1 操作正在进行中与完成时的进度条

## 展示持续的活动的指示器

- 为了展示一个持续的(indeterminate)活动的指示器,用`setProgress(0, 0, true)`方法把指示器添加进notification, 然后发布这个notification。前两个参数忽略, 第三个参数决定indicator 还是 indeterminate。结果是指示器与进度条有同样的样式, 除了它的动画正在进行。
- 在操作开始的时候发布notification, 动画将会一直进行直到你更新notification。当操作完成时, 调用`setProgress(0, 0, false)`方法, 然后更新notification来移除这个动画指示器。一定要这么做, 否则即使你操作完成了, 动画还是会在那运行。同时也要记得更新notification的文字来显示操作完成。
- 为了观察持续的活动的指示器是如何工作的, 看前面的代码。定位到下面的几行:

```
// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(id, mBuilder.build());
```

- 将你找到的代码用下面的几行代码代替, 注意`setProgress()`方法的第三个参数设置成了true, 表示进度条是 indeterminate类型的。

```
// Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(id, mBuilder.build());
```

- 结果显示在图2中:

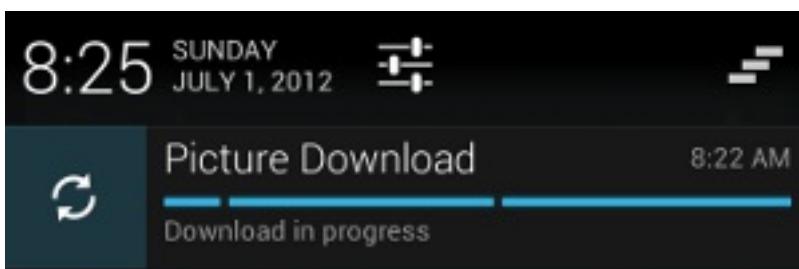


图2 正在进行的活动的指示器

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/search/index.html>

# 增加搜索功能

Android的内置搜索功能，能够在app中方便地为所有用户提供一个统一的搜索体验。根据设备所运行的Android版本，有两种方式可以在你的app中实现搜索。本节课程涵盖如何像Android 3.0中介绍的那样用[SearchView](#)添加搜索，使用系统提供的默认搜索框来向下兼容旧版本Android。

## Lessons

- [建立搜索界面](#)

学习如何向你的app中添加搜索界面，如何设置activity去处理搜索请求

- [保存并搜索数据](#)

学习在SQLite虚拟数据库表中用简单的方法储存和搜索数据

- [保持向后兼容](#)

通过使用搜索功能来学习如何向下兼容旧版本设备

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/search/setup.html>

# 建立搜索界面

从Android 3.0开始，在action bar中使用[SearchView](#)作为item，是在你的app中提供搜索的一种更好方法。像其他所有在action bar中的item一样，你可以定义[SearchView](#)在有足够空间的时候总是显示，或设置为一个折叠操作(collapsible action)，一开始[SearchView](#)作为一个图标显示，当用户点击图标时再显示搜索框占据整个action bar。

**Note:**在本课程的后面，你会学习对那些不支持[SearchView](#)的设备，如何使你的app向下兼容至Android 2.1版本。

## 添加SearchView到中action bar中添加

为了在action bar中添加[SearchView](#)，在你的工程目录res/menu/中创建一个名为options\_menu.xml的文件，再把下列代码添加到文件中。这段代码定义了如何创建search item，比如使用的图标和item的标题。collapseActionView属性允许你的[SearchView](#)占据整个action bar，在不使用的时候折叠成普通的action bar item。由于在手持设备中action bar的空间有限，建议使用collapsibleActionView属性来提供更好的用户体验。

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/search"
          android:title="@string/search_title"
          android:icon="@drawable/ic_search"
          android:showAsAction="collapseActionView|ifRoom"
          android:actionViewClass="android.widget.SearchView" />
</menu>
```

**Note:**如果你的menu item已经有一个XML文件，你可以只把<item>元素添加入文件。

要在action bar中显示[SearchView](#)，把XML菜单资源(res/menu/options\_menu.xml)填充到你的activity中的[onCreateOptionsMenu\(\)](#)方法：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);

    return true;
}
```

如果你立即运行你的app，[SearchView](#)就会显示在你app的action bar中，但还无法使用。你现在需要定义[SearchView](#)如何运行。

## 创建一个检索配置

检索配置(searchable configuration)在 res/xml/searchable.xml文件中定义了[SearchView](#)如何运行。检索配置中至少要包含一个 android:label属性，与Android manifest中的<application>或<activity> android:label属性值相同。但我们还是建议添加 android:hint属性来告诉用户应该在搜索框中输入什么内容：

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

<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
    android:hint="@string/search_hint" />
```

在你的应用的manifest文件中，声明一个指向res/xml/searchable.xml文件的元素，来告诉你的应用在哪里能找到检索配置。在你想要显示[SearchView](#)的<activity>中声明<meta-data>元素：

```
<activity ... >
    ...
    <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable" />
</activity>
```

在你之前创建的[onCreateOptionsMenu\(\)](#)方法中，调用[setSearchableInfo\(SearchableInfo\)](#)把[SearchView](#)和检索配置关联在一起：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);

    // 关联检索配置和SearchView
    SearchManager searchManager =
        (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView =
        (SearchView) menu.findItem(R.id.search).getActionView();
    searchView.setSearchableInfo(
        searchManager.getSearchableInfo(getApplicationContext()));

    return true;
}
```

调用[getSearchableInfo\(\)](#)返回一个[SearchableInfo](#)由检索配置XML文件创建的对象。检索配置与[SearchView](#)正确关联后，当用户提交一个搜索请求时，[SearchView](#)会以[ACTION\\_SEARCH](#) intent启动一个activity。所以你现在需要一个能过滤这个intent和处理搜索请求的activity。

## 创建一个检索activity

当用户提交一个搜索请求时，[SearchView](#)会尝试以[ACTION\\_SEARCH](#)启动一个activity。检索activity会过滤[ACTION\\_SEARCH](#) intent并在某种数据集中根据请求进行搜索。要创建一个检索activity，在你选择的activity中声明对[ACTION\\_SEARCH](#) intent过滤：

```
<activity android:name=".SearchResultsActivity" ... >
    ...
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    ...
</activity>
```

在你的检索activity中，通过在[onCreate\(\)](#)方法中检查[ACTION\\_SEARCH](#) intent来处理它。

**Note:**如果你的检索activity在single top mode下启动(`android:launchMode="singleTop"`)，也要在[onNewIntent\(\)](#)方法中处理[ACTION\\_SEARCH](#) intent。在single top mode下你的activity只有一个会被创建，而随后启动的activity将不会在栈中创建新的activity。这种启动模式很有用，因为用户可以在当前activity中进行搜索，而不用在每次搜索时都创建一个activity实例。

```
public class SearchResultsActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        handleIntent(getIntent());
    }

    @Override
    protected void onNewIntent(Intent intent) {
        ...
        handleIntent(intent);
    }

    private void handleIntent(Intent intent) {

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUE
                //通过某种方法，根据请求检索你的数据
        }
    }
    ...
}
```

如果你现在运行你的app，[SearchView](#)就能接收用户的搜索请求，以[ACTION\\_SEARCH](#) intent启动你的检索activity。现在就由你来解决如何依据请求来储存和搜索数据。

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/search/search.html>

# 保存并搜索数据

有很多方法可以储存你的数据，比如储存在线上的数据库，本地的SQLite数据库，甚至是文本文件。你自己来选择最适合你应用的存储方式。本节课程会向你展示如何创建一个健壮的可以提供全文搜索的SQLite虚拟表。并从一个每行有一组单词-解释对的文件中将数据填入。

## 创建虚拟表

虚拟表与SQLite表的运行方式类似，但虚拟表是通过回调来向内存中的对象进行读取和写入，而不是通过数据库文件。要创建一个虚拟表，首先为该表创建一个类：

```
public class DatabaseTable {  
    private final DatabaseOpenHelper mDatabaseOpenHelper;  
  
    public DatabaseTable(Context context) {  
        mDatabaseOpenHelper = new DatabaseOpenHelper(context);  
    }  
}
```

在DatabaseTable类中创建一个继承[SQLiteOpenHelper](#)的内部类。你必须重写类[SQLiteOpenHelper](#)中定义的abstract方法，才能在必要的时候创建和更新你的数据库表。例如，下面一段代码声明了一个数据库表，用来储存字典app所需的单词。

```
public class DatabaseTable {  
  
    private static final String TAG = "DictionaryDatabase";  
  
    //字典的表中将要包含的列项  
    public static final String COL_WORD = "WORD";  
    public static final String COL_DEFINITION = "DEFINITION";  
  
    private static final String DATABASE_NAME = "DICTIONARY";  
    private static final String FTS_VIRTUAL_TABLE = "FTS";  
    private static final int DATABASE_VERSION = 1;  
  
    private final DatabaseOpenHelper mDatabaseOpenHelper;  
  
    public DatabaseTable(Context context) {  
        mDatabaseOpenHelper = new DatabaseOpenHelper(context);  
    }  
  
    private static class DatabaseOpenHelper extends SQLiteOpenHelper {  
  
        private final Context mHelperContext;  
        private SQLiteDatabase mDatabase;  
  
        private static final String FTS_TABLE_CREATE =  
            "CREATE VIRTUAL TABLE " + FTS_VIRTUAL_TABLE +  
            " USING fts3 (" +  
            COL_WORD + ", " +  
            COL_DEFINITION + ")";  
  
        DatabaseOpenHelper(Context context) {  
            super(context, DATABASE_NAME, null, DATABASE_VERSION);  
            mHelperContext = context;  
        }  
  
        @Override
```

```
public void onCreate(SQLiteDatabase db) {
    mDatabase = db;
    mDatabase.execSQL(FTS_TABLE_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.w(TAG, "Upgrading database from version " + oldVersion
          + " to " + newVersion + ", which will destroy all old data.");
    db.execSQL("DROP TABLE IF EXISTS " + FTS_VIRTUAL_TABLE);
    onCreate(db);
}
}
```

## 填入虚拟表

现在，表需要数据来储存。下面的代码会向你展示如何读取一个内容为单词和解释的文本文件(位于res/raw/definitions.txt)，如何解析文件与如何将文件中的数据按行插入虚拟表中。为防止UI锁死这些操作会在另一条线程中执行。将下面的一段代码添加到你的DatabaseOpenHelper内部类中。

**Tip:**你也可以设置一个回调来通知你的UI activity线程的完成结果。

```
private void loadDictionary() {
    new Thread(new Runnable() {
        public void run() {
            try {
                loadWords();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }).start();
}

private void loadWords() throws IOException {
    final Resources resources = mHelperContext.getResources();
    InputStream inputStream = resources.openRawResource(R.raw.defi
    BufferedReader reader = new BufferedReader(new InputStreamReader(
        try {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] strings = TextUtils.split(line, "-");
                if (strings.length < 2) continue;
                long id = addWord(strings[0].trim(), strings[1].trim());
                if (id < 0) {
                    Log.e(TAG, "unable to add word: " + strings[0].tri
                }
            }
        } finally {
            reader.close();
        }
    }

    public long addWord(String word, String definition) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(COL_WORD, word);
        initialValues.put(COL_DEFINITION, definition);

        return mDatabase.insert(FTS_VIRTUAL_TABLE, null, initialValues
    }
}
```

任何恰当的地方，都可以调用loadDictionary()方法向表中填入数据。一个比较好的

地方是DatabaseOpenHelper类的[onCreate\(\)](#)方法中，紧随创建表之后：

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    mDatabase = db;  
    mDatabase.execSQL(FTS_TABLE_CREATE);  
    loadDictionary();  
}
```

## 搜索请求

当你的虚拟表创建好并填入数据后，根据[SearchView](#)提供的请求搜索数据。将下面的方法添加到DatabaseTable类中，用来创建搜索请求的SQL语句：

```
public Cursor getWordMatches(String query, String[] columns) {
    String selection = COL_WORD + " MATCH ?";
    String[] selectionArgs = new String[] {query+"*"};
    return query(selection, selectionArgs, columns);
}

private Cursor query(String selection, String[] selectionArgs, String
SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
builder.setTables(FTS_VIRTUAL_TABLE);

Cursor cursor = builder.query(mDatabaseOpenHelper.getReadableD
    columns, selection, selectionArgs, null, null, null);

if (cursor == null) {
    return null;
} else if (!cursor.moveToFirst()) {
    cursor.close();
    return null;
}
return cursor;
}
```

调用getWordMatches()来搜索请求。任何符合的结果返回到[Cursor](#)中，可以直接遍历或是建立一个[ListView](#)。这个例子是在检索activity的handleIntent()方法中调用getWordMatches()。请记住，因为之前创建的intent filter，检索activity会在[ACTION\\_SEARCH](#) intent中额外接收请求作为变量存储：

```
DatabaseTable db = new DatabaseTable(this);

...
private void handleIntent(Intent intent) {

    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        Cursor c = db.getWordMatches(query, null);
        //执行Cursor并显示结果
    }
}
```

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/search/backward-compat.html>

# 保持向下兼容

[SearchView](#)和action bar只在Android 3.0以及以上版本可用。为了支持旧版本平台，你可以回到搜索对话框。搜索框是系统提供的UI，在调用时会覆盖在你的应用的最顶端。

## 设置最小和目标API级别

要设置搜索对话框，首先在你的manifest中声明你要支持旧版本设备，并且目标平台为Android 3.0或更新版本。当你这么做之后，你的应用会自动地在Android 3.0或以上使用action bar，在旧版本的设备使用传统的目录系统：

```
<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="15">  
<application>  
    ...
```

## 为旧版本设备提供搜索对话框

要在旧版本设备中调用搜索对话框，可以在任何时候，当用户从选项目录中选择搜索项时，就会调用onSearchRequested()。因为Android 3.0或以上会在action bar中显示SearchView(就像在第一节课中演示的那样)，所以当用户选择目录的搜索项时，只有Android 3.0以下版本的会调用onOptionsItemSelected()。

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.search:  
            onSearchRequested();  
            return true;  
        default:  
            return false;  
    }  
}
```

## 在运行时检查Android的构建版本

在运行时，检查设备的版本可以保证在旧版本设备中，不使用不支持的[SearchView](#)。在我们这个例子中，这一操作在[onCreateOptionsMenu\(\)](#)方法中：

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.options_menu, menu);  
  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {  
        SearchManager searchManager =  
            (SearchManager) getSystemService(Context.SEARCH_SERVICE);  
        SearchView searchView =  
            (SearchView) menu.findItem(R.id.search).getActionView();  
        searchView.setSearchableInfo(  
            searchManager.getSearchableInfo(getApplicationContext()));  
        searchView.setIconifiedByDefault(false);  
    }  
    return true;  
}
```

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/app-indexing/index.html>

# 使得你的App内容可被Google搜索

随着移动app变得越来越普遍，用户不仅仅从网站上查找相关信息，也在他们安装的app上查找。你可以使Google能够抓取你的app内容，当内容与你自己的网页一致时，Google搜索的结果会将你的app作为目标展示给用户。

通过为你的activity提供intent filter，可以使Google搜索展示你的app中特定的内容。Google搜索应用索引(Google Search app indexing)通过在用户搜索结果的网页链接旁附上相关的app内容链接，补充了这一功能。使用移动设备的用户可以在他们的搜索结果中点击链接来打开你的app，使他们能够直接浏览你的app中的内容，而不需要打开网页。

要启用Google搜索应用索引，你需要把有关app与网页之间联系的信息提供给Google。这个过程包括下面几个步骤：

1. 通过在你的app manifest中添加intent filter来开启链接到你的app中指定内容的深度链接。
2. 在你的网站中的相关页面添加注解或Sitemap文件
3. 选择允许谷歌爬虫(Googlebot)在Google Play store中通过APK抓取，建立app内容索引。在早起采用者计划(early adopter program)中作为参与者加入时，会自动选择允许。

这节课，会向你展示如何启用深度链接和建立应用内容索引，使用户可以从移动设备搜索结果直接打开此内容。

## Lessons

- [为App内容开启深度链接](#)

演示如何添加intent filter来启用链接app内容的深度链接

- [为索引指定App内容](#)

演示如何给网站的metadata添加注解，使Google的算法能为app内容建立索引

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/app-indexing/deep-linking.html>

# 为App内容开启深度链接

为使Google能够抓取你的app内容，并允许用户从搜索结果进入你的app，你必须给你的app manifest中相关的activity添加intent filter。这些intent filter能使深度链接与你的任何activity相连。例如，用户可以在购物app中，点击一条深度链接来浏览一个介绍了自己所搜索的产品的页面。

# 为你的深度链接添加Intent filter

要创建一条与你的app内容相连的深度链接，添加一个包含了以下这些元素和属性值的intent filter到你的manifest中：

## <action>

指定ACTION\_VIEW的操作，使得Google搜索可以触及intent filter。

## <data>

添加一个或多个<data>标签，每一个标签代表一种activity对URI格式的解析，<data>必须至少包含android:scheme属性。

你可以添加额外的属性来改善activity所接受的URI类型。例如，你或许有几个activity可以接受相似的URI，它们仅仅是路径名不同。在这种情况下，使用android:path属性或它的变形(pathPattern或pathPrefix)，使系统能辨别对不同的URI路径应该启动哪个activity。

## <category>

包括BROWSABLE category。BROWSABLE category对于使intent filter能被浏览器访问是必要的。没有这个category，在浏览器中点击链接无法解析到你的app。DEFAULT category是可选的，但建议添加。没有这个category，activity只能使用app组件名称以显示(explicit)intent启动。

下面的一段XML代码向你展示，你应该如何在manifest中为深度链接指定一个intent filter。URI “example://gizmos” 和 “<http://www.example.com/gizmos>” 都能够解析到这个activity。

```
<activity
    android:name="com.example.android.GizmosActivity"
    android:label="@string/title_gizmos" >
    <intent-filter android:label="@string/filter_title_viewgizmos" >
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- 接受以"example://gizmos"开头的 URIs -->
        <data android:scheme="example"
              android:host="gizmos" />
        <!-- 接受以"http://www.example.com/gizmos"开头的 URIs -->
        <data android:scheme="http"
              android:host="www.example.com"
              android:pathPrefix="gizmos" />
    </intent-filter>
</activity>
```

当你把包含有指定activity内容的URI的intent filter添加到你的app manifest后，Android就可以在你的app运行时，为app与匹配URI的Intent建立路径。

学习更多关于定义intent filter，见[Allow Other Apps to Start Your Activity](#)

## 从传入的意图读取数据

一旦系统通过一个intent filter启动你的activity，你可以使用由[Intent](#)提供的数据来决定需要处理什么。调用[getData\(\)](#)和[getAction\(\)](#)方法来取出传入[Intent](#)中的数据与操作。你可以在activity生命周期的任何时候调用这些方法，但一般情况下你应该在前期回调中调用如[onCreate\(\)](#)或[onStart\(\)](#)。

这个是一段代码，展示如何从[Intent](#)中取出数据：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    Intent intent = getIntent();  
    String action = intent.getAction();  
    Uri data = intent.getData();  
}
```

遵守下面这些惯例来提高用户体验：

- 深度链接应直接为用户打开内容，不需要任何提示，插播式广告页和登录页面。要确保用户能看到app的内容，即使之前从没打开过这个应用。当用户从启动器打开app时，可以在操作结束后给出提示。这个准则也同样适用于网站的[first click free](#)体验。
- 遵循[Navigation with Back and Up](#)中的设计指导，来使你的app能够满足用户通过深度链接进入app后，向后导航的需求。

## 测试你的深度链接

你可以使用[Android Debug Bridge](#)和activity管理(am)工具来测试你指定的intent filter URI，能否正确解析到正确的app activity。你可以在设备或者模拟器上运行adb命令。

测试intent filter URI的一般adb语法是：

```
$ adb shell am start  
    -W -a android.intent.action.VIEW  
    -d <URI> <PACKAGE>
```

例如，下面的命令视图浏览与指定URI相关的目标app activity。

```
$ adb shell am start  
    -W -a android.intent.action.VIEW  
    -d "example://gizmos" com.example.android
```

编写: [Lin-H](#) - 校对:

原文: <http://developer.android.com/training/app-indexing/enabling-app-indexing.html>

# 为索引指定App内容

Google的网页爬虫机器([Googlebot](#))会抓取页面，并为Google搜索引擎建立索引，也能为你的Android app内容建立索引。通过选择加入这一功能，你可以允许Googlebot通过抓取在Google Play Store中的APK内容，为你的app内容建立索引。要指出哪些app内容你想被Google索引，只需要添加链接元素到现有的[Sitemap](#)文件，或添加到你的网站中每个页面的<head>元素中，以相同的方式为你的页面添加。

**Note:**现在，Google搜索app索引功能从以前只限于英文Android app，转变为开放给参与了早期开放计划(early adopter program)的所有开发者。你可以通过提交[App Indexing Expression of Interest](#)表格来注册成为参与者。

你所共享给Google搜索的深度链接必须按照下面的URI格式：

```
android-app://<package_name>/<scheme>/<host_path>
```

构成URI的各部分是：

- **package\_name** 代表在[Google Play Developer Console](#)中所列出来的你的APK的包名。
- **scheme** 匹配你的intent filter的URI方案。
- **host\_path** 找出你的应用中所指定的内容。

下面的几节叙述如何添加一个深度链接URI到你的Sitemap或网页中。

## 添加深度链接(Deep link)到你的Sitemap

要在你的[Sitemap](#)中为Google搜索app索引(Google Search app indexing)添加深度链接的注解，使用`<xhtml:link>`标签，并指定用作替代URI的深度链接。

例如，下面一段XML代码向你展示如何使用`<loc>`标签指定一个链接到你的页面的链接，以及如何使用`<xhtml:link>`标签指定链接到你的Android app的深度链接。

```
<?xml version="1.0" encoding="UTF-8" ?>
<urlset
  xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <url>
    <loc>example://gizmos</loc>
    <xhtml:link
      rel="alternate"
      href="android-app://com.example.android/example/gi
    </url>
    ...
</urlset>
```

## 添加深度链接到你的网页中

除了在你的Sitemap文件中，为Google搜索app索引指定深度链接外，你还可以在你的HTML标记网页中给深度链接添加注解。你可以在`<head>`标签内这么做，为每一个页面添加一个`<link>`标签，并指定用作替代URI的深度链接。

例如，下面的一段HTML代码向你展示如何在页面中指定一个URL为`example://gizmos`的相应的深度链接。

```
<html>
<head>
    <link rel="alternate"
          href="android-app://com.example.android/example/gizmos"
          ...
</head>
<body> ... </body>
```

## 允许Google通过你的app抓取URL请求

一般来说，你可以通过使用[robots.txt](#)文件，来控制Googlebot如何抓取你网站上的公开访问的URL。当Googlebot为你的app内容建立索引后，你的app可以把HTTP请求当做一般操作。但是，这些请求会被视为从Googlebot发出，发送到你的服务器上。因此，你必须正确配置你的服务器上的`robots.txt`文件来允许这些请求。

例如，下面的`robots.txt`指示向你展示，如何允许你网站上的特定目录(如 `/api/`)能被你的app访问，并限制Googlebot访问你的网站上的其他目录。

```
User-Agent: Googlebot
Allow: /api/
Disallow: /
```

学习更多关于如何修改`robots.txt`，来控制页面抓取，详见[Controlling Crawling and Indexing Getting Started](#)。

编写:[riverfeng](#)

校对:

# 为多屏幕设计

从小屏手机到大屏电视， android拥有数百种不同屏幕尺寸的设备，因此，设计兼容不同屏幕尺寸的应用程序满足不同的用户体验就变得非常重要。

但是，只是单纯的兼容不同的设备类型是远远不够的。每个不同的屏幕尺寸都给用户体验带来不同的可能性和挑战。所以，为了充分的满足用户，你的应用不仅要支持多屏幕，更要针对每个屏幕配置优化你的用户体验。

这个课程就将教你如何针对不同屏幕配置来优化你的UI。

本课程提供了一个简单的示例[NewsReader](#)。这个示例中提供的代码对多屏幕适配是非常好的一个练习，并且你也可以将这个示例中的代码运用到你自己的项目中。

注意：这节课中相关的例子为了兼容android3.0以下的版本使用了support library中的fragment，如果你需要使用该示例，请先下载support library并添加到例子中。

# Lessons

- [支持不同屏幕尺寸](#)

这节课主要告诉你如何设计能适配多种不同尺寸的屏幕（通过使用灵活的规格（dimensions），RelativeLayout，屏幕尺寸和方向限定词（qualifier），别名过滤器（alias filter）和点9图片）。

- [支持不同的屏幕密度](#)

这节课将演示如何支持不同像素密度的屏幕（使用密度独立像素（dip）以及为不同的密度提供合适的位图（bitmap））。

- [实现自适应UI流（Flows）](#)

这节课将说的是如何通过实现UI流（flow）的方式来适配不同屏幕尺寸和密度的组合（运行时检测活动布局，根据当前布局正确的响应，处理屏幕配置的变化）。

编写:[riverfeng](#)

校对:

# 兼容不同的屏幕大小

这节课教你如何通过以下几种方式支持多屏幕：

- 1: 确保你的布局能自适应屏幕
- 2: 根据你的屏幕配置提供合适的UI布局
- 3: 确保你当前的布局适合当前的屏幕。
- 4: 提供合适的位图（bitmap）

## 使用“wrap\_content”和“match\_parent”

为了确保你的布局能灵活的适应不同的屏幕尺寸，针对一些view组件，你应该使用wrap\_content和match\_parent来设置他们的宽和高。如果你使用了wrap\_content，view的宽和高会被设置为该view所包含的内容的大小值。如果是match\_parent（在API 8之前是fill\_parent）则被设置为该组件的父控件的大小。

通过使用wrap\_content和match\_parent尺寸值代替硬编码的尺寸，你的视图将分别只使用控件所需要的空间或者被拓展以填充所有有效的空间。比如：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:layout_width="match_parent"
        android:id="@+id/linearLayout1"
        android:gravity="center"
        android:layout_height="50dp">
        <ImageView android:id="@+id/imageView1"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:src="@drawable/logo"
            android:paddingRight="30dp"
            android:layout_gravity="left"
            android:layout_weight="0" />
        <View android:layout_height="wrap_content"
            android:id="@+id/view1"
            android:layout_width="wrap_content"
            android:layout_weight="1" />
        <Button android:id="@+id/categorybutton"
            android:background="@drawable/button_bg"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:layout_width="120dp"
            style="@style/CategoryButtonStyle"/>
    </LinearLayout>

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineListFragment"
        android:layout_width="match_parent" />
</LinearLayout>
```

注意上面的例子使用wrap\_content和match\_parent来指定组件尺寸而不是使用固定的尺寸。这样就能使你的布局正确的适配不同的屏幕尺寸和屏幕配置（这里的配置主要是指屏幕的横竖屏切换）。

例如，下图演示的就是该布局在竖屏和横屏模式下的效果，注意组件的尺寸是自动适应宽和高的。



图1：News Reader示例app（左边竖屏，右边横屏）。

## 使用绝对布局 (RelativeLayout)

你可以使用LinearLayout以及wrap\_content和match\_parent组合来构建复杂的布局，但是LinearLayout却不允许你精准的控制它子view的关系，子view在LinearLayout中只能简单一个接一个的排成行。如果你需要你的子view不只是简简单单的排成行的排列，更好的方法是使用RelativeLayout，它允许你指定你布局中控件与控件之间的关系，比如，你可以指定一个子view在左边，另一个则在屏幕的右边。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dp"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@+id/ok"
        android:layout_alignTop="@+id/ok"
        android:text="Cancel" />
</RelativeLayout>
```



图2: QVGA (小尺寸屏幕) 屏幕下截图



图3: WSVGA (大尺寸屏幕) 屏幕下截图

注意：尽管组件的尺寸发生了变化，但是它的子view之间的关系还是通过RelativeLayout.LayoutParams已经指定好了。

## 使用据尺寸限定词

(译者注：这里的限定词只要是指在编写布局文件时，将布局文件放在加上类似 large, sw600dp等这样限定词的文件夹中，以此来告诉系统根据屏幕选择对应的布局文件，比如下面例子的layout-large文件夹)

从上一节的学习里程中，我们知道如何编写灵活的布局或者相对布局，它们都能通过拉伸或者填充控件来适应不同的屏幕，但是它们却无法为每个不同屏幕尺寸提供最好的用户体验。因此，你的应用不应该只是实现灵活的布局，同时也应该为不同的屏幕配置提供几种不同的布局方式。你可以通过配置限定（configuration qualifiers）来做这件事情，它能在运行时根据你当前设备的配置（比如不同的屏幕尺寸设计了不同的布局）来选择合适的资源。

比如，很多应用都为大屏幕实现了“两个方框”模式（应用可能在一个方框中实现一个list，另外一个则实现list的content），平板和电视都是大到能在一个屏幕上适应两个方框，但是手机屏幕却只能单个显示。所以，如果你想实现这些布局，你就需要以下文件：

res/layout/main.xml.单个方框（默认）布局：

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineFragment"
        android:layout_width="match_parent" />

</LinearLayout>
```

res/layout-large/main.xml,两个方框布局：

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>

    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />

</LinearLayout>
```

注意第二个布局文件的目录名字“large qualifier”，在大尺寸的设备屏幕时（比如7寸平板或者其他大屏幕的设备）就会选择该布局文件，而其他比较小的设备则会选择没有限定词的另一个布局（也就是第一个布局文件）。

## 使用最小宽度限定词

在Android3.2之前，开发者还有一个困难，那就是Android设备的“large”屏幕尺寸，其中包括Dell Streak（设备名称），老版Galaxy Tab和一般的7寸平板，有很多的应用都想针对这些不同的设备（比如5和7寸的设备）定义不同的布局，但是这些设备都被定义为了large尺寸屏幕。也是因为这个，所以Android在3.2的时候开始使用最小宽度限定词。

最小宽度限定词允许你根据设备的最小宽度（dp单位）来指定不同布局。比如，传统的7寸平板最小宽度为600dp，如果你希望你的UI能够在这样的屏幕上显示两个方框（一个方框的显示在小屏幕上），你可以使用上节中提到的同样的两个布局文件，不同的是，使用sw600来指定两个方框的布局使用在最小宽度为600dp的设备上。

res/layout/main.xml,单个方框（默认）布局：

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineFragment"
        android:layout_width="match_parent" />

</LinearLayout>
```

res/layout-sw600dp/main.xml,两个方框布局：

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>

    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />

</LinearLayout>
```

这样意味着当你的设备的最小宽度等于600dp或者更大时，系统选择layout-sw600dp/main.xml（两个方框）的布局，而小一点的屏幕则会选择layout/main.xml（单个方框）的布局。然而，在3.2之前的设备上，这样做并不是很好的选择。因为3.2之前还没有将sw600dp作为一个限定词出现，所以，你还是需要使用large限定词来做。因此，你还是应该要有一个布局文件名为res/layout-large/main.xml，和res/layout-sw600dp/main.xml一样。在下一节中，你将学到如何避免像这样出现重复的布局文件。

## 使用布局别名

最小宽度限定词只能在Android3.2或者更高的版本上使用。因此，你还是需要使用抽象尺寸（small, normal, large, xlarge）来兼容以前的版本。比如，你想要将你的UI设计为在手机上只显示一个方框的布局，而在7寸平板或电视，或者其他大屏幕设备上显示多个方框的布局，你可能得提供这些文件：

res/layout/main.xml: 单个方框布局

res/layout-large: 多个方框布局

res/layout-sw600dp: 多个方框布局

最后两个文件都是一样的，因为其中一个将会适配Android3.2的设备，而另外一个则会适配其他Android低版本的平板或者电视。为了避免这些重复的文件（维护让人感觉头痛就是因为这个），你可以使用别名文件。比如，你可以定义如下布局：res/layout/main.xml，单个方框布局 res/layout/main\_twopans.xml，两个方框布局 然后添加这两个文件：  
res/values-large/layout.xml:

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

res/values-sw600dp/layout.xml:

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

最后两个文件拥有相同的内容，但它们并没有真正意义上的定义布局。它们只是将 main\_twopanes 设置成为了别名main，它们分别处在large和sw600dp选择器中，所以它们能适配Android任何版本的平板和电视（在3.2之前平板和电视可以直接匹配large，而3.2或者以上的则匹配sw600dp）。

## 使用方向限定词

有一些布局不管是在横向还是纵向的屏幕配置中都能显示的非常好，但是更多的时候，适当的调整一下会更好。在News Reader应用例子中，以下是布局在不同屏幕尺寸和方向的行为：

小屏幕，纵向：一个方框加logo 小屏幕，横向：一个方框加logo 7寸平板，纵向：一个方框加action bar 7寸平板，横向：两个宽方框加action bar 10寸平板，纵向：两个窄方框加action bar 10寸平板，横向：两个宽方框加action bar 电视，横向：两个宽方框加action bar

这些每个布局都会再res/layout目录下定义一个xml文件，如此，应用就能根据屏幕配置的变化根据别名匹配到对应的布局来适应屏幕。

res/layout/onpane.xml:

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineFragment"
        android:layout_width="match_parent" />
</LinearLayout>
```

res/layout/onepane\_with\_bar.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:layout_width="match_parent"
        android:id="@+id/linearLayout1"
        android:gravity="center"
        android:layout_height="50dp">
        <ImageView android:id="@+id/imageView1"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:src="@drawable/logo"
            android:paddingRight="30dp"
            android:layout_gravity="left"
            android:layout_weight="0" />
        <View android:layout_height="wrap_content"
            android:id="@+id/view1"
            android:layout_width="wrap_content"
            android:layout_weight="1" />
        <Button android:id="@+id/categorybutton"
            android:background="@drawable/button_bg"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:layout_width="120dp" />
    </LinearLayout>
</LinearLayout>
```

```
        style="@style/CategoryButtonStyle" />
</LinearLayout>

<fragment android:id="@+id/headlines"
    android:layout_height="fill_parent"
    android:name="com.example.android.newsreader.HeadlineListFragment"
    android:layout_width="match_parent" />
</LinearLayout>
```

res/layout/twopanes.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineListFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>
```

res/layout/twopanes\_narrow.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlineListFragment"
        android:layout_width="200dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>
```

现在所有可能的布局我们都已经定义了，唯一剩下的问题是使用方向限定词来匹配对应的布局给屏幕。这时候，你就可以使用布局别名的功能了：

res/values/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/onepane_with_breadcrumbs
    <bool name="has_two_panes">false</bool>
```

```
</resources>
```

res/values-sw600dp-land/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes</item>
    <bool name="has_two_panes">true</bool>
</resources>
```

res/values-sw600dp-port/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/onepane</item>
    <bool name="has_two_panes">false</bool>
</resources>
```

res/values-large-land/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes</item>
    <bool name="has_two_panes">true</bool>
</resources>
```

res/values-large-port/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes_narrow
    <bool name="has_two_panes">true</bool>
</resources>
```

## 使用点9图片

支持不同的屏幕尺寸同时也意味着你的图片资源也必须能兼容不同的屏幕尺寸。比如，一个button的背景图片就必须要适应该button的各种形状。

如果你在使用组件时可以改变图像的大小，你很快就会发现这是一个不明确的选择，因为运行的时候，图片会被拉伸或者压缩（这样容易造成图像失真）。避免这种情况的解决方案就是使用点9图片，这是一种能够指定哪些区域能够或者不能够拉伸的特殊png文件。

因此，在设计的图像需要与组件一起变大变小时，一定要使用点9.若要将位图转换为点9，你可以用一个普通的图像开始（下图，是在4倍变焦情况下的图像显示）。

你可以通过sdk中的draw9patch程序（位于tools/directory目录下）来画点9图片。通过沿左侧和顶部边框绘制像素来标记应该被拉伸的区域。也可以通过沿右侧和底部边界绘制像素来标记。就像下图所示一样：

请注意，上图沿边界的黑色像素。在顶部边框和左边框的那些表明图像的可拉伸区域，右边和底部边框则表示内容应该放置的地方。

此外，注意.9.png这个格式，你也必须用这个格式，因为框架会检测这是一个点9图片而不是一个普通图片。

当你将这个应用到组件的背景的时候（通过设置  
android:background="@drawable/button"）， android框架会自动正确的拉伸图像以适应按钮的大小，下图就是各种尺寸中的显示效果：

编写:[riverfeng](#)

校对:

# 兼容不同的屏幕密度

这节课将教你如何通过提供不同的资源和使用独立分辨率（dp）来支持不同的屏幕密度。

## 使用密度独立像素 (dp)

在做UI布局的时候有一个常见的问题，你必须要避免，就是当你设计的布局的时候不要使用规定像素来定义你UI的距离和尺寸。使用像素单位来做UI的单位是会有问题的。因为，不同的屏幕有不同的像素密度，所以，同样单位的像素在不同的设备上会有不同的物理尺寸。因此，在指定单位的时候，通常使用dp或者sp。一个dp代表一个密度独立像素，也就相当于在每英寸160点的屏幕上， $1\text{dp} = 1\text{px}$ 。sp也是一个基本的单位，不过它主要是用在文本尺寸上（它也是一种尺寸规格独立的像素），所以，你在定义文本尺寸的时候应该使用这种规格单位（不要使用在布尺寸上）。

例如，当你是定两个view之间的空间时，应该使用dp而不是px：

```
<Button android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/clickme"  
        android:layout_marginTop="20dp" />
```

当指定文本尺寸时，始终应该使用sp：

```
<TextView android:layout_width="match_parent"  
         android:layout_height="wrap_content"  
         android:textSize="20sp" />
```

## 提供可供选择的图片

因为Android能运行在很多不同屏幕密度的设备上，所以，你应该针对不同的但是常用的几种密度提供不同的bitmap资源：小屏幕（low），medium（中），high（高）以及超高（extra-high）密度。这将能帮助你在所有的屏幕密度中得到非常好的图形质量和性能。

为了提供更好的用户体验，你应该使用以下几种规格为不同的屏幕密度提供位图资源：

```
xhdpi:2.0  
hdpi:1.5  
mdpi:1.0 (标准线)  
ldpi:0.75
```

这也就意味着如果在xhdpi设备上你需要一个200x200的图片，那么你则需要一张150x150的图片用于hdpi，100x100的用于mdpi以及75x75的用户ldpi设备。

然后将这些图片资源放到res/对应的目录下面，系统会自动根据当前屏幕密度自动去选择合适的资源进行加载：

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

这样放置图片资源后，不过你什么时候使用@drawable/awesomeimage，系统都会给予屏幕的dp来选择合适的图片。

如果你想知道更多关于如何为你的应用程序创建icon资源，你可以看看Icon设计指南[Icon Design Guidelines](#)。

编写:[riverfeng](#)

校对:

# 实现可适应的UI流程

在你应用已经可以显示UI的基础上，UI的流程可能会不一样。比如，当你的应用在有两个方框的模式中，点击左边方框的item时，内容显示在右边方框中。如果是在只有一个方框的模式中，当你点击某个item的时候，内容则显示在一个新的activity中。

## 确定当前布局

当你在实现不同布局的时候，首先，你应该确定用户在当前的情况下看到的view应该是个什么样子。比如，你可能想知道当前用户到底是处于“单个方框”的模式还是“多个方框”的模式。这个时候，你就可以通过查询指定的view是不是存在并是否显示来判断当前的模式：

```
public class NewsReaderActivity extends FragmentActivity {  
    boolean mIsDualPane;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main_layout);  
  
        View articleView = findViewById(R.id.article);  
        mIsDualPane = articleView != null &&  
                      articleView.getVisibility() == View.VISIBLE;  
    }  
}
```

注意：使用代码查询id为“article”的view是否可见比直接硬编码查询指定的布局更加的灵活。

另一个关于如何适配已经存在的不同组件的例子是在组件执行操作之前先检查它是否是可用的。比如，在News Reader示例中，有一个按钮点击后打开一个菜单，但是这个按钮仅仅只在Android3.0之后的版本中才能显示（因为这个函数是在API 11中ActionBar中才能有的）。所以，在给这个按钮添加事件之间，你可以这样做：

```
Button catButton = (Button) findViewById(R.id.categorybutton);  
OnClickListener listener = /* create your listener here */;  
if (catButton != null) {  
    catButton.setOnClickListener(listener);  
}
```

## 根据当前布局响应

根据当前不同的布局有一些操作肯定会带来不一样的结果。比如，在News Reader示例中，当你点击headlines列表中的某一条headline时，如果你的UI是在多个方框模式中，内容会显示在右边的方框中，如果你的UI是在单个方框模式中，内容则会显示在一个新的独立的Activity中：

```
@Override  
public void onHeadlineSelected(int index) {  
    mArtIndex = index;  
    if (mIsDualPane) {  
        /* display article on the right pane */  
        mArticleFragment.displayArticle(mCurrentCat.getArticle(index));  
    } else {  
        /* start a separate activity */  
        Intent intent = new Intent(this, ArticleActivity.class);  
        intent.putExtra("catIndex", mCatIndex);  
        intent.putExtra("artIndex", index);  
        startActivity(intent);  
    }  
}
```

同样，如果你的应用程序时多个方框的模式，那么它应该在导航栏中设置带有选项卡的action bar。而如果是单框模式，那么导航栏应该设置为spinner widget。所以，你的代码应该检查哪个方案是最合适的：

```
final String CATEGORIES[] = { "Top Stories", "Politics", "Economy"  
  
public void onCreate(Bundle savedInstanceState) {  
    ....  
    if (mIsDualPane) {  
        /* use tabs for navigation */  
        actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_TABS);  
        int i;  
        for (i = 0; i < CATEGORIES.length; i++) {  
            actionBar.addTab(actionBar.newTab().setText(CATEGORIES[i]).setTabListener(handler));  
        }  
        actionBar.setSelectedNavigationItem(selTab);  
    }  
    else {  
        /* use list navigation (spinner) */  
        actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_LIST);  
        SpinnerAdapter adap = new ArrayAdapter(this,  
            R.layout.headline_item, CATEGORIES);  
        actionBar.setListNavigationCallbacks(adap, handler);  
    }  
}
```

## 在其他Activity中复用Fragment

在设计为多屏幕适配的UI时有一个复用的原则：将你的界面变为单独部分，这样它能在某些屏幕配置上被实现为一个方框，而在其他屏幕配置中，则被实现为一个单独的activity。例如，在News Reader中，新闻内容文字在大屏幕上显示在屏幕右边的方框中，而在小屏幕中，则是由单独的activity显示的。

像这样的情况，你就应该在不同的activity中使用同一个Fragment，以此来避免代码的重复，而达到代码复用的效果。比如，ArticleFragment在多个方框模式下是这样用的：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>
```

在小屏幕中，它又是如下方式被复用的（没有布局文件）：

```
ArticleFragment frag = new ArticleFragment();
getSupportFragmentManager().beginTransaction().add(android.R.id.co
```

当然，如果将这个fragment定义在XML布局文件中，也有同样的效果，但是在这个例子中，则没有必要，因为这个article fragment是这个activity的唯一组件。

当你在设计fragment的时候，有一个非常重要的知识点：不要为某个特定的activity设计耦合度高的fragment。通常的做法是，通过定义抽象接口，并在接口中定义需要与该fragment进行交互的activity的抽象方法，然后与该fragment进行交互的activity实现这些抽象接口方法的具体方法。

例如，在News Reader中，HeadlinesFragment就很好的诠释了这一点：

```
public class HeadlinesFragment extends ListFragment {
    ...
    OnHeadlineSelectedListener mHeadlineSelectedListener = null;

    /* Must be implemented by host activity */
    public interface OnHeadlineSelectedListener {
        public void onHeadlineSelected(int index);
    }
    ...

    public void setOnHeadlineSelectedListener(OnHeadlineSelectedLi
```

```
        mHeadlineSelectedListener = listener;
    }
}
```

然后，当用户选择了一个headline item之后， fragment将通知对应的activity指定监听事件（而不是通过硬编码的方式去通知）：

```
public class HeadlinesFragment extends ListFragment {
    ...
    @Override
    public void onItemClick(AdapterView<?> parent,
                           View view, int position, long id) {
        if (null != mHeadlineSelectedListener) {
            mHeadlineSelectedListener.onHeadlineSelected(position)
        }
    }
    ...
}
```

这种技术在[支持平板与手持设备\(Supporting Tablets and Handsets\)](#)有更加详细的介绍。

## 处理屏幕配置变化

如果使用的是单独的activity来实现你界面的不同部分，你需要注意的是，屏幕变化（如旋转变化）的时候，你也应该根据屏幕配置的变化来改变你UI的变化。

例如，在传统的Android3.0或以上版本的7寸平板上，News Reader示例在竖屏的时候使用独立的activity显示文章内容，而在横屏的时候，则使用两个方框的模式（即内容显示在右边的方框中）。这也就意味着，当用户在竖屏模式下观看文章的时候，你需要检测屏幕是否被改变为了横屏，如果改变了，则结束当前activity并返回到主activity中，这样，UI就能显示为两个方框的模式了。

```
public class ArticleActivity extends FragmentActivity {  
    int mCatIndex, mArtIndex;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mCatIndex = getIntent().getExtras().getInt("catIndex", 0);  
        mArtIndex = getIntent().getExtras().getInt("artIndex", 0);  
  
        // If should be in two-pane mode, finish to return to main  
        if (getResources().getBoolean(R.bool.has_two_panes)) {  
            finish();  
            return;  
        }  
        ...  
    }  
}
```

编写:[xrayzh](#)

校对:

原文:<http://developer.android.com/training/tv/index.html>

# 为TV进行设计

基于Android平台的智能TV，让你可以在家中的TV屏幕上运行你常用的Android应用。在Google应用商店中，有数千个应用已经针对TV设备进行了优化。本课程将演示如何优化你的Android应用，设计TV的布局，使之能够适应用户距离TV几米远并且用遥控器操作的情况。

## Dependencies and Prerequisites

Android 2.0 (API level 5) or higher

# Lessons

- 为TV优化布局

本节课演示如何对TV屏幕上的布局进行优化。在TV屏幕上，应用的布局具有以下特性：

- 固定不变的“横屏（landscape）”模式
- 高分辨率显示
- 远距离的交互环境（10-foot UI）

- 为TV优化导航

本节课演示如何针对TV设计导航，包括：

- 手持遥控器导航
- 提供导航反馈
- 在屏幕上提供便捷的控制选项

- 处理TV不支持的功能

通常情况下，有许多硬件功能在TV上都不可用。本节课演示如何替代这些缺失的功能，或者在运行时检测并关闭这些缺失的功能。

编写:[xrayzh](#)

校对:

原文:<http://developer.android.com/training/tv/optimizing-layouts-tv.html>

# 为TV优化布局

When your application is running on a television set, you should assume that the user is sitting about ten feet away from the screen. This user environment is referred to as the 10-foot UI. To provide your users with a usable and enjoyable experience, you should style and lay out your UI accordingly..

This lesson shows you how to optimize layouts for TV by:

- Providing appropriate layout resources for landscape mode.
- Ensuring that text and controls are large enough to be visible from a distance.
- Providing high resolution bitmaps and icons for HD TV screens.

## **Design Landscape Layouts**

编写:

校对:

为TV优化导航

编写:

校对:

# 处理不支持TV的功能

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/custom-views/index.html>

# 创建自定义View

Android的framework有大量的Views用来与用户进行交互并显示不同种类的数据。但是有时候你的程序有个特殊的需求，而Android内置的views组件并不能实现。这一章节会演示如何创建你自己的views，并使得它们是robust与reusable的。

## 依赖和要求

Android 2.1 (API level 7) 或更高

## 你也可以看

- [Custom Components](#)
- [Input Events](#)
- [Property Animation](#)
- [Hardware Acceleration](#)
- [Accessibility developer guide](#)

## 例子

Download the sample [CustomView.zip](#)

# Lesson

- 创建一个View类

创建一个像内置的view，有自定义属性并支持[ADT](#) layout编辑器。

- 自定义Drawing

使用Android graphics系统使你的view拥有独特的视觉效果。

- 使得View是可交互的

用户期望view对操作反应流畅自然。这节课会讨论如何使用gesture detection, physics, 和 animation使你的用户界面有专业的水准。

- 优化View

不管你的UI如何的漂亮，如果不能流畅运行用户也不会喜欢。学习如何避免一般的性能问题，和如何使用硬件加速来使你的自定义图像运行更流畅。

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/custom-views/create-view.html>

# 创建自定义的View类

设计良好的类总是相似的。它使用一个好用的接口来封装一个特定的功能，它有效的使用CPU与内存，等等。为了成为一个设计良好的类，自定义的view应该：

- 遵守Android标准规则。
- 提供自定义的风格属性值并能够被Android XML Layout所识别。
- 发出可访问的事件。
- 能够兼容Android的不同平台。

Android的framework提供了许多基类与XML标签用来帮助你创建一个符合上面要求的View。这节课会介绍如何使用Android framework来创建一个view的核心功能。

## Subclass a View

Android framework里面定义的view类都继承自View。你自定义的view也可以直接继承View，或者你可以通过继承既有的一个子类(例如Button)来节约一点时间。

为了允许Android Developer Tools能够识别你的view，你必须至少提供一个constructor，它包含一个Content与一个AttributeSet对象作为参数。这个constructor允许layout editor创建并编辑你的view的实例。

```
class PieChart extends View {  
    public PieChart(Context context, AttributeSet attrs) {  
        super(context, attrs);  
    }  
}
```

## Define Custom Attributes

为了添加一个内置的View到你的UI上，你需要通过XML属性来指定它的样式与行为。为了实现自定义的view的行为，你应该：

- 为你的view在资源标签下定义自设的属性
- 在你的XML layout中指定属性值
- 在运行时获取属性值
- 把获取到的属性值应用在你的view上

为了定义自设的属性，添加 资源到你的项目中。放置于res/values/attrs.xml文件中。下面是一个attrs.xml文件的示例：

```
<resources>
    <declare-styleable name="PieChart">
        <attr name="showText" format="boolean" />
        <attr name="labelPosition" format="enum">
            <enum name="left" value="0"/>
            <enum name="right" value="1"/>
        </attr>
    </declare-styleable>
</resources>
```

上面的代码声明了2个自设的属性，**showText**与**labelPosition**，它们都归属于PieChart的项目下的styleable实例。styleable实例的名字，通常与自定义的view名字一致。尽管这并没有严格规定要遵守这个convention，但是许多流行的代码编辑器都依靠这个命名规则来提供statement completion。

一旦你定义了自设的属性，你可以在layout XML文件中使用它们。唯一不同的是你自设的属性是归属于不同的命名空间。不是属

于<http://schemas.android.com/apk/res/android>的命名空间，它们归属  
于[http://schemas.android.com/apk/res/\[your package name\]](http://schemas.android.com/apk/res/[your package name])。例如，下面演示了如何为PieChart使用上面定义的属性：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res/com.example.customviews"
    <com.example.customviews.charting.PieChart
        custom:showText="true"
        custom:labelPosition="left" />
</LinearLayout>
```

为了避免输入长串的namespace名字，示例上面使用了xmlns指令，这个指令可以指派custom作

为<http://schemas.android.com/apk/res/com.example.customviews>namespace的别名。你也可以选择其他的别名作为你的namespace。

请注意，如果你的view是一个inner class，你必须指定这个view的outer class。同样的，如果PieChart有一个inner class叫做PieView。为了使用这个类中自设的属性，你应该使用[com.example.customviews.charting.PieChart\\$PieView](com.example.customviews.charting.PieChart$PieView)。

## Apply Custom Attributes

当view从XML layout被创建的时候，在xml标签下的属性值都是从resource下读取出来并传递到view的constructor作为一个AttributeSet参数。尽管可以从AttributeSet中直接读取数值，可是这样做有些弊端（没有看懂下面的两个原因）：

- 拥有属性的资源并没有经过分解
- Styles并没有运用上

取而代之的是，通过obtainStyledAttributes()来获取属性值。这个方法会传递一个[TypedArray](#)对象，它是间接referenced并且styled的。

Android资源编译器帮你做了许多工作来使调用[obtainStyledAttributes\(\)](#)更简单。对res目录里的每一个<declare-styleable>资源，自动生成的R.java文件定义了存放属性ID的数组和常量，常量用来索引数组中每个属性。你可以使用这些预先定义的常量来从[TypedArray](#)中读取属性。这里就是PieChart类如何读取它的属性：

```
public PieChart(Context context, AttributeSet attrs) {  
    super(context, attrs);  
    TypedArray a = context.getTheme().obtainStyledAttributes(  
        attrs,  
        R.styleable.PieChart,  
        0, 0);  
  
    try {  
        mShowText = a.getBoolean(R.styleable.PieChart_showText, false);  
        mTextPos = a.getInteger(R.styleable.PieChart_labelPosition,  
    } finally {  
        a.recycle();  
    }  
}
```

请注意TypedArray对象是一个shared资源，必须被在使用后进行回收。

## Add Properties and Events

Attributes是一个强大的控制view的行为与外观的方法，但是他们仅仅能够在view被初始化的时候被读取到。为了提供一个动态的行为，需要暴露出一些合适的getter与setter方法。下面的代码演示了如何使用这个技巧：

```
public boolean isShowText() {  
    return mShowText;  
}  
  
public void setShowText(boolean showText) {  
    mShowText = showText;  
    invalidate();  
    requestLayout();  
}
```

请注意，在setShowText方法里面有调用[invalidate\(\)](#) and [requestLayout\(\)](#)。当view的某些内容发生变化的时候，需要调用invalidate来通知系统对这个view进行redraw，当某些元素变化会引起组件大小变化时，需要调用requestLayout方法。

自定义的view也需要能够支持响应事件的监听器。例如，PieChart暴露了一个自设的事件OnCurrentItemChanged来通知监听器，用户已经切换了焦点到一个新的组件上。

我们很容易忘记了暴露属性与事件，特别是当你是这个view的唯一用户时。请花费一些时间来仔细定义你的view的交互。一个好的规则是总是暴露任何属性与事件。

## Design For Accessibility

Your custom view should support the widest range of users. This includes users with disabilities that prevent them from seeing or using a touchscreen. To support users with disabilities, you should:

- Label your input fields using the android:contentDescription attribute
- Send accessibility events by calling sendAccessibilityEvent() when appropriate.
- Support alternate controllers, such as D-pad and trackball

For more information on creating accessible views, see [Making Applications Accessible](#) in the Android Developers Guide.

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/custom-view/custom-draw.html>

# 实现自定义View的绘制

自定义view最重要的一个部分是自定义它的外观。根据你的程序的需求，自定义绘制动作可能简单也可能很复杂。这节课会演示一些最常见的操作。

## Override onDraw()

重绘一个自定义的view最重要的步骤是重写onDraw()方法。onDraw()的参数是一个Canvas对象。Canvas类定义了绘制文本，线条，图像与许多其他图形的方法。你可以在onDraw方法里面使用那些方法来创建你的UI。

在你调用任何绘制方法之前，你需要创建一个Paint对象。

## Create Drawing Objects

android.graphics framework把绘制定义为下面两类:

- 绘制什么, 由Canvas控制
- 如何绘制, 由Paint控制

例如Canvas提供绘制一条直线的方法, Paint提供直线颜色。所以在绘制之前, 你需要创建一个或者多个Paint对象。

```
private void init() {  
    mTextPaint = new Paint(Paint.ANTI_ALIAS_FLAG);  
    mTextPaint.setColor(mTextColor);  
    if (mTextHeight == 0) {  
        mTextHeight = mTextPaint.getTextSize();  
    } else {  
        mTextPaint.setTextSize(mTextHeight);  
    }  
  
    mPiePaint = new Paint(Paint.ANTI_ALIAS_FLAG);  
    mPiePaint.setStyle(Paint.Style.FILL);  
    mPiePaint.setTextSize(mTextHeight);  
  
    mShadowPaint = new Paint(0);  
    mShadowPaint.setColor(0xff101010);  
    mShadowPaint.setMaskFilter(new BlurMaskFilter(8, BlurMaskFilter  
  
    ...
```

刚开始就创建对象是一个重要的优化技巧。Views会被频繁的重新绘制, 初始化许多绘制对象需要花费昂贵的代价。在onDraw方法里面创建绘制对象会严重影响到性能并使得你的UI显得卡顿。

## Handle Layout Events

为了正确的绘制你的view，你需要知道view的大小。复杂的自定义view通常需要根据在屏幕上的大小与形状执行多次layout计算。你不应该去估算这个view在屏幕上的显示大小。即使只有一个程序会使用你的view，仍然是需要处理屏幕大小不同，密度不同，方向不同所带来的影响。

尽管view有许多方法是用来计算大小的，但是大多数是不需要重写的。如果你的view不需要特别的控制它的大小，唯一需要重写的方法是[onSizeChanged\(\)](#)。

onSizeChanged()，当你的view第一次被赋予一个大小时，或者你的view大小被更改时会被执行。在onSizeChanged方法里面计算位置，间距等其他与你的view大小值。

当你的view被设置大小时，layout manager(布局管理器)假定这个大小包括所有的view的内边距(padding)。当你计算你的view大小时，你必须处理内边距的值。这段PieChart.onSizeChanged()中的代码演示该怎么做：

```
// Account for padding
float xpad = (float)(getPaddingLeft() + getPaddingRight());
float ypad = (float)(getPaddingTop() + getPaddingBottom());

// Account for the label
if (mShowText) xpad += mTextWidth;

float ww = (float)w - xpad;
float hh = (float)h - ypad;

// Figure out how big we can make the pie.
float diameter = Math.min(ww, hh);
```

如果你想更加精确的控制你的view的大小，需要重写[onMeasure\(\)](#)方法。这个方法的参数是View.MeasureSpec，它会告诉你的view的父控件的大小。那些值被包装成int类型，你可以使用静态方法来获取其中的信息。

这里是一个实现[onMeasure\(\)](#)的例子。在这个例子中PieChart试着使它的区域足够大，使pie可以像它的label一样大：

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
    // Try for a width based on our minimum
    int minw = getPaddingLeft() + getPaddingRight() + getSuggestedMinimumWidth();
    int w = resolveSizeAndState(minw, widthMeasureSpec, 1);

    // Whatever the width ends up being, ask for a height that would
    // get as big as it can
    int minh = MeasureSpec.getSize(w) - (int)mTextWidth + getPaddingTop();
    int h = resolveSizeAndState(MeasureSpec.getSize(w) - (int)mTextWidth, heightMeasureSpec);

    setMeasuredDimension(w, h);
}
```

上面的代码有三个重要的事情需要注意：

- 计算的过程有把view的padding考虑进去。这个在后面会提到，这部分是view所控制的。
- 帮助方法resolveSizeAndState()是用来创建最终的宽高值的。这个方法会通过比较view的需求大小与spec值返回一个合适的View.MeasureSpec值，并传递到onMeasure方法中。
- onMeasure()没有返回值。它通过调用setMeasuredDimension()来获取结果。调用这个方法是强制执行的，如果你遗漏了这个方法，会出现运行时异常。

# Draw!

每个view的onDraw都是不同的，但是有下面一些常见的操作：

- 绘制文字使用drawText()。指定字体通过调用setTypeface(), 通过setColor()来设置文字颜色.
- 绘制基本图形使用drawRect(), drawOval(), drawArc(). 通过setStyle()来指定形状是否需要filled, outlined.
- 绘制一些复杂的图形，使用Path类. 通过给Path对象添加直线与曲线, 然后使用drawPath()来绘制图形. 和基本图形一样，paths也可以通过setStyle来设置是outlined, filled, both.
- 通过创建LinearGradient对象来定义渐变。调用setShader()来使用LinearGradient。
- 通过使用drawBitmap来绘制图片.

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Draw the shadow
    canvas.drawOval(
        mShadowBounds,
        mShadowPaint
    );

    // Draw the label text
    canvas.drawText(mData.get(mCurrentItem).mLabel, mTextX, mTextY,

    // Draw the pie slices
    for (int i = 0; i < mData.size(); ++i) {
        Item it = mData.get(i);
        mPiePaint.setShader(it.mShader);
        canvas.drawArc(mBounds,
            360 - it.mEndAngle,
            it.mEndAngle - it.mStartAngle,
            true, mPiePaint);
    }

    // Draw the pointer
    canvas.drawLine(mTextX, mPointerY, mPointerX, mPointerY, mTextP
    canvas.drawCircle(mPointerX, mPointerY, mPointerSize, mTextPain
}
```

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/custom-view/make-interactive.html>

# 使得View可交互

绘制UI仅仅是创建自定义View的一部分。你还需要使得你的View能够以模拟现实世界的方式来进行反馈。Objects应该总是与现实情景能够保持一致。例如，图片不应该突然消失又从另外一个地方出现，因为在现实世界里面不会发生那样的事情。正确的应该是，图片从一个地方移动到另外一个地方。

用户应该可以感受到UI上的微小变化，并对这些变化反馈到现实世界中。例如，当用户做fling(迅速滑动)的动作，应该在滑动开始与结束的时候给用户一定的反馈。

这节课会演示如何使用Android framework的功能来为自定义的View添加那些现实世界中的行为。

## Handle Input Gestures

像许多其他UI框架一样，Android提供一个输入事件模型。用户的动作会转换成触发一些回调函数的事件，你可以重写这些回调方法来定制你的程序应该如何响应用户的输入事件。在Android中最常用的输入事件是touch，它会触发[onTouchEvent\(android.view.MotionEvent\)](#)的回调。重写这个方法来处理touch事件：

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    return super.onTouchEvent(event);  
}
```

Touch事件本身并不是特别有用。如今的touch UI定义了touch事件之间的相互作用，叫做gestures。例如tapping,pulling,flinging与zooming。为了把那些touch的源事件转换成gestures, Android提供了[GestureDetector](#)。

通过传入[GestureDetector.OnGestureListener](#)的一个实例构建一个GestureDetector。如果你只是想要处理几种gestures(手势操作)你可以继承[GestureDetector.SimpleOnGestureListener](#)，而不用实现[GestureDetector.OnGestureListener](#)接口。例如，下面的代码创建一个继承[GestureDetector.SimpleOnGestureListener](#)的类，并重写[onDown\(MotionEvent\)](#)。

```
class mListener extends GestureDetector.SimpleOnGestureListener {  
    @Override  
    public boolean onDown(MotionEvent e) {  
        return true;  
    }  
}  
mDetector = new GestureDetector(PieChart.this.getContext(), new mL
```

不管你是否使用[GestureDetector.SimpleOnGestureListener](#)，你必须总是实现onDown()方法，并返回true。这一步是必须的，因为所有的gestures都是从onDown()开始的。如果你在onDown()里面返回false，系统会认为你想要忽略后续的gesture，那么GestureDetector.OnGestureListener的其他回调方法就不会被执行到了。一旦你实现了[GestureDetector.OnGestureListener](#)并且创建了[GestureDetector](#)的实例，你可以使用你的[GestureDetector](#)来中止你在onTouchEvent里面收到的touch事件。

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    boolean result = mDetector.onTouchEvent(event);  
    if (!result) {  
        if (event.getAction() == MotionEvent.ACTION_UP) {  
            stopScrolling();  
            result = true;  
        }  
    }  
    return result;  
}
```

当你传递一个touch事件到onTouchEvent()时，若这个事件没有被认为是gesture中的一部分，它会返回false。你可以执行自定义的gesture-deception代码。

## Create Physically Plausible(貌似可信的) Motion

Gestures是控制触摸设备的一种强有力的方式，但是除非你能够产出一个合理的触摸反馈，否则将是违反用户直觉的。一个很好的例子是fling手势，用户迅速的在屏幕上移动手指然后抬手离开屏幕。这个手势应该使得UI迅速的按照fling的方向进行滑动，然后慢慢停下来，就像是用户旋转一个飞轮一样。

幸运的是，Android有提供帮助类来模拟这些物理行为。

```
@Override  
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,  
                        float velocityY) {  
    mScroller.fling(currentX, currentY, velocityX / SCALE, velocityY);  
    postInvalidate();  
}
```

**Note:** 尽管速率是通过GestureDetector来计算的，许多开发者感觉使用这个值使得fling动画太快。通常把x与y设置为4到8倍的关系。

```
if (!mScroller.isFinished()) {  
    mScroller.computeScrollOffset();  
    setPieRotation(mScroller.getCurry());  
}
```

Scroller类会为你计算滚动位置，但是他不会自动把哪些位置运用到你的view上面。你有责任确保View获取并运用到新的坐标。你有两种方法来实现这件事情：

- 在调用fling()之后执行postInvalidate()，这是为了确保能强制进行重画。这个技术需要每次在onDraw里面计算过scroll offsets(滚动偏移量)之后调用postInvalidate()。
- 使用[ValueAnimator](#)

第二个方法使用起来会稍微复杂一点，但是它更有效率并且避免了不必要的重画的view进行重绘。缺点是ValueAnimator是从API Level 11才有的。因此他不能运用到3.0的系统之前的版本上。

**Note:** ValueAnimator虽然是API 11才有的，但是你还是可以在最低版本低于3.0的系统上使用它，做法是在运行时判断当前的API Level，如果低于11则跳过。

```
mScroller = new Scroller(getContext(), null, true);  
mScrollAnimator = ValueAnimator.ofFloat(0, 1);  
mScrollAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {  
    @Override  
    public void onAnimationUpdate(ValueAnimator valueAnimator) {  
        if (!mScroller.isFinished()) {  
            mScroller.computeScrollOffset();  
            setPieRotation(mScroller.getCurry());  
        }  
    }  
});  
mScroller.startScroll(currentX, currentY, scrollOffset, scrollOffset);  
mScroller.setDuration(1000);  
mScrollAnimator.setDuration(1000);  
mScrollAnimator.start();
```

```
    } else {
        mScrollAnimator.cancel();
        onScrollFinished();
    }
});
```

## Make Your Transitions Smooth

用户期待一个UI之间的切换是能够平滑过渡的。UI元素需要做到渐入淡出来取代突然出现与消失。Android从3.0开始有提供[property animation framework](#),用来使得平滑过渡变得更加容易。

如果你不想改变View的属性，只是做一些动画的话，你可以使用ObjectAnimator.

```
mAutoCenterAnimator = ObjectAnimator.ofInt(PieChart.this, "PieRotation", 0, targetAngle);
mAutoCenterAnimator.setIntValues(targetAngle);
mAutoCenterAnimator.setDuration(AUTOCENTER_ANIM_DURATION);
mAutoCenterAnimator.start();
```

如果你想改变的是view的某些基础属性，你可以使用[ViewPropertyAnimator](#),它能够同时执行多个属性的动画。

```
animate().rotation(targetAngle).setDuration(ANIM_DURATION).start()
```

编写:[kesenhoo](#)

校对:

# 优化自定义View

前面的课程学习到了如何创建设计良好的View，并且能够使之在手势与状态切换时得到正确的反馈。下面要介绍的是如何使得view能够执行更快。为了避免UI显得卡顿，你必须确保动画能够保持在60fps以上。

## Do Less, Less Frequently

为了加速你的view，对于频繁调用的方法，需要尽量减少不必要的代码。先从onDraw开始，需要特别注意不应该在这里做内存分配的事情，因为它会导致GC，从而导致卡顿。在初始化或者动画间隙期间做分配内存的动作。不要在动画正在执行的时候做内存分配的事情。

你还需要尽可能的减少onDraw被调用的次数，大多数时候导致onDraw都是因为调用了invalidate()。因此请尽量减少调用invalidate()的次数。如果可能的话，尽量调用含有4个参数的invalidate()方法而不是没有参数的invalidate()。没有参数的invalidate会强制重绘整个view。

另外一个非常耗时的操作是请求layout。任何时候执行requestLayout()，会使得Android UI系统去遍历整个View的层级来计算出每一个view的大小。如果找到有冲突的值，它会需要重新计算好几次。另外需要尽量保持View的层级是扁平化的，这样对提高效率很有帮助。

如果你有一个复杂的UI，你应该考虑写一个自定义的ViewGroup来执行他的layout操作。与内置的view不同，自定义的view可以使得程序仅仅测量这一部分，这避免了遍历整个view的层级结构来计算大小。

## Use Hardware Acceleration

从Android 3.0开始，Android的2D图像系统可以通过GPU来加速。GPU硬件加速可以提高许多程序的性能。但是这并不是说它适合所有的程序。

参考[Hardware Acceleration](#) 来学习如何在程序中启用加速。

一旦你开启了硬件加速，性能的提示并不一定可以明显察觉到。移动GPU在某些例如 scaling,rotating与translating的操作中表现良好。但是对其他一些任务则表现不佳。

在下面的例子中，绘制pie是相对来说比较费时的。解决方案是把pie放到一个子view中，并设置View使用LAYER\_TYPE\_HARDWARE来进行加速。

```
private class PieView extends View {

    public PieView(Context context) {
        super(context);
        if (!isInEditMode()) {
            setLayerType(View.LAYER_TYPE_HARDWARE, null);
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        for (Item it : mData) {
            mPiePaint.setShader(it.mShader);
            canvas.drawArc(mBounds,
                360 - it.mEndAngle,
                it.mEndAngle - it.mStartAngle,
                true, mPiePaint);
        }
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int ol
        mBounds = new RectF(0, 0, w, h);
    }

    RectF mBounds;
}
```

通过这样的修改以后，PieChart.PieView.onDraw()只会在第一次现实的时候被调用。之后，pie chart会被缓存为一张图片，并通过GPU来进行重画不同的角度。

缓存图片到hardware layer会消耗video memory，而video memory又是有限的。基于这样的考虑，仅仅在用户触发scrolling的时候使用LAYER\_TYPE\_HARDWARE，在其他时候，使用LAYER\_TYPE\_NONE。

编写:

校对:

# 创建向后兼容的UI

编写:

校对:

# 抽象新的APIs

编写:

校对:

# 代理至新的APIs

编写:

校对:

# 使用旧的APIs实现新API的效果

编写:

校对:

# 使用版本敏感的组件

编写:[KOST](#), 校对:

原文:<http://developer.android.com/training/accessibility/index.html>

# 实现辅助功能

当我们需要尽可能扩大我们用户的基数的时候，就要开始注意我们软件的可达性了(*Accessibility* 易接近，可亲性)。界面中的小提示对大多数用户而言是可行的，比如说当按钮被按下时视觉上的变化，但是对于那些视力上有些缺陷的用户而言效果就不是那么理想了。

本章将给您演示如何最大化利用Android框架中的Accessibility特性。包括如何利用焦点导航(*focus navigation*)与内容描述(*content description*)对你的应用的可达性进行优化。也包括了创建Accessibility Service，使用户与应用（不仅仅是你自己的应用）之间的交互更加容易。

# 课程

- [开发Accessibility应用](#)

学习如何让你的程序更易用，具有可达性。允许使用键盘或者十字键(*directional pad*)来进行导航，利用Accessibility Service特性设置标签或执行事件来打造更舒适的用户体验。

- [编写 Accessibility Services](#)

编写一个Accessibility Service来监听可达性事件，利用这些不同类型的事件和内容描述来帮助用户与应用的交互。本例将会实现利用一个TTS引擎来向用户发出语音提示的功能。

编写:K0ST

校对:

# 开发辅助程序

本课程将教您：

1. 添加内容描述(*Content Descriptions*)
2. 设计焦点导航 (*Focus Navigation*)
3. 执行可达性事件(*Accessibility Events*)
4. 测试你的程序

Android平台本身有一些专注可达性的特性，这些特性可以帮助你专门为那些视觉上或生理上有缺陷的人在应用上做特别的优化。然而，正确的优化方式或最简单利用这个特性的方法往往不是那么显而易见的。本课程将给您演示如何利用和实现这些策略和平台的特性，构建一个更好的具有可达性的Android应用。

## 添加内容描述

一个好的交互界面上的元素通常不需要特别使用一个标签来表明这个元素的作用。例如对于一个任务型应用来说，一个项目旁边的勾选框表达的意思就非常明确，或者对于一个文件管理应用，垃圾桶的图标表达的意思也非常清除。然而对于具有视觉障碍的用户来说，其他类型的UI交互提示是有必要的。

幸运的是，我们可以很轻松的给一个UI元素加上标签，这样类似于[TalkBack](#)这样的基于语音的Accessibility Service就可以将标签的内容朗读出来。如果你的标签在整个应用的声明周期中不太可能会发生变化(比如‘停止’或者‘购买’)，你就可以在XML布局文件中对`android:contentDescription`属性进行修改。例子如下：

```
<Button  
    android:id="@+id/pause_button"  
    android:src="@drawable/pause"  
    android:contentDescription="@string/pause"/>
```

然而，在很多情况下描述的内容是基于上下文环境的，比如说一个开关按钮的状态，或者在list中一片可选的数据项。在运行时编辑内容描述可以使用`setContentDescription()`方法，例子如下：

```
String contentDescription = "Select " + strValues[position];  
label.setContentDescription(contentDescription);
```

将这个添加进您的代码是提高您应用可达性的最简单的方法。尝试着将那些有用的地方都加入内容描述，但是要避免像web开发者那样将所有的元素都标注，那样会产生大量的无用信息。比如说，不要将应用图标的内容描述设置为‘应用图标’。这仅仅会对用户的浏览产生干扰。

来试试吧！下载TalkBack(谷歌开发的一款可达性应用)，在**Settings > Accessibility > TalkBack**将它开启。然后使用你的应用听听看TalkBack发出的语音提示。

## 设计焦点导航

你的应用除了支持触摸操作外，更应该支持其他的导航方式。很多Android设备不仅仅提供了触摸屏，还提供了其他的导航硬件比如说十字键、方向键、轨迹球等等。除此之外，最新的Android发行版本也支持蓝牙或USB的外接设备，比如键盘等等。

为了实现这种方式的导航，一切用户可以用来可导航的元素(*navigational elements*)都需要设置为`focusable`（聚焦），这个设置也可以在运行时通过`View.setFocusable()`方法来进行设定，或者也可以在XML布局文件中使用`android:focusable`来设置。

每个UI控件有四个属

性，`android:nextFocusUp, android:nextFocusDown, android:nextFocusLeft, android:nextFocusRight` 用户在导航时可以利用这些属性来指定下一个焦点的位置。系统会自动根据布局的方向来确定导航的顺序，如果在您的应用中系统提供的方案不合适，您可以用这些属性来进行修改。

比如说，下面就是一个关于按钮和标签的例子，他们都是可聚焦的(*focusable*)，按向下键会将焦点从按钮移到文字上，按向上会重新将焦点移到按钮上。

```
<Button android:id="@+id/doSomething"
    android:focusable="true"
    android:nextFocusDown="@+id/label"
    ... />
<TextView android:id="@+id/label"
    android:focusable="true"
    android:text="@string/labelText"
    android:nextFocusUp="@+id/doSomething"
    ... />
```

证实您的应用运行正确的直观方法，最简单的方式就是在Android虚拟机里运行您的应用，然后使用虚拟器的方向键来在各个元素之间导航，使用OK按钮来代替触摸元素的事件。

## 填充可达性事件

如果你在你的Android框架中使用了View组件，当你选中了一个View或者是焦点变化的时候，可达性事件(*AccessibilityEvent*)都会产生。这些事件会被传递到Accessibility Service中进行处理，实现一些辅助功能，如语音提示等。

如果你写了一个自定义的View，请确保它在合适的时候产生事件。使用*sendAccessibilityEvent(int)*函数可以产生可达性事件，其中的参数表示事件的类型。完整的可达性事件类型可查阅[AccessibilityEvent](#)参考文档。

比如说，你拓展了一个图片的View，你希望在它聚焦的时候使用键盘打字可以在其中插入题注，这时候发送一个*TYPE\_VIEW\_TEXT\_CHANGED*事件就非常合理，尽管它不是本身就构建在这个图片View中的。产生事件的代码如下：

```
public void onTextChanged(String before, String after) {  
    ...  
    if (AccessibilityManager.getInstance(mContext).isEnabled()) {  
        sendAccessibilityEvent(AccessibilityEvent.TYPE_VIEW_TEXT_C  
    }  
    ...  
}
```

## 测试你的程序

请确保您在添加可达性功能后测试它的有效性。为了测试内容描述可可达性事件，请安装并启用一个Accessibility Service。其中的一个选择就是使用TalkBack，它是一个免费的开源的屏幕读取软件，可在Google Play上进行下载。Service启动后，请测试您应用中所有的功能，同时听听TalkBack的语音反馈。

同时，尝试着用一个方向控制器来控制你的应用，而非使用触摸的方式。你可以使用一个物理设备，比如十字键、轨迹球等。如果没有条件，可以使用android虚拟器，它提供了虚拟的按键控制。

在测试导航与反馈的同时，和没有任何视觉提示的情况下，你应该对你的应用大概是一个什么样子有所认识。出现问题就修复优化它们，你最终就会开发出一个更易用可达的Android程序。

编写:K0ST

校对:

# 开发辅助服务

本课程将教您：

1. 创建Accessibility Service
2. 配置你的Accessibility Service
3. 响应AccessibilityEvents
4. 从View层级中提取更多信息

Accessibility Service是Android系统框架提供给安装在设备上应用的一个可选的导航反馈特性。Accessibility Service 可以替代应用与用户交流反馈，比如将文本转化为语音提示，或是用户的手指悬停在屏幕上一个较重要的区域时的触摸反馈等。本课程将教您如何创建一个Accessibility Service，同时处理来自应用的信息，并将这些信息反馈给用户。

## 创建Accessibility Service

Accessibility Service可以绑定在一个正常的应用中，或者是单独的一个Android项目都可以。创建一个Accessibility Service的步骤与创建普通Service的步骤相似，在你的项目中创建一个继承于[AccessibilityService](#)的类：

```
package com.example.android.apis.accessibility;

import android.accessibilityservice.AccessibilityService;

public class MyAccessibilityService extends AccessibilityService {
    ...
    @Override
    public void onAccessibilityEvent(AccessibilityEvent event) {
    }

    @Override
    public void onInterrupt() {
    }

    ...
}
```

与其他Service类似，你必须在manifest文件当中声明这个Service。记得标明它监听处理了android.accessibilityservice事件，以便Service在其他应用产生[AccessibilityEvent](#)的时候被调用。

```
<application ...>
    ...
    <service android:name=".MyAccessibilityService">
        <intent-filter>
            <action android:name="android.accessibilityservice.AccessibilityService" />
        </intent-filter>
        ...
    </service>
    ...
</application>
```

如果你为这个Service创建了一个新项目，且仅仅是一个Service而不准备做成一个应用，那么你就可以移除启动的Activity(一般为MainActivity.java)，同样也记得在manifest中将这个Activity声明移除。

## 配置你的Accessibility Service

设置Accessibility Service的配置变量会告诉系统如何让Service运行与何时运行。你希望响应哪种类型的事件？Service是否对所有的应用有效还是对部分指定包名的应用有效？使用哪些不同类型的反馈？

你有两种设置这些变量属性的方法，一种向下兼容的办法是通过代码来进行设定，使用`setServiceInfo(android.accessibilityservice.AccessibilityServiceInfo)`。你需要重写(*override*)`onServiceConnected()`方法，并在这里进行Service的配置。

```
@Override  
public void onServiceConnected() {  
    // Set the type of events that this service wants to listen to  
    // won't be passed to this service.  
    info.eventTypes = AccessibilityEvent.TYPE_VIEW_CLICKED |  
        AccessibilityEvent.TYPE_VIEW_FOCUSED;  
  
    // If you only want this service to work with specific applica-  
    // package names here. Otherwise, when the service is activat-  
    // ed to events from all applications.  
    info.packageNames = new String[]  
        {"com.example.android.myFirstApp", "com.example.androi-  
            // Set the type of feedback your service will provide.  
            info.feedbackType = AccessibilityServiceInfo.FEEDBACK_SPOKEN;  
  
            // Default services are invoked only if no package-specific or  
            // application-specific for the type of AccessibilityEvent generated. This service  
            // is application-specific, so the flag isn't necessary. If this  
            // were a general-purpose service, it would be worth considering sett-  
            // ing the DEFAULT flag.  
  
            // info.flags = AccessibilityServiceInfo.DEFAULT;  
  
            info.notificationTimeout = 100;  
  
            this.setServiceInfo(info);  
    }  
}
```

在Android 4.0之后，就用另一种方式来设置了：通过设置XML文件来进行配置。一些特性的选项比如`canRetrieveWindowContent`仅仅可以在XML可以配置。对于上面所示的相应的配置，利用XML配置如下：

```
<accessibility-service  
    android:accessibilityEventTypes="typeViewClicked|typeViewFocu-  
    android:packageNames="com.example.android.myFirstApp, com.exa-  
    android:accessibilityFeedbackType="feedbackSpoken"  
    android:notificationTimeout="100"  
    android:settingsActivity="com.example.android.apis.accessibil-  
    android:canRetrieveWindowContent="true"  
/>
```

---

如果你确定是通过XML进行配置，那么请确保在manifest文件中通过< meta-data >标签指定这个配置文件。假设此配置文件存放的地址为：res/xml/serviceconfig.xml，那么标签应该如下：

```
<service android:name=".MyAccessibilityService">
    <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService" />
    </intent-filter>
    <meta-data android:name="android.accessibilityservice"
        android:resource="@xml/serviceconfig" />
</service>
```

## 响应Accessibility Event

现在你的Service已经配置好并可以监听Accessibility Event了，来写一些响应这些事件的代码吧！首先就是要重写`onAccessibilityEvent(AccessibilityEvent event)`方法，在这个方法中，使用`getEventType()`来确定事件的类型，使用`getContentDescription()`来提产生这个事件的View相关的文本标签。

```
@Override
public void onAccessibilityEvent(AccessibilityEvent event) {
    final int eventType = event.getEventType();
    String eventText = null;
    switch(eventType) {
        case AccessibilityEvent.TYPE_VIEW_CLICKED:
            eventText = "Focused: ";
            break;
        case AccessibilityEvent.TYPE_VIEW_FOCUSED:
            eventText = "Focused: ";
            break;
    }

    eventText = eventText + event.getContentDescription();

    // Do something nifty with this text, like speak the composed
    // back to the user.
    speakToUser(eventText);
    ...
}
```

## 从View层级中提取更多信息

这一步并不是必要步骤，但是却非常有用。Android 4.0版本中增加了一个新特性，就是能够用AccessibilityService来遍历View层级，并从产生Accessibility事件的组件与它的父子组件中提取必要的信息。为了实现这个目的，你需要在XML文件中进行如下的配置：

1. 立即获取到产生这个事件的Parent
2. 在这个Parent中寻找文本标签或勾选框
3. 如果找到，创建一个字符串来反馈给用户，提示内容和是否已勾选。
4. 如果当遍历View的时候某处返回了null值，那么就直接结束这个方法。

```
// Alternative onAccessibilityEvent, that uses AccessibilityNodeInfo

@Override
public void onAccessibilityEvent(AccessibilityEvent event) {

    AccessibilityNodeInfo source = event.getSource();
    if (source == null) {
        return;
    }

    // Grab the parent of the view that fired the event.
    AccessibilityNodeInfo rowNode = getListItemNodeInfo(source);
    if (rowNode == null) {
        return;
    }

    // Using this parent, get references to both child nodes, the
    // AccessibilityNodeInfo labelNode = rowNode.getChild(0);
    if (labelNode == null) {
        rowNode.recycle();
        return;
    }

    AccessibilityNodeInfo completeNode = rowNode.getChild(1);
    if (completeNode == null) {
        rowNode.recycle();
        return;
    }

    // Determine what the task is and whether or not it's complete
    // the text inside the label, and the state of the check-box.
    if (rowNode.getChildCount() < 2 || !rowNode.getChild(1).isChecked())
        rowNode.recycle();
    return;
}

CharSequence taskLabel = labelNode.getText();
final boolean isComplete = completeNode.isChecked();
String completeStr = null;

if (isComplete) {
    completeStr = getString(R.string.checked);
```

```
    } else {
        completeStr = getString(R.string.not_checked);
    }
    String reportStr = taskLabel + completeStr;
    speakToUser(reportStr);
}
```

现在你已经实现了一个完整可运行的Accessibility Service。尝试着调整它与用户的交互方式吧！比如添加语音引擎，或者添加震动来提供触觉上的反馈都是不错的选择！

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/index.html>

# 管理系统UI

系统Bar是用来展示通知、表现设备状态和完成设备导航的屏幕区域。通常上来说，系统Bar包括状态栏和导航栏，他们一般都是与程序同时显示在屏幕上的。而照片、视频等这类沉浸式的应用可以临时弱化系统Bar图标来创造一个更加专注的体验环境，甚至可以完全隐藏系统Bar。

如果你对Android Design指南很熟悉，你应该已经知道遵照标准的Android设计UI指南与使用模式来设计App的重要性。在你修改系统Bar之前，你应该仔细的考虑一下用户的需求与预期，因为它们是操作设备和观察设备状态的常规途径。

这节课描述了如何在不同版本的Android上隐藏或淡化系统Bar，来营造一个沉浸式的用户体验，同时做到快速的访问与操作系统Bar。

# 课程

- [淡化系统Bar](#)

学习如何淡化和隐藏状态栏与导航栏。

- [隐藏状态栏](#)

学习如何在不同版本的Android上隐藏状态栏。

- [隐藏导航栏](#)

学习如何隐藏导航栏。

- [全屏沉浸式应用](#)

学习如何在你的App中创建沉浸模式。

- [响应UI可见性的变化](#)

学习如何注册一个监听器来监听系统UI可见性的变化，以便于相应的调整App的UI。

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/dim.html>

# 淡化系统Bar

本课程将向你讲解如何在Android 4.0(*API level 14*)与更高的的系统版本上淡化系统Bar(状态栏与导航栏)。早起版本的Android没有提供一个自带的方法来淡化系统Bar。

当你使用这个方法的时候，内容区域并不会发生大小的变化，只是系统栏的图标会收起来。一旦用户触摸状态栏或者是导航栏的时候，这两个系统栏都会变得完全可见。这种方法的优势是系统栏仍然可见，但是它们的细节被隐藏掉了，因此可以在不牺牲快捷访问系统Bar的情况下创建一个沉浸式的体验。

这节课将教您

1. 淡化状态栏和导航栏
2. 显示状态栏和导航栏

同时您应该阅读

[ActionBar Api指南](#)

[Android 设计指南](#)

## 淡化状态栏和系统栏

在版本为4.0以上的Android系统上，你可以像如下使用SYSTEM\_UI\_FLAG\_LOW\_PROFILE这个标签。

```
// This example uses decor view, but you can use any visible view.  
View decorView = getActivity().getWindow().getDecorView();  
int uiOptions = View.SYSTEM_UI_FLAG_LOW_PROFILE;  
decorView.setSystemUiVisibility(uiOptions);
```

一旦用户触摸到了状态栏或者是系统栏，这个标签就会被清除，使系统栏重新显示出来。在标签被清除的情况下，如果你想重新淡化系统栏就必须重新设定它。

图片1展示了一个图库中的图片，界面的系统栏都已被淡化（需要注意的是图库应用完全隐藏状态栏，而不是淡化它）；注意导航栏（图片的右侧）上变暗的白色的小点，他们代表了被隐藏的导航操作。



图1. 淡化的系统栏

图2展示的是同一张图片，系统栏处于显示的状态。

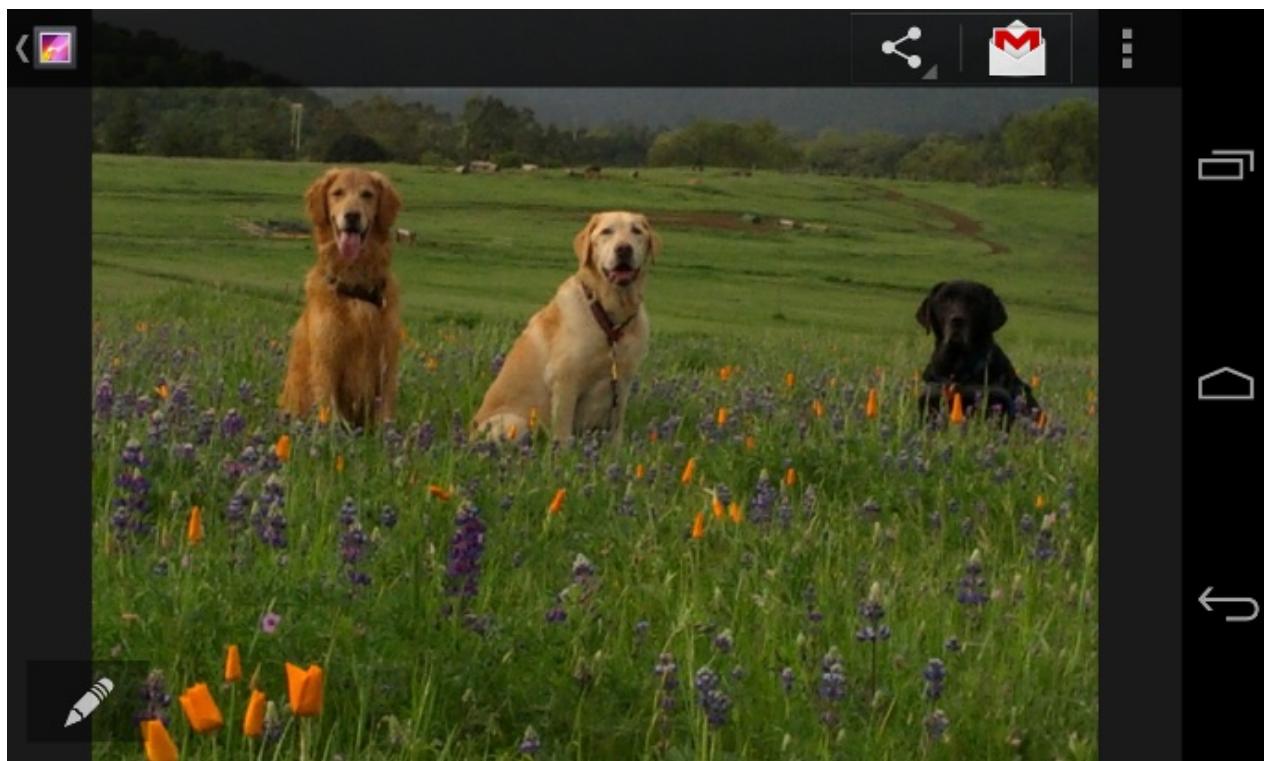


图2.显示的系统栏

## 显示状态栏与导航栏

如果你想动态的清除显示标签，你可以使用`setSystemUiVisibility()`方法：

```
View decorView = getActivity().getWindow().getDecorView();
// Calling setSystemUiVisibility() with a value of 0 clears
// all flags.
decorView.setSystemUiVisibility(0);
```

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/status.html>

# 隐藏系统Bar

这节课将教您

1. 在4.0及以下版本中隐藏状态栏
2. 在4.1及以上版本中隐藏状态栏
3. 在4.4及以上版本中隐藏状态栏
4. 让内容显示在状态栏之后
5. 同步状态栏与Action Bar的变化

同时您应该阅读

[ActionBar API指南](#)

[Android设计指南](#)

本课程将教您如何在不同版本的Android下隐藏状态栏。隐藏状态栏（或者是导航栏）可以让内容得到更多的展示空间，从而提供一个更加沉浸式的用户体验。

图1展示了显示状态栏的界面



图2展示了隐藏状态栏的界面。请注意，ActionBar这个时候也被隐藏了。请永远不要在隐藏状态栏的时候显示ActionBar。



## 在4.0及以下版本中隐藏状态栏

在Android 4.0及更低的版本中，你可以通过设置WindowManager来隐藏状态栏。你可以动态的隐藏，或者在你的manifest文件中设置activity的主题。如果你的应用的状态栏在运行过程中会一直隐藏，那么推荐你使用改写manifest设定主题的方法（严格上来讲，即便设置了manifest你也可以动态的改变主题）。

```
<application
    ...
    android:theme="@android:style/Theme.Holo.NoActionBar.Fullscreen"
    ...
</application>
```

设置主题的优势是：

- 易于维护，切不容易出错
- 有更流畅的UI转换，因为在初始化你的Activity之前，系统已经得到了需要渲染UI的信息

另一方面我们可以选择使用WindowManager来动态隐藏状态栏。这个方法可以更简单的在用户与App进行交互式展示与隐藏状态栏。

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // If the Android version is lower than Jellybean, use this
        // the status bar.
        if (Build.VERSION.SDK_INT < 16) {
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_LAYOUT_IN_SCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        }
        setContentView(R.layout.activity_main);
    }
    ...
}
```

当你设置WindowManager标签之后（无论是通过activity主题还是动态设置），这个标签就会一直生效直到你清除它。

设置了FLAG\_LAYOUT\_IN\_SCREEN之后，你可以使用与启用FLAG\_FULLSCREEN后相同的屏幕区域。这个方法防止了状态栏隐藏和展示的时候内容区域的大小变化。

## 在4.1及以上版本中隐藏状态栏

在Android 4.1(API level 16)以及更高的版本中，你可以使用`setSystemUiVisibility()`来进行动态隐藏。`setSystemUiVisibility()`在View层面设置了UI的标签，然后这些设置被整合到了Window层面。`setSystemUiVisibility()`给了你一个比设置 WindowManager 标签更加粒度化的操作。下面的代码隐藏了状态栏：

```
View decorView = getWindow().getDecorView();
// Hide the status bar.
int uiOptions = View.SYSTEM_UI_FLAG_FULLSCREEN;
decorView.setSystemUiVisibility(uiOptions);
// Remember that you should never show the action bar if the
// status bar is hidden, so hide that too if necessary.
ActionBar actionBar = getActionBar();
actionBar.hide();
```

注意以下几点：

- 一旦UI标签被清除(比如跳转到另一个activity),如果你还想隐藏状态栏你就必须再次设定它。详细可以看第五节如何监听并响应UI可见性的变化。
- 在不同的地方设置UI标签是有所区别的。如果你在activity的`onCreate()`方法中隐藏系统栏，当用户按下home键系统栏就会重新显示。当用户再重新打开activity的时候，`onCreate()`不会被调用，所以系统栏还会保持可见。如果你想让在不同activity之间切换时，系统UI保持不变，你需要在`onReasume()`与`onWindowFocusChaned()`里设定UI标签。
- `setSystemUiVisibility()`仅仅在被调用的View显示的时候才会生效。
- 当从View导航到别的地方时，用`setSystemUiVisibility()`设置的标签会被清除。

## 让内容显示在状态栏之后

在Android 4.1及以上版本，你可以将应用的内容显示在状态栏之后，这样当状态栏显示与隐藏的时候，内容区域的大小就不会发生变化。要做到这个效果，我们需要用到SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN这个标志。同时，你也有可能需要SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE这个标志来帮助你的应用维持一个稳定的布局。

当你是用这种方法的时候，你就需要来确保应用中特定区域不会被系统栏掩盖（比如地图应用中一些自带的操作区域）。如果被覆盖了应用就会无法使用。在大多数的情况下，你可以在布局文件中添加`android:fitsSystemWindows`标签，设置它为true。它会调整父ViewGroup使它留出特定区域给系统栏，对于大多数应用这种方法就足够了。

在一些情况下，你可能需要修改默认的padding大小来获取合适的布局。为了控制内容区域的布局相对系统栏（它占据了一个叫做“内容嵌入”content insets的区域）的位置，你可以重写`fitSystemWindows(Rect insets)`方法。当窗口的内容嵌入区域发生变化时，`fitSystemWindows()`方法会被view的hierarchy调用，让View做出相应的调整适应。重写这个方法你就可以按你的意愿处理嵌入区域与应用的布局。

## 同步状态栏与Action Bar的变化

在Android 4.1及以上的版本，为了防止在Action Bar隐藏和显示的时候布局发生变化，你可以使用Action Bar的overlay模式。在Overlay模式中，Activity的布局占据了所有可能的空间，好像Action Bar不存在一样，系统会在布局的上方绘制ActionBar。虽然这会遮盖住上方的一些布局，但是当Action Bar显示或者隐藏的时候，系统就不需要重新改变布局区域的大小，使之无缝的变化。

要启用Action Bar的overlay模式，你需要创建一个继承自Action Bar主题的自定义主题，将`android:windowActionBarOverlay`属性设置为true。要了解详细信息，请参考[添加Action Bar](#)课程中的Action Bar的覆盖层叠。

设置`SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN`来让你的activity使用的屏幕区域与设置`SYSTEM_UI_FLAG_FULLSCREEN`时的区域相同。当你需要隐藏系统UI时，使用`SYSTEM_UI_FLAG_FULLSCREEN`。这个操作也同时隐藏了Action Bar（因为`windowActionBarOverlay="true"`），当同时显示与隐藏ActionBar与状态栏的时候，使用一个动画来让他们相互协调。

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/navigation.html>

# 隐藏导航Bar

这节课将教您

1. 在4.0及以上版本中隐藏导航栏
2. 让内容显示在导航栏之后

本节课程将教您如何对导航栏进行隐藏，这个特性是Android 4.0版本中引入的。

即便本课程关注如何隐藏导航栏，但是在实际的开发中，你最好让状态栏与导航栏同时消失。在保证导航栏易于再次访问的情况下，隐藏导航栏与状态栏使内容区域占据了整个显示空间，因此提供了一个更加沉浸式的用户体验。



## 在4.0及以上版本中隐藏导航栏

你可以在Android 4.0以及以上版本，使用SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION标志来隐藏导航栏。代码如下：

```
View decorView = getWindow().getDecorView();
// Hide both the navigation bar and the status bar.
// SYSTEM_UI_FLAG_FULLSCREEN is only available on Android 4.1 and
// a general rule, you should design your app to hide the status bar
// hide the navigation bar.
int uiOptions = View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
               | View.SYSTEM_UI_FLAG_FULLSCREEN;
decorView.setSystemUiVisibility(uiOptions);
```

注意以下几点

- 使用这个方法时，触摸屏幕的任何一个区域都是使导航栏（与状态栏）重新显示。用户的交互使这个标志SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION被清除。
- 一旦这个标志被清除了，如果你想再次隐藏导航栏，你就需要重新对这个标志进行设定。在一节[响应UI可见性的变化](#)中，将详细讲解应用监听系统UI变化来做出相应的调整操作。
- 在不同的地方设置UI标签是有所区别的。如果你在activity的onCreate()方法中隐藏系统栏，当用户按下home键系统栏就会重新显示。当用户再重新打开activity的时候，onCreate()不会被调用，所以系统栏还会保持可见。如果你想让在不同activity之间切换时，系统UI保持不变，你需要在onReasume()与onWindowFocusChaned()里设定UI标签。
- setSystemUiVisibility()仅仅在被调用的View显示的时候才会生效。
- 当从View导航到别的地方时，用setSystemUiVisibility()设置的标签会被清除。

## 让内容显示在导航栏之后

在Android 4.1与更高的版本中，你可以让应用的内容显示在导航栏的后面，这样当导航栏展示或隐藏的时候内容区域就不会发生布局大小的变化。可以使  
用SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION标志来做到这个效果。同时，你也有  
可能需要SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE这个标志来帮助你的应用维持一个稳定的  
布局。

当你使用这种方法的时候，就需要你来确保应用中特定区域不会被系统栏掩盖。更详细的信息可以浏览[隐藏状态栏](#)一节。

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/immersive.html>

# 全屏沉浸式应用

这节课将教您

1. 确定一种方式
2. 使用非粘性沉浸模式
3. 使用粘性沉浸模式

Adnroid 4.4中引入为`setSystemUiVisibility()`引入了一个新标签`SYSTEM_UI_FLAG_IMMERSIVE`, 它可以让应用进入真正的全屏模式。当这个标签与`SYSTEM_UI_FLAG_HIDE_NAVIGATION`和`SYSTEM_UI_FLAG_FULLSCREEN`一起使用的时候, 导航栏和状态栏就会隐藏, 让你的应用可以接受屏幕上任何地方的触摸事件。

当沉浸式全屏模式启用的时候, 你的activity会继续接受各类的触摸事件。用户可以通过在边缘区域向内滑动来让系统栏重新显示。这个操作清空了`SYSTEM_UI_FLAG_HIDE_NAVIGATION`(和`SYSTEM_UI_FLAG_FULLSCREEN`, 如果有的话)两个标志, 因此系统栏重新变得可见。如果设置了的话, 这个操作同时也触发了`View.OnSystemUiVisibilityChangeListener`。然而, 如果你想让系统栏在一段时间后自动隐藏的话, 你应该使用`SYSTEM_UI_FLAG_IMMERSIVE_STICKY`标签。请注意, 'sticky'版本的标签不会触发任何的监听器, 因为在这个模式下展示的系统栏是处于暂时的状态。

图1展示了各种不同的“沉浸式”状态



在上图中:

1. 非沉浸模式——展示了应用进入沉浸模式之前的状态。也展示了设置`IMMERSIVE`标签后用户滑动展示系统栏的状态。用户滑动后, `SYSTEM_UI_FLAG_HIDE_NAVIGATION`和`SYSTEM_UI_FLAG_FULLSCREEN`就会被清除, 系统栏就会重新显示并保持可见。请注意, 最好的方式就是让所有的UI控件与系统栏的显示隐藏保持同步, 这样可以减少屏幕显示所处的状态, 同时提供了更无缝平滑的用户体验。因此所有的UI控件跟随系统栏一同显示。一旦应用进入了沉浸模式, UI控件也跟随着系统栏一同隐藏。为了确保UI的可见性与系统栏保持一致, 我们需要一个监听器`View.OnSystemUiVisibilityChangeListener`来监听系统栏的变化。这在

下一节中将详细讲解。

2. 提示气泡——第一次进入沉浸模式时，系统将会显示一个提示气泡，提示用户如何再让系统栏显示出来。请注意，如果为了测试你想强制显示提示气泡，你可以先将应用设为沉浸模式，然后按下电源键进入锁屏模式，并在5秒中之后打开屏幕。
3. 沉浸模式——这张图展示了隐藏了系统栏和其他UI控件的状态。你可以设置IMMERSIVE和IMMERSIVE\_STICKY来进入这个状态。
4. 粘性标签——这就是你设置了IMMERSIVE\_STICKY标签时的UI状态，用户会向内滑动以展示系统栏。半透明的系统栏会临时的进行显示，一段时间后自动隐藏。滑动的操作并不会清空任何标签，也不会触发系统UI可见性的监听器，因为暂时显示的导航栏并不被认为是一种可见的状态。

注意，immersive类的标签只有在

与SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION,SYSTEM\_UI\_FLAG\_FULLSCREEN中一个或两个一起使用的时候才会生效。你可以只使用其中的一个，但是一般情况下你需要同时隐藏状态栏和导航栏以达到沉浸的效果。

## 确定一种方式

SYSTEM\_UI\_FLAG\_IMMERSIVE与SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY都提供了沉浸式的体验，但是在上面的描述中，他们是不一样的，下面讲解一下什么时候该用哪一种标签。

- 如果你在写一款图书浏览器、新闻杂志阅读器，请将IMMERSIVE标签与SYSTEM\_UI\_FLAG\_FULLSCREEN,SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION一起使用。因为用户可能会经常访问Action Bar和一些UI控件，又不希望在翻页的时候有其他的东西进行干扰。IMMERSIVE在该种情况下就是个很好的选择。
- 如果你在打造一款真正的沉浸式应用，而且你希望屏幕边缘的区域也可以与用户进行交互，并且他们也不会经常访问系统UI。这个时候就要将IMMERSIVE\_STICKY和SYSTEM\_UI\_FLAG\_FULLSCREEN SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION两个标签一起使用。比如做一款游戏或者绘图应用就很合适。
- 如果你在打造一款视频播放器，并且需要少量的用户交互操作。你可能就需要之前版本的一些方法了（从Android 4.0开始）。对于这种应用，简单的使用SYSTEM\_UI\_FLAG\_FULLSCREEN与SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION就足够了，不需要使用immersive标签。

# 使用非粘性沉浸模式

当你使用SYSTEM\_UI\_FLAG\_IMMERSIVE标签的时候，它是基于其他设置过的标签(SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION和SYSTEM\_UI\_FLAG\_FULLSCREEN)来隐藏系统栏的。当用户向内滑动，系统栏重新显示并保持可见。

用其他的UI标签

(如SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION和SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE来防止系统栏隐藏时内容区域大小发生变化是一种很不错的方法。你也需要确保ActionBar和其他系统UI控件同时进行隐藏。下面这段代码展示了如何在不改变内容区域大小的情况下，隐藏与显示状态栏和导航栏。

```
// This snippet hides the system bars.  
private void hideSystemUI() {  
    // Set the IMMERSIVE flag.  
    // Set the content to appear under the system bars so that the  
    // doesn't resize when the system bars hide and show.  
    mDecorView.setSystemUiVisibility(  
        View.SYSTEM_UI_FLAG_LAYOUT_STABLE  
        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION  
        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN  
        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION // hide nav bar  
        | View.SYSTEM_UI_FLAG_FULLSCREEN // hide status bar  
        | View.SYSTEM_UI_FLAG_IMMERSIVE);  
}  
  
// This snippet shows the system bars. It does this by removing all  
// except for the ones that make the content appear under the system bars.  
private void showSystemUI() {  
    mDecorView.setSystemUiVisibility(  
        View.SYSTEM_UI_FLAG_LAYOUT_STABLE  
        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION  
        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);  
}
```

你可能同时也希望在如下的几种情况下使用IMMERSIVE标签来提供更好的用户体验：

- 注册一个监听器来监听系统UI的变化。
- 实现onWindowFocusChanged()函数。如果窗口获取了焦点，你可能需要对系统栏进行隐藏。如果窗口失去了焦点，比如说弹出了一个对话框或菜单，你可能需要取消那些将要在Handler.postDelayed()或其他地方的隐藏操作。
- 实现一个GestureDetector，它监听了onSingleTapUp(MotionEvent)事件。可以使用户点击内容区域来切换系统栏的显示状态。单纯的点击监听可能不是最好的解决方案，因为当用户在屏幕上拖动手指的时候（假设点击的内容占据了整个屏幕），这个事件也会被触发。

更多关于此话题的讨论，可以观看这个视频 [DevBytes: Android 4.4 Immersive Mode](#)

## 使用粘性沉浸模式

当使用了SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY标签的时候，向内滑动的操作会让系统栏临时显示，并处于半透明的状态。此时没有标签会被清除，系统UI可见性监听器也不会被触发。如果用户没有进行操作，系统栏会在一段时间内自动隐藏。

图2展示了当使用IMMERSIVE\_STICKY标签时，半透明的系统栏展示与又隐藏的状态。



下面是一段实现代码。一旦窗口获取了焦点，只要简单的设置IMMERSIVE\_STICKY与上面讨论过的其他标签即可。

```
@Override  
public void onWindowFocusChanged(boolean hasFocus) {  
    super.onWindowFocusChanged(hasFocus);  
    if (hasFocus) {  
        decorView.setSystemUiVisibility(  
            View.SYSTEM_UI_FLAG_LAYOUT_STABLE  
            | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION  
            | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN  
            | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION  
            | View.SYSTEM_UI_FLAG_FULLSCREEN  
            | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY);  
    }  
}
```

---

注意，如果你想实现IMMERSIVE\_STICKY的自动隐藏效果，同时也需要展示你自己的UI控件。你只需要使用IMMERSIVE与Handler.postDelayed()或其他类似的东西，让它几秒后重新进入沉浸模式即可。

编写:[KOST](#) - 校对

原文:<http://developer.android.com/training/system-ui/visibility.html>

# 响应UI可见性的变化

本节课将教你如何注册监听器来监听系统UI可见性的变化。这个方法在将系统栏与你自己的UI控件进行同步操作时很有用。

## 注册监听器

为了获取系统UI可见性变化的通知，我们需要对View注册View.OnSystemUiVisibilityChangeListener监听器。通常上来说，这个View是用来控制导航的可见性的。

例如你可以添加如下代码在onCreate中

```
View decorView = getWindow().getDecorView();
decorView.setOnSystemUiVisibilityChangeListener
    (new View.OnSystemUiVisibilityChangeListener() {
        @Override
        public void onSystemUiVisibilityChange(int visibility) {
            // Note that system bars will only be "visible" if none of
            // LOW_PROFILE, HIDE_NAVIGATION, or FULLSCREEN flags are set.
            if ((visibility & View.SYSTEM_UI_FLAG_FULLSCREEN) == 0) {
                // TODO: The system bars are visible. Make any desired
                // adjustments to your UI, such as showing the action
                // other navigational controls.
            } else {
                // TODO: The system bars are NOT visible. Make any desired
                // adjustments to your UI, such as hiding the action bar
                // other navigational controls.
            }
        }
    });
});
```

保持系统栏和UI同步是一种很好的方式，比如当状态栏显示或隐藏的时候进行Action Bar的显示和隐藏。

编写:[kesenhoo](#)

校对:

# 用户输入

These classes cover various subjects of user input, such as touch screen gestures and text input through on-screen input methods and hardware keyboards.

## [Using Touch Gestures](#)

How to write apps that allow users to interact with the touch screen via touch gestures.

## [Handling Keyboard Input](#)

How to specify the appearance and behaviors of soft input methods (such as on-screen keyboards) and how to optimize the experience with hardware keyboards.

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/index.html>

# 使用触摸手势

这一章节讲述，如何编写允许用户通过触摸手势进行交互的app。Android提供了多种API帮你创建和检测手势。

尽管你的app不应该依赖于触摸手势来完成基本操作（因为某些情况下手势是不用的），但为你的app添加基于触摸的交互，将会大大地提高app的可用性以及吸引力。

为了给用户提供一致、直觉性的使用体验，你的app应该遵守Android触摸手势的惯常做法。[手势设计指南](#)展示了Android app中的常用手势。同样，设计指南也提供了[触摸反馈](#)的相关内容。

# Lessons

- [检测常用的手势](#)

学习如何使用[GestureDetector](#)检测基本的触摸手势,如滑动,惯性滑动以及双击。

- [追踪手势移动](#)

学习如何追踪手势移动。

- [Scroll手势动画](#)

学习如何使用scrollers ([Scrollers](#)以及[OverScroll](#)) 来产生滚动动画以响应触摸事件。

- [处理多触摸手势](#)

学习如何检测多点(手指)触摸手势。

- [拖拽与缩放](#)

学习如何实现基于触摸的拖拽与缩放。

- [管理ViewGroup中的触摸事件](#)

学习如何管理[ViewGroup](#)中的触摸事件, 以确保事件能被正确地分发到目标views。

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/detector.html>

# 检测常用的手势

当用户用一根或多根手指触碰屏幕时，一个“触摸手势”就发生了，并且你的应用可把这样的触摸方式解释成一种特定的手势。手势检测有以下两个相应的阶段：

1. 收集触摸事件的相关数据。
2. 分析这些数据，看它们是否满足任何一种你的app所支持的手势的标准。

## 支持库中的类

本节课程的示例程序使用的是[GestureDetectorCompat](#)类和[MotionEventCompat](#)类。这些类都在[Support Library](#)中。你可以通过使用支持库中的类，来为运行着Android1.6及以上系统的设备提供兼容性功能。需要注意的一点是，[MotionEventCompat](#)类不是[MotionEvent](#)类的替代品，而是提供了一些静态工具类函数，你可以把[MotionEvent](#)对象作为参数传递给这些函数，从而得到与事件相关的动作(action)。

## 收集数据

当用户用一根或多根手指触碰屏幕时，接收触摸事件的View的[onTouchEvent\(\)](#)函数就会被回调。对于一系列连续的触摸事件（位置、压力、大小、额外的一根手指等等），[onTouchEvent\(\)](#)会被调用若干次，并且最终识别为一种手势。

手势开始于用户刚触摸屏幕时，其后系统会持续地追踪用户手指的位置，用户手指都离开屏幕时手势结束。在整个交互期间，[MotionEvent](#)被分发给[onTouchEvent\(\)](#)函数，来提供每次交互的详细信息。你的app可以使用[MotionEvent](#)提供的数据，来判断是否发生了某种特定的手势。

### 为Activity或View捕获触摸事件

为了捕获Activity或View中的触摸事件，你可以重写[onTouchEvent\(\)](#)回调函数。

接下来的代码段中使用了[getActionMasked\(\)](#)函数，该函数可以从event参数中获得用户执行的动作。它提供了一些触摸的原始数据，你可以使用这些数据，来判断是否发生了某个特定手势。

```
public class MainActivity extends Activity {  
    ...  
    // This example shows an Activity, but you would use the same approach  
    // you were subclassing a View.  
    @Override  
    public boolean onTouchEvent(MotionEvent event) {  
  
        int action = MotionEventCompat.getActionMasked(event);  
  
        switch(action) {  
            case (MotionEvent.ACTION_DOWN) :  
                Log.d(DEBUG_TAG, "Action was DOWN");  
                return true;  
            case (MotionEvent.ACTION_MOVE) :  
                Log.d(DEBUG_TAG, "Action was MOVE");  
                return true;  
            case (MotionEvent.ACTION_UP) :  
                Log.d(DEBUG_TAG, "Action was UP");  
                return true;  
            case (MotionEvent.ACTION_CANCEL) :  
                Log.d(DEBUG_TAG, "Action was CANCEL");  
                return true;  
            case (MotionEvent.ACTION_OUTSIDE) :  
                Log.d(DEBUG_TAG, "Movement occurred outside bounds  
                      of current screen element");  
                return true;  
            default :  
                return super.onTouchEvent(event);  
        }  
    }  
}
```

然后，你可以自行处理这些事件，来判断是否出现了某个手势。当你需要检测自定义手势时，你可以使用这种方式。然而，如果你的app仅仅需要使用一些常见的手势，如双击，

长按，快速滑动（fling）等，你可以使用[GestureDetector](#)类来完成。[GestureDetector](#)可以让你更简单地检测常见手势，并且无需自行处理单个的触摸事件。相关内容将会在下面的[Detect Gestures](#)中讨论。

## 捕获单个view对象的触摸事件

除了使用[onTouchEvent\(\)](#)来捕获触摸事件，你也可以使用[setOnTouchListener\(\)](#)函数给任意[View](#)对象关联一个[View.OnTouchListener](#)对象来捕获触摸事件。这样做可以让你不继承已有的[View](#)，也能监听它的触摸事件。比如：

```
View myView = findViewById(R.id.my_view);
myView.setOnTouchListener(new OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        // ... Respond to touch events
        return true;
    }
});
```

创建listener对象时，谨防对[ACTION\\_DOWN](#)事件返回false。如果返回false，会导致listener对象监听不到后续的[ACTION\\_MOVE](#)、[ACTION\\_UP](#)等系列事件。这是因为[ACTION\\_DOWN](#)事件是所有触摸事件的开端。

如果你正在写一个自定义View，你也可以像上面描述的那样重写[onTouchEvent\(\)](#)函数。

# 检测手势

Android提供了[GestureDetector](#)类来检测一般手势。它支持的手势包括[onDown\(\)](#), [onLongPress\(\)](#),[onFling\(\)](#)等。你可以把[GestureDetector](#)和上面描述的[onTouchEvent\(\)](#)函数结合在一起使用。

## 检测所有支持的手势

当你实例化一个[GestureDetectorCompat](#)对象时，需要一个实现了[GestureDetector.OnGestureListener](#)接口的对象作为参数。当某个特定的触摸事件发生时，[GestureDetector.OnGestureListener](#)就会通知用户。为了让你的[GestureDetector](#)对象能到接收到触摸事件，你需要重写View或Activity的[onTouchEvent\(\)](#)函数，并且把所有捕获到的事件传递给detector对象。

接下来的代码段中，on型的函数返回值是true意味着你已经处理完这个触摸事件了。如果返回false，则会把事件沿view栈传递，直到触摸事件被成功地处理了。

运行下面的代码段，来了解你与触摸屏交互时动作（action）是如何触发的，以及每个触摸事件[MotionEvent](#)中的内容。你也会了解到一个简单的交互会产生多少的数据。

```
public class MainActivity extends Activity implements
    GestureDetector.OnGestureListener,
    GestureDetector.OnDoubleTapListener {

    private static final String DEBUG_TAG = "Gestures";
    private GestureDetectorCompat mDetector;

    // Called when the activity is first created.
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Instantiate the gesture detector with the
        // application context and an implementation of
        // GestureDetector.OnGestureListener
        mDetector = new GestureDetectorCompat(this, this);
        // Set the gesture detector as the double tap
        // listener.
        mDetector.setOnDoubleTapListener(this);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        this.mDetector.onTouchEvent(event);
        // Be sure to call the superclass implementation
        return super.onTouchEvent(event);
    }

    @Override
    public boolean onDown(MotionEvent event) {
        Log.d(DEBUG_TAG, "onDown: " + event.toString());
        return true;
    }
}
```

```
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
                           float velocityX, float velocityY) {
        Log.d(DEBUG_TAG, "onFling: " + event1.toString() + event2.toString());
        return true;
    }

    @Override
    public void onLongPress(MotionEvent event) {
        Log.d(DEBUG_TAG, "onLongPress: " + event.toString());
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceY) {
        Log.d(DEBUG_TAG, "onScroll: " + e1.toString() + e2.toString());
        return true;
    }

    @Override
    public void onShowPress(MotionEvent event) {
        Log.d(DEBUG_TAG, "onShowPress: " + event.toString());
    }

    @Override
    public boolean onSingleTapUp(MotionEvent event) {
        Log.d(DEBUG_TAG, "onSingleTapUp: " + event.toString());
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent event) {
        Log.d(DEBUG_TAG, "onDoubleTap: " + event.toString());
        return true;
    }

    @Override
    public boolean onDoubleTapEvent(MotionEvent event) {
        Log.d(DEBUG_TAG, "onDoubleTapEvent: " + event.toString());
        return true;
    }

    @Override
    public boolean onSingleTapConfirmed(MotionEvent event) {
        Log.d(DEBUG_TAG, "onSingleTapConfirmed: " + event.toString());
        return true;
    }
}
```

## 检测支持的部分手势

如果你仅仅只想处理几种手势，你可以选择继承[GestureDetector.SimpleOnGestureListener](#)类，而不是实

现GestureDetector.OnGestureListener接口。

GestureDetector.SimpleOnGestureListener类实现了所有的on型函数，并且都返回false。因此，你可以仅仅重写你所需要的函数。比如，下面的代码段中创建了一个继承GestureDetector.SimpleOnGestureListener的类，并且只重写了onFling()和onDown()函数。

无论你是否使用GestureDetector.OnGestureListener类，最好都实现onDown()函数并且返回true。这是因为所有的手势都是由onDown()消息开始的。如果你让onDown()函数返回false，就像GestureDetector.SimpleOnGestureListener类默认的那样，系统会假定你想忽略手势的剩余部分，GestureDetector.OnGestureListener中的其他函数也就永远不会被调用。这可能让你的app出现意想不到的问题。仅仅当你真的想忽略整个手势时，你才应该让onDown()函数返回false。

```
public class MainActivity extends Activity {

    private GestureDetectorCompat mDetector;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mDetector = new GestureDetectorCompat(this, new MyGestureL
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        this.mDetector.onTouchEvent(event);
        return super.onTouchEvent(event);
    }

    class MyGestureListener extends GestureDetector.SimpleOnGestur
        private static final String DEBUG_TAG = "Gestures";

        @Override
        public boolean onDown(MotionEvent event) {
            Log.d(DEBUG_TAG, "onDown: " + event.toString());
            return true;
        }

        @Override
        public boolean onFling(MotionEvent event1, MotionEvent eve
            float velocityX, float velocityY) {
            Log.d(DEBUG_TAG, "onFling: " + event1.toString() + event
            return true;
        }
    }
}
```

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/movement.html>

# 追踪手势移动

本节课程讲述如何追踪手势移动。

每当触摸位置、压力、大小发生变化时，[onTouchEvent\(\)](#)函数都会随着新的[ACTION\\_MOVE](#)事件参数被重新调用一次。正如[检测常用的手势](#)中描述的那样，触摸事件全部都记录在[onTouchEvent\(\)函数](#)的[MotionEvent](#)参数中。

因为基于手指的触摸的交互方式并不总是非常精确，所以检测触摸事件更多的是基于手势移动而非简单地触摸。为了帮助app区分基于移动的手势（如滑动）和非移动手势（如简单地点击），Android引入了“touch slop”的概念。Touch slop是指用户触摸事件在可被识别为移动手势前，移动过的那一段像素距离。关于这一主题的更多讨论，可以在[管理ViewGroup中的触摸事件](#)中查看。

根据你的app的需求，有多种追踪手势移动的方式可以选择。比如：

- 追踪手指的起始和终止位置（比如，把屏幕上的对象从A点移动到B点）
- 根据x、y轴坐标，追踪手指移动的方向。
- 追踪历史状态。你可以通过调用[MotionEvent](#)的[getHistorySize\(\)](#)函数获得一个手势的历史尺寸大小。你可以通过移动事件的[getHistorical<Value>](#)系列函数获得事件之前的位置、尺寸、时间以及按压力(pressures)。当你需要绘制用户手指痕迹时，历史状态非常有用，比如触摸绘图。查看[MotionEvent](#)来了解更多细节。
- 追踪手指在触摸屏上滑过的速度。

## 追踪速度

你可以让移动手势简单地基于手指滑动过的距离或(和)方向。但是速度经常也是追踪手势特性的一个决定性因素，甚至是判断一个手势是否发生的依据。为了让速度计算更容易，Android提供了[VelocityTracker](#)类以及[支持库](#)中的[VelocityTrackerCompat](#)类。[VelocityTracker](#)类可以帮助你追踪触摸事件中的速度因素。如果速度是你的手势的一个判断标准，比如快速滑动(fling)，那么这些类是很有用的。

下面是一个简单的例子，说明了[VelocityTracker](#)中API函数的用处。

```
public class MainActivity extends Activity {
    private static final String DEBUG_TAG = "Velocity";
    ...
    private VelocityTracker mVelocityTracker = null;
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int index = event.getActionIndex();
        int action = event.getActionMasked();
        int pointerId = event.getPointerId(index);

        switch(action) {
            case MotionEvent.ACTION_DOWN:
                if(mVelocityTracker == null) {
                    // Retrieve a new VelocityTracker object to work with.
                    mVelocityTracker = VelocityTracker.obtain();
                }
                else {
                    // Reset the velocity tracker back to its initial values.
                    mVelocityTracker.clear();
                }
                // Add a user's movement to the tracker.
                mVelocityTracker.addMovement(event);
                break;
            case MotionEvent.ACTION_MOVE:
                mVelocityTracker.addMovement(event);
                // When you want to determine the velocity, call
                // computeCurrentVelocity(). Then call getXVelocity()
                // and getYVelocity() to retrieve the velocity for
                mVelocityTracker.computeCurrentVelocity(1000);
                // Log velocity of pixels per second
                // Best practice to use VelocityTrackerCompat when
                Log.d("MainActivity", "X velocity: " +
                    VelocityTrackerCompat.getXVelocity(mVelocityTracker,
                        pointerId));
                Log.d("MainActivity", "Y velocity: " +
                    VelocityTrackerCompat.getYVelocity(mVelocityTracker,
                        pointerId));
                break;
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                // Return a VelocityTracker object back to be reused.
                mVelocityTracker.recycle();
                break;
        }
    }
}
```

```
        }
        return true;
    }
}
```

注意：需要注意的是，你应该在ACTION\_MOVE事件后计算速度，而不是在ACTION\_UP事件后。在ACTION\_UP事件之后计算，x、y方向的速度都会是0。

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/scroll.html>

# Scroll手势动画

Android中通常使用[ScrollView](#)类来实现滚动（scroll）。任何可能超过父类边界的布局都应该嵌套在一个[ScrollView](#)中，以提供一个由系统框架管理的可滚动的view。仅仅在某些特殊情形下，才需要实现一个自定义scroller。本节课程描述了这样一个情形：使用scrollers显示滚动效果来响应触摸手势。

你可以使用scrollers([Scroller](#)或者[OverScroller](#))收集数据，这些数据可用来产生滚动动画以响应一个触摸事件。这两个类很相似，但是[OverScroller](#)有一些函数，在平移或惯性滑动手势后，能向用户指出他们已经达到内容尽头了。[InteractiveChart](#)例子使用了[EdgeEffect](#)类（实际上是[EdgeEffectCompat](#)类），用来在用户到达内容尽头时显示发光效果。

注意：比起Scroller类，我们更推荐使用[OverScroller](#)类来产生滚动动画。[OverScroller](#)类为老设备提供了很好的向后兼容性。另外需要注意的是，当你自己实现滚动时，通常只需要使用scrollers。如果你把布局嵌套在[ScrollView](#)和[HorizontalScrollView](#)中，它们会帮你把这些做好。

通过使用平台标准的滚动物理定律（摩擦、速度等），scroller可随着时间产生滚动动画。实际上，scroller本身不会绘制任何东西。Scrollers只是随着时间的推移帮你追踪滚动的偏移量，但它们不会自动地把这些位置应用到你的view上。你需要以某种让你的滚动动画更流畅的速度，来获取并使用新的坐标。

## 理解术语Scrolling

在Android中，“Scrolling”这个词根据不同情景有着不同的含义。

**Scrolling**是指视窗（viewport）（指你正在看的内容所在的‘窗口’）移动的一般过程。当朝x轴和y轴方向滚动时，就叫做平移。示例程序提供的**InteractiveChart**类，展示了两种不同类型的scrolling，即拖拽与快速滑动。

- 拖拽(dragging)是scrolling的一种类型，发生在用户在触摸屏上拖拽手指时。通常可以重写[GestureDetector.OnGestureListener的onScroll\(\)](#)函数来简单地处理拖拽。关于拖拽的更多讨论，可以查看[拖拽与缩放](#)章节。
- 快速滑动(fling)这种类型的scrolling，发生在用户快速拖拽并抬高手指时。当用户抬高手指后，你通常想继续保持scrolling(移动视窗)，但是会保持减速直到视窗停止移动。可以重写[GestureDetector.OnGestureListener的onFling\(\)](#)函数来实现快速滑动的处理。这也是本节课程的做法。

虽然经常会把使用scroller对象与快速滑动手势结合起来，但在任何你想让UI展示scrolling动画来响应触摸事件的地方，他们都可以被拿来使用。比如，你可以重写[onTouchEvent\(\)](#)函数，来直接处理触摸事件，并且产生一个scrolling效果或“对齐到页”动画(snapping to page)来响应这些触摸事件。

## 实现基于触摸的Scrolling

本节讲述如何使用一个scroller。下面的代码段来自InteractiveChart样例的类中。它使用了[GestureDetector][GestureDetector\_url]，并且重写了[GestureDetector.SimpleOnGestureListener](#)的[onFling\(\)](#)函数。它使用[OverScroller](#)来追踪快速滑动手势。在快速滑动手势完成后，如果用户到达内容尽头，应用会显示发光的效果。

注意：InteractiveChart样例程序展示了一个可缩放、平移、滑动的表格。在接下来的代码段中，**mContentRect**表示view中的一块方形坐标区域，该区域将被用来绘制表格。在任意给定的时间点，整个表格都有某一部分会被绘制在这个区域内。**mCurrentViewport**表示表格中当前在屏幕上可见的那一部分。因为像素偏移量通常当作整型处理，所以**mContentRect**是**Rect**类型的。因为图表的区域范围是数值型/浮点型值，所以**mCurrentViewport**是**RectF**类型的。

代码段的第一部分展示了[onFling\(\)](#)函数的实现：

```
//当前视窗（viewport）。这个矩形表示图表当前的可视区域范围。  
//视窗是app中用户可通过触摸手势操作的那部分。  
private RectF mCurrentViewport =  
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);  
  
// The current destination rectangle (in pixel coordinates) into which  
// chart data should be drawn.  
private Rect mContentRect;  
  
private OverScroller mScroller;  
private RectF mScrollerStartViewport;  
...  
private final GestureDetector.SimpleOnGestureListener mGestureList  
    = new GestureDetector.SimpleOnGestureListener() {  
        @Override  
        public boolean onDown(MotionEvent e) {  
            // Initiates the decay phase of any active edge effects.  
            releaseEdgeEffects();  
            mScrollerStartViewport.set(mCurrentViewport);  
            // Aborts any active scroll animations and invalidates.  
            mScroller.forceFinished(true);  
            ViewCompat.postInvalidateOnAnimation(InteractiveLineGraphV  
                return true;  
        }  
        ...  
        @Override  
        public boolean onFling(MotionEvent e1, MotionEvent e2,  
            float velocityX, float velocityY) {  
            fling((int) -velocityX, (int) -velocityY);  
            return true;  
        }  
    };  
  
private void fling(int velocityX, int velocityY) {
```

```

// Initiates the decay phase of any active edge effects.
releaseEdgeEffects();
// Flings use math in pixels (as opposed to math based on the
Point surfaceSize = computeScrollSurfaceSize();
mScrollerStartViewport.set(mCurrentViewport);
int startX = (int) (surfaceSize.x * (mScrollerStartViewport.le
    AXIS_X_MIN) / (
    AXIS_X_MAX - AXIS_X_MIN));
int startY = (int) (surfaceSize.y * (AXIS_Y_MAX -
    mScrollerStartViewport.bottom) / (
    AXIS_Y_MAX - AXIS_Y_MIN));
// Before flinging, aborts the current animation.
mScroller.forceFinished(true);
// Begins the animation
mScroller.fling(
    // Current scroll position
    startX,
    startY,
    velocityX,
    velocityY,
    /*
     * Minimum and maximum scroll positions. The minimum s
     * position is generally zero and the maximum scroll p
     * is generally the content size less the screen size.
     * content width is 1000 pixels and the screen width i
     * pixels, the maximum scroll offset should be 800 pix
     */
    0, surfaceSize.x - mContentRect.width(),
    0, surfaceSize.y - mContentRect.height(),
    // The edges of the content. This comes into play when
    // the EdgeEffect class to draw "glow" overlays.
    mContentRect.width() / 2,
    mContentRect.height() / 2);
// Invalidates to trigger computeScroll()
ViewCompat.postInvalidateOnAnimation(this);
}

```

当[onFling\(\)](#)函数调用[postInvalidateOnAnimation\(\)](#)时，它会触发[computeScroll\(\)](#)来更新x、y的值。通常一个子view用scroller对象来产生滚动动画时会这样做，就如上面的例子一样。

大多数views直接通过[scrollTo\(\)](#)函数传递scroller对象的x、y坐标值。接下来的[computeScroll\(\)](#)函数的实现采用了一种不同的方式。它调用[computeScrollOffset\(\)](#)函数来获得当前位置的x、y值。当满足边缘显示发光效果的条件时（图表已被放大显示，x或y值超过边界，并且app当前没有显示overscroll），这段代码会设置overscroll发光效果，并调用[postInvalidateOnAnimation\(\)](#)函数来让view失效重绘：

```

// Edge effect / overscroll tracking objects.
private EdgeEffectCompat mEdgeEffectTop;
private EdgeEffectCompat mEdgeEffectBottom;
private EdgeEffectCompat mEdgeEffectLeft;
private EdgeEffectCompat mEdgeEffectRight;

private boolean mEdgeEffectTopActive;

```

```
private boolean mEdgeEffectBottomActive;
private boolean mEdgeEffectLeftActive;
private boolean mEdgeEffectRightActive;

@Override
public void computeScroll() {
    super.computeScroll();

    boolean needsInvalidate = false;

    // The scroller isn't finished, meaning a fling or programmatic
    // operation is currently active.
    if (mScroller.computeScrollOffset()) {
        Point surfaceSize = computeScrollSurfaceSize();
        int currX = mScroller.getCurrX();
        int currY = mScroller.getCurrY();

        boolean canScrollX = (mCurrentViewport.left > AXIS_X_MIN
                              || mCurrentViewport.right < AXIS_X_MAX);
        boolean canScrollY = (mCurrentViewport.top > AXIS_Y_MIN
                              || mCurrentViewport.bottom < AXIS_Y_MAX);

        /*
         * If you are zoomed in and currX or currY is
         * outside of bounds and you're not already
         * showing overscroll, then render the overscroll
         * glow edge effect.
         */
        if (canScrollX
            && currX < 0
            && mEdgeEffectLeft.isFinished()
            && !mEdgeEffectLeftActive) {
            mEdgeEffectLeft.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller))
            mEdgeEffectLeftActive = true;
            needsInvalidate = true;
        } else if (canScrollX
            && currX > (surfaceSize.x - mContentRect.width())
            && mEdgeEffectRight.isFinished()
            && !mEdgeEffectRightActive) {
            mEdgeEffectRight.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller))
            mEdgeEffectRightActive = true;
            needsInvalidate = true;
        }

        if (canScrollY
            && currY < 0
            && mEdgeEffectTop.isFinished()
            && !mEdgeEffectTopActive) {
            mEdgeEffectTop.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller))
            mEdgeEffectTopActive = true;
            needsInvalidate = true;
        } else if (canScrollY
```

```

        && currY > (surfaceSize.y - mContentRect.height())
        && mEdgeEffectBottom.isFinished()
        && !mEdgeEffectBottomActive) {
    mEdgeEffectBottom.onAbsorb((int)
        OverScrollerCompat.getCurVelocity(mScroller))
    mEdgeEffectBottomActive = true;
    needsInvalidate = true;
}
...
}

```

这是缩放部分的代码：

```

// Custom object that is functionally similar to Scroller
Zoomer mZoomer;
private PointF mZoomFocalPoint = new PointF();
...

// If a zoom is in progress (either programmatically or via double
// touch), performs the zoom.
if (mZoomer.computeZoom()) {
    float newWidth = (1f - mZoomer.getCurrZoom()) *
        mScrollerStartViewport.width();
    float newHeight = (1f - mZoomer.getCurrZoom()) *
        mScrollerStartViewport.height();
    float pointWithinViewportX = (mZoomFocalPoint.x -
        mScrollerStartViewport.left)
        / mScrollerStartViewport.width();
    float pointWithinViewportY = (mZoomFocalPoint.y -
        mScrollerStartViewport.top)
        / mScrollerStartViewport.height();
    mCurrentViewport.set(
        mZoomFocalPoint.x - newWidth * pointWithinViewportX,
        mZoomFocalPoint.y - newHeight * pointWithinViewportY,
        mZoomFocalPoint.x + newWidth * (1 - pointWithinViewportX),
        mZoomFocalPoint.y + newHeight * (1 - pointWithinViewportY));
    constrainViewport();
    needsInvalidate = true;
}
if (needsInvalidate) {
    ViewCompat.postInvalidateOnAnimation(this);
}

```

这是上面代码段中调用过的**computeScrollSurfaceSize()**函数。他会计算当前可滚动部分的尺寸，以像素为单位。举例来说，如果整个图表区域都是可见的，它的值就简单地等于**mContentRect**的大小。如果图表两个方向上都放大到200%，此函数返回的尺寸在水平、垂直方向上都会大两倍。

```

private Point computeScrollSurfaceSize() {
    return new Point(
        (int) (mContentRect.width() * (AXIS_X_MAX - AXIS_X_MIN)
            / mCurrentViewport.width())),

```

```
        (int) (mContentRect.height() * (AXIS_Y_MAX - AXIS_Y_MIN  
            / mCurrentViewport.height())));
}
```

查看[ViewPager类的源代码](#)，可以发现另一个关于scroller的用法示例。它用滚动来响应flings，使用scrolling来实现“对齐到页”(snapping to page)动画。

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/multi.html>

# 处理多触摸手势

多点触摸触手势是指在同一时间有多点（手指）触碰屏幕。本节课程讲述如何检测多点触摸手势。

## 追踪多点

当多个手指同时触摸屏幕时，系统会产生如下的触摸事件：

- [ACTION\\_DOWN](#)-给触摸屏幕的第一个点。此事件是手势的开端。第一触摸点的数据在[MotionEvent](#)中的索引总是0。
- [ACTION\\_POINTER\\_DOWN](#)-给除第一点外出现在屏幕上的额外的点。这个点的数据在[MotionEvent](#)中的索引，可以通过[getActionIndex\(\)](#)函数获得。
- [ACTION\\_MOVE](#)-按压手势时发生变化。
- [ACTION\\_POINTER\\_UP](#)-当非第一点离开屏幕时发送此消息。
- [ACTION\\_UP](#)-当最后一点离开屏幕时发送此消息。

你可以通过每个点的索引以及id，单独地追踪[MotionEvent](#)中的每个点。

- Index: [MotionEvent](#)把每个点的信息都存储在一个数组中。点在数组中的位置就是该点的索引值。大多数用来与点交互的[MotionEvent](#)函数都是以索引值作为参数的，而不是点的ID。
- ID: 每个点也都对应提供了一个ID，该ID在整个手势期间一直存在，以便你单独地追踪每个点。

每个独立的点在移动事件中出现的次序是不固定的。因此，从一个事件到另一个事件，点的索引值是可以改变的，但点的ID在它的生命周期内是保证不会改变的。使用[getPointerId\(\)](#)可以获得一个点的ID，在手势随后的移动事件中，就可以用该ID来追踪这个点。对于随后一系列的事件，可以使用[findPointerIndex\(\)](#)函数，来获得对应给定ID的点在移动事件中的索引值。如下：

```
private int mActivePointerId;

public boolean onTouchEvent(MotionEvent event) {
    ...
    // Get the pointer ID
    mActivePointerId = event.getPointerId(0);

    // ... Many touch events later...

    // Use the pointer ID to find the index of the active pointer
    // and fetch its position
    int pointerIndex = event.findPointerIndex(mActivePointerId);
    // Get the pointer's current position
    float x = event.getX(pointerIndex);
    float y = event.getY(pointerIndex);
}
```

## 获取MotionEvent的动作

你应该始终使用[getActionMasked\(\)](#)函数（或者更好用[MotionEventCompat.getActionMasked\(\)](#)这个兼容版本）来获取[MotionEvent](#)的动作(action)。与旧的[getAction\(\)](#)函数不同的是，[getActionMasked\(\)](#)本就是设计用来处理多点触摸的。它会返回执行过的动作的掩码值，不包括点的索引位。你可以使用[getActionIndex\(\)](#)来获得与该动作关联的点的索引值。这在接下来的代码段中可以看到。

注意：这个样例使用的是[MotionEventCompat](#)类。这个类位于[Support Library](#)中。你应该使用[MotionEventCompat](#)类，来提供对更多平台的支持。需要注意的一点是，[MotionEventCompat](#)并不是[MotionEvent](#)类的替代品。准确来说，它提供了一些静态工具类函数，你可以把[MotionEvent](#)对象作为参数传递给这些函数，来得到与事件相关的动作。

```
int action = MotionEventCompat.getActionMasked(event);
// Get the index of the pointer associated with the action.
int index = MotionEventCompat.getActionIndex(event);
int xPos = -1;
int yPos = -1;

Log.d(DEBUG_TAG, "The action is " + actionToString(action));

if (event.getPointerCount() > 1) {
    Log.d(DEBUG_TAG, "Multitouch event");
    // The coordinates of the current screen contact, relative to
    // the responding View or Activity.
    xPos = (int)MotionEventCompat.getX(event, index);
    yPos = (int)MotionEventCompat.getY(event, index);

} else {
    // Single touch event
    Log.d(DEBUG_TAG, "Single touch event");
    xPos = (int)MotionEventCompat.getX(event, index);
    yPos = (int)MotionEventCompat.getY(event, index);
}

...
// Given an action int, returns a string description
public static String actionToString(int action) {
    switch (action) {

        case MotionEvent.ACTION_DOWN: return "Down";
        case MotionEvent.ACTION_MOVE: return "Move";
        case MotionEvent.ACTION_POINTER_DOWN: return "Pointer Down";
        case MotionEvent.ACTION_UP: return "Up";
        case MotionEvent.ACTION_POINTER_UP: return "Pointer Up";
        case MotionEvent.ACTION_OUTSIDE: return "Outside";
        case MotionEvent.ACTION_CANCEL: return "Cancel";
    }
    return "";
}
```

}

关于多点触摸的更多内容以及示例，可以查看[拖拽与缩放](#)章节。

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/scale.html>

# 拖拽与缩放

本节课程讲述，如何使用触摸手势拖拽、缩放屏幕上的对象，使用[onTouchEvent\(\)](#)来截获触摸事件。

## 拖拽一个对象

如果你的目标版本为3.0或以上，你可以使用[View.OnDragListener](#)监听内置的drag-and-drop事件，[拖拽与释放](#)中有更多描述。

使用触摸手势在屏幕上拖拽一个对象是很常见的操作。接下来的代码段让用户可以拖拽屏幕上的图片。需要注意以下几点：

- 拖拽操作时，即使有额外的手指放置到屏幕上，app也必须保持对最初的点（手指）的追踪。比如，想象在拖拽图片时，用户放置了第二根手指在屏幕上，并且抬起了第一根手指。如果你的app只是单独地追踪每个点，它会把第二个点当做默认的点，并且把图片移到该点的位置。
- 为了防止这种情况发生，你的app需要区分初始点以及之后任意的触摸点。要做到这一点，它需要追踪[处理多触摸手势](#)中提到过的[ACTION\\_POINTER\\_DOWN](#)、[ACTION\\_POINTER\\_UP](#)事件。每当第二根手指按下或拿起时，[ACTION\\_POINTER\\_DOWN](#)、[ACTION\\_POINTER\\_UP](#)事件就会传递给[onTouchEvent\(\)](#)回调函数。
- 当[ACTION\\_POINTER\\_UP](#)事件发生时，示例程序会移除对该点的索引值的引用，确保操作中的点的ID(the active pointer ID)不会引用已经不在触摸屏上的触摸点。这种情况下，app会选择另一个触摸点来作为操作中(active)的点，并保存它当前的x、y值。由于在[ACTION\\_MOVE](#)事件时，这个保存的位置会被用来计算屏幕上的对象将要移动的距离，所以app会始终根据正确的触摸点来计算移动的距离。

下面的代码段允许用户拖拽屏幕上的对象。它会记录操作中的点（active pointer）的初始位置，计算触摸点移动过的距离，再把对象移动到新的位置。如上所述，它也正确地处理了额外触摸点的可能。

需要注意的是，代码段中使用了[getActionMasked\(\)](#)函数。你应该始终使用这个函数（或者更好用[MotionEventCompat.getActionMasked\(\)](#)这个兼容版本）来获得[MotionEvent](#)对应的动作(action)。不像旧的[getAction\(\)](#)函数，[getActionMasked\(\)](#)就是设计用来处理多点触摸的。它会返回执行过的动作的掩码值，不包括该点的索引位。

```
// The 'active pointer' is the one currently moving our object.
private int mActivePointerId = INVALID_POINTER_ID;

@Override
public boolean onTouchEvent(MotionEvent ev) {
    // Let the ScaleGestureDetector inspect all events.
    mScaleDetector.onTouchEvent(ev);

    final int action = MotionEventCompat.getActionMasked(ev);

    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            final int pointerIndex = MotionEventCompat.getActionIndex();
            final float x = MotionEventCompat.getX(ev, pointerIndex);
            final float y = MotionEventCompat.getY(ev, pointerIndex);

            // Remember where we started (for dragging)
```

```
mLastTouchX = x;
mLastTouchY = y;
// Save the ID of this pointer (for dragging)
mActivePointerId = MotionEventCompat.getPointerId(ev, 0);
break;
}

case MotionEvent.ACTION_MOVE: {
    // Find the index of the active pointer and fetch its position
    final int pointerIndex =
        MotionEventCompat.findPointerIndex(ev, mActivePointerId);

    final float x = MotionEventCompat.getX(ev, pointerIndex);
    final float y = MotionEventCompat.getY(ev, pointerIndex);

    // Calculate the distance moved
    final float dx = x - mLastTouchX;
    final float dy = y - mLastTouchY;

    mPosX += dx;
    mPosY += dy;

    invalidate();

    // Remember this touch position for the next move event
    mLastTouchX = x;
    mLastTouchY = y;

    break;
}

case MotionEvent.ACTION_UP: {
    mActivePointerId = INVALID_POINTER_ID;
    break;
}

case MotionEvent.ACTION_CANCEL: {
    mActivePointerId = INVALID_POINTER_ID;
    break;
}

case MotionEvent.ACTION_POINTER_UP: {

    final int pointerIndex = MotionEventCompat.getActionIndex(ev);
    final int pointerId = MotionEventCompat.getPointerId(ev, pointerIndex);

    if (pointerId == mActivePointerId) {
        // This was our active pointer going up. Choose a new
        // active pointer and adjust accordingly.
        final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
        mLastTouchX = MotionEventCompat.getX(ev, newPointerIndex);
        mLastTouchY = MotionEventCompat.getY(ev, newPointerIndex);
        mActivePointerId = MotionEventCompat.getPointerId(ev, newPointerIndex);
    }
    break;
}
```

```
    }
}
return true;
}
```

## 通过拖拽平移

前一部分展示了一个拖拽屏幕上对象的例子。另一个常见的场景是平移（panning），是指用户通过拖拽移动引起x、y轴方向发生滚动(scrolling)。上面的代码段直接截获了[MotionEvent](#)动作来实现拖拽。这一部分的代码段，利用了平台对常用手势的内置支持。它重写了[GestureDetector.SimpleOnGestureListener](#)的[onScroll\(\)](#)函数。

更详细地说，当用户拖拽手指来平移内容时，[onScroll\(\)](#)函数就会被调用。[onScroll\(\)](#)函数只会在手指按下的情况下被调用，一旦手指离开屏幕了，要么手势终止，要么快速滑动(fling)手势开始（如果手指在离开屏幕前快速移动了一段距离）。关于滚动与快速滑动的更多讨论，可以查看[Scroll手势动画](#)章节。

这里是[onScroll\(\)](#)的相关代码段：

```
// The current viewport. This rectangle represents the currently visible chart domain and range.  
private RectF mCurrentViewport =  
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);  
  
// The current destination rectangle (in pixel coordinates) into which chart data should be drawn.  
private Rect mContentRect;  
  
private final GestureDetector.SimpleOnGestureListener mGestureList  
    = new GestureDetector.SimpleOnGestureListener() {  
    ...  
  
    @Override  
    public boolean onScroll(MotionEvent e1, MotionEvent e2,  
                           float distanceX, float distanceY) {  
        // Scrolling uses math based on the viewport (as opposed to map coordinates).  
  
        // Pixel offset is the offset in screen pixels, while viewport offset is the offset within the current viewport.  
        float viewportOffsetX = distanceX * mCurrentViewport.width()  
            / mContentRect.width();  
        float viewportOffsetY = -distanceY * mCurrentViewport.height()  
            / mContentRect.height();  
        ...  
        // Updates the viewport, refreshes the display.  
        setViewportBottomLeft(  
            mCurrentViewport.left + viewportOffsetX,  
            mCurrentViewport.bottom + viewportOffsetY);  
        ...  
        return true;  
    }  
}
```

[onScroll\(\)](#)函数中滑动视窗(viewport)来响应触摸手势的实现：

```
/**  
 * Sets the current viewport (defined by mCurrentViewport) to the  
 * X and Y positions. Note that the Y value represents the topmost
```

```
* and thus the bottom of the mCurrentViewport rectangle.  
*/  
private void setViewportBottomLeft(float x, float y) {  
    /*  
     * Constrains within the scroll range. The scroll range is sim  
     * extremes (AXIS_X_MAX, etc.) minus the viewport size. For ex  
     * extremes were 0 and 10, and the viewport size was 2, the sc  
     * be 0 to 8.  
    */  
  
    float curWidth = mCurrentViewport.width();  
    float curHeight = mCurrentViewport.height();  
    x = Math.max(AXIS_X_MIN, Math.min(x, AXIS_X_MAX - curWidth));  
    y = Math.max(AXIS_Y_MIN + curHeight, Math.min(y, AXIS_Y_MAX));  
  
    mCurrentViewport.set(x, y - curHeight, x + curWidth, y);  
  
    // Invalidates the View to update the display.  
    ViewCompat.postInvalidateOnAnimation(this);  
}
```

# 使用触摸手势进行缩放

如同[检测常用手势](#)章节中提到的，[GestureDetector](#)可以帮助你检测Android中的常见手势，例如滚动，快速滚动以及长按。对于缩放，Android也提供了[ScaleGestureDetector](#)类。当你想让view能识别额外的手势时，你可以配合使用[GestureDetector](#)和[ScaleGestureDetector](#)类。

为了报告检测到的手势事件，手势检测需要使用作为构造函数参数的listener对象。[ScaleGestureDetector](#)使用[ScaleGestureDetector.OnScaleGestureListener](#)。Android提供了[ScaleGestureDetector.SimpleOnScaleGestureListener](#)类作为帮助类，如果你不是关注所有的手势事件，你可以自行拓展(extend)它。

## 基本的缩放示例

下面的代码段展示了缩放功能中的基本部分。

```
private ScaleGestureDetector mScaleDetector;
private float mScaleFactor = 1.0f;

public MyCustomView(Context mContext) {
    ...
    // View code goes here
    ...
    mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
}

@Override
public boolean onTouchEvent(MotionEvent ev) {
    // Let the ScaleGestureDetector inspect all events.
    mScaleDetector.onTouchEvent(ev);
    return true;
}

@Override
public void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    canvas.save();
    canvas.scale(mScaleFactor, mScaleFactor);
    ...
    // onDraw() code goes here
    ...
    canvas.restore();
}

private class ScaleListener
    extends ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        mScaleFactor *= detector.getScaleFactor();

        // Don't let the object get too small or too large.
        mScaleFactor = Math.max(0.1f, Math.min(mScaleFactor, 5.0f));
    }
}
```

```
        invalidate();
        return true;
    }
}
```

## 更加复杂的缩放示例

这是本章节提供的InteractiveChart样例中一个更复杂的示范。通过使用[ScaleGestureDetector](#)中的"span"([getCurrentSpanX/Y](#))和"focus"  
([getFocusX/Y](#))，InteractiveChart样例支持滚动（平移）以及多指缩放。

```
@Override
private RectF mCurrentViewport =
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);
private Rect mContentRect;
private ScaleGestureDetector mScaleGestureDetector;
...
public boolean onTouchEvent(MotionEvent event) {
    boolean retVal = mScaleGestureDetector.onTouchEvent(event);
    retVal = mGestureDetector.onTouchEvent(event) || retVal;
    return retVal || super.onTouchEvent(event);
}

/**
 * The scale listener, used for handling multi-finger scale gesture
 */
private final ScaleGestureDetector.OnScaleGestureListener mScaleGe
    = new ScaleGestureDetector.SimpleOnScaleGestureListener()
/**/
    * This is the active focal point in terms of the viewport. Co
    * variable but kept here to minimize per-frame allocations.
    */
    private PointF viewportFocus = new PointF();
    private float lastSpanX;
    private float lastSpanY;

    // Detects that new pointers are going down.
@Override
public boolean onScaleBegin(ScaleGestureDetector scaleGestureD
    lastSpanX = ScaleGestureDetectorCompat.
        getCurrentSpanX(scaleGestureDetector);
    lastSpanY = ScaleGestureDetectorCompat.
        getCurrentSpanY(scaleGestureDetector);
    return true;
}

@Override
public boolean onScale(ScaleGestureDetector scaleGestureDetect
    float spanX = ScaleGestureDetectorCompat.
        getCurrentSpanX(scaleGestureDetector);
    float spanY = ScaleGestureDetectorCompat.
        getCurrentSpanY(scaleGestureDetector);
```

```
        float newWidth = lastSpanX / spanX * mCurrentViewport.widt
        float newHeight = lastSpanY / spanY * mCurrentViewport.hei

        float focusX = scaleGestureDetector.getFocusX();
        float focusY = scaleGestureDetector.getFocusY();
        // Makes sure that the chart point is within the chart reg
        // See the sample for the implementation of hitTest().
        hitTest(scaleGestureDetector.getFocusX(),
                scaleGestureDetector.getFocusY(),
                viewportFocus);

        mCurrentViewport.set(
            viewportFocus.x
                - newWidth * (focusX - mContentRect.left)
                / mContentRect.width(),
            viewportFocus.y
                - newHeight * (mContentRect.bottom - focusY)
                / mContentRect.height(),
            0,
            0);
        mCurrentViewport.right = mCurrentViewport.left + newWidth;
        mCurrentViewport.bottom = mCurrentViewport.top + newHeight
        ...
        // Invalidates the View to update the display.
        ViewCompat.postInvalidateOnAnimation(InteractiveLineGraphV

        lastSpanX = spanX;
        lastSpanY = spanY;
        return true;
    }
};
```

编写: [Andrwyw](#) - 校对:

原文: <http://developer.android.com/training/gestures/viewgroup.html>

# 管理ViewGroup中的触摸事件

处理[ViewGroup](#)中的触摸事件需要特别注意，因为通常情况下都是[ViewGroup](#)中的子view处理不同的触摸事件，而不是[ViewGroup](#)自己处理。为了确保每个view能正确地接受到它们想要的触摸事件，可以重载[onInterceptTouchEvent\(\)](#)函数。

## 在ViewGroup中截获触摸事件

每当在ViewGroup的表面上检测到一个触摸事件，包括它子view的表面，onInterceptTouchEvent()都会被调用。如果onInterceptTouchEvent()返回true，MotionEvent就被截获了，表示它不再会被传递到子view了，而是传递给该父view的onTouchEvent()方法。

onInterceptTouchEvent()方法让父view能够在它的子view之前处理触摸事件。如果你让onInterceptTouchEvent()返回true，则之前处理触摸事件的子view会收到ACTION\_CANCEL消息，并且该点之后的事件会被发送给该父view的onTouchEvent()函数，进行通常地处理。onInterceptTouchEvent()也可以返回false，这样在事件沿view层级分发到可通过onTouchEvent()处理它的目标前，父view可以简单地观察该事件。

接下来的代码段中，MyViewGroup继承自ViewGroup。MyViewGroup有多个子view。如果你水平地拖动手指经过某个子view，该子view不会接收到触摸事件，而是MyViewGroup处理这些触摸事件来滚动它的内容。然而，如果你点击子view中的button，或垂直地滚动子view，则父view不会截获这些触摸事件，因为子view本就是预订目标。在这些情况下，onInterceptTouchEvent()应该返回false，MyViewGroup的onTouchEvent()也不会被调用。

```
public class MyViewGroup extends ViewGroup {

    private int mTouchSlop;

    ...

    ViewConfiguration vc = ViewConfiguration.get(view.getContext());
    mTouchSlop = vc.getScaledTouchSlop();

    ...

    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        /*
         * This method JUST determines whether we want to intercept
         * If we return true, onTouchEvent will be called and we do
         * scrolling there.
         */
        final int action = MotionEventCompat.getActionMasked(ev);

        // Always handle the case of the touch gesture being completed
        if (action == MotionEvent.ACTION_CANCEL || action == MotionEvent.ACTION_UP) {
            // Release the scroll.
            mIsScrolling = false;
            return false; // Do not intercept touch event, let the child handle it
        }

        switch (action) {
            case MotionEvent.ACTION_MOVE: {
                if (mIsScrolling) {
                    // We're currently scrolling, so yes, intercept
                }
            }
        }
    }
}
```

```
// touch event!
    return true;
}

// If the user has dragged her finger horizontally
// the touch slop, start the scroll

// left as an exercise for the reader
final int xDiff = calculateDistanceX(ev);

// Touch slop should be calculated using ViewConfig
// constants.
if (xDiff > mTouchSlop) {
    // Start scrolling!
    mIsScrolling = true;
    return true;
}
break;
}

...
}

// In general, we don't want to intercept touch events. They
// handled by the child view.
return false;
}

@Override
public boolean onTouchEvent(MotionEvent ev) {
    // Here we actually handle the touch event (e.g. if the action
    // scroll this container).
    // This method will only be called if the touch event was
    // onInterceptTouchEvent
    ...
}
}
```

注意[ViewGroup](#)也提供了[requestDisallowInterceptTouchEvent\(\)](#)方法。当它的子view不想该父view和祖先view通过[onInterceptTouchEvent\(\)](#)截获它的触摸事件时，[ViewGroup](#)会调用改方法。

## 使用ViewConfiguration的常量

上面的代码段中使用了当前的[ViewConfiguration](#)来初始化mTouchSlop变量。你可以使用[ViewConfiguration](#)类来获取Android系统常用的一些距离、速度、时间值。

“Touch slop”是指在用户触摸事件可被识别为移动手势前,移动过的那一段像素距离。Touch slop通常用来预防用户在做一些其他操作时意外地滑动，例如触摸屏幕上的元素时。

另外两个常用的[ViewConfiguration](#)函数是[getScaledMinimumFlingVelocity\(\)](#)和[getScaledMaximumFlingVelocity\(\)](#)。这两个函数会返回初始化一个快速滑动(fling)的最小、最大速度（分别地），以像素每秒为测量单位。如：

```
ViewConfiguration vc = ViewConfiguration.get(view.getContext());
private int mSlop = vc.getScaledTouchSlop();
private int mMinFlingVelocity = vc.getScaledMinimumFlingVelocity()
private int mMaxFlingVelocity = vc.getScaledMaximumFlingVelocity()

...
case MotionEvent.ACTION_MOVE: {
    ...
    float deltaX = motionEvent.getRawX() - mDownX;
    if (Math.abs(deltaX) > mSlop) {
        // A swipe occurred, do something
    }
}

...
case MotionEvent.ACTION_UP: {
    ...
    } if (mMinFlingVelocity <= velocityX && velocityX <= mMaxFling
         && velocityY < velocityX) {
        // The criteria have been satisfied, do something
    }
}
```

## 扩展view的可触摸区域

Android提供了[TouchDelegate](#)类让父view扩展子view的可触摸区域，扩展后的区域可超过子view本身的边界。这在子view很小，但需要一个更大的触摸区域时非常有用。如果需要，你也可以使用这种方式来实现对子view的触摸区域的收缩。

在下面的例子中，[ImageButton](#)对象是这个"delegate view"（是指触摸区域将被父view扩展的那个子view）。这是布局文件：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android:id="@+id/parent_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <ImageButton android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@null"
        android:src="@drawable/icon" />
</RelativeLayout>
```

下面的代码段做了这样几件事：

- 获得父view对象并发送一段[Runnable](#)到UI线程。这会确保父view在调用[getHitRect\(\)](#)函数前会布局它的子view。[getHitRect\(\)](#)函数会获得子view在父view坐标系中的点击矩形（触摸区域）。
- 找到[ImageButton](#)子view，然后调用[getHitRect\(\)](#)来获得它的触摸区域的边界。
- 扩展[ImageButton](#)的点击矩形的边界。
- 实例化一个[TouchDelegate](#)对象，并把扩展过的点击矩形和[ImageButton](#)子view作为参数传递给它。
- 设置父view的[TouchDelegate](#)，这样在touch delegate边界内的点击就会传递到该子view上。

在[ImageButton](#)子view的touch delegate范围内，父view会接收到所有的触摸事件。如果触摸事件发生在子view自身的点击矩形中，父view会把触摸事件交给子view处理。

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Get the parent view
        View parentView = findViewById(R.id.parent_layout);

        parentView.post(new Runnable() {
            // Post in the parent's message queue to make sure the
            // lays out its children before you call getHitRect()
            @Override
            public void run() {
                // The bounds for the delegate view (an ImageButto
```

```
// in this example)
Rect delegateArea = new Rect();
ImageButton myButton = (ImageButton) findViewById(
myButton.setEnabled(true);
myButton.setOnClickListener(new View.OnClickListener
    @Override
    public void onClick(View view) {
        Toast.makeText(MainActivity.this,
            "Touch occurred within ImageButton",
            Toast.LENGTH_SHORT).show();
    }
));

// The hit rectangle for the ImageButton
myButton.getHitRect(delegateArea);

// Extend the touch area of the ImageButton beyond
// on the right and bottom.
delegateArea.right += 100;
delegateArea.bottom += 100;

// Instantiate a TouchDelegate.
// "delegateArea" is the bounds in local coordinates
// the containing view to be mapped to the delegate
// "myButton" is the child view that should receive
// events.
TouchDelegate touchDelegate = new TouchDelegate(de
    myButton);

// Sets the TouchDelegate on the parent view, such
// within the touch delegate bounds are routed to
if (View.class.isInstance(myButton.getParent())) {
    ((View) myButton.getParent()).setTouchDelegate
}
}

});
```

编写:[zhaochunqi](#)

校对:

# 处理键盘输入

Android 系统展示了一个屏幕上的键盘- 被称为软输入法- 当一个文本域在UI中接收到聚焦时。为了提供最好的用户体验，你可以指定你期望的输入类型(电话号码或Email地址)和输入法的表现形式(是否需要自动纠正拼写错误)。

除了屏幕上的输入法，Android也支持实体键盘，所以充分利用可能的外接键盘来优化用户的交互体验是很重要的。

# Lessons

- [指定输入法](#)

学习如何表现特定的虚拟输入法，如为电话号码、网址和其他一些格式所设计的。同样应该学习如何指定建议的操作如确定(**Done**)或者下一步(**Next**)。

- [处理输入法的显示](#)

学习如何指定合适展示软键盘输入法，如何让你的布局适合因为输入法而减少的屏幕空间。

- [支持键盘导航](#)

学习如何验证用户能够使用键盘导航到你的应用以及如何对导航顺序做出相应的改变。

- [处理键盘行为](#)

学习如何对用户的键盘输入进行回应。

编写:[zhaochunqi](#)

校对:

# 指定输入法类型

每个文本域期待特定的文本类型，如Email，电话号码，或者纯文本。为应用中的每一个文本域指定特定的输入类型以便系统展示更为合适的软键盘输入法(比如屏幕上键盘)是很重要的。

## 指定键盘类型

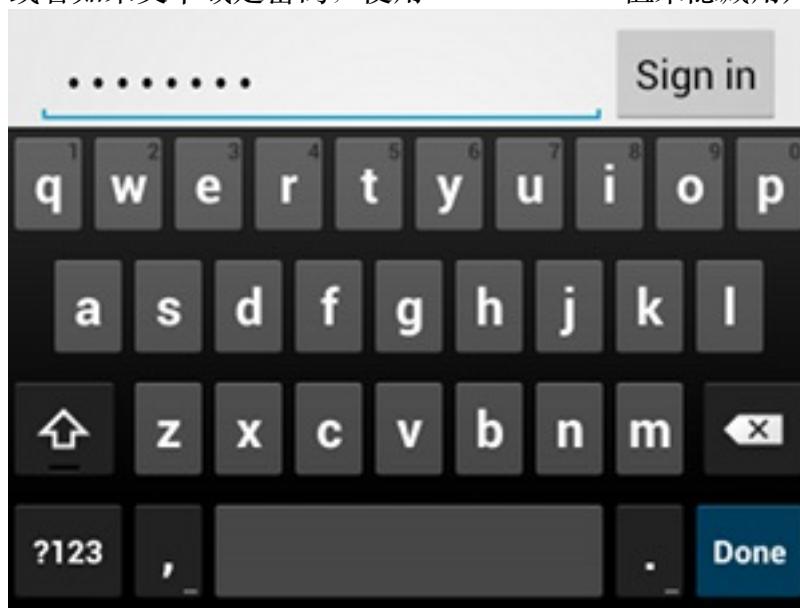
你总是可以为你的文本域定义输入法通过添加`android:inputType` 属性到 `<EditText>` 元素中。

举例来说，如果你想要一个为输入电话号码的输入法，使用"phone"值：



```
<EditText  
    android:id="@+id/phone"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/phone_hint"  
    android:inputType="phone" />
```

或者如果文本域是密码，使用"textPassword"值来隐藏用户的输入：



```
<EditText  
    android:id="@+id/password"  
    android:hint="@string/password_hint"  
    android:inputType="textPassword"  
    ... />
```

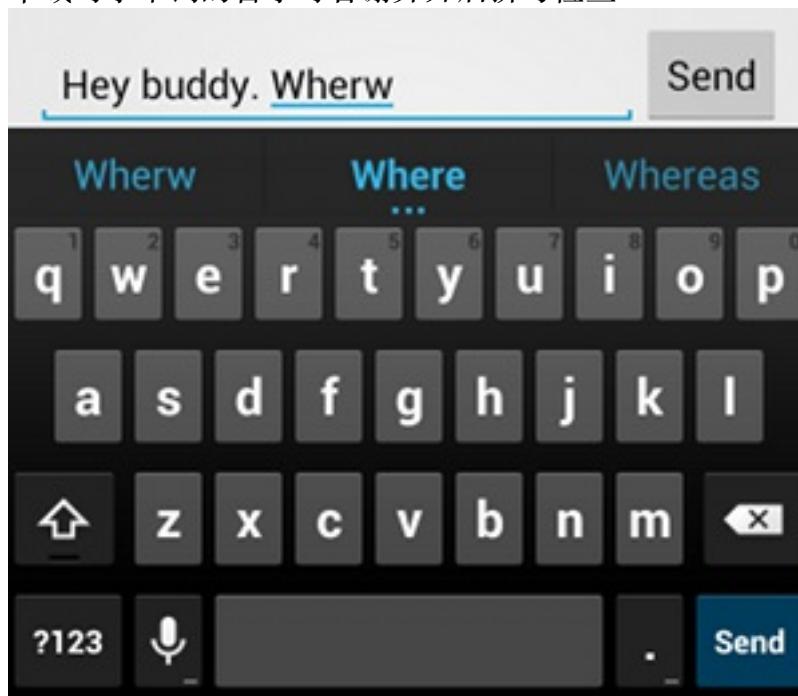
有几种可供选择的值在`android:inputType`属性中记录，一些值可以组合起来实现特定的输入

法表现和附加的行为。

## 开启拼写建议和其他的行为

android:inputType属性允许你为输入法指定不同的行为。最为重要的是，如果你的文本域是为基本的文本输入(如短信息)，你应该使用"textAutoCorrect"来开启拼写检查。

你可以组合不同的行为和输入法形式通过textAutoCorrect这个属性。如：如何创建一个文本域句子单词的首字母答谢并开启拼写检查：



```
<EditText  
    android:id="@+id/message"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:inputType=  
        "textCapSentences|textAutoCorrect"  
    ... />
```

## 指定输入法的行为

多数的软键盘会在底部角落里为用户提供一个合适的动作按钮来触发当前文本域的操作。默认情况下，系统使用下一步(Next)或者确认(DONE)除非你的文本域允许多行(如 android:inputType="textMultiLine")，这种情况下，动作按钮就是回车换行。然而，你可以制定额外的动作一边更适合你的文本域，比如SEND和GO。

指定特定的动作按钮，将 [android:imeOptions](#) 属性的值设为"actionSend" 或 "actionSearch"。



如：

```
<EditText  
    android:id="@+id/search"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/search_hint"  
    android:inputType="text"  
    android:imeOptions="actionSend" />
```

然后你可以通过为 [EditText](#) 定义 [TextView.OnEditorActionListener](#) 来监听动作按钮的启动。在监听器中，对输入法编辑器对合适的回应的动作ID对应在 [EditorInfo](#) 类中，如 [IME\\_ACTION\\_SEND](#)。例如：

```
EditText editText = (EditText) findViewById(R.id.search);  
editText.setOnEditorActionListener(new OnEditorActionListener() {  
    @Override  
    public boolean onEditorAction(TextView v, int actionBarId, KeyEvent  
        handled = false;  
        if (actionId == EditorInfo.IME_ACTION_SEND) {  
            sendMessage();  
            handled = true;  
        }  
        return handled;  
    }  
});
```

编写:[zhaochunqi](#)

校对:

# 处理输入法可见性

当输入焦点移入或移出可编辑当文本域时，Android会相应的显示或隐藏输入法(如屏幕输入法)。系统也会决定你的输入法上方UI和文本域的显示。举例来说，当屏幕上竖直空间被压缩时，文本域可能填充所有的输入法上方的空间。对于多数的应用来说，这些默认的行为基本就足够了。

然而，在一些事例中，你可能会想要更加直接的控制输入法的显示，指定你的布局在在输入法显示时候的表现。这节课会向你解释如何控制和回应输入法的可见性。

## 在Activity启动时显示输入法

尽管Android会在Activity启动时给予第一个文本域焦点，但是并不会显示输入法。因为进入文本可能并不是activity中的首要任务，所以这为是很合理的。可是，如果进入文本确实是首要的任务(如登录界面)，可能需要用到输入法默认显示。

为了在activity启动时展示输入法，添加[android:windowSoftInputMode](#) 属性到<activity>元素中，使用 "stateVisible"，如下：

```
<application ... >
    <activity
        android:windowSoftInputMode="stateVisible" ... >
        ...
    </activity>
    ...
</application>
```

注意：如果用户设备有一个实体键盘，软键盘输入法可能不显示。

## 需要时显示输入法

如果在activity生命周期中有一个方法在想要确保输入法是可见的，可以使用[InputMethodManager](#) 来实现。

举例来说，下面的方法调用了一个需要用户填写文本的[View](#)，调用了[requestFocus\(\)](#) 来获取焦点，然后 [showSoftInput\(\)](#)来打开输入法。

```
public void showSoftKeyboard(View view) {  
    if (view.requestFocus()) {  
        InputMethodManager imm = (InputMethodManager)  
            getSystemService(Context.INPUT_METHOD_SERVICE);  
        imm.showSoftInput(view, InputMethodManager.SHOW_IMPLICIT);  
    }  
}
```

注意：一旦输入法设定可见了，你不应该用程序来隐藏。系统会在用户结束文本域的任务的时候隐藏，或者可以使用系统控制(如返回键)来隐藏。

## 指定你的UI回应方式

当你的输入法显示在屏幕上，减少了UI中的可用空间。系统会为你的UI的可见区的UI做调整但是可能并非很正确。为了确保你应用的最佳表现，你应该在UI的剩余空间中展示你想要展示的系统界面。

为了声明你在activity中的合适的对待，使用 android:windowSoftInputMode 属性在你的清单文件中的<activity>元素使用某个"adjust"值。

举例来说，为了确保系统会在可用空间中重新调整布局的大小。为了确保你所有的布局内容都是可用的(尽管可能需要滑动)使用"adjustResize":

```
<application ... >
    <activity
        android:windowSoftInputMode="adjustResize" ... >
        ...
    </activity>
    ...
</application>
```

你可以结合调整和使用上面的[初始输入法可见性](#)来指定：

```
<activity
    android:windowSoftInputMode="stateVisible|adjustResize" ...
    ...
</activity>
```

如果你的UI中包含用户可能需要在文本输入时立即执行的事情，那么使用"adjustResize"时很重要的。例如，如果你使用相对布局在屏幕底部放置一个按钮，使用"adjustResize"来重新调整大小，使得按钮栏出现在输入法上方。

编写:[zhaochunqi](#)

校对:

# 兼容键盘导航

除了软键盘输入法(如屏幕键盘)以外，Android支持物理键盘连接到设备上。一个键盘不仅提供为文本输入提供方便地模式，而提供一个合适的方法来导航和与应用交互。尽管多数的手持设备像手机使用触摸作为主要的交互方式，平板和一些其他的设备正在逐步流行起来，许多用户喜欢外接键盘。

随着更多的Android设备提供这种体验，为你的应用添加通过键盘进行交互的支持优化是很重要的。这节课介绍了怎样为键盘导航提供更好的支持。

注意：对那些没有使用可见导航提示的应用来说，在应用中支持方向性的导航对于应用的可用性也是很重要的。完全支持方向性导航在你的应用中还可以帮助你使用诸如[uiautomator](#)进行[自动化用户界面测试化](#)。

# 测试你的应用

可能用户已经在你的应用中使用键盘导航了，因为Android系统默认开启了大多是必要的行为。

所有由Android framework(如Button和EditText) 提供的交互 widgets是可获得焦点的。这意味着用户可以使手控设备如D-pad或键盘或widgets发亮或者其他一些获得输入焦点的行为改变外观。

为了测试你的应用：

1. 在实体键盘的设备上安装你的应用

如果你没有带实体键盘的设备，连接一个蓝牙键盘或者USB键盘(尽管并不是所有的设备都支持USB连接)

你还可以使用Android虚拟机

1. 在AVD管理器中，或者点击New Device或者选择一个已存在的文档点击Clone.
  2. 在出现的窗口中，确保键盘和D-pad开启。
2. 为了验证你的应用，只是用Tab键来进行UI导航，确保每一个UI控制的焦点如预期的一致。查看每个不在预期焦点的实例。
  3. 从头开始，使用方向键(键盘上的箭头键)来控制你应用的导航。从每一个在你UI中的焦点元素，按上、下、左、右。查看每个不在预期焦点的实例

如果你遇到任何使用Tab键或方向键不如预期，在布局文件中指定应该的焦点，如下面几部分所讨论的。

## 处理Tab导航

当一个用户使用键盘上到Tab键导航到你的应用时，系统会在元素之间传递焦点，取决于他们在布局文件中的显示顺序。如果你使用相对布局，在屏幕上的元素顺序与文件中元素的顺序不一致，那样你可能需要手动的指定焦点顺序。

举例来说，在下面的布局文件中，两个对其右边的按钮和一个对齐第二个按钮导航。为了把焦点从第一个按钮传递到文本域，然后再传递到第二个按钮，布局文件需要清楚的为每一个可聚焦的元素定义焦点顺序，使用属性`android:nextFocusForward`:

```
<RelativeLayout ...>
    <Button
        android:id="@+id/button1"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:nextFocusForward="@+id/editText1"
        ... />
    <Button
        android:id="@+id/button2"
        android:layout_below="@+id/button1"
        android:nextFocusForward="@+id/button1"
        ... />
    <EditText
        android:id="@+id/editText1"
        android:layout_alignBottom="@+id/button2"
        android:layout_toLeftOf="@+id/button2"
        android:nextFocusForward="@+id/button2"
        ... />
    ...
</RelativeLayout>
```

现在焦点从button1到button2再到editText1改成了合适的按照出现在屏幕上顺序到从button1到editText1再到button2.

## 处理直接的导航

用户也能够使用键盘上的方向键在你的app中导航(这种行为与在D-pad和轨迹球中的导航一致)。系统提供了一个最佳猜测对于哪个视图应该给予焦点在一个基于方向的基于布局文件的在屏幕上展现的布局。然而有时，系统会猜测错误。

如果你的系统没有传递焦点到合适的视图中在导航到一个给定的视图中的时候，指定一个视图使用如下的属性：

- android:nextFocusUp
- android:nextFocusDown
- android:nextFocusLeft
- android:nextFocusRight

每一个属性设计了下一个接受焦点的视图当用户导航到那个方向时，如指定当view的ID一样。举例来说：

```
<Button  
    android:id="@+id/button1"  
    android:nextFocusRight="@+id/button2"  
    android:nextFocusDown="@+id/editText1"  
    ... />  
<Button  
    android:id="@+id/button2"  
    android:nextFocusLeft="@+id/button1"  
    android:nextFocusDown="@+id/editText1"  
    ... />  
<EditText  
    android:id="@+id/editText1"  
    android:nextFocusUp="@+id/button1"  
    ... />
```

编写:[zhaochunqi](#)

校对:

# 处理按键动作

当预估给予可编辑当文本域焦点时，如一个[EditText](#)元素，而且用户拥有一个实体键盘连接，所有当输入由系统处理。然而如果你想接管或直接处理键盘输入键盘操作，通过实现接口[KeyEvent.Callback](#)的回调方法，如[onKeyDown\(\)](#)和[onKeyMultiple\(\)](#)。

Activity和View类都实现了[KeyEvent.Callback](#)的接口，所以通常你只需要在这些重写回调方法来适当的扩展这些类。

注意：当使用KeyEvent类和相关的API处理键盘事件时，你期望的应该是只从实体键盘中接收。你永远不应该指望从一个软键盘(如屏幕键盘)来接受点击事件。

## 处理单个按键点击事件

处理单个的按键点击，实现合适的 [onKeyDown\(\)](#) 或 [onKeyUp\(\)](#)。通常，你使用[onKeyUp\(\)](#)来确保你只接收一个事件。如果用户点击并按住按钮不放，[onKeyDown\(\)](#)会被调用多次。

举例，这是一个对一些按键控制游戏的实现：

```
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
    switch (keyCode) {
        case KeyEvent.KEYCODE_D:
            moveShip(MOVE_LEFT);
            return true;
        case KeyEvent.KEYCODE_F:
            moveShip(MOVE_RIGHT);
            return true;
        case KeyEvent.KEYCODE_J:
            fireMachineGun();
            return true;
        case KeyEvent.KEYCODE_K:
            fireMissile();
            return true;
        default:
            return super.onKeyUp(keyCode, event);
    }
}
```

## 处理修饰键

为了对修饰键进行回应如一个组合Shift和Control修饰键，你可以查询[KeyEvent](#)传递到回调方法。一些方法提供一些信息关于修饰键如getModifiers() 和 getMetaState()。然而，最简单的解决方案时检查你关心的按键是否被按下了的方法，如[isShiftPressed\(\)](#) 和 [isCtrlPressed\(\)](#)。

例如，有一个[onKeyDown\(\)](#) 的实现，当Shift键和一个其他当键按下当时时候做一些额外的处理：

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    switch (keyCode) {
        ...
        case KeyEvent.KEYCODE_J:
            if (event.isShiftPressed()) {
                fireLaser();
            } else {
                fireMachineGun();
            }
            return true;
        case KeyEvent.KEYCODE_K:
            if (event.isShiftPressed()) {
                fireSeekingMissle();
            } else {
                fireMissile();
            }
            return true;
        default:
            return super.onKeyDown(keyCode, event);
    }
}
```

编写:

校对:

# 兼容游戏控制器

待认领进行编写，有意向的小伙伴，可以直接修改对应的markdown文件，进行提交！

编写:

校对:

# 处理控制器输入动作

待认领进行编写，有意向的小伙伴，可以直接修改对应的markdown文件，进行提交！

编写:

校对:

# 支持不同的Android系统版本

待认领进行编写，有意向的小伙伴，可以直接修改对应的markdown文件，进行提交！

编写:

校对:

# 支持多个控制器

待认领进行编写，有意向的小伙伴，可以直接修改对应的markdown文件，进行提交！

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/best-background.html>

# 后台任务

These classes show you how to run jobs in the background to boost your application's performance and minimize its drain on the battery.

## [Running in a Background Service](#)

How to improve UI performance and responsiveness by sending work to a Service running in the background

## [Loading Data in the Background](#)

How to use CursorLoader to query data without affecting UI responsiveness.

## [Managing Device Awake State](#)

How to use repeating alarms and wake locks to run background jobs.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/run-background-service/index.html>

# 在IntentService中执行后台任务

除非你特别指定，否则大部分在前台UI界面上的操作都执行在一个叫做UI Thread的特殊线程中。这可能会导致某些问题，因为耗时操作可能会干扰界面的响应性能。为了避免这样的问题，Android Framework提供了几个类，用来帮助你把那些耗时操作移动到后台线程中执行。那些类中最常用的就是[IntentService](#)。

这一章节会讲到如何实现一个IntentService，向它发送任务并反馈它的结果给其他模块。

## Lessons

- [Creating a Background Service:创建IntentService](#)  
学习如何创建一个IntentService。
- [Sending Work Requests to the Background Service:发送任务请求到IntentService](#)  
学习如何发送工作任务到IntentService。
- [Reporting Work Status:报告后台任务的执行状态](#)  
学习如何使用Intent与LocalBroadcastManager在Activit与IntentService之间进行交互。

编写:[kesenhoo](#)

校对:

# Creating a Background Service: 创建 IntentService

IntentService为执行一个操作在单个后台线程，提供了一种直接的实现方式。它可以处理一个长时间操作的任务并确保不影响到UI的响应性。而且IntentService的执行并不受UI的生命周期的影响。

IntentService有下面几个局限性：

- 不可以直接和UI做交互。为了把他执行的结果体现在UI上，需要发送给Activity。
- 工作任务队列是顺序执行的，如果一个任务正在IntentService中执行，此时你再发送一个任务请求，这个任务会一直等待直到前面一个任务执行完毕。
- 正在执行的任务无法打断。

然而，在大多数情况下，IntentService都是简单后台任务操作的理想选择。

这节课会演示如何创建继承的IntentService。同样也会演示如何创建必须实现的回调[onHandleIntent\(\)](#)。最后，还会解释如何在manifest文件中定义这个IntentService。

## 1) 创建IntentService

为了给你的app创建一个IntentService， 定义一个类， extends IntentService， 在里面override onHandleIntent()方法， 如下所示：

```
public class RSSPullService extends IntentService {  
    @Override  
    protected void onHandleIntent(Intent workIntent) {  
        // Gets data from the incoming Intent  
        String dataString = workIntent.getDataString();  
        ...  
        // Do work here, based on the contents of dataString  
        ...  
    }  
}
```

注意一个普通Service组件的其他回调， 例如onStartCommand()会被IntentService自动触发。在IntentService中， 要避免override那些回调。

## 2)在Manifest文件中定义IntentService

IntentService需要在manifest文件的标签下进行定义，如下所示：

```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    ...
    <!--
        Because android:exported is set to "false",
        the service is only available to this app.
    -->
    <service
        android:name=".RSSPullService"
        android:exported="false"/>
    ...
</application>
```

android:name属性指明了IntentService的名字。

注意标签并没有包含任何intent filter。因为发送任务给IntentService的Activity需要使用显式Intent，所以不需要filter。这也意味着只有在同一个app或者其他使用同一个UserID的组件才能够访问到这个Service。

至此，已经学习了IntentService的基础知识，下节会学习如何发送任务到IntentService。

编写:[kesenhoo](#)

校对:

# Sending Work Requests to the Background Service:发送任务请求到IntentService

前一篇文章演示了如何创建一个IntentService类。这次会演示如何通过发送一个Intent来触发IntentService执行任务。这个Intent可以传递一些数据给IntentService。可以在Activity或者Fragment的任何时间点发送这个Intent。

为了创建一个工作请求并发送到IntentService。需要先创建一个explicit Intent，添加数据到intent，然后通过执行[startService\(\)](#) 把它发送到IntentService。

下面的是代码示例：

- 创建一个新的显式的Intent用来启动IntentService。

```
/*
 * Creates a new Intent to start the RSSPullService
 * IntentService. Passes a URI in the
 * Intent's "data" field.
 */
mServiceIntent = new Intent(getActivity(), RSSPullService.class);
mServiceIntent.setData(Uri.parse(dataUrl));
```

- 执行startService()

```
// Starts the IntentService
getActivity().startService(mServiceIntent);
```

注意：可以在Activity或者Fragment的任何位置发送任务请求。

一旦执行了startService()，IntentService在自己本身的[onHandleIntent\(\)](#)方法里面开始执行这个任务。

下一步是如何把工作任务的执行结果返回给发送任务的Activity或者Fragment。下节课会演示如何使用[BroadcastReceiver](#)来完成这个任务。

编写:[kesenhoo](#)

校对:

# Reporting Work Status:报告后台任务的执行状态

这章节会演示如何回传IntentService中执行的任务状态与结果给发送方。例如，回传任务的状态给Activity并进行更新UI。推荐的方式是使用[LocalBroadcastManager](#)，这个组件可以限制broadcast只在自己的App中进行传递。

## Report Status From an IntentService

为了在IntentService中向其他组件发送任务状态，首先创建一个Intent并在data字段中包含需要传递的信息。作为一个可选项，还可以给这个Intent添加一个action与data URI。

下一步，通过执行[LocalBroadcastManager.sendBroadcast\(\)](#)来发送Intent。Intent被发送到任何有注册接受它的组件中。为了获取到LocalBroadcastManager的实例，可以执行`getInstance()`。代码示例如下：

```
public final class Constants {  
    ...  
    // Defines a custom Intent action  
    public static final String BROADCAST_ACTION =  
        "com.example.android.threadsample.BROADCAST";  
    ...  
    // Defines the key for the status "extra" in an Intent  
    public static final String EXTENDED_DATA_STATUS =  
        "com.example.android.threadsample.STATUS";  
    ...  
}  
public class RSSPullService extends IntentService {  
    ...  
    /*  
     * Creates a new Intent containing a Uri object  
     * BROADCAST_ACTION is a custom Intent action  
     */  
    Intent localIntent =  
        new Intent(Constants.BROADCAST_ACTION)  
            // Puts the status into the Intent  
            .putExtra(Constants.EXTENDED_DATA_STATUS, status);  
    // Broadcasts the Intent to receivers in this app.  
    LocalBroadcastManager.getInstance(this).sendBroadcast(localInt  
    ...  
}
```

下一步是在发送任务的组件中接收发送出来的broadcast数据。

## Receive Status Broadcasts from an IntentService

为了接受广播的数据对象，需要使用BroadcastReceiver的子类并实现[BroadcastReceiver.onReceive\(\)](#)的方法，这里可以接收LocalBroadcastManager发出的广播数据。

```
// Broadcast receiver for receiving status updates from the Intent
private class ResponseReceiver extends BroadcastReceiver
{
    // Prevents instantiation
    private DownloadStateReceiver() {
    }
    // Called when the BroadcastReceiver gets an Intent it's registered to receive
    @Override
    public void onReceive(Context context, Intent intent) {
        ...
        /*
         * Handle Intents here.
         */
        ...
    }
}
```

一旦定义了BroadcastReceiver，也应该定义actions，categories与data用来做广播过滤。为了实现这些，需要使用[IntentFilter](#)如下所示：

```
// Class that displays photos
public class DisplayActivity extends FragmentActivity {
    ...
    public void onCreate(Bundle stateBundle) {
        ...
        super.onCreate(stateBundle);
        ...
        // The filter's action is BROADCAST_ACTION
        IntentFilter mStatusIntentFilter = new IntentFilter(
            Constants.BROADCAST_ACTION);

        // Adds a data filter for the HTTP scheme
        mStatusIntentFilter.addDataScheme("http");
    }
}
```

为了给系统注册这个BroadcastReceiver，需要通过LocalBroadcastManager执行[registerReceiver\(\)](#)的方法。如下所示：

```
// Instantiates a new DownloadStateReceiver
DownloadStateReceiver mDownloadStateReceiver =
    new DownloadStateReceiver();
// Registers the DownloadStateReceiver and its intent filter
LocalBroadcastManager.getInstance(this).registerReceiver(
    mDownloadStateReceiver,
    mStatusIntentFilter);
...
```

---

一个BroadcastReceiver可以处理多种类型的广播数据。每个广播数据都有自己的ACTION。这个功能使得不用定义多个不同的BroadcastReceiver来分别处理不同的ACTION数据。为BroadcastReceiver定义另外一个IntentFilter，只需要创建一个新的IntentFilter并重复执行registerReceiver()即可。例如：

```
/*
 * Instantiates a new action filter.
 * No data filter is needed.
 */
statusIntentFilter = new IntentFilter(Constants.ACTION_ZOO
...
// Registers the receiver with the new filter
LocalBroadcastManager.getInstance(getActivity()).registerR
    mDownloadStateReceiver,
    mIntentFilter);
```

发送一个广播并不会start或者resume一个Activity。BroadcastReceiver可以接收广播数据，即使是你的是app是在后台运行中。但是这不会强迫使得你的app变成foreground的。如果想在app不可见的时候通知用户一个后台的事件，建议使用[Notification](#)。永远不要为了响应一个广播而去启动Activity。

---

笔者评论：使用LocalBroadcastManager结合IntentService其实是一种很典型高效的做法，同时也更符合OO的思想，通过广播注册与反注册的方式，对两个组件进行解耦。如果使用Handler传递到后台线程作为回调，容易带来的内存泄漏。原因是：匿名内部类对外面的Activity持有引用，如果在Activity被销毁的时候，没有对Handler进行显式的解绑，会导致Activity无法正常销毁，这样自然就有了内存泄漏。当然，如果用文章中的方案，通常也要记得在Activity的onPause的时候进行unRegisterReceiver，除非你有充足的理由为解释这里为何要继续保留。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/load-data-background/index.html>

# 使用CursorLoader在后台加载数据

从[ContentProvider](#)查询你需要显示的数据是比较耗时的。如果你在Activity中直接执行查询的操作，那么有可能导致Activity出现ANR的错误。即使没有发生ANR，用户也会看到一个令人烦恼的UI延迟。为了避免那些问题，你应该在另外一个线程中执行查询的操作，等待查询操作完成，然后再显示查询结果。

你可以通过使用[CursorLoader](#)来实现，它会在后台异步查询数据并在查询结束时和Activity重新进行连接。CursorLoader不仅仅能够实现在后台查询数据，还能够在查询数据发生变化时自动执行重新查询的操作。

这节课会介绍如何使用CursorLoader来执行一个后台查询数据的操作。在这节课中的演示代码使用的是[v4 Support Library](#)中的类。

下载演示代码

[ThreadSample](#)

## Lessons

- [使用CursorLoader执行查询任务：Running a Query with a CursorLoader](#)

学习如何使用CursorLoader在后台执行查询操作。

- [处理查询的结果：Handling the Results](#)

学习如何处理从CursorLoader查询到的数据，以及在loader框架重置CursorLoader时如何解除当前Cursor的引用。

编写:[kesenhoo](#)

校对:

# 使用CursorLoader执行查询任务

CursorLoader依靠ContentProvider在后台执行一个异步的查询操作，并且返回数据给调用它的Activity或者FragmentActivity。这使得Activity 或者 FragmentActivity 能够在查询任务正在执行的时候可以与用户继续其他的交互。

## 定义使用CursorLoader的Activity

为了在Activity或者FragmentActivity中使用CursorLoader，需要实现[LoaderCallbacks](#)接口。CursorLoader会触发这些回调方法；这节课与下节课会详细介绍每一个回调方法。

例如，下面演示了FragmentActivity如何使用CursorLoader。

```
public class PhotoThumbnailFragment extends FragmentActivity implements  
    LoaderManager.LoaderCallbacks<Cursor> {  
    ...  
}
```

## 初始化查询

为了初始化查询，需要执行[LoaderManager.initLoader\(\)](#)。这个方法可以初始化后台任务。你可以在用户输入查询条件之后触发初始化的操作，如果你不需要用户输入数据作为查询条件，你可以触发这个方法在onCreate()或者onCreateView()。例如：

```
// Identifies a particular Loader being used in this component
private static final int URL_LOADER = 0;
...
/* When the system is ready for the Fragment to appear, this does
 * the Fragment's View
 */
public View onCreateView(
    LayoutInflater inflater,
    ViewGroup viewGroup,
    Bundle bundle) {
    ...
    /*
     * Initializes the CursorLoader. The URL_LOADER value is equivalent
     * to onCreateLoader().
     */
    getLoaderManager().initLoader(URL_LOADER, null, this);
    ...
}
```

**Note:** getLoaderManager()仅仅是在Fragment类中可以直接访问。为了在FragmentActivity中获取到LoaderManager，需要执行getSupportLoaderManager()。

## 开始查询

一旦后台任务被初始化好，它会执行你实现的回调方法[onCreateLoader\(\)](#)。为了启动查询任务，会在这个方法里面返回CursorLoader。你可以初始化一个空的CursorLoader然后使用它的方法来定义你的查询条件，或者你可以在初始化CursorLoader对象的时候就同时定义好查询条件：

```
/*
 * Callback that's invoked when the system has initialized the Loader
 * is ready to start the query. This usually happens when initLoader()
 * called. The loaderID argument contains the ID value passed to the
 * initLoader() call.
 */
@Override
public Loader<Cursor> onCreateLoader(int loaderID, Bundle bundle)
{
    /*
     * Takes action based on the ID of the Loader that's being created
     */
    switch (loaderID) {
        case URL_LOADER:
            // Returns a new CursorLoader
            return new CursorLoader(
                getActivity(),           // Parent activity context
                mDataUrl,               // Table to query
                mProjection,            // Projection to return
                null,                   // No selection clause
                null,                   // No selection arguments
                null                    // Default sort order
            );
        default:
            // An invalid id was passed in
            return null;
    }
}
```

一旦后台查询任务获取到了这个Loader对象，就开始在后台执行查询的任务。当查询完成之后，会执行[onLoadFinished\(\)](#)这个回调函数，关于这些内容会在下一节讲到。

编写:[kesenhoo](#)

校对:

# 处理查询的结果

正如前面一节课讲到的，你应该在[onCreateLoader\(\)](#)的回调里面使用CursorLoader执行加载数据的操作。接下去Loader会提供查询数据的结果给Activity或者FragmentActivity实现的[LoaderCallbacks.onLoadFinished\(\)](#)回调方法。这个回调方法的参数之一是[Cursor](#)，它包含了查询的数据。你可以使用Cursor对象来更新需要显示的数据或者进行下一步的处理。

除了[onCreateLoader\(\)](#)与[onLoadFinished\(\)](#)，你也需要实现[onLoaderReset\(\)](#)。这个方法在CursorLoader检测到[Cursor](#)上的数据发生变化的时候会被触发。当数据发生变化时，系统会触发重新查询的操作。

## Handle Query Results

为了显示CursorLoader返回的Cursor数据，需要使用实现AdapterView的类，并为这个类绑定一个实现了CursorAdapter的Adapter。系统会自动把Cursor中的数据显示到View上。

你可以在显示数据之前建立View与Adapter的关联。然后在[onLoadFinished\(\)](#)的时候把Cursor与Adapter进行绑定。一旦你把Cursor与Adapter进行绑定之后，系统会自动更新View。当Cursor上的内容发生改变的时候，也会触发这些操作。

例如：

```
public String[] mFromColumns = {
    DataProviderContract.IMAGE_PICTURENAME_COLUMN
};
public int[] mToFields = {
    R.id.PictureName
};
// Gets a handle to a List View
ListView mListview = (ListView) findViewById(R.id.dataList);
/*
 * Defines a SimpleCursorAdapter for the ListView
 *
 */
SimpleCursorAdapter mAdapter =
    new SimpleCursorAdapter(
        this, // Current context
        R.layout.list_item, // Layout for a single row
        null, // No Cursor yet
        mFromColumns, // Cursor columns to use
        mToFields, // Layout fields to use
        0 // No flags
    );
// Sets the adapter for the view
mListview.setAdapter(mAdapter);
...
/*
 * Defines the callback that CursorLoader calls
 * when it's finished its query
 */
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    ...
    /*
     * Moves the query results into the adapter, causing the
     * ListView fronting this adapter to re-display
     */
    mAdapter.changeCursor(cursor);
}
```

## Delete Old Cursor References

当Cursor失效的时候，CursorLoader会被重置。这通常发生在Cursor相关的数据改变的时候。在重新执行查询操作之前，系统会执行你的[onLoaderReset\(\)](#)回调方法。在这个回调方法中，你应该删除当前Cursor上的所有数据，避免发生内存泄露。一旦onLoaderReset()执行结束，CursorLoader就会重新执行查询操作。

例如：

```
/*
 * Invoked when the CursorLoader is being reset. For example, this
 * called if the data in the provider changes and the Cursor becomes
 */
@Override
public void onLoaderReset(Loader<Cursor> loader) {

    /*
     * Clears out the adapter's reference to the Cursor.
     * This prevents memory leaks.
     */
    mAdapter.changeCursor(null);
}
```

编写:[lttowq](#), 校对:

原文:<http://developer.android.com/training/scheduling/index.html>

# 管理设备的唤醒状态

当你的一个Android设备闲置时，屏幕将会变暗，然后关闭屏幕，最后关闭CPU。这是减少你的设备电量的快速消耗，然而，有些时候，你的应用程序可能需要不同的行为时间：

- 1.例如游戏或电影应用需要保持屏幕亮着
- 2.其他的应用也许不需要屏幕开着，但或许会请求CPU保持运行直到一个关键操作结束。

这节课描述如何在必要的时候保持设备唤醒但消耗它的电量。

# Lesson

## Keeping the Device Awake

- 学习当需要时如何保持屏幕和CPU唤醒，同时减少对电池的生命周期的影响。

## Scheduling Repeating Alarms

- 学习使用重复闹钟对于发生在生命周期外之应用的作业调度，即使应用没有运行或者设备处于睡眠状态。

编写:[lttowq](#)(未验证)

校对:

# 保持设备唤醒

为了避免电源消耗，一个Android设备闲置时会迅速进入睡眠状态。然而这里有些时间当一个应用需要唤醒屏幕或者CPU并且保持唤醒完成一些任务。

你采取的方法依赖于你的应用的需要。但是，一般规则是你应该使用最轻量级的方法对的应用，减小你的应用对系统资源的影响。接下来的部分描述怎样处理当设备默认睡眠的行为与你请求不相容的情况。

## 保持屏幕亮着

确定你的应用需要保持屏幕变亮，比如游戏与电影的应用。最好的方式是使用[FLAG\\_KEEP\\_SCREEN\\_ON](#)在你的Activity（仅在Activity不在Service或其他组件里）例如：

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);  
    }  
}
```

这个方法的优点像解锁([在Keep the CPU On讨论](#))，它不需要特殊的权限，平台正确管理你的用户在应用之间移动，不需要你的应用担心释放未使用资源。

另外一种实现在你的应用布局xml文件里，通过使用[android:keepScreenOn](#)属性：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:keepScreenOn="true">  
    ...  
</RelativeLayout>
```

使用[android:keepScreenOn="true"](#)与使用[FLAG\\_KEEP\\_SCREEN\\_ON](#)等效。你能无论使用哪个方法对你的应用都不错。编程方式设置该标志在你Activity的优点，它让你的编程后清除标志，从而使屏幕关闭该选项。

注意：你不需要清除[FLAG\\_KEEP\\_SCREEN\\_ON](#)便签除非你不在想屏幕呆在你正在运行的应用里面（例如：如果你想要屏幕延时在一个确定的周期静止）。窗口管理照顾确保正确事情发生当你的应用进入后台或者返回前台。但是如果你明确清除从而允许屏幕再次关闭，使用[clearFlags\(\)](#)；

`getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);`

# 保持CPU运行

如果你需要保持CPU运行为了完成一些工作在设备睡眠，你可以使用[PowerManager](#)系统服务特性回调唤醒锁。唤醒锁允许你应用控制本地设备电源状态。

创建和支持唤醒锁能有个引人注目的影响对本地设备电源周期。因此你使用唤醒锁仅仅当你确实需要，并保持他们尽可能短的时间尽可能。例如，你不应该需要使用唤醒锁在一个Activity中。以上描述，如果你想保持屏幕亮着在你的Activity，使用[FLAG KEEP SCRRE ON](#)。

一种合理情况对于使用唤醒锁可能在后台服务需要抢占一个唤醒锁保持CPU运行当屏幕关闭时。再一次，可是，这个实践应该最小因为它影响电池周期。

为了使用唤醒锁，首先你增加[WAKE\\_LOCK](#)权限在应用主要清单文件：

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

如果你的应用包括一个广播接收器使用这个服务做一些工作，你能管理你唤醒锁通过一个[WakefulBroadcastReceiver](#)作为描述[Using a WajefulBroadcastReceiver](#)这是优先的方法。如果你的应用不允许这个模式，这里告诉你之间设置唤醒锁：

```
PowerManager powerManager = (PowerManager) getSystemService(POWER_WAKELOCK);
WakeLock wakelock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
    "MyWakelockTag");
wakelock.acquire();
```

为了释放唤醒锁，使用[wakelock.release\(\)](#).这释放你的要求的CPU，它是重要的对于释放一个唤醒锁当你的应用使用完毕，避免消耗电量。

## 使用WakefulBroadcastReceiver

使用一个广播接收器结合一个服务让你管理循环周期在后台任务。[WakefulBroadcastReceiver](#)是一个特殊广播接收器类型小心创建和管理一个[PARTIAL\\_WAKE\\_LOCK](#)对于你的应用程序。[WakefulBroadcastReceiver](#)忽略任务对于一个[Service](#)（典型的一个[IntentService](#)），当确保设备不转换到睡眠状态。如果你不支持一个唤醒锁当转换工作对于一个服务，你实际上允许设备返回睡眠状态在工作完成之前。网络结果是应用可能没有完成正在做的工作直到一些任意点在未来，不是你想要的。

首先你增加[WakefulBroadcastReceiver](#)在你的主manifest文件里面，作为其他广播接收器。

```
<receiver android:name=".MyWakefulReceiver"></receiver>
```

代码开始在MyIntentService的方法[startWakefulService\(\)](#).这个方法是完成[startService\(\)](#).除了[WakefulBroadcastReceiver](#)支持唤醒锁当服务开启。通信停止与[startWakefulService\(\)](#)支持一个额外验证唤醒锁。

```
public class MyWakefulReceiver extends WakefulBroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
```

```
    Intent service = new Intent(COntext,MyIntentService.class);
    startWakefulService(context,service);
}
```

当服务完成，回调[MyWakefulReceiver.completeWakefulIntent\(\)](#)释放唤醒锁。[completeWakefulIntent\(\)](#)方法有它的相同参数停止从[WakefulBroadcastReceiver](#)：

```
public class MyWakefulReceiver extends IntentService{
    public static final int NOTIFICATION_ID = 1;
    private NotificationManager mNotificationManager;
    NotificationCompat.Builder builder;
    public MyIntentService() {
        Bundle extra = intent.getExtras();
        MyWakefulReceiver.completeWakefulIntent(intent);
    }
}
```

[下一课：调度重复闹钟](#)

编写:[lttowq](#)(未验证)

校对:

# 调度重复的闹钟

闹钟([基本类AlarmManager](#))给你一种方式执行基本时间操作你app生命周期外。例如你可以使用闹钟初始化一个长时间操作，例如开启一个服务一天为了下载天气预报。

## 闹钟的特性

- 它们让设置次数或间距点燃你意图
- 你可以使用它结合广播接收器开启服务和执行其他操作
- 它们执行在你的应用之外，所以你可以使用它触发事件或动作即使你的app没有运行或设备处于睡眠状态。
- 它们可以帮助你app最小化资源请求。你可以调度无需依赖的或者连续运行在后台的服务。

## 注意

对于定时操作保证结果在你app生命周期之内，替代可虑使用[Handler类](#)结合[定时器与线程](#)。这个方法给你Android更好的控制系统资源。

## 理解交替使用

一个重复闹钟是相对简单的机制和有限的灵活性。它或许不是最好的选择对于你的app，特别是如果你的app需要触发网络操作。一个坏的闹钟设计能造成电池漏电和让一个有意义的服务负载。

一个普通的情景对于触发一个你的app同步数据和一个服务器的生命周期外的操作。这个案例你可能冒险使用一个重复的闹钟。但是你自己服务器是本地你的app数据，使用[Google Cloud Messaging](#)(GCM)在结合你的[sync adapter](#)是更好解决方案比[AlarmManager](#)。一个同步适配器给你所有相同的调度选项作为[AlarmManager](#)，但是它提供你更灵活性。比如：一个同步可能基本在“新数据”消息从服务器或设备（细节见[Running a Sync Adapter](#)）。用户活动或静止，一天的时间或更久。看下面两个链接对于什么时候怎样使用GCM和同步适配器细节的讨论。1.[The App Clinic: Cricket\(需出墙\)](#)

2.[DevBytes: Efficient Data Transfers](#) (需出墙)

## 最好的练习

每个选择让你设计你的重复闹钟可以用序列在你的app使用或滥用系统资源。例如，想象一个流行的app和一个服务器同步。如果你同步操作在计时器的操作上，每一个app的实例同步在11: 00P.M，服务器负载造成高延时或者甚至“服务器拒绝”。下面是使用闹钟的建议：

- 增加随意（颤动）对任何网络请求触发作为一个重复闹钟的结果；
  - 做本地任务当一个闹钟触发。“本地任务”意味任何事情不需要敲击服务器或请求一个数据从服务器
  - 在相同的时间，调度闹钟包含网络请求点燃相同定时周期
- 保持你的闹钟频率最小；
- 不是必要的情况不要唤醒设备(这个行为被闹钟决定，细节在[Choose an alarm type](#))；
- 不要使用你的闹钟触发时间比它精确； 使用[setInexactRepeating](#))替代[setRepeating](#)).当你使用[setInexactRepeating](#))，Android同步重复闹钟从多个app和在相同的时间点燃它。这减少系统必须唤醒设备的数目，以此减少电源能耗。Android4.4 (API Level19)，所有的重复闹钟是不精确的。注意当[setInexactRepeating](#))是一个改进[setRepeating](#)),它能覆盖一个服务如果每个app的实例撞击服务器在相同的时间。因此对于网络请求，增加相同随意的闹钟、作为以上描述。
- 避免在你的闹钟基础上计时如果可能 重复的闹钟在预测触发器的没有扫描好的基础上。使用[ELAPSED\\_REALTIME](#)如果你能。不同的闹钟类型描述的细节在下面选项。

# 设置重复闹钟

下面的描述，重制闹钟作为一个好的选择有规律调度时间或数据备份。一个重复闹钟好如下特性：

- 一个闹钟类型的讨论见[Choose an alarm type](#).
- 一个触发时间。如果触发时间你制定在过去，闹钟触发立即执行。
- 闹钟间距。例如，某一天、每个小时、每5秒等等。
- 一个行将发生的意图是当闹钟被触发。当你设置一秒闹钟使用相同悬而未决的意图，它能替换原始闹钟。

## 选择一个闹钟的类型

其中第一个考虑是使用什么类型重置闹钟。这里有两个通用的计时器闹钟：“elapsed real time”和“real time clock”。elapsed real time使用“计时自从系统引导”作为引用，和real time clock使用UTC计时。这意味着elapsed real time适合设置一个闹钟在一段时间基础上(例如：一个闹钟点燃每30秒)且它不受地区和时区的影响。real time clock类型更好适配闹钟依赖当前时区。两个类型有“唤醒”版本，都能唤醒设备的CPU如果屏幕关闭。这确保闹钟将启动在调度时间、这是有用的如果你的ap有一个时间依赖。例如，如果它有一个限制窗口将启动当你的设备在下个唤醒。如果你简单的需要那种启动特殊意图(例如：每半小时)，使用其中elapsed real time类型。一般，这是更好的选择。如果你需要闹钟启动在一天中特殊的时间，然后选择某个计real time clock类型。注意，但是这个方法有些缺点——app或许不会翻译好对于其他地区，如果用户改变设备时间设置，它可能造成意外行为在你app。使用一个真实时间计时的闹钟类型也不会扫描好，综上，我们建议你使用elapsed real time。

这里列出类型：

- [ELAPSED\\_REALTIME](#)-点燃悬而未决意图在计时基础上从设备被引导，但是不需要唤醒设备。The elapsed 时间包括一些次数在设备处于睡眠期间。
- [ELAPSED\\_REALTIME\\_WAKEUP](#)唤醒设备并且启动悬而未决意图后指定过去的时间长度自从设备启动。
- [RTC](#)点燃悬而未决的意图在指定时间但是没有唤醒设备。
- [RTC\\_WAKEUP](#)唤醒设备在悬而未决意图在指定时间。

## ELAPSED\_REALTIME\_WAKEUP例子

这里有一些相同案例使用[ELAPSED\\_REALTIME\\_WAKEUP](#)

唤醒设备启动闹钟在30分钟内和每30分钟后：

```
// Hopefully your alarm will have a lower frequency than this!
alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    AlarmManager.INTERVAL_HALF_HOUR,
    AlarmManager.INTERVAL_HALF_HOUR, alarmIntent);
```

唤醒设备在启动一次（无重复）闹钟在每分钟内：

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager) context.getSystemService(Context.ALARM_SE
```

```
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);
alarmMgr.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    SystemClock.elapsedRealtime() +
    60 * 1000, alarmIntent);
```

## RTC案例

这里有相同案例使用[RTC\\_WAKEUP](#)

唤醒设备启动闹钟在约定时间2:00PM，一天内重复一次：

```
// Set the alarm to start at approximately 2:00 p.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 14);
// With setInexactRepeating(), you have to use one of the AlarmManager
// constants--in this case, AlarmManager.INTERVAL_DAY.
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
    AlarmManager.INTERVAL_DAY, alarmIntent);
```

唤醒设备启动闹钟在8:30am,每20分钟后在启动：

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);
// Set the alarm to start at 8:30 a.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 8);
calendar.set(Calendar.MINUTE, 30)
// setRepeating() lets you specify a precise custom interval--in this
// case, every 20 minutes.
alarmMgr.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
    1000 * 60 * 20, alarmIntent);
```

## 闹钟启动时间的设定

上面的描述，选择闹钟类型对于创建闹钟是第一步。第二不是设定闹钟启动的时间。对于大多数的app，[setInexactRepeating](#))是正确的选择。当你使用这个方法，Android同步多个不精确的重复闹钟和启动它们在相同的时间。这减少电源的能耗。

对真实的app有严格的要求-例如，闹钟需要精确启动在8:30am和每隔一小时之后-使用[setRepeating](#))，但是你应该避免使用精确地闹钟如果可能。

伴随[setInexactRepeating](#))，你不能指定客户意图一种方式你能[setRepeating](#))。你使用间距常量，例如[INTERVAL\\_FIFTEEN\\_MINUTES](#),[INTERVAL\\_DAY](#) 等等。见[AlarmManager](#)里面的完整的列表。

## 取消闹钟

依赖你app，你可能想报考一些取消闹钟的能力。取消闹钟回调([cancel](#))在你闹钟管理器，通过[PendingIntent](#)你不在启动，例如：

```
// If the alarm has been set, cancel it.  
if (alarmMgr!= null) {  
    alarmMgr.cancel(alarmIntent);  
}
```

## 启动闹钟当你设备启动时

默认的设置是所有的的闹钟被取消当一个设备关闭时。为了阻止发生，你可以你的app自动重启一个重复闹钟如果用户重启设备。这确保在[AlarmManager](#)将继续不需要用户手动启动闹钟。

这里的步骤：

1. 设置[RECEIVE\\_BOOT\\_COMPLETED](#)权限在你app主菜单（manifest）允许你的app接受[ACTION\\_BOOT\\_COMPLETED](#)在广播后系统完成启动（如果你app已经运行通过用户至少一次才有效）

```
<uses-permission android:name="android.permission.RECEIVE_BO
```

2. 实现一个[BroadcastReceiver](#)接收广播；

```
public class SampleBootReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {  
            // Set the alarm here.  
        }  
    }  
}
```

3. 增加接收器在你app的mainfest文件里面与一个意图过滤，过滤器在[ACTION\\_BOOT\\_COMPLETED](#)：

```
<receiver android:name=".SampleBootReceiver"  
         android:enabled="false">  
    <intent-filter>  
        <action android:name="android.intent.action.BOOT_COMPLETED" />  
    </intent-filter>  
</receiver>
```

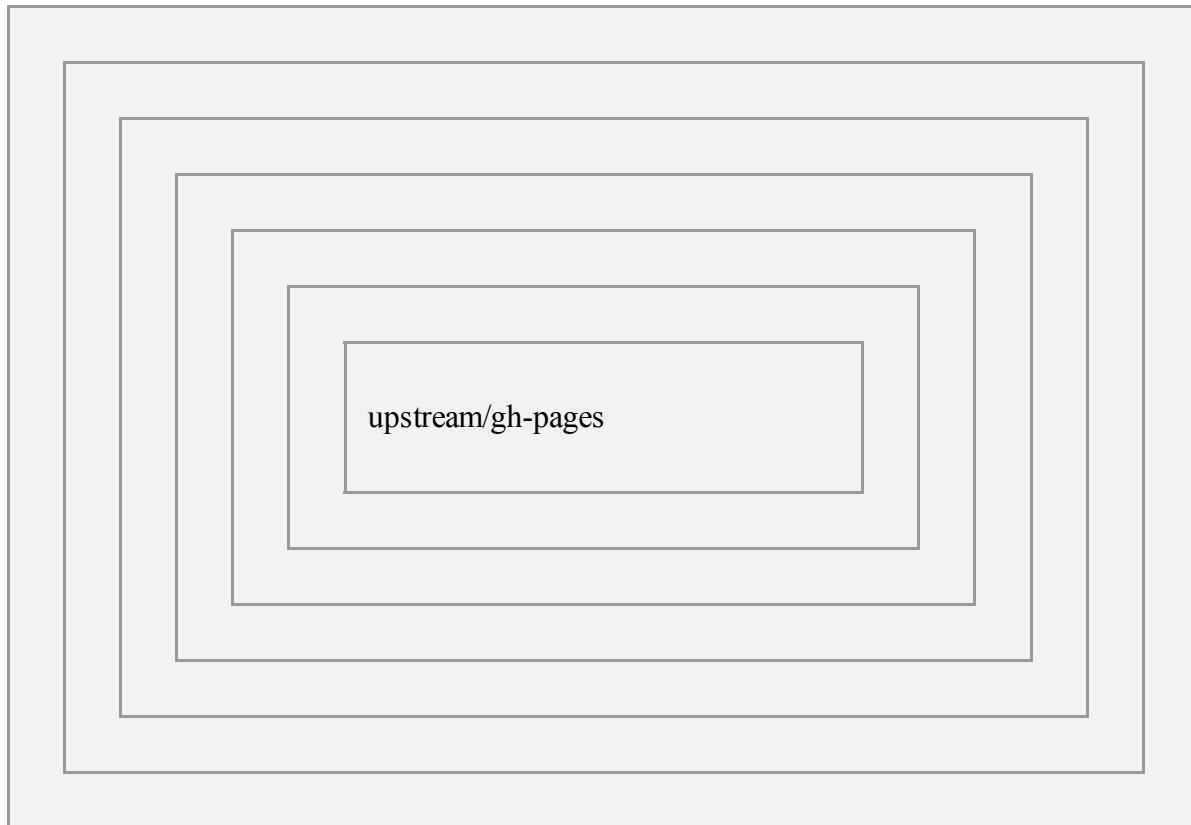
注意在mainfiest文件里，引导接收器设置`android:enabled="false"`。这意味着接收器将不会回调除非app程序显式地启用它。这是阻止不必要引导接收器的回调。你能启动一个接收器（例如：如果用户设置如下一个闹钟）如下：

```
ComponentName receiver = new ComponentName(context, SampleBootRece
PackageManager pm = context.getPackageManager();
pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP);
```

一旦你启动接器，它将保持启动，即使用户重启设备。总之,编程式启动接收器重写manifest里的设置，几多次重启。接收器将保持启动直到你的app不在使用它。你可以禁用一个接收器（例如：如果用户取消一个闹钟）如下：

```
ComponentName receiver = new ComponentName(context, SampleBootRece
PackageManager pm = context.getPackageManager();
pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
    PackageManager.DONT_KILL_APP);
```

## 下一课：最佳性能实践



编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/best-performance.html>

# 性能优化

These classes and articles help you build an app that's smooth, responsive, and uses as little battery as possible.

## [Managing Your App's Memory](#)

How to keep your app's memory footprint small in order to improve performance on a variety of mobile devices.

## [Performance Tips](#)

How to optimize your app's performance in various ways to improve its responsiveness and battery efficiency.

## [Improving Layout Performance](#)

How to identify problems in your app's layout performance and improve the UI responsiveness.

## [Optimizing Battery Life](#)

How to minimize the amount of power your app requires by adapting to current power conditions and performing power-hungry tasks at proper intervals.

## [Sending Operations to Multiple Threads](#)

How to improve the performance and scalability of long-running operations by dispatching work to multiple threads.

## [Keeping Your App Responsive](#)

How to keep your app responsive to user interaction so the UI does not lock-up and display an "Application Not Responding" dialog.

## [JNI Tips](#)

How to efficiently use the Java Native Interface with the Android NDK.

## [SMP Primer for Android](#)

Tips for coding Android apps on symmetric multiprocessor systems.

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/articles/memory.html>

# 管理应用的内存

Random Access Memory(RAM)在任何软件开发环境中都是一个很宝贵的资源。这一点在物理内存通常很有限的移动操作系统上，显得尤为突出。尽管Android的Dalvik虚拟机扮演了常规的垃圾回收的角色，但这并不意味着你可以忽视app的内存分配与释放的时机与地点。

为了GC能够从你的app中及时回收内存，你需要避免Memory Leaks(这通常由引用的不能释放而导致)并且在适当的时机(下面会讲到的lifecycle callbacks)来释放引用。对于大多数apps来说，Dalvik的GC会自动把离开活动线程的对象进行回收。

这篇文章会解释Android如何管理app的进程与内存分配，并且你可以在开发Android应用的时候主动的减少内存的使用。关于Java的资源管理机制，请参加其它书籍或者线上材料。如果你正在寻找如何分析你的内存使用情况的文章，请参考这里[Investigating Your RAM Usage](#)。

# 第1部分:Android是如何管理内存的

Android并没有提供内存的交换区(Swap space)，但是它有使用[paging](#)与[memory-mapping\(mmapping\)](#)的机制来管理内存。这意味着任何你修改的内存(无论是通过分配新的对象还是访问到mmapped pages的内容)都会贮存在RAM中，而且不能被paged out。因此唯一完整释放内存的方法是释放那些你可能hold住的对象的引用，这样使得它能够被GC回收。只有一种例外是：如果系统想要在其他地方进行reuse。

## 1)共享内存

Android通过下面几个方式在不同的Process中来共享RAM:

- 每一个app的process都是从同一个被叫做Zygote的进程中fork出来的。Zygote进程在系统启动并且载入通用的framework的代码与资源之后开始启动。为了启动一个新的程序进程，系统会fork Zygote进程生成一个新的process，然后在新的process中加载并运行app的代码。这使得大多数的RAM pages被用来分配给framework的代码与资源，并在应用的所有进程中进行共享。
- 大多数static的数据被mmapped到一个进程中。这不仅仅使得同样的数据能够在进程间进行共享，而且使得它能够在需要的时候被paged out。例如下面几种static的数据：
  - Dalvik code (by placing it in a pre-linked .odex file for direct mmapping)
  - App resources (by designing the resource table to be a structure that can be mmapped and by aligning the zip entries of the APK)
  - Traditional project elements like native code in .so files.
- 在许多地方，Android通过显式的分配共享内存区域(例如ashmem或者gralloc)来实现一些动态RAM区域的能够在不同进程间的共享。例如，window surfaces在app与screen compositor之间使用共享的内存，cursor buffers在content provider与client之间使用共享的内存。

关于如何查看app所使用的共享内存，请查看[Investigating Your RAM Usage](#)

## 2)分配与回收内存

这里有下面几点关于Android如何分配与回收内存的事实：

- 每一个进程的Dalvik heap都有一个限制的虚拟内存范围。这就是逻辑上讲的heap size，它可以随着需要进行增长，但是会有一个系统为它所定义的上限。
- 逻辑上讲的heap size和实际物理上使用的内存数量是不等的，Android会计算一个叫做Proportional Set Size(PSS)的值，它记录了那些和其他进程进行共享的内存大小。  
(假设共享内存大小是10M，一共有20个Process在共享使用，根据权重，可能认为其中有0.3M才是真正算是你的进程所使用的)
- Dalvik heap与逻辑上的heap size不吻合，这意味着Android并不会去做heap中的碎片整理用来关闭空闲区域。Android仅仅会在heap的尾端出现不使用的空间时才会做收缩逻辑heap size大小的动作。但是这并不是意味着被heap所使用的物理内存大小不能被收缩。在垃圾回收之后，Dalvik会遍历heap并找出不使用的pages，然后使用madvise把那些pages返回给kernal。因此，成对的allocations与deallocations大块的数据可以使得物理内存能够被正常的回收。然而，回收碎片化的内存则会使得效率低下很多，因为那些碎片化的分配页面也许会被其他地方所共享到。

## 3)限制应用的内存

为了维持多任务的功能环境，Android为每一个app都设置了一个硬性的heap size限制。准

确的heap size限制随着不同设备的不同RAM大小而各有差异。如果你的app已经到了heap的限制大小并且再尝试分配内存的话，会引起OutOfMemoryError的错误。

在一些情况下，你也许想要查询当前设备的heap size限制大小是多少，然后决定cache的大小。可以通过getMemoryClass()来查询。这个方法会返回一个整数，表明你的app heap size限制是多少megabates。

## 4) 切换应用

当用户在不同应用之间进行切换的时候，不是使用交换空间的办法。Android会把那些不含foreground组件的进程放到LRU cache中。例如，当用户刚刚启动了一个应用，这个时候为它创建了一个进程，但是当用户离开这个应用，这个进程并没有离开。系统会把这个进程放到cache中，如果用户后来回到这个应用，这个进程能够被resued，从而实现app的快速切换。

如果你的应用有一个被缓存的进程，它被保留在内存中，并且当前不再需要它了，这会对系统的整体性能有影响。因此当系统开始进入低内存状态时，它会由系统根据LRU的规则与其他因素选择杀掉某些进程，为了保持你的进程能够尽可能长久的被cached，请参考下面的章节学习何时释放你的引用。

更对关于不在foreground的进程是Android是如何决定kill掉哪一类进程的问题，请参考[Processes and Threads](#).

## 第2部分:你的应用该如何管理内存

你应该在开发过程的每一个阶段都考虑到RAM的有限性，甚至包括在开发开始之前的设计阶段。有许多种设计与实现方式，他们有着不同的效率，尽管是对同样一种技术的不断组合与演变。

为了使得你的应用效率更高，你应该在设计与实现代码时，遵循下面的技术要点。

### 1)珍惜Services资源

如果你的app需要在后台使用service，除非它被触发执行一个任务，否则其他时候都应该是非运行状态。同样需要注意当这个service已经完成任务后停止service失败引起的泄漏。

当你启动一个service，系统会倾向为了这个Service而一直保留它的Process。这使得process的运行代价很高，因为系统没有办法把Service所占用的RAM让给其他组件或者被Paged out。这减少了系统能够存放到LRU缓存当中的process数量，它会影响app之间的切换效率。它甚至会导致系统内存使用不稳定，从而无法继续Hold住所有目前正在运行的Service。

限制你的service的最好办法是使用[IntentService](#)，它会在处理完扔给它的intent任务之后尽快结束自己。更多信息，请阅读[Running in a Background Service](#)。

当一个service已经不需要的时候还继续保留它，这对Android应用的内存管理来说是最糟糕的错误之一。因此千万不要贪婪的使得一个Service持续保留。不仅仅是因为它会使得你的app因RAM的限制而性能糟糕，而且用户会发现那些行为奇怪的app并且卸载它。

### 2)当你的UI隐藏时释放内存

当用户切换到其它app并且你的app UI不再可见时，你应该释放你的UI上占用的任何资源。在这个时候释放UI资源可以显著的增加系统cached process的能力，它会对用户的质量体验有着直接的影响。

为了能够接收到用户离开你的UI时的通知，你需要实现Activity类里面的[onTrimMemory\(\)](#)回调方法。你应该使用这个方法来监听到[TRIM\\_MEMORY\\_UI\\_HIDDEN](#)级别，它意味着你的UI已经隐藏，你应该释放那些仅仅被你的UI使用的资源。

请注意：你的app仅仅会在所有UI组件的被隐藏的时候接收到onTrimMemory()的回调并带有参数TRIM\_MEMORY\_UI\_HIDDEN。这与onStop()的回调是不同的，onStop会在activity的实例隐藏时会执行，例如当用户从你的app的某个activity跳转到另外一个activity时onStop会被执行。因此你应该实现onStop回调，并且在此回调里面释放activity的资源，例如网络连接，unregister广播接收者。除非接收到[onTrimMemory\(TRIM\\_MEMORY\\_UI\\_HIDDEN\)](#)的回调，否者你不应该释放你的UI资源。这确保了用户从其他activity切回来时，你的UI资源仍然可用，并且可以迅速恢复activity。

### 3)当内存紧张时释放部分内存

在你的app生命周期的任何阶段，onTrimMemory回调方法同样可以告诉你整个设备的内存资源已经开始紧张。你应该根据onTrimMemory方法中的内存级别来进一步决定释放哪些资源。

- [TRIM\\_MEMORY\\_RUNNING\\_MODERATE](#):你的app正在运行并且不会被列为可杀死的。但是设备正运行于低内存状态下，系统开始开始激活杀死LRU Cache中的Process的机制。

- [TRIM\\_MEMORY\\_RUNNING\\_LOW](#): 你的app正在运行且没有被列为可杀死的。但是设备正运行于更低内存的状态下，你应该释放不用的资源用来提升系统性能，这会直接影响了你的app的性能。
- [TRIM\\_MEMORY\\_RUNNING\\_CRITICAL](#): 你的app仍在运行，但是系统已经把LRU Cache中的大多数进程都已经杀死，因此你应该立即释放所有非必须的资源。如果系统不能回收到足够的RAM数量，系统将会清除所有的LRU缓存中的进程，并且开始杀死那些之前被认为不应该杀死的进程，例如那个进程包含了一个运行中的Service。

同样，当你的app进程正在被cached时，你可能会接受到从onTrimMemory()中返回的下面的值之一：

- [TRIM\\_MEMORY\\_BACKGROUND](#): 系统正运行于低内存状态并且你的进程正处于LRU缓存名单中最不容易杀掉的位置。尽管你的app进程并不是处于被杀掉的高危险状态，系统可能已经开始杀掉LRU缓存中的其他进程了。你应该释放那些容易恢复的资源，以便于你的进程可以保留下来，这样当用户回退到你的app的时候才能够迅速恢复。
- [TRIM\\_MEMORY\\_MODERATE](#): 系统正运行于低内存状态并且你的进程已经接近LRU名单的中部位置。如果系统开始变得更加内存紧张，你的进程是有可能被杀死的。
- [TRIM\\_MEMORY\\_COMPLETE](#): 系统正运行与低内存的状态并且你的进程正处于LRU名单中最容易被杀掉的位置。你应该释放任何不影响你的app恢复状态的资源。

因为onTrimMemory()的回调是在**API 14**才被加进来的，对于老的版本，你可以使用[onLowMemory](#))回调来进行兼容。onLowMemory相当与TRIM\_MEMORY\_COMPLETE。

**Note:** 当系统开始清除LRU缓存中的进程时，尽管它首先按照LRU的顺序来操作，但是它同样会考虑进程的内存使用量。因此消耗越少的进程则越容易被留下来。

## 4) 检查你应该使用多少的内存

正如前面提到的，每一个Android设备都会有不同的RAM总大小与可用空间，因此不同设备为app提供了不同大小的heap限制。你可以通过调用[getMemoryClass\(\)](#)来获取你的app的可用heap大小。如果你的app尝试申请更多的内存，会出现OutOfMemory的错误。

在一些特殊的情景下，你可以通过在manifest的application标签下添加largeHeap=true的属性来声明一个更大的heap空间。如果你这样做，你可以通过[getLargeMemoryClass\(\)](#)来获取到一个更大的heap size。

然而，能够获取更大heap的设计本意是为了一小部分会消耗大量RAM的应用(例如一个大图片的编辑应用)。不要轻易的因为你需要使用大量的内存而去请求一个大的heap size。只有当你清楚的知道哪里会使用大量的内存并且为什么这些内存必须被保留时才去使用large heap. 因此请尽量少使用large heap。使用额外的内存会影响系统整体的用户体验，并且会使得GC的每次运行时间更长。在任务切换时，系统的性能会变得大打折扣。

另外，large heap并不一定能够获取到更大的heap。在某些有严格限制的机器上，large heap的大小和通常的heap size是一样的。因此即使你申请了large heap，你还是应该通过执行getMemoryClass()来检查实际获取到的heap大小。

## 5) 避免bitmaps的浪费

当你加载一个bitmap时，仅仅需要保留适配当前屏幕设备分辨率的数据即可，如果原图高

于你的设备分辨率，需要做缩小的动作。请记住，增加bitmap的尺寸会对内存呈现出2次方的增加，因为X与Y都在增加。

**Note:**在Android 2.3.x (API level 10)及其以下, bitmap对象是的pixel data是存放在native内存中的，它不便于调试。然而，从Android 3.0(API level 11)开始，bitmap pixel data是分配在你的app的Dalvik heap中，这提升了GC的工作并且更加容易Debug。因此如果你的app使用bitmap并在旧的机器上引发了一些内存问题，切换到3.0以上的机器上进行Debug。

## 6) 使用优化的数据容器

利用Android Framework里面优化过的容器类，例如[SparseArray](#), [SparseBooleanArray](#), 与[LongSparseArray](#). 通常的HashMap的实现方式更加消耗内存，因为它需要一个额外的实例对象来记录Mapping操作。另外，SparseArray更加高效在于他们避免了对key与value的autobox自动装箱，并且避免了装箱后的解箱。

## 7) 请注意内存开销

对你所使用的语言与库的成本与开销有所了解，从开始到结束，在设计你的app时谨记这些信息。通常，表面上看起来无关痛痒(innocuous)的事情也许实际上会导致大量的开销。例如：

- Enums的内存消耗通常是static constants的2倍。你应该尽量避免在Android上使用enums。
- 在Java中的每一个类(包括匿名内部类)都会使用大概500 bytes。
- 每一个类的实例花销是12-16 bytes。
- 往HashMap添加一个entry需要额一个额外占用的32 bytes的entry对象。

## 8) 请注意代码“抽象”

通常，开发者使用抽象简单的作为"好的编程实践",因为抽象能够提升代码的灵活性与可维护性。然而，抽象会导致一个显著的开销:通常他们需要同等量的代码用于可执行。那些代码会被map到内存中。因此如果你的抽象没有显著的提升效率，应该尽量避免他们。

## 9) 为序列化的数据使用nano protobufs

[Protocol buffers](#)是由Google为序列化结构数据而设计的，一种语言无关，平台无关，具有良好扩展性的协议。类似XML，却比XML更加轻量，快速，简单。如果你需要为你的数据实现协议化，你应该在客户端的代码中总是使用nano protobufs。通常的协议化操作会生成大量繁琐的代码，这容易给你的app带来许多问题:增加RAM的使用量，显著增加APK的大小，更慢的执行速度，更容易达到DEX的字符限制。

关于更多细节，请参考[protobuf readme](#)的"Nano version"章节。

## 10) Avoid dependency injection frameworks

使用类似[Guice](#)或者[RoboGuice](#)等framework injection包是很有效的，因为他们能够简化你的代码。

RoboGuice 2 smoothes out some of the wrinkles in your Android development experience and makes things simple and fun. Do you always forget to check for null when you getIntent().getExtras()? RoboGuice 2 will help you. Think casting findViewById() to a

TextView shouldn't be necessary? RoboGuice 2 is on it. RoboGuice 2 takes the guesswork out of development. Inject your View, Resource, System Service, or any other object, and let RoboGuice 2 take care of the details.

然而，那些框架会通过扫描你的代码执行许多初始化的操作，这会导致你的代码需要大量的RAM来map代码。但是mapped pages会长时间的被保留在RAM中。

## 11) 谨慎使用external libraries

很多External library的代码都不是为移动网络环境而编写的，在移动客户端则显示的效率不高。至少，当你决定使用一个external library的时候，你应该针对移动网络做繁琐的porting与maintenance的工作。

即使是针对Android而设计的library，也很可能是很危险的，因为每一个library所做的事情都是不一样的。例如，其中一个lib使用的是nano protobufs，而另外一个使用的是micro protobufs。那么这样，在你的app里面有2种protobuf的实现方式。这样的冲突同样可能发生在输出日志，加载图片，缓存等等模块里面。

同样不要陷入为了1个或者2个功能而导入整个library的陷阱。如果没有一个合适的库与你的需求相吻合，你应该考虑自己去实现，而不是导入一个大而全的解决方案。

## 12) 优化整体性能

官方有列出许多优化整个app性能的文章：[Best Practices for Performance](#)。这篇文章就是其中之一。有些文章是讲解如何优化app的CPU使用效率，有些是如何优化app的内存使用效率。

你还应该阅读[optimizing your UI](#)来为layout进行优化。同样还应该关注lint工具所提出的建议，进行优化。

## 13) 使用ProGuard来剔除不需要的代码

[ProGuard](#)能够通过移除不需要的代码，重命名类，域与方法等对代码进行压缩，优化与混淆。使用ProGuard可以使你的代码更加紧凑，这样能够使用更少mapped代码所需要的RAM。

## 14) 对最终的APK使用zipalign

在编写完所有代码，并通过编译系统生成APK之后，你需要使用[zipalign](#)对APK进行重新校准。如果你不做这个步骤，会导致你的APK需要更多的RAM，因为一些类似图片资源的东西不能被mapped。

**Notes:** Google Play不接受没有经过zipalign的APK。

## 15) 分析你的RAM使用情况

一旦你获取到一个相对稳定的版本后，需要分析你的app整个生命周期内使用的内存情况，并进行优化，更多细节请参考[Investigating Your RAM Usage](#)。

## 16) 使用多进程

如果合适的话，有一个更高级的技术可以帮助你的app管理内存使用：通过把你的app组件

切分成多个组件，运行在不同的进程中。这个技术必须谨慎使用，大多数app都不应该运行在多个进程中。因为如果使用不当，它会显著增加内存的使用，而不是减少。当你的app需要在后台运行与前台一样的大量的任务的时候，可以考虑使用这个技术。

一个典型的例子是创建一个可以长时间后台播放的Music Player。如果整个app运行在一个进程中，当后台播放的时候，前台的那些UI资源也没有办法得到释放。类似这样的app可以切分成2个进程：一个用来操作UI，另外一个用来后台的Service.

你可以通过在manifest文件中声明'android:process'属性来实现某个组件运行在另外一个进程的操作。

```
<service android:name=".PlaybackService"
         android:process=":background" />
```

更多关于使用这个技术的细节，请参考原文，链接如下。

<http://developer.android.com/training/articles/memory.html>

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/articles/perf-tips.html>

# Performance Tips

这篇文章主要是介绍了一些小细节的优化技巧，当这些小技巧综合使用起来的时候，对于整个App的性能提升还是有作用的，只是不能较大幅度的提升性能而已。选择合适的算法与数据结构才应该是你首要考虑的因素，在这篇文章中不会涉及这方面。你应该使用这篇文章中的小技巧作为平时写代码的习惯，这样能够提升代码的效率。

通常来说，高效的代码需要满足下面两个规则：

- 不要做冗余的动作
- 如果能避免，尽量不要分配内存

代码的执行效果会受到设备CPU,设备内存,系统版本等诸多因素的影响。为了确保代码能够在不同设备上都运行良好，需要最大化代码的效率。

## 避免创建不必要的对象

虽然GC可以回收不用的对象，可是为这些对象分配内存，并回收它们同样是需要耗费资源的。因此请尽量避免创建不必要的对象，有下面一些例子来说明这个问题：

- 如果你需要返回一个String对象，并且你知道它最终会需要连接到一个StringBuffer，请修改你的实现方式，避免直接进行连接操作，应该采用创建一个临时对象来做这个操作。
- 当从输入的数据集中抽取出Strings的时候，尝试返回原数据的substring对象，而不是创建一个重复的对象。

一个稍微激进点的做法是把所有多维的数据分解成1维的数组：

- 一组int数据要比一组Integer对象要好很多。可以得知，两组1维数组要比一个2维数组更加的有效率。同样的，这个道理可以推广至其他原始数据类型。
- 如果你需要实现一个数组用来存放(Foo,Bar)的对象，尝试分解为Foo[]与Bar[]要比(Foo,Bar)好很多。(当然，为了某些好的API的设计，可以适当做一些妥协。但是在自己的代码内部，你应该多多使用分解后的容易。)

通常来说，需要避免创建更多的对象。更少的对象意味着更少的GC动作，GC会对用户体验有比较直接的影响。

## 选择Static而不是Virtual

如果你不需要访问一个对象的值域,请保证这个方法是static类型的,这样方法调用将快15%-20%。这是一个好的习惯,因为你可以从方法声明中得知调用无法改变这个对象的状态。

## 常量声明为Static Final

先看下面这种声明的方式

```
static int intValue = 42;  
static String strValue = "Hello, world!";
```

编译器会使用方法来初始化上面的值，之后访问的时候会需要先到它那里查找，然后才返回数据。我们可以使用static final来提升性能：

```
static final int intValue = 42;  
static final String strValue = "Hello, world!";
```

这时再也不需要上面的那个方法来做多余的查找动作了。所以，请尽可能的为常量声明为**static final**类型的。

## 避免内部的Getters/Setters

像C++等native language,通常使用getters(`i = getCount()`)而不是直接访问变量(`i = mCount`).这是编写C++的一种优秀习惯，而且通常也被其他面向对象的语言所采用，例如C#与Java，因为编译器通常会做inline访问，而且你需要限制或者调试变量，你可以在任何时候在getter/setter里面添加代码。然而，在Android上，这是一个糟糕的写法。Virtual method的调用比起直接访问变量要耗费更多。那么合理的做法是：在面向对象的设计当中应该使用getter/setter，但是在类的内部你应该直接访问变量. 没有JIT(Just In Time Compiler)时，直接访问变量的速度是调用getter的3倍。有JIT时,直接访问变量的速度是通过getter访问的7倍。请注意，如果你使用[ProGuard](#), 你可以获得同样的效果，因为ProGuard可以为你inline accessors.

## 使用增强的For循环

请比较下面三种循环的方法：

```
static class Foo {
    int mSplat;
}

Foo[] mArray = ...

public void zero() {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one() {
    int sum = 0;
    Foo[] localArray = mArray;
    int len = localArray.length;

    for (int i = 0; i < len; ++i) {
        sum += localArray[i].mSplat;
    }
}

public void two() {
    int sum = 0;
    for (Foo a : mArray) {
        sum += a.mSplat;
    }
}
```

- zero()是最慢的，因为JIT没有办法对它进行优化。
- one()稍微快些。
- two()在没有做JIT时是最快的，可是如果经过JIT之后，与方法one()是差不多一样快的。它使用了增强的循环方法for-each。

所以请尽量使用for-each的方法，但是对于ArrayList，请使用方法one()。

## 使用包级访问而不是内部类的私有访问

参考下面一段代码

```
public class Foo {  
    private class Inner {  
        void stuff() {  
            Foo.this.doStuff(Foo.this.mValue);  
        }  
    }  
  
    private int mValue;  
  
    public void run() {  
        Inner in = new Inner();  
        mValue = 27;  
        in.stuff();  
    }  
  
    private void doStuff(int value) {  
        System.out.println("Value is " + value);  
    }  
}
```

Foo\$Inner里面有访问外部类的一个变量。这样的做法会给系统造成额外的麻烦，请尽量避免。

## 避免使用**float**类型

Android系统中**float**类型的数据存取速度是**int**类型的一半，尽量优先采用**int**类型。

## 使用库函数

尽量使用System.arraycopy()等一些封装好的库函数，它的效率是手动编写copy实现的9倍多。

**Tip:** Also see Josh Bloch's Effective Java, item 47.

## 谨慎使用native函数

当你需要把已经存在的native code迁移到Android，请谨慎使用JNI。如果你要使用JNI,请学习[JNI Tips](#)

## 关于性能的误区

在没有做JIT之前，使用一种确切的数据类型确实要比抽象的数据类型速度要更有效率。（例如，使用HashMap要比Map效率更高。）有误传效率要高一倍，实际上只是6%左右。而且，在JIT之后，他们直接并没有大多差异。

## 关于测量

上面文档中出现的数据是Android的实际运行效果。我们可以用[Traceview](#)来测量，但是测量的数据是没有经过JIT优化的，所以实际的效果应该是要比测量的数据稍微好些。

关于如何测量与调试，还可以参考下面两篇文章：

- [Profiling with Traceview and dmtracedump](#)
- [Analysing Display and Performance with Systrace](#)

编写: [allenlsy](#), 校对:

原文: <http://developer.android.com/training/improving-layouts/index.html>

# 提升Layout的性能

Layout 是 Android 应用中直接影响用户体验的关键部分。如果实现的不好，你的 Layout 会导致稍微费内存一点的程序变得非常卡顿。Android SDK 带有帮助你找到 Layout 性能问题的工具。使用它，你会用最小的内存空间实现流畅的 UI。

# 课程

## 优化 Layout 的层级

就像一个复杂的网页会减慢载入速度，你的 Layout 结构如果太复杂，也会造成性能问题。本节教你如何使用 SDK 自带工具来检查 Layout 并找到瓶颈。

## 使用 标签重用 Layout

如果你的程序 UI 在不同地方重复使用某个 Layout，那本节教你如何创建高效的，可重用的 Layout 部件，并把它们“包含”到 UI Layout 中。

## 按需载入视图

除了简单的把一个 Layout 包含到另一个中，你可能还想在程序开始后，仅当你的 Layout 对用户可见时才开始载入。本节告诉你如何分步载入 Layout 来提高初始性能。

## Making ListView Scrolling Smooth

如果你有一个包含复杂或者每个项 (item) 包含很多数据的 ListView，那么上下滚动的性能可能会降低。本节给你一些关于如何把滚动变得更流畅的提示。

编写: [allenlsy](#)

校对:

# 优化layout的层级

一个常见的误区是，用最基础的 Layout 结构可以使 Layout 性能提高。然而，你的程序的每个组件和 Layout 都需要初始化、布置位置和绘制。例如，嵌套的 LinearLayout 可能会使得 View 的层级结构很深。此外，嵌套使用了 layout\_weight 参数的 LinearLayout 的计算量会尤其大，因为每个子元素都需要被测量两次。这对需要多次重复 inflate 的 Layout 尤其需要注意，比如使用 ListView 或 GridView 时。

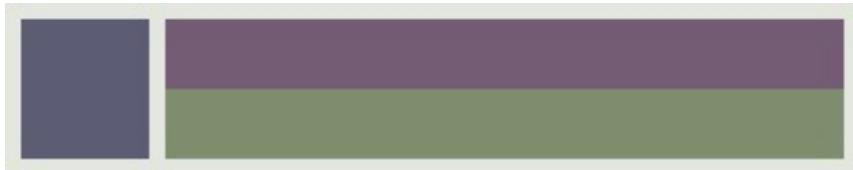
本课中，你将学习使用 [Hierarchy Viewer](#)（层级浏览器，译者注）和 [Layoutopt](#)（Layout优化工具，译者注）来检查和优化 Layout。

## 检查 Layout

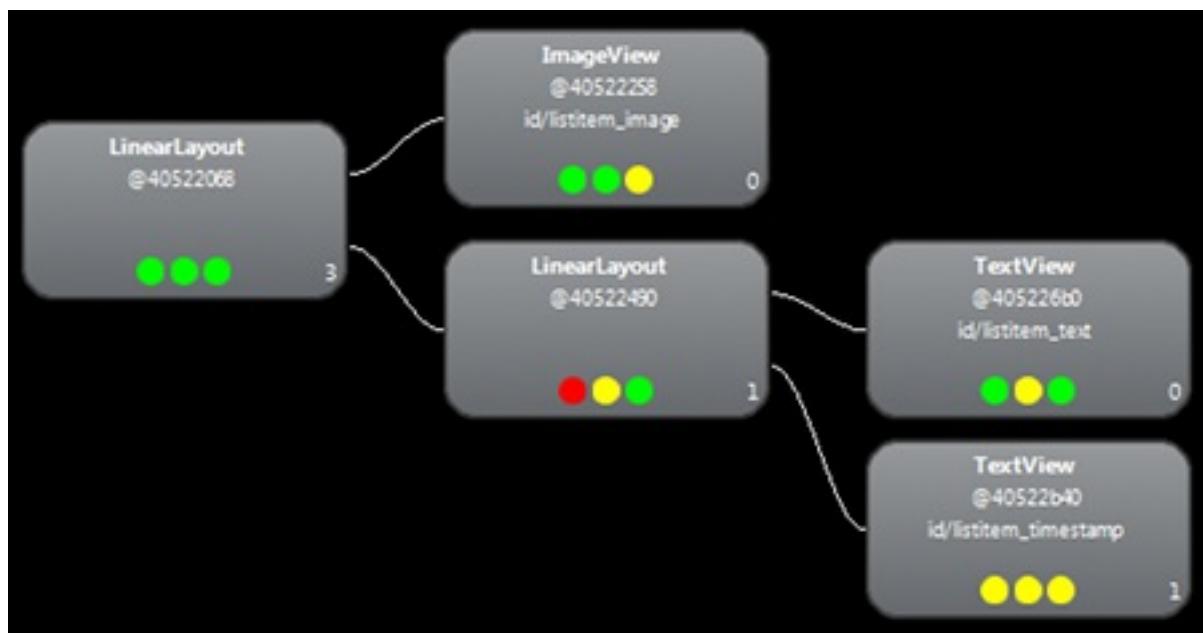
Android SDK 工具带有有一个叫做 [Hierarchy Viewer](#) 的工具，能够在程序运行时分析 Layout。你可以用这个工具找到 Layout 的性能瓶颈。

Hierarchy Viewer 会让你选择设备或者模拟器上正在运行的进程，然后显示其 Layout 的树型结构。每个块上的交通灯分别代表了它在测量、布置和绘画时的性能，帮你找出瓶颈部分。

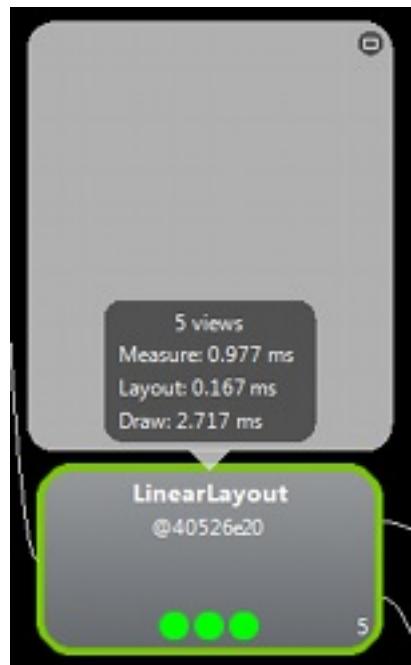
比如，下图是 ListView 中一个列表项的 Layout。它的左边是一个小位图，右边是两个层叠的文字（Text）。像这种需要被多次 inflate 的 Layout，优化它们会有事半功倍的效果。



hierarchyviewer 这个工具在 <sdk>/tools/ 中。当打开时，它显示一张可使用设备的列表，和它正在运行的组件。点击 **Load View Hierarchy** 来查看所选组件的层级。比如，下图就是前一个图中所示 Layout 的层级关系。



这张图中，你可以看到一个三层结构，其中在布置右下角的 TextView 的时候有问题。点击其中的项会显示每个步骤所花费的时间。这样，谁花了多长时间在什么哪个步骤上面，就清晰可见了。

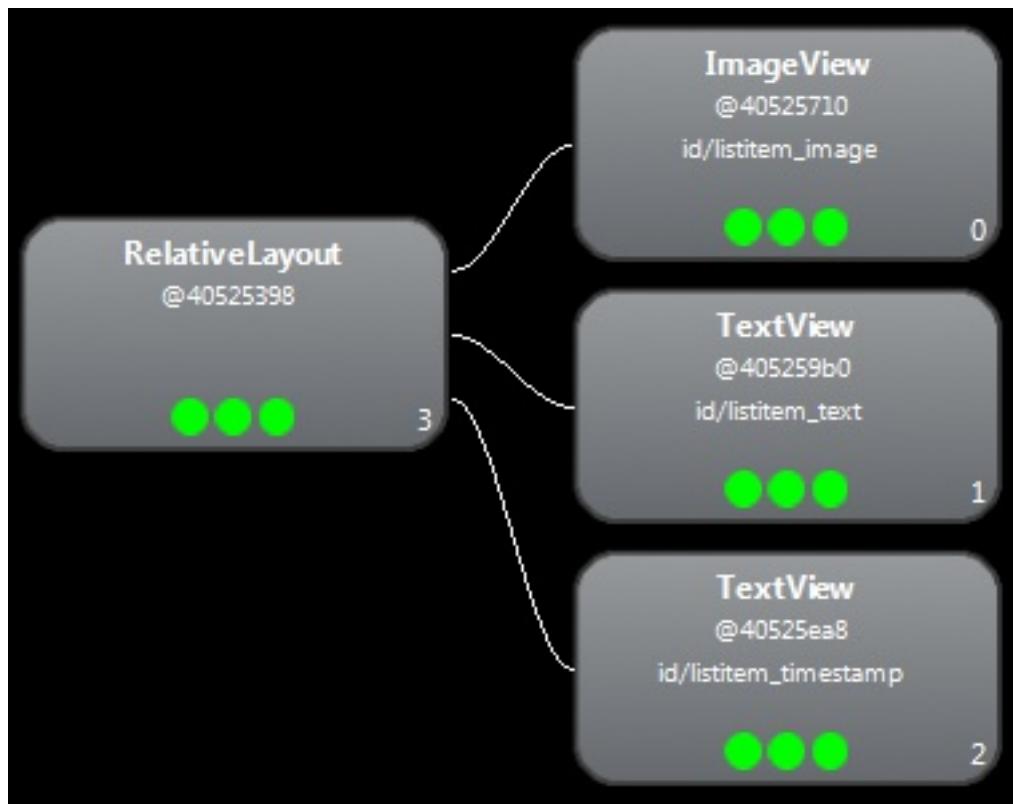


可以看到，渲染一个完整的列表项的时间就是：

- 测量: 0.977ms
- 布置: 0.167ms
- 绘制: 2.717ms

## 修正 Layout

因为上面的 Layout 性能太慢，原因在这个嵌套的 LinearLayout，解决的办法可能是将 Layout 层级变浅变宽，而不是又窄又深。RelativeLayout 作为根节点时就可以达到目的。所以，当换成基于 RelativeLayout 的设计时，你的 Layout 变成了两层。新的 Layout 长成这样：



现在渲染列表项的时间：

- 测量: 0.598ms
- 布置: 0.110ms
- 绘制: 2.146ms

可能看起来是很小的进步，但是由于它对列表中每个项都有效，这个时间要翻倍。

更明显的性能差距，是当使用在 LinearLayout 中使用 `layout_weight` 的时候，因为会减慢“测量”的速度。这只是一个正确使用各种 Layout 的例子，当你使用 `layout_weight` 时一定要慎重。

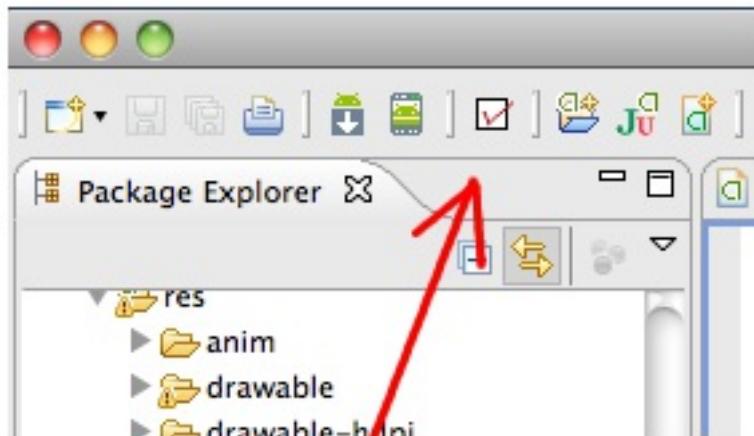
# 使用 Lint

大部分叫做 lint 的编程工具，都是类似于代码规范的检测工具。比如 JSLint, CSSLinkt, JSONLint 等等。译者注。

经常运行 [Lint](#) 工具来检查 Layout 可能的优化方法，是个很好的实践。Lint 已经取代了 Layoutopt 工具，它拥有更强大的功能。Lint 中包含的一些检测[规则](#)有：

- 使用复合 drawable —— 用一个 drawable 替代一个包含 ImageView 和 TextView 的 LinearLayout 时会更有效率。
- 合并根 frame —— 如果 FrameLayout 是 Layout 的根节点，并且没有使用padding 或者背景等，那么用 merge 标签替代他们会稍微高效些。
- 没用的子节点 —— 一个没有子节点或者背景的 Layout 应该被去掉，来提高性能
- 没用的父节点 —— 一个节点如果只有一个子节点，并且它不是 ScrollView 或根节点，并且它没有背景，这样的节点应该直接被子节点取代。
- 太深的 Layout —— Layout 的嵌套层数太深对性能有很大影响。尝试使用更扁平的 Layout，比如 RelativeLayout 或 GridLayout 来提高性能。一般最多不超过 10 层。

另一个使用 Lint 的好处就是，它内置于 ADT Eclipse (ADT16+) 中。Lint 在你导出apk文件、编辑XML文件或使用 Layout 编辑器时会自动运行。手动强制运行 Lint，在 Eclipse 的工具栏中按这个：



当使用 Eclipse 的时候，Lint 有自动修复、提示建议和直接跳转到问题处的功能。如果你没有使用 Eclipse，你也可以通过命令行运行 Lint。更多信息，参见 [tools.android.com](http://tools.android.com)。

编写: [allenlsy](#)

校对:

# 使用include标签重用layouts

虽然 Android 提供很多小的可重用的交互组件，你仍然可能需要重用复杂一点的组件，这也许会用到 Layout。为了高效重用整个的 Layout，你可以使用 `<include/>` 和 `<merge/>` 标签把其他 Layout 嵌入当前 Layout。

重用 Layout 非常强大，它让你可以创建复杂的可重用 Layout。比如，一个 yes/no 按钮面板，或者带有文字的自定义进度条。这还意味着，任何在多个 Layout 中重复出现的元素可以被提取出来，单独管理设计，再添加到 Layout 中。所以，当你要添加一个自定义 View 来实现单独的 UI 组件时，你可以更简单的直接重用某个 Layout 文件。

## 创建可重用 Layout

如果你已经知道你需要重用的 Layout，先创建以德新的 Layout XML 文件。比如，以下是一个来自 G-Kenya codelab 的 Layout，定义了一个需要添加到每个 Activity 中的标题栏 (titlebar.xml)：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/titlebar_bg">

    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gafricalogo" />
</FrameLayout>
```

根节点就是你想添加入的 Layout 类型。当需要显示这个标题栏时，就应该添加这个 Layout。

## 使用 <include> 标签

使用 <include> 标签，可以在 Layout 中添加可重用的组件。比如，这里有一个来自 G-Kenya codelab 的 Layout 需要包含上面的那个标题栏：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/app_bg"
    android:gravity="center_horizontal">

    <include layout="@layout/titlebar"/>

    <TextView android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:text="@string/hello"
              android:padding="10dp" />

    ...

</LinearLayout>
```

你也可以覆写被 include 元素的根元素的所有 Layout 参数（任何 android:layout\_\* 属性）。比如：

```
<include android:id="@+id/news_title"
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         layout="@layout/title"/>
```

然而，如果你要在 <include> 中覆写某些属性，你必须先覆写 android:layout\_height 和 android:layout\_width。

## 使用 <merge> 标签

<merge /> 标签在你嵌套 Layout 时取消了 UI 层级中冗余的 ViewGroup。比如，如果你有一个 Layout 是一个竖直方向的 LinearLayout，其中包含两个连续的 View 可以在别的 Layout 中重用，那么你会做一个 LinearLayout 来包含这两个 View，以便重用。不过，当使用另一个 LinearLayout 来嵌套这个可重用的 LinearLayout 时，这种嵌套 LinearLayout 的方式除了减慢你的 UI 性能外没有任何意义。

为了避免这种情况，你可以用 <merge> 元素来替代可重用 Layout 的根节点。例如：

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/add"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/delete"/>

</merge>
```

现在，当你要将这个 Layout 包含到另一个 Layout 中时（并且使用了 <include/> 标签），系统会直接把两个 Button 放到 Layout 中，而不会有太多的 Layout 被嵌套。

编写: [allenlsy](#)

校对:

# 按需加载视图

有时你的 Layout 会用到不怎么重用的复杂视图。不管它是列表项 细节，进度显示器，或是撤销时的提示信息，你可以仅在需要的时候载入它们，提高 UI 渲染速度。

## 定义 ViewStub

[ViewStub](#) 是一个轻量的视图，不需要大小信息，也不会在被加入的 Layout 中绘制任何东西。每个 ViewStub 只需要设置 android:layout 属性来指定需要被 inflate 的 Layout 类型。

以下 ViewStub 是一个半透明的进度条覆盖层。功能上讲，它应该只在新的数据项被导入到应用程序时可见。

```
<ViewStub  
    android:id="@+id/stub_import"  
    android:inflatedId="@+id/panel_import"  
    android:layout="@layout/progress_overlay"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom" />
```

## 载入 ViewStub Layout

当你要载入用 ViewStub 声明的 Layout 时，要么用 `setVisibility(View.VISIBLE)` 设置它的可见性，要么调用其 `inflate()` 方法。

```
((ViewStub) findViewById(R.id.stub_import)).setVisibility(View.VIS  
// or  
View importPanel = ((ViewStub) findViewById(R.id.stub_import)).inf
```

注意：`inflate()` 方法会在渲染完成后返回被 `inflate` 的视图，所以你不需要再调用 `findViewById()` 去查找这个元素，如果你需要和这个 Layout 交互的话。

一旦 ViewStub 可见或是被 `inflate` 了，ViewStub 元素就不存在了。取而代之的是被 `inflate` 的 Layout，其 id 是 ViewStub 上的 `android:inflatedId` 属性。（ViewStub 的 `android:id` 属性仅在 ViewStub 可见以前可用）

注意：ViewStub 的一个缺陷是，它目前不支持使用 `<merge/>` 标签的 Layout。

编写: [allenlsy](#)

校对:

# 使得ListView滑动流畅

保持程序流畅的关键，是让主线程（UI 线程）不要进行大量运算。你要确保在其他线程执行磁盘读写、网络读写或是 SQL 操作等。为了测试你的应用的状态，你可以启用 [StrictMode](#)（严格模式，译者注）。

## 使用后台线程

你应该把主线程中的耗时间的操作，提取到一个后台线程（也叫做“工人线程”，英文 worker thread）中，使得主线程只关注 UI 绘画。很多时候，使用  [AsyncTask](#) 是一个简单的在主线程以外进行操作的方法。系统使用一个全局队列来排列所有的 AsyncTask，并且一次运行其中的  [execute\(\)](#) 方法。这个行为是全局的，这意味着你不需要考虑自己定义线程池的事情。

在以下的例子中，一个 AsyncTask 被用于在后台线程载入图片，并在载入完成后把图片放入 UI。当图片正在载入时，它还回显示一个进度转盘。

```
// Using an AsyncTask to load the slow images in a background thread
new AsyncTask<ViewHolder, Void, Bitmap>() {
    private ViewHolder v;

    @Override
    protected Bitmap doInBackground(ViewHolder... params) {
        v = params[0];
        return mFakeImageLoader.getImage();
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        super.onPostExecute(result);
        if (v.position == position) {
            // If this item hasn't been recycled already, hide the
            // progress and set and show the image
            v.progress.setVisibility(View.GONE);
            v.icon.setVisibility(View.VISIBLE);
            v.icon.setImageBitmap(result);
        }
    }
}.execute(holder);
```

从 Android 3.0 (API level 11) 开始，AsyncTask 有个新特性，那就是它可以在多个 CPU 核上运行。你可以调用  [executeOnExecutor\(\)](#) 来自动根据核数，在多核上执行任务，而不是调用  [execute\(\)](#)。

## 在 View Holder 中填入视图对象

你的代码可能在 ListView 滑动时经常使用 `findViewById()`，这样会降低性能。即使是 Adapter 返回一个用于回收的 inflate 后的视图，你仍然需要查看这个元素并更新它。避免频繁调用 `findViewById()` 的方法之一，就是使用 View Holder（视图占位符）设计模式。

一个 ViewHolder 对象存储了他的标签下的每个视图。这样你不用频繁查找这个元素。第一，你需要创建一个类来存储你会用到的视图。比如：

```
static class ViewHolder {  
    TextView text;  
    TextView timestamp;  
    ImageView icon;  
    ProgressBar progress;  
    int position;  
}
```

然后，在 Layout 的类中生成一个 ViewHolder 对象：

```
ViewHolder holder = new ViewHolder();  
holder.icon = (ImageView) convertView.findViewById(R.id.listitem_i  
holder.text = (TextView) convertView.findViewById(R.id.listitem_te  
holder.timestamp = (TextView) convertView.findViewById(R.id.listit  
holder.progress = (ProgressBar) convertView.findViewById(R.id.prog  
convertView.setTag(holder);
```

这样你就可以轻松获取每个视图，而不是用 `findViewById()` 来不断查找视图，节省了宝贵的运算时间。

编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/monitoring-device-state/index.html>

# 优化电池寿命

显然，手持设备的电量需要引起很大的重视。通过这一系列的课程，可以学会如何根据设备电池状态来改变App的某些行为与功能。

通过在断开连接时关闭后台服务，在电量减少时减少更新数据的频率等等操作可以在不影响用户体验的前提下，确保App对电池寿命的影响减到最小。

编写:[kesenhoo](#)

校对:

# **Monitoring the Battery Level and Charging State**[监测电池的电量与充电状态]

当你想通过改变后台更新操作的频率来减少对电池寿命的影响，那么先手需要检查当前电量与充电状态。

电池的电量与是否在充电状态会影响到一个程序去执行更新的操作。当设备在进行AC充电时，程序做任何操作都不太会受到电量的影响，所以在大多数时候，我们可以在设备充电时做很多想做的事情（刷新数据，下载文件等），相反的，如果设备没有在充电状态，那么我们就需要尽量减少设备的更新操作等来延长电池的续航能力。

同样的，我们可以通过检查电池目前的电量来减少甚至停止一些更新操作。

## 1) Determine the Current Charging State [判断当前充电状态]

[BatteryManager](#)会广播一个带有电池与充电详情的[Sticky Intent](#) 因为广播的是一个sticky intent，那么不需要注册BroadcastReceiver。仅仅只需要简单的call一个参null参数的registerReceiver()方法。

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);
```

我们可以从intent里面提取出当前的充电状态与是否通过USB或者AC充电器来充电。

```
// Are we charging / charged?
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS,
    boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
        status == BatteryManager.BATTERY_STATUS_FULL;

// How are we charging?
int chargePlug = battery.getIntExtra(BatteryManager.EXTRA_PLUGGED,
    boolean usbCharge = chargePlug == BATTERY_PLUGGED_USB;
    boolean acCharge = chargePlug == BATTERY_PLUGGED_AC;
```

我们可以从intent里面提取出当前的充电状态与是否通过USB或者AC充电器来充电。通常的做法是在使用AC充电时最大化后台更新操作，在使用USB充电时降低更新操作，不在充电状态时，最小化更新操作。

## 2) Monitor Changes in Charging State[监测充电状态的改变]

充电状态随时可能改变，显然，需要通过检查充电状态的改变来通知App改变某些行为。

BatteryManager会在设备连接或者断开充电器的时候广播一个action。接收到这个广播是很重要的，即使我们的app没有在运行。特别是在是否接收这个广播会对app决定后台更新频率产生影响的前提下。因此很有必要在manifest文件里面注册一个监听来接收ACTION\_POWER\_CONNECTED与ACTION\_POWER\_DISCONNECTED的intent。

```
<receiver android:name=".PowerConnectionReceiver">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_POWER_CONNE
        <action android:name="android.intent.action.ACTION_POWER_DISCO
    </intent-filter>
</receiver>
```

```
public class PowerConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int status = intent.getIntExtra(BatteryManager.EXTRA_STATU
        boolean isCharging = status == BatteryManager.BATTERY_STATU
                           status == BatteryManager.BATTERY_STATU

        int chargePlug = intent.getIntExtra(BatteryManager.EXTRA_P
        boolean usbCharge = chargePlug == BATTERY_PLUGGED_USB;
        boolean acCharge = chargePlug == BATTERY_PLUGGED_AC;
    }
}
```

### 3) Determine the Current Battery Level[判断当前电池电量]

在一些情况下，获取到当前电池电量也是很有帮助的。我们可以在获知电量少于某个级别的时候减少某些后台操作。我们可以从获取到电池状态的intent中提取出电池电量与容量等信息。

```
int level = battery.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = battery.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
float batteryPct = level / (float) scale;
```

## 4)Monitor Significant Changes in Battery Level[检测电量的有效改变]

虽然我们可以轻易的不间断的检测电池状态，但是这并不是必须的。通常来说，我们只需要检测电量的某些有效改变，特别是设备在进入或者离开低电量状态的时候。下面的例子，电量监听器只会在设备电量进入低电量或者离开低电量的时候才会触发，仅仅需要监听ACTION\_BATTERY\_LOW与ACTION\_BATTERY\_OKAY.

```
<receiver android:name=".BatteryLevelReceiver">
<intent-filter>
    <action android:name="android.intent.action.ACTION_BATTERY_LOW"/>
    <action android:name="android.intent.action.ACTION_BATTERY_OKAY">
    </intent-filter>
</receiver>
```

通常都需要在进入低电量的情况下，关闭所有后台程序来维持设备的续航，因为这个时候做任何的更新等操作都是无谓的，很可能在你还没有来的及操作刚才更新的内容的时候就自动关机了。 In many cases, the act of charging a device is coincident with putting it into a dock. The next lesson shows you how to determine the current dock state and monitor for changes in device docking.

编写:[kesenhoo](#)

校对:

# Determining and Monitoring the Docking State and Type [判断并监测设备的停驻状态与类型]

在上一课中有这样一句话：In many cases, the act of charging a device is coincident with putting it into a dock.

在很多情况下，为设备充电也是一种设备停驻方式

Android设备能够有好几种停驻状态。包括车载模式，家庭模式与数字对战模拟模式[这个有点奇怪]。停驻状态通常与充电状态是非常密切关联的。

停驻模式会如何影响更新频率这完全取决于app的设置。我们可以选择在桌面模式下频繁的更新数据也可以选择在车载模式下关闭更新操作。相反的，你也可以选择在车载模式下最大化更新交通数据频率。

停驻状态也是以sticky intent方式来广播的，这样可以通过查询intent里面的数据来判断是否目前处于停驻状态，处于哪种停驻状态。

## 1) Determine the Current Docking State [判断当前停驻状态]

因为停驻状态的广播内容也是sticky intent(ACTION\_DOCK\_EVENT)，所以不需要注册 BroadcastReceiver。

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_DOCK_EVENT);
Intent dockStatus = context.registerReceiver(null, ifilter);

int dockState = battery.getIntExtra(EXTRA_DOCK_STATE, -1);
boolean isDocked = dockState != Intent.EXTRA_DOCK_STATE_UNDOCKED;
```

## 2) Determine the Current Dock Type[判断当前停驻类型]

一共有下面4中停驻类型：

- Car
  - Desk
  - Low-End (Analog) Desk: API level 11开始才有
  - High-End (Digital) Desk: API level 11开始才有

通常仅仅需要像下面一样检查当前停驻类型：

### 3)Monitor for Changes in the Dock State or Type[监测停驻状态或者类型的改变]

只需要像下面一样注册一个监听器：

```
<action android:name="android.intent.action.ACTION.Dock_Event"/>
```

Receiver获取到信息后可以像上面那样检查需要的数据。

编写:[kesenhoo](#)

校对:

# **Determining and Monitoring the Connectivity Status[判断并监测网络连接状态]**

通常我们会有一些计划的任务，比如重复闹钟，后台定时启动的任务等。但是如果我们的网络没有连接上，那么就没有必要启动那些需要连接网络的任务。我们可以使用ConnectivityManager来检查是否连接上网络，是何种网络。[通过网络的连接状况改变，相应的改变app的行为，减少无谓的操作，从而延长设备的续航能力]

## 1) Determine if You Have an Internet Connection[判断当前是否有网络连接]

显然如果没有网络连接，那么就没有必要做那些需要联网的事情。下面是一个检查是否有网络连接的例子：

```
ConnectivityManager cm =  
    (ConnectivityManager) context.getSystemService(Context.CONN  
  
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();  
boolean isConnected = activeNetwork.isConnectedOrConnecting();
```

## 2) Determine the Type of your Internet Connection[判断连接网络的类型]

设备通常可以有移动网络，WiMax,Wi-Fi与以太网连接等类型。通过查询当前活动的网络类型，可以根据网络的带宽做适合的事情。

```
boolean isWiFi = activeNetwork.getType() == ConnectivityManager.TY
```

使用移动网络会比Wi-Fi花费代价更大，所以多数情况下，在移动网络情况下减少一些数据的获取操作，同样，一些像下载文件等操作需要等有Wi-Fi的情况下才开始。如果已经关闭了更新操作，那么需要监听网络切换，当有比较好的网络时重新启动之前取消的操作。

### 3)Monitor for Changes in Connectivity[监测网络连接的切换]

当网络连接被改变的时候，ConnectivityManager会broadcast CONNECTIVITY\_ACTION ("android.net.conn.CONNECTIVITY\_CHANGE") 的动作消息。我们需要在manifest文件里面注册一个带有像下面action一样的Receiver:

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
```

通常网络的改变会比较频繁，我们没有必要不间断的注册监听网络的改变。通常我们会在有Wi-Fi的时候进行下载动作，若是网络切换到移动网络则通常会暂停当前下载，监听到恢复到Wi-Fi的情况则开始恢复下载。

编写:[kesenhoo](#)

校对:

# Manipulating Broadcast Receivers On Demand[按需操控广播接收者]

简单的方法是为我们监测的状态创建一个BroadcastReceiver，并在manifest中为每一个状态进行注册监听。然后，每一个Receiver根据当前设备的状态来简单重新安排下一步执行的任务。[这句话感觉理解有点问题]

上面那个方法的副作用是，设备会在每次收到广播都被唤醒，这有点超出期望，因为有些广播是不希望唤醒设备的。

更好的方法是根据程序运行情况开启或者关闭广播接收者。这样的话，那些在manifest中注册的receivers仅仅会在需要的时候才被激活。

## 1) Toggle and Cascade State Change Receivers to Improve Efficiency[切换是否开启这些状态Receivers来提高效率]

我们可以使用PackageManager来切换任何一个在manifest里面定义好的组件的开启状态。可以使用下面的方法来开启或者关闭任何一个broadcast receiver:

```
ComponentName receiver = new ComponentName(context, myReceiver.class);
PackageManager pm = context.getPackageManager();
pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP)
```

使用这种技术，如果我们判断到网络链接已经断开，那么可以在这个时候关闭除了connectivity-change的之外的所有Receivers。

相反的，一旦重新建立网络连接，我们可以停止监听网络链接的改变。而仅仅在执行需要联网的操作之前判断当前网络是否可以用即可。

你可以使用上面同样的技术来暂缓一个需要带宽的下载操作。可以开启receiver来监听是否连接上Wi-Fi来重新开启下载的操作。

编写:[AllenZheng1991](#)

校对:

原文地址: <http://developer.android.com/training/multiple-threads/index.html>

# 多线程操作

如果你把一个会长时间运行且数据密集的操作分割成一个个小的操作，然后运行在多个线程上，它的执行速度和效率都会得到提高。在一个有多核CPU的设备上，系统可以并行运行多个线程，而不是让每个操作在等待其它操作执行完后再伺机执行。例如，如果要解码大量的图片文件并以缩略图的形式把图片显示在屏幕上，当你每个解码单独用一个线程去执行时，会发现速度快了很多。

这一部分向你展示了如何在一个Android应用中创建和使用多线程，以及如何使用一个线程池对象（thread pool object）。你还将了解到如何通过代码运行一个线程，以及如何让你创建的一个线程和UI线程之间进行通信。

# 课程

## 在一个线程中执行一段特定的代码

学习如何通过实现[Runnable](#)接口定义一个线程类，让你写的代码能在单独的一个线程中执行。

## 为多线程创建线程池

学习如何创建一个能管理线程池和任务队列的对象，需要使用一个叫[ThreadPoolExecutor](#)的类。

## 在线程池中的一个线程里执行代码

学习如何让线程池里的一个线程执行一个任务。

## 与UI线程通信

学习如何让线程池里的一个普通线程与UI线程进行通信。

编写: [AllenZheng1991](#)

校对:

原文地址: <http://developer.android.com/training/multiple-threads/define-runnable.html>

# 在一个线程中执行一段特定的代码

这一课向你展示了如何通过实现[Runnable](#)接口得到一个能在重写的[Runnable.run\(\)](#)方法中执行一段代码的单独的线程。另外你可以传递一个[Runnable](#)对象到另一个对象，然后这个对象可以把它附加到一个线程，并执行它。一个或多个执行特定操作的[Runnable](#)对象有时也被称为一个任务。

[Thread](#)和[Runnable](#)只是两个基本的线程类，通过他们能发挥的作用有限，但是他们是强大的Android线程类的基础类，例如Android中的[HandlerThread](#),[AsyncTask](#)和[IntentService](#)都是以它们为基础。[Thread](#)和[Runnable](#)同时也是[ThreadPoolExecutor](#)类的基础。[ThreadPoolExecutor](#)类能自动管理线程和任务队列，甚至可以并行执行多个线程。

## 定义一个实现Runnable接口的类

直接了当的方法是通过实现[Runnable](#)接口去定义一个线程类。例如：

```
public class PhotoDecodeRunnable implements Runnable {  
    ...  
    @Override  
    public void run() {  
        /*  
         * 把你想要在线程中执行的代码写在这里  
         */  
        ...  
    }  
    ...  
}
```

## 实现run()方法

在一个类里，[Runnable.run\(\)](#) 包含执行了的代码。通常在[Runnable](#) 中执行任何操作都是可以的，但需要记住的是，因为[Runnable](#) 不会在UI线程中运行，所以它不能直接更新UI对象，例如[View](#) 对象。为了与UI对象进行通信，你必须使用另一项技术，在[Communicate with the UI Thread\(与UI线程进行通信\)](#) 这一课中我们会对其进行描述。

在[run\(\)](#)方法的开始的地方通过调用参数为[THREAD\\_PRIORITY\\_BACKGROUND](#) 的[Process.setThreadPriority\(\)](#)方法来设置线程使用的是后台运行优先级。这个方法减少了通过[Runnable](#)创建的线程和和UI线程之间的资源竞争。

你还应该通过在[Runnable](#) 自身中调用[Thread.currentThread\(\)](#)来存储一个引用到[Runnable](#) 对象的线程。

下面这段代码展示了如何创建[run\(\)](#)方法：

```
class PhotoDecodeRunnable implements Runnable {  
    ...  
    /*  
     * 定义要在这个任务中执行的代码  
     */  
    @Override  
    public void run() {  
        // 把当前的线程变成后台执行的线程  
        android.os.Process.setThreadPriority(android.os.Process.TH  
        ...  
        /*  
         * 在PhotoTask实例中存储当前线程，以至于这个实例能中断这个线程  
         */  
        mPhotoTask.setImageDecodeThread(Thread.currentThread());  
        ...  
    }  
    ...  
}
```

编写: [AllenZheng1991](#)

校对:

原文地址: <http://developer.android.com/training/multiple-threads/create-threadpool.html>

# 为多线程创建线程池

在前面的课程中展示了如何在单独的一个线程中执行一个任务。如果你的线程只想执行一次，那么上一课的内容已经能满足你的需要了。如果你想在一个数据集中重复执行一个任务，而且你只需要一个执行运行一次。这时，使用一个[IntentService](#)将能满足你的需求。为了在资源可用的时候自动执行任务，或者允许不同的任务同时执行（或前后两者），你需要提供一个管理线程的集合。为了做这个管理线程的集合，使用一个[ThreadPoolExecutor](#)实例，当一个线程在它的线程池中变得不受约束时，它会运行队列中的一个任务。为了能执行这个任务，你所需要做的就是把它加入到这个队列。

一个线程池能运行多个并行的任务实例，因此你要能保证你的代码是线程安全的，从而你需要给会被多个线程访问的变量附上同步代码块(synchronized block)。当一个线程在对一个变量进行写操作时，通过这个方法将能阻止另一个线程对该变量进行读取操作。典型的，这种情况会发生在静态变量上，但同样它也能突然发生在任意一个只实例化一次。为了学到更多的相关知识，你可以阅读[进程与线程\(Processes and Threads\)](#)这一API指南。

# 定义线程池类

在自己的类中实例化[ThreadPoolExecutor](#)类。在这个类里需要做以下事：

## 1. 为线程池使用静态变量

为了有一个单一控制点用来限制CPU或网络资源[Runnable](#)类型，你可能想有一个能管理每一个线程的线程池，且每个线程都会是单个实例。比如，你可以把这个作为一部分添加到你的全局变量的声明中去：

```
public class PhotoManager {  
    ...  
    static {  
        ...  
        // Creates a single static instance of PhotoManager  
        sInstance = new PhotoManager();  
    }  
    ...  
}
```

## 2. 使用私有构造方法

让构造方法私有从而保证这是一个单例，这意味着你不需要在同步代码块(synchronized block)中额外访问这个类：

```
public class PhotoManager {  
    ...  
    /**  
     * Constructs the work queues and thread pools used to download  
     * and decode images. Because the constructor is marked private,  
     * it's unavailable to other classes, even in the same package.  
     */  
    private PhotoManager() {  
        ...  
    }  
}
```

## 3. 通过调用线程池类里的方法开启你的任务

在线程池类中定义一个能添加任务到线程池队列的方法。例如：

```
public class PhotoManager {  
    ...  
    // Called by the PhotoView to get a photo  
    static public PhotoTask startDownload(  
        PhotoView imageView,  
        boolean cacheFlag) {  
        ...  
        // Adds a download task to the thread pool for execution  
        sInstance.  
            mDownloadThreadPool.  
            execute(downloadTask.getHTTPDownloadRunnable());  
        ...  
    }  
}
```

```
}
```

#### 4. 在构造方法中实例化一个**Handler**, 且将它附加到你APP的UI线程。

一个**Handler**允许你的APP安全地调用UI对象（例如[View](#)对象）的方法。大多数UI对象只能从UI线程安全地被修改。这个方法将会在[与UI线程进行通信\(Communicate with the UI Thread\)](#)这一课中进行详细的描述。例如：

```
private PhotoManager() {  
    ...  
    // Defines a Handler object that's attached to the UI thread.  
    mHandler = new Handler(Looper.getMainLooper()) {  
        /*  
         * handleMessage() defines the operations to perform when  
         * the Handler receives a new Message to process.  
         */  
        @Override  
        public void handleMessage(Message inputMessage) {  
            ...  
        }  
        ...  
    }  
}
```

# 确定线程池的参数

一旦有了整体的类结构,你可以开始定义线程池了。为了初始化一个[ThreadPoolExecutor](#)对象, 你需要提供以下数值:

## 1. 线程池的初始化大小和最大的大小

这个是指最初分配给线程池的线程数量, 以及线程池中允许的最大线程数量。在线程池中拥有的线程数量主要取决于你的设备的CPU内核数。这个数字可以从系统环境中获得:

```
public class PhotoManager {  
    ...  
    /*  
     * Gets the number of available cores  
     * (not always the same as the maximum number of cores)  
     */  
    private static int NUMBER_OF_CORES =  
        Runtime.getRuntime().availableProcessors();  
}
```

这个数字可能不反映设备的物理核心数量, 因为一些设备根据系统负载关闭了一个或多个CPU内核, 对于这样的设备, [availableProcessors\(\)](#) 方法返回的是处于活动状态的内核数量, 可能少于设备的实际内核总数。

## 2. 线程保持活动状态的持续时间和时间单位

这个是指线程被关闭前保持空闲状态的持续时间。这个持续时间通过时间单位值进行解译, 是[TimeUnit\(\)](#)中定义的常量之一。

## 3. 一个任务队列

这个传入的队列由[ThreadPoolExecutor](#)获取的[Runnable](#)对象组成。为了执行一个线程中的代码, 一个线程池管理者从先进先出的队列中取出一个[Runnable](#)对象且把它附加到一个线程。当你创建线程池时需要提供一个队列对象, 这个队列对象类必须实现[BlockingQueue](#)接口。为了满足你的APP的需求, 你可以选择一个Android SDK中已经存在的队列实现类。为了学习更多相关知识, 你可以看一下[ThreadPoolExecutor](#)类的概述。下面是一个使用[LinkedBlockingQueue](#)实现的例子:

```
public class PhotoManager {  
    ...  
    private PhotoManager() {  
        ...  
        // A queue of Runnables  
        private final BlockingQueue<Runnable> mDecodeWorkQueue;  
        ...  
        // Instantiates the queue of Runnables as a LinkedBlockingQueue  
        mDecodeWorkQueue = new LinkedBlockingQueue<Runnable>();  
        ...  
    }  
    ...  
}
```

## 创建一个线程池

为了创建一个线程池，可以通过调用[ThreadPoolExecutor\(\)](#)构造方法初始化一个线程池管理者对象，这样就能创建和管理一组可约束的线程了。如果线程池的初始化大小和最大大小相同，[ThreadPoolExecutor](#)在实例化的时候就会创建所有的线程对象。例如：

```
private PhotoManager() {
    ...
    // Sets the amount of time an idle thread waits before termination
    private static final int KEEP_ALIVE_TIME = 1;
    // Sets the Time Unit to seconds
    private static final TimeUnit KEEP_ALIVE_TIME_UNIT = TimeUnit.SECONDS;
    // Creates a thread pool manager
    mDecodeThreadPool = new ThreadPoolExecutor(
        NUMBER_OF_CORES,           // Initial pool size
        NUMBER_OF_CORES,           // Max pool size
        KEEP_ALIVE_TIME,
        KEEP_ALIVE_TIME_UNIT,
        mDecodeWorkQueue);
}
```

编写: [AllenZheng1991](#)

校对:

原文地址: <http://developer.android.com/training/multiple-threads/run-code.html>

# 在线程池中的一个线程里执行代码

在前面的课程中向你展示了如何去定义一个可以管理线程池且能在他们中执行任务代码的类。在这一课中我们将向你展示如何在线程池中执行任务代码。为了达到这个目的，你需要把任务添加到线程池的工作队列中去，当一个线程变成可运行状态时，[ThreadPoolExecutor](#)从工作队列中取出一个任务，然后在该线程中执行。

这节课同时也向你展示了如何去停止一个正在执行的任务，这个任务可能在刚开始执行时是你想要的，但后来发现它所做的工作并不是你所需要的。你可以取消线程正在执行的任务，而不是浪费处理器的运行时间。例如你正在从网络上下载图片且对下载的图片进行了缓存，当检测到正在下载的图片在缓存中已经存在时，你可能希望停止这个下载任务。当然，这都取决于你怎么样去编写你的APP，因为可能压根你就不会在开始下载前进行检测。

## 在线程池的一个线程中执行一个任务

为了在一个特定的线程池的线程里开启一个任务，可以通过调用[ThreadPoolExecutor.execute\(\)](#)，它需要提供一个[Runnable](#)类型的参数，这个调用会把该任务添加到这个线程池中的工作队列。当一个空闲的线程进入可执行状态时，线程管理者从工作队列中取出等待时间最长的那个任务，并且在线程中执行它。

```
public class PhotoManager {  
    public void handleState(PhotoTask photoTask, int state) {  
        switch (state) {  
            // The task finished downloading the image  
            case DOWNLOAD_COMPLETE:  
                // Decodes the image  
                mDecodeThreadPool.execute(  
                    photoTask.getPhotoDecodeRunnable());  
                ...  
            }  
            ...  
        }  
        ...  
    }  
}
```

当[ThreadPoolExecutor](#)在一个线程中开启一个[Runnable](#)后，它会自动调用[Runnable](#)的[run\(\)](#)方法。

## 中断一段正在执行的代码

为了停止执行一个任务，你必须中断执行这个任务的线程。在准备做这件事之前，当你创建一个任务时，你需要存储处理该任务的线程。例如：

```
class PhotoDecodeRunnable implements Runnable {  
    // Defines the code to run for this task  
    public void run() {  
        /*  
         * Stores the current Thread in the  
         * object that contains PhotoDecodeRunnable  
         */  
        mPhotoTask.setImageDecodeThread(Thread.currentThread());  
        ...  
    }  
    ...  
}
```

想要中断一个线程，你可以调用[Thread.interrupt\(\)](#)。需要注意的是这些线程对象都被系统控制，系统可以在你的APP进程之外修改他们。因为这个原因，在你要中断一个线程时，你需要把这段代码放在一个同步代码块中对这个线程的访问加锁来解决这个问题。例如：

```
public class PhotoManager {  
    public static void cancelAll() {  
        /*  
         * Creates an array of Runnables that's the same size as the  
         * thread pool work queue  
         */  
        Runnable[] runnableArray = new Runnable[mDecodeWorkQueue.size()];  
        // Populates the array with the Runnables in the queue  
        mDecodeWorkQueue.toArray(runnableArray);  
        // Stores the array length in order to iterate over the array  
        int len = runnableArray.length;  
        /*  
         * Iterates over the array of Runnables and interrupts each one.  
         */  
        synchronized (sInstance) {  
            // Iterates over the array of tasks  
            for (int runnableIndex = 0; runnableIndex < len; runnableIndex++) {  
                // Gets the current thread  
                Thread thread = runnableArray[taskArrayIndex].mThread;  
                // If the Thread exists, post an interrupt to it  
                if (null != thread) {  
                    thread.interrupt();  
                }  
            }  
        }  
        ...  
    }  
}
```

在大多数情况下，通过调用[Thread.interrupt\(\)](#)能立即中断这个线程，然而他只能停止那些处

于等待状态的线程，却不能中断那些占据CPU或者长时间连接网络的任务。为了避免减慢或造成系统死锁，在尝试进行中断操作前，你应该测试当前是否存在处于挂起状态的中断请求：

```
/*
 * Before continuing, checks to see that the Thread hasn't
 * been interrupted
 */
if (Thread.interrupted()) {
    return;
}
...
// Decodes a byte array into a Bitmap (CPU-intensive)
BitmapFactory.decodeByteArray(
    imageBuffer, 0, imageBuffer.length, bitmapOptions);
...
```

编写:[AllenZheng1991](#)

校对:

原文地址: <http://developer.android.com/training/multiple-threads/communicate-ui.html>

# 与UI线程通信

在前面的课程中你学习了如何在一个被[ThreadPoolExecutor](#)管理的线程中开启一个任务。最后这一节课将会向你展示如何从执行的任务中发送数据给运行在UI线程中的对象。这个功能允许你的任务可以做后台工作，然后把得到的结果数据转移给UI元素使用，例如位图数据。

任何一个APP都有自己特定的一个线程用来运行UI对象，比如[View](#)对象，这个线程我们称之为UI线程。只有运行在UI线程中的对象能访问运行在其它线程中的对象。因为你的任务执行的线程来自一个线程池而不是执行在UI线程，所以他们不能访问UI对象。为了把数据从一个后台线程转移到UI线程，需要使用一个运行在UI线程里的[Handler](#)。

## 在UI线程中定义一个Handler

[Handler](#)属于Android系统的线程管理框架的一部分。一个[Handler](#)对象用于接收消息和执行处理消息的代码。一般情况下，如果你为一个新线程创建了一个[Handler](#)，你还需要创建一个[Handler](#)，让它与一个已经存在的线程关联，用于这两个线程之间的通信。如果你把一个[Handler](#)关联到UI线程，处理消息的代码就会在UI线程中执行。

你可以在一个用于创建你的线程池的类的构造方法中实例化一个[Handler](#)对象，并把它定义为全局变量，然后通过使用[Handler\(Looper\)](#)这一构造方法实例化它，用于关联到UI线程。[Handler\(Looper\)](#)这一构造方法需要传入了一个[Looper](#)对象，它是Android系统的线程管理框架中的另一部分。当你在一个特定的[Looper](#)实例的基础上去实例化一个[Handler](#)时，这个[Handler](#)与[Looper](#)运行在同一个线程里。例如：

```
private PhotoManager() {  
    ...  
    // Defines a Handler object that's attached to the UI thread  
    mHandler = new Handler(Looper.getMainLooper()) {  
        ...  
    };  
}
```

在这个[Handler](#)里需要重写[handleMessage\(\)](#)方法。当这个[Handler](#)接收到由另外一个线程管理的[Handler](#)发送过来的新消息时，Android系统会自动调用这个方法，而所有线程对应的[Handler](#)都会收到相同信息。例如：

```
/*  
 * handleMessage() defines the operations to perform when  
 * the Handler receives a new Message to process.  
 */  
@Override  
public void handleMessage(Message inputMessage) {  
    // Gets the image task from the incoming Message object  
    PhotoTask photoTask = (PhotoTask) inputMessage.obj;  
    ...  
}  
...  
}
```

下一部分将向你展示如何用[Handler](#)转移数据。

# 把数据从一个任务中转移到UI线程

为了从一个运行在后台线程的任务对象中转移数据到UI线程中的一个对象，首先需要存储任务对象中的数据和UI对象的引用；接下来传递任务对象和状态码给实例化[Handler](#)的那个对象。在这个对象里，发送一个包含任务对象和状态的[Message](#)给[Handler](#)也运行在UI线程中，所以它可以把数据转移到UI线程。

## 在任务对象中存储数据

比如这里有一个[Runnable](#)，它运行在一个编码了一个[Bitmap](#)且存储这个[Bitmap](#)到父类[PhotoTask](#)对象里的后台线程。这个[Runnable](#)同样也存储了状态码[DECODE\\_STATE\\_COMPLETED](#)。

```
// A class that decodes photo files into Bitmaps
class PhotoDecodeRunnable implements Runnable {

    ...
    PhotoDecodeRunnable(PhotoTask downloadTask) {
        mPhotoTask = downloadTask;
    }
    ...
    // Gets the downloaded byte array
    byte[] imageBuffer = mPhotoTask.getByteBuffer();
    ...
    // Runs the code for this task
    public void run() {
        ...
        // Tries to decode the image buffer
        returnBitmap = BitmapFactory.decodeByteArray(
            imageBuffer,
            0,
            imageBuffer.length,
            bitmapOptions
        );
        ...
        // Sets the ImageView Bitmap
        mPhotoTask.setImage(returnBitmap);
        // Reports a status of "completed"
        mPhotoTask.handleDecodeState(DECODE_STATE_COMPLETED);
        ...
    }
    ...
}
```

[PhotoTask](#)类还包含一个用于给[ImageView](#)显示[Bitmap](#)的handler。虽然[Bitmap](#)和[ImageView](#)的引用在同一个对象中，但你不能把这个[Bitmap](#)分配给[ImageView](#)去显示，因为它们并没有运行在UI线程中。

这时，下一步应该发送这个状态给[PhotoTask](#)对象。

## 发送状态取决于对象层次

*PhotoTask*是下一个层次更高的对象，它包含将要展示数据的编码数据和[View](#)对象的引用。它会收到一个来自*PhotoDecodeRunnable*的状态码，并把这个状态码单独传递到一个包含线程池和[Handler](#)实例的对象：

```
public class PhotoTask {  
    ...  
    // Gets a handle to the object that creates the thread pools  
    sPhotoManager = PhotoManager.getInstance();  
    ...  
    public void handleDecodeState(int state) {  
        int outState;  
        // Converts the decode state to the overall state.  
        switch(state) {  
            case PhotoDecodeRunnable.DECODE_STATE_COMPLETED:  
                outState = PhotoManager.TASK_COMPLETE;  
                break;  
            ...  
        }  
        ...  
        // Calls the generalized state method  
        handleState(outState);  
    }  
    ...  
    // Passes the state to PhotoManager  
    void handleState(int state) {  
        /*  
         * Passes a handle to this task and the  
         * current state to the class that created  
         * the thread pools  
         */  
        sPhotoManager.handleState(this, state);  
    }  
    ...  
}
```

## 转移数据到UI

从*PhotoTask*对象那里，*PhotoManager*对象收到了一个状态码和一个*PhotoTask*对象的handler。因为状态码是*TASK\_COMPLETE*，所以创建一个[Message](#)应该包含状态和任务对象，然后把它发送给[Handler](#)：

```
public class PhotoManager {  
    ...  
    // Handle status messages from tasks  
    public void handleState(PhotoTask photoTask, int state) {  
        switch (state) {  
            ...  
            // The task finished downloading and decoding the image  
            case TASK_COMPLETE:  
                /*  
                 * Creates a message for the Handler  
                 * with the state and the task object  
                 */  
        }  
    }  
}
```

```
        */
    Message completeMessage =
        mHandler.obtainMessage(state, photoTask);
    completeMessage.sendToTarget();
    break;
    ...
}
...
}
```

最终，[Handler.handleMessage\(\)](#)会检查每个传入进来的[Message](#)，如果状态码是[TASK\\_COMPLETE](#)，这时任务就完成了，而传入的[Message](#)里的[PhotoTask](#)对象里同时包含一个[Bitmap](#)和一个[ImageView](#)。因为[Handler.handleMessage\(\)](#)运行在UI线程里，所以它能安全地转移[Bitmap](#)数据给[ImageView](#)：

```
private PhotoManager() {
    ...
    mHandler = new Handler(Looper.getMainLooper()) {
        @Override
        public void handleMessage(Message inputMessage) {
            // Gets the task from the incoming Message obj
            PhotoTask photoTask = (PhotoTask) inputMessage
            // Gets the ImageView for this task
            PhotoView localView = photoTask.getPhotoView()
            ...
            switch (inputMessage.what) {
                ...
                // The decoding is done
                case TASK_COMPLETE:
                    /*
                     * Moves the Bitmap from the task
                     * to the View
                     */
                    localView.setImageBitmap(photoTask.get
                    break;
                ...
                default:
                    /*
                     * Pass along other messages from the
                     */
                    super.handleMessage(inputMessage);
            }
            ...
        }
        ...
    }
    ...
}
...
}
```

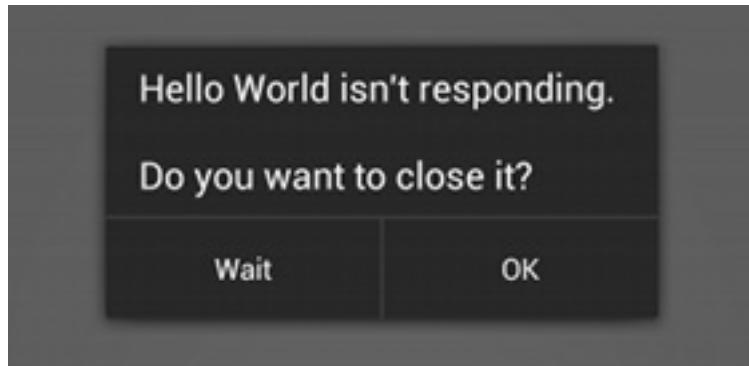
编写:[kesenhoo](#), 校对:

原文:<http://developer.android.com/training/articles/perf-anr.html>

# 避免出现程序无响应ANR(Keeping Your App Responsive)

可能你写的代码在性能测试上表现良好，但是你的应用仍然有时候会反应迟缓(sluggish)，停顿(hang)或者长时间卡死(freeze)，或者应用处理输入的数据花费时间过长。对于你的应用来说最糟糕的事情是出现"程序无响应(Application Not Responding)" (ANR)的警示框。

在Android中，系统通过显示ANR警示框来保护程序的长时间无响应。对话框如下：



此时，你的应用已经经历过一段时间的无法响应了，因此系统提供用户可以退出应用的选择。为你的程序提供良好的响应性是至关重要的，这样才能够避免系统为用户显示ANR的警示框。

这节课描述了Android系统是如何判断一个应用不可响应的。这节课还会提供程序编写的指导原则，确保你的程序保持响应性。

## 是什么导致了ANR?(What Triggers ANR?)

通常来说，系统会在程序无法响应用户的输入事件时显示ANR。例如，如果一个程序在UI线程执行I/O操作(通常是网络请求或者是文件读写)，这样系统就无法处理用户的输入事件。或者是应用在UI线程花费了太多的时间用来建立一个复杂的在内存中的数据结构，又或者是在一个游戏程序的UI线程中执行了一个复杂耗时的计算移动的操作。确保那些计算操作高效是很重要的，不过即使是最高效的代码也是需要花时间执行的。

对于你的应用中任何可能执行时间长的操作，你都不应该执行在UI线程。你可以创建一个工作线程，把那些操作都执行在工作线程中。这确保了UI线程(这个线程会负责处理UI事件)能够顺利执行，也预防了系统因代码僵死而崩溃。因为UI线程是和类级别相关联的，你可以把相应性作为一个类级别(class-level)的问题(相比来说，代码性能则属于方法级别(method-level)的问题)

在Android中，程序的响应性是由Activity Manager与Window Manager系统服务来负责监控的。当系统监测到下面的条件之一时会显示ANR的对话框：

- 对输入事件(例如硬件点击或者屏幕触摸事件)，5秒内都无响应。
- BroadReceiver不能够在10秒内结束接收到任务。

## 如何避免ANRs(How to Avoid ANRs)

Android程序通常是执行在默认的UI线程(也可以成为main线程)中的。这意味着在UI线程中执行的任何长时间的操作都可能触发ANR，因为程序没有给自己处理输入事件或者broadcast事件的机会。

因此，任何执行在UI线程的方法都应该尽可能的简短快速。特别是，在activity的生命周期的关键方法onCreate()与onResume()方法中应该尽可能的做比较少的事情。类似网络或者DB操作等可能长时间执行的操作，或者是类似调整bitmap大小等需要长时间计算的操作，都应该执行在工作线程中。(在DB操作中，可以通过异步的网络请求)。

为了执行一个长时间的耗时操作而创建一个工作线程最方便高效的方式是使用AsyncTask。只需要继承AsyncTask并实现doInBackground()方法来执行任务即可。为了把任务执行的进度呈现给用户，你可以执行publishProgress()方法，这个方法会触发onProgressUpdate()的回调方法。在onProgressUpdate()的回调方法中(它执行在UI线程)，你可以执行通知用户进度的操作，例如：

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    // Do the long-running work in here
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    // This is called each time you call publishProgress()
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    // This is called when doInBackground() is finished
    protected void onPostExecute(Long result) {
        showNotification("Downloaded " + result + " bytes");
    }
}
```

为了能够执行这个工作线程，只需要创建一个实例并执行execute()：

```
new DownloadFilesTask().execute(url1, url2, url3);
```

相比起AsycnTask来说，创建自己的线程或者HandlerThread稍微复杂一点。如果你想这样做，你应该通过`Process.setThreadPriority()`并传递`THREAD_PRIORITY_BACKGROUND`来设置线程的优先级为"`background`"。如果你不通过这种方式来给线程设置一个低的优先级，那么这个线程仍然会使得你的应用显得卡顿，因为这个线程默认与UI线程有着同样的优先级。

如果你实现了Thread或者HandlerThread，请确保你的UI线程不会因为等待工作线程的某个任务而去执行Thread.wait()或者Thread.sleep(). UI线程不应该去等待工作线程完成某个任务，你的UI现场应该提供一个Handler给其他工作线程，这样工作线程能够通过这个Handler在任务结束的时候通知UI线程。使用这样的方式来设计你的应用程序可以使得你的程序UI线程保持响应性，以此来避免ANR。

BroadcastReceiver有特定执行时间的限制说明了broadcast receivers应该做的是：简短快速的任务，避免执行费时的操作，例如保存数据或者注册一个Notification。正如在UI线程中执行的方法一样，程序应该避免在broadcast receiver中执行费时的长任务。但不是采用通过工作线程来执行复杂的任务的方式，你的程序应该启动一个IntentService来响应intent broadcast的长时间任务。

**Tip:** 你可以使用StrictMode来帮助寻找因为不小心加入到UI线程的潜在的长时间执行的操作，例如网络或者DB相关的任务。

## 增加响应性(Reinforce Responsiveness)

通常来说，100ms - 200ms是用户能够察觉到卡顿的上限。这样的话，下面有一些避免ANR的技巧：

- 如果你的程序需要响应正在后台加载的任务，在你的UI中可以显示ProgressBar来显示进度。
- 对游戏程序，在工作线程执行计算的任务。
- 如果你的程序在启动阶段有一个耗时的初始化操作，可以考虑显示一个闪屏，要么尽快的显示主界面，然后马上显示一个加载的对话框，异步加载数据。无论哪种情况，你都应该显示一个进度信息，以免用户感觉程序有卡顿的情况。
- 使用性能测试工具，例如Systrace与Traceview来判断程序中影响响应性的瓶颈。

编写: [pedant](#), 校对:

原文: <http://developer.android.com/training/articles/perf-jni.html>

# JNI Tips

JNI全称Java Native Interface。它为托管代码（使用Java编程语言编写）与本地代码（使用C/C++编写）提供了一种交互方式。它是与厂商无关的（vendor-neutral）,支持从动态共享库中加载代码，虽然这样会稍显麻烦，但有时这是相当有效的。

如果你对JNI还不是太熟悉，可以先通读[Java Native Interface Specification](#)这篇文章来对JNI如何工作以及哪些特性可用有个大致的印象。这种接口的一些方面不能立即一读就显而易见，所以你会发现接下来的几个章节很有用处。

# JavaVM 及 JNIEnv

JNI定义了两种关键数据结构，“JavaVM”和“JNIEnv”。它们本质上都是指向函数表指针的指针（在C++版本中，它们被定义为类，该类包含一个指向函数表的指针，以及一系列可以通过这个函数表间接地访问对应的JNI函数的成员函数）。JavaVM提供“调用接口（invocation interface）”函数，允许你创建和销毁一个JavaVM。理论上你可以在一个进程中拥有多个JavaVM对象，但安卓只允许一个。

JNIEnv提供了大部分JNI功能。你定义的所有本地函数都会接收JNIEnv作为第一个参数。

JNIEnv是用作线程局部存储。因此，你不能在线程间共享一个**JNIEnv**变量。如果在一段代码中没有其它办法获得它的JNIEnv，你可以共享JavaVM对象，使用GetEnv来取得该线程下的JNIEnv（如果该线程有一个JavaVM的话；见下面的AttachCurrentThread）。

JNIEnv和JavaVM的在C声明是不同于在C++的声明。头文件“jni.h”根据它是以C还是以C++模式包含来提供不同的类型定义（typedefs）。因此，不建议把JNIEnv参数放到可能被两种语言引入的头文件中（换一句话说：如果你的头文件需要#define \_\_cplusplus，你可能不得不在任何涉及到JNIEnv的内容处都要做些额外的工作）。

# 线程

所有的线程都是Linux线程，由内核统一调度。它们通常从托管代码中启动（使用Thread.start），但它们也能够在其他任何地方创建，然后连接（attach）到JavaVM。例如，一个用pthread\_create启动的线程能够使用JNI AttachCurrentThread或AttachCurrentThreadAsDaemon函数连接到JavaVM。在一个线程成功连接（attach）之前，它没有JNIEnv，不能够调用**JNI**函数。

连接一个本地环境创建的线程会触发构造一个java.lang.Thread对象，然后其被添加到主线程群组（main ThreadGroup），以让调试器可以探测到。对一个已经连接的线程使用AttachCurrentThread不做任何操作（no-op）。

安卓不能中止正在执行本地代码的线程。如果正在进行垃圾回收，或者调试器已发出了中止请求，安卓会在下一次调用JNI函数的时候中止线程。

连接过的（attached）线程在它们退出之前必须通过**JNI**调用**DetachCurrentThread**。如果你觉得直接这样编写不太优雅，在安卓2.0（Eclair）及以上，你可以使用pthread\_key\_create来定义一个析构函数，它将会在线程退出时被调用，你可以在那儿调用DetachCurrentThread（使用生成的key与pthread\_setspecific将JNIEnv存储到线程局部空间内；这样JNIEnv能够作为参数传入到析构函数当中去）。

# jclass, jmethodID, jfieldID

如果你想在本地代码中访问一个对象的字段（field），你可以像下面这样做：

- 对于类，使用FindClass获得类对象的引用
- 对于字段，使用GetFieldId获得字段ID
- 使用对应的方法（例如GetIntField）获取字段下面的值

类似地，要调用一个方法，你首先得获得一个类对象的引用，然后是方法ID（method ID）。这些ID通常是指向运行时内部数据结构。查找到它们需要些字符串比较，但一旦你实际去执行它们获得字段或者做方法调用是非常快的。

如果性能是你看重的，那么一旦查找出这些值之后在你的本地代码中缓存这些结果是非常有用的。因为每个进程当中的JavaVM是存在限制的，存储这些数据到本地静态数据结构中是非常合理的。

类引用（class reference），字段ID（field ID）以及方法ID（method ID）在类被卸载前都是有效的。如果与一个类加载器（ClassLoader）相关的所有类都能够被垃圾回收，但是这种情况在安卓上是罕见甚至不可能出现，只有这时类才被卸载。注意虽然jclass是一个类引用，但是必须要调用**NewGlobalRef**保护起来（见下个章节）。

当一个类被加载时如果你想缓存些ID，而后当这个类被卸载后再次载入时能够自动地更新这些缓存ID，正确做法是在对应的类中添加一段像下面的代码来初始化这些ID：

```
/*
 * 我们在一个类初始化时调用本地方法来缓存一些字段的偏移信息
 * 这个本地方法查找并缓存你感兴趣的class/field/method ID
 * 失败时抛出异常
 */
private static native void nativeInit();

static {
    nativeInit();
}
```

在你的C/C++代码中创建一个nativeClassInit方法以完成ID查找的工作。当这个类被初始化时这段代码将会执行一次。当这个类被卸载后而后再次载入时，这段代码将会再次执行。

# 局部和全局引用

每个传入本地方法的参数，以及大部分JNI函数返回的每个对象都是“局部引用”。这意味着它只在当前线程的当前方法执行期间有效。即使这个对象本身在本地方法返回之后仍然存在，这个引用也是无效的。

这同样适用于所有 jobject 的子类，包括 jclass，jstring，以及 jarray（当 JNI 扩展检查是打开的时候，运行时会警告你对大部分对象引用的误用）。

如果你想持有一个引用更长的时间，你就必须使用一个全局 (“global”) 引用了。

NewGlobalRef 函数以一个局部引用作为参数并且返回一个全局引用。全局引用能够保证在你调用 DeleteGlobalRef 前都是有效的。

这种模式通常被用在缓存一个从 FindClass 返回的 jclass 对象的时候，例如：

```
jclass localClass = env->FindClass("MyClass");
jclass globalClass = reinterpret_cast<jclass>(env->NewGlobalRef(lo
```

所有的 JNI 方法都接收局部引用和全局引用作为参数。相同对象的引用却可能具有不同的值。例如，用相同对象连续地调用 NewGlobalRef 得到返回值可能是不同的。为了检查两个引用是否指向的是同一个对象，你必须使用 **IsSameObject** 函数。绝不要在本地代码中用 == 符号来比较两个引用。

得出的结论就是你绝不要在本地代码中假定对象的引用是常量或者是唯一的。代表一个对象的 32 位值从方法的一次调用到下一次调用可能有不同的值。在连续的调用过程中两个不同的对象却可能拥有相同的 32 位值。不要使用 jobject 的值作为 key.

开发者需要“不过度分配”局部引用。在实际操作中这意味着如果你正在创建大量的局部引用，或许是通过对象数组，你应该使用 DeleteLocalRef 手动地释放它们，而不是寄希望 JNI 来为你做这些。实现上只预留了 16 个局部引用的空间，所以如果你需要更多，要么你删掉以前的，要么使用 EnsureLocalCapacity / PushLocalFrame 来预留更多。

注意 jfieldID 和 jmethodID 是 **映射类型 (opaque types)**，不是对象引用，不应该被传入到 NewGlobalRef。原始数据指针，像 GetStringUTFChars 和 GetByteArrayElements 的返回值，也都不是对象（它们能够在线程间传递，并且在调用对应的 Release 函数之前都是有效的）。

还有一种不常见的情况值得一提，如果你使用 AttachCurrentThread 连接 (attach) 了本地进程，正在运行的代码在线程分离 (detach) 之前决不会自动释放局部引用。你创建的任何局部引用必须手动删除。通常，任何在循环中创建局部引用的本地代码可能都需要做一些手动删除。

# UTF-8、UTF-16字符串

Java编程语言使用UTF-16格式。为了便利，JNI也提供了支持[变形UTF-8 \(Modified UTF-8\)](#)的方法。这种变形编码对于C代码是非常有用的，因为它将\u0000编码成0xc0 0x80，而不是0x00。最惬意的事情是你能在具有C风格的以\0结束的字符串上计数，同时兼容标准的libc字符串函数。不好的一面是你不能传入随意的UTF-8数据到JNI函数而还指望它正常工作。

如果可能的话，直接操作UTF-16字符串通常更快些。安卓当前在调用GetStringChars时不需要拷贝，而GetStringUTFChars需要一次分配并且转换为UTF-8格式。注意**UTF-16**字符串不是以零终止字符串，\u0000是被允许的，所以你需要像对jchar指针一样地处理字符串的长度。

不要忘记Release你Get的字符串。这些字符串函数返回jchar或者jbyte，都是指向基本数据类型的C格式的指针而不是局部引用。它们在Release调用之前都保证有效，这意味着当本地方法返回时它们并不主动释放。

传入NewStringUTF函数的数据必须是**变形UTF-8**格式。一种常见的错误情况是，从文件或者网络流中读取出的字符数据，没有过滤直接使用NewStringUTF处理。除非你确定数据是7位的ASCII格式，否则你需要剔除超出7位ASCII编码范围（high-ASCII）的字符或者将它们转换为对应的变形UTF-8格式。如果你没那样做，UTF-16的转换结果可能不会是你要的结果。JNI扩展检查将会扫描字符串，然后警告你那些无效的数据，但是它们将不会发现所有潜在的风险。

# 原生类型数组

JNI提供了一系列函数来访问数组对象中的内容。对象数组的访问只能一次一条，但如果原生类型数组以C方式声明，则能够直接进行读写。

为了让接口更有效率而不受VM实现的制约，GetArrayElements系列调用允许运行时返回一个指向实际元素的指针，或者是分配些内存然后拷贝一份。不论哪种方式，返回的原始指针在相应的Release调用之前都保证有效（这意味着，如果数据没被拷贝，实际的数组对象将会受到牵制，不能重新成为整理堆空间的一部分）。你必须释放（Release）每个你通过Get得到的数组。同时，如果Get调用失败，你必须确保你的代码在之后不会去尝试调用Release来释放一个空指针（NULL pointer）。

你可以用一个非空指针作为isCopy参数的值来决定数据是否会被拷贝。这相当有用。

Release类的函数接收一个mode参数，这个参数的值可选的有下面三种。而运行时具体执行的操作取决于它返回的指针是指向真实数据还是拷贝出来的那份。

- 0
  - 真实的：实际数组对象不受到牵制
  - 拷贝的：数据将会复制回去，备份空间将会被释放。
- JNI\_COMMIT
  - 真实的：不做任何操作
  - 拷贝的：数据将会复制回去，备份空间将不会被释放。
- JNI\_ABORT
  - 真实的：实际数组对象不受到牵制.之前的写入不会被取消。
  - 拷贝的：备份空间将会被释放；里面所有的变更都会丢失。

检查isCopy标识的一个原因是对你一个数组做出变更后确认你是否需要传入JNI\_COMMIT来调用Release函数。如果你交替地执行变更和读取数组内容的代码，你也许可以跳过无操作（no-op）的JNI\_COMMIT。检查这个标识的另一个可能的原因是使用JNI\_ABORT可以更高效。例如，你也许想得到一个数组，适当地修改它，传入部分到其他函数中，然后丢掉这些修改。如果你知道JNI是你做了一份新的拷贝，就没有必要再创建另一份“可编辑的（editable）”的拷贝了。如果JNI传给你的是原始数组，这时你就需要创建一份你自己的拷贝了。

另一个常见的错误（在示例代码中出现过）是认为当isCopy是false时你就可以不调用Release。实际上是没有这种情况的。如果没有分配备份空间，那么初始的内存空间会受到牵制，位置不能被垃圾回收器移动。

另外注意JNI\_COMMIT标识没有释放数组，你最终需要使用一个不同的标识再次调用Release。

# 区间数组

当你想做的只是拷出或者拷进数据时，可以选择调用像GetArrayElements和GetStringChars这类非常有用的函数。想想下面：

```
jbyte* data = env->GetByteArrayElements(array, NULL);  
if (data != NULL) {  
    memcpy(buffer, data, len);  
    env->ReleaseByteArrayElements(array, data, JNI_ABORT);  
}
```

这里获取到了数组，从当中拷贝出开头的len个字节元素，然后释放这个数组。根据代码的实现，Get函数将会牵制或者拷贝数组的内容。上面的代码拷贝了数据（为了可能的第二次），然后调用Release；这当中JNI\_ABORT确保不存在第三份拷贝了。

另一种更简单的实现方式：

```
env->GetByteArrayRegion(array, 0, len, buffer);
```

这种方式有几个优点：

- 只需要调用一个JNI函数而不是两个，减少了开销。
- 不需要指针或者额外的拷贝数据。
- 减少了开发人员犯错的风险-在某些失败之后忘记调用Release不存在风险。

类似地，你能使用SetArrayRegion函数拷贝数据到数组，使用GetStringRegion或者GetStringUTFRegion从String中拷贝字符。

# 异常

当异常发生时你一定不能调用大部分的**JNI**函数。你的代码收到异常（通过函数的返回值，`ExceptionCheck`，或者`ExceptionOccurred`），然后返回，或者清除异常，处理掉。

当异常发生时你被允许调用的**JNI**函数有：

- `DeleteGlobalRef`
- `DeleteLocalRef`
- `DeleteWeakGlobalRef`
- `ExceptionCheck`
- `ExceptionClear`
- `ExceptionDescribe`
- `ExceptionOccurred`
- `MonitorExit`
- `PopLocalFrame`
- `PushLocalFrame`
- `ReleaseArrayElements`
- `ReleasePrimitiveArrayCritical`
- `ReleaseStringChars`
- `ReleaseStringCritical`
- `ReleaseStringUTFChars`

许多**JNI**调用能够抛出异常，但通常提供一种简单的方式来检查失败。例如，如果 `NewString` 返回一个非空值，你不需要检查异常。然而，如果你调用一个方法（使用一个像 `CallObjectMethod` 的函数），你必须一直检查异常，因为当一个异常抛出时它的返回值将不会是有效的。

注意中断代码抛出的异常不会展开本地调用堆栈信息，Android 也还不支持 C++ 异常。`JNI Throw` 和 `ThrowNew` 指令仅仅是在当前线程中放入一个异常指针。从本地代码返回到托管代码时，异常将会被注意到，得到适当的处理。

本地代码能够通过调用 `ExceptionCheck` 或者 `ExceptionOccurred` 捕获到异常，然后使用 `ExceptionClear` 清除掉。通常，抛弃异常而不处理会导致些问题。

没有内建的函数来处理 `Throwable` 对象自身，因此如果你想得到异常字符串，你需要找出 `Throwable Class`，然后查找到 `getMessage "()Ljava/lang/String;"` 的方法 ID，调用它，如果结果非空，使用 `GetStringUTFChars`，得到的结果你可以传到 `printf(3)` 或者其它相同功能的函数输出。

# 扩展检查

JNI的错误检查很少。错误发生时通常会导致崩溃。Android也提供了一种模式，叫做CheckJNI，这当中JavaVM和JNIEnv函数表指针被换成了函数表，它在调用标准实现之前执行了一系列扩展检查的。

额外的检查包括：

- 数组：试图分配一个长度为负的数组。
- 坏指针：传入一个不完整jarray/jclass/jobject/jstring对象到JNI函数，或者调用JNI函数时使用空指针传入到一个不能为空的参数中去。
- 类名：传入了除“java/lang/String”之外的类名到JNI函数。
- 关键调用：在一个“关键的(critical)”get和它对应的release之间做出JNI调用。
- 直接的ByteBuffers：传入不正确的参数到NewDirectByteBuffer。
- 异常：当一个异常发生时调用了JNI函数。
- JNIEnvs：在错误的线程中使用一个JNIEnv。
- jfieldIDs：使用一个空jfieldID，或者使用jfieldID设置了一个错误类型的值到字段（比如说，试图将一个StringBuilder赋给String类型的域），或者使用一个静态字段下的jfieldID设置到一个实例的字段（instance field）反之亦然，或者使用的一个类的jfieldID却来自另一个类的实例。
- jmethodIDs：当调用Call\*Method函数时时使用了类型错误的jmethodID：不正确的返回值，静态/非静态的不匹配，this的类型错误（对于非静态调用）或者错误的类（对于静态类调用）。
- 引用：在类型错误的引用上使用了DeleteGlobalRef/DeleteLocalRef。
- 释放模式：调用release使用一个不正确的释放模式（其它非0，JNI\_ABORT，JNI\_COMMIT的值）。
- 类型安全：从你的本地代码中返回了一个不兼容的类型（比如说，从一个声明返回String的方法却返回了StringBuilder）。
- UTF-8：传入一个无效的变形UTF-8字节序列到JNI调用。

（方法和域的可访问性仍然没有检查：访问限制对于本地代码并不适用。）

有几种方法去启用CheckJNI。

如果你正在使用模拟器，CheckJNI默认是打开的。

如果你有一台root过的设备，你可以使用下面的命令序列来重启运行时（runtime），启用CheckJNI。

```
adb shell stop  
adb shell setprop dalvik.vm.checkjni true  
adb shell start
```

随便哪一种，当运行时（runtime）启动时你将会在你的日志输出中见到如下的字符：

```
D AndroidRuntime: CheckJNI is ON
```

如果你有一台常规的设备，你可以使用下面的命令：

```
adb shell setprop debug.checkjni 1
```

这将不会影响已经在运行的app，但是从那以后启动的任何app都将打开CheckJNI(改变属性为其它值或者只是重启都将会再次关闭CheckJNI)。这种情况下，你将会在下一次app启动时，在日志输出中看到如下字符：

```
D Late-enabling CheckJNI
```

# 本地库

你可以使用标准的System.loadLibrary方法来从共享库中加载本地代码。在你的本地代码中较好的做法是：

- 在一个静态类初始化时调用System.loadLibrary（见之前的一个例子中，当中就使用了nativeClassInit）。参数是“未加修饰（undecorated）”的库名称，因此要加载“libfubar.so”，你需要传入“fubar”。
- 提供一个本地函数：**jint JNI\_OnLoad(JavaVM vm, void reserved)**
- 在JNI\_OnLoad中，注册所有你的本地方法。你应该声明方法为“静态的（static）”因此名称不会占据设备上符号表的空间。

JNI\_OnLoad函数在C++中的写法如下：

```
jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env;
    if (vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6
        return -1;
    }

    // 使用env->FindClass得到jclass
    // 使用env->RegisterNatives注册本地方法

    return JNI_VERSION_1_6;
}
```

你也可以使用共享库的全路径来调用System.load。对于Android app，你也许会发现从context对象中得到应用私有数据存储的全路径是非常有用的。

上面是推荐的方式，但不是仅有的实现方式。显式注册不是必须的，提供一个JNI\_OnLoad函数也不是必须的。你可以使用基于特殊命名的“发现（discovery）”模式来注册本地方法（更多细节见：[JNI spec](#)），虽然这并不可取。因为如果一个方法的签名错误，在这个方法实际第一次被调用之前你是不会知道的。

关于JNI\_OnLoad另一点注意的是：任何你在JNI\_OnLoad中对FindClass的调用都发生在用作加载共享库的类加载器的上下文（context）中。一般FindClass使用与“调用栈”顶部方法相关的加载器，如果当中没有加载器（因为线程刚刚连接）则使用“系统（system）”类加载器。这就使得JNI\_OnLoad成为一个查寻及缓存类引用很便利的地方。

# 64位机问题

Android当前设计为运行在32位的平台上。理论上它也能够构建为64位的系统，但那不是现在的目标。当与本地代码交互时，在大多数情况下这不是你需要担心的，但是如果你打算存储指针变量到对象的整型字段（integer field）这样的本地结构中，这就变得非常重要了。为了支持使用64位指针的架构，你需要使用**long**类型而不是**int**类型的字段来存储你的本地指针。

# 不支持的特性/向后兼容性

除了下面的例外，支持所有的JNI 1.6特性：

- DefineClass没有实现。Android不使用Java字节码或者class文件，因此传入二进制class数据将不会有效。

对Android以前老版本的向后兼容性，你需要注意：

- 本地函数的动态查找 在Android 2.0(Eclair)之前，在搜索方法名称时，字符“\$”不会转换为对应的“\_00024”。要使它正常工作需要使用显式注册方式或者将本地方法的声明移出内部类。
- 分离线程 在Android 2.0(Eclair)之前，使用pthread\_key\_create析构函数来避免“退出前线程必须分离”检查是不可行的（运行时(runtime)也使用了一个pthread key析构函数，因此这是一场看谁先被调用的竞赛）。
- 全局弱引用 在Android 2.0(Eclair)之前，全局弱引用没有被实现。如果试图使用它们，老版本将完全不兼容。你可以使用Android平台版本号常量来测试系统的支持性。在Android 4.0 (Ice Cream Sandwich)之前，全局弱引用只能传给NewLocalRef, NewGlobalRef, 以及DeleteWeakGlobalRef（强烈建议开发者在使用全局弱引用之前都为它们创建强引用hard reference，所以这不应该在所有限制当中）。从Android 4.0 (Ice Cream Sandwich)起，全局弱引用能够像其它任何JNI引用一样使用了。
- 局部引用 在Android 4.0 (Ice Cream Sandwich)之前，局部引用实际上是直接指针。Ice Cream Sandwich为了更好地支持垃圾回收添加了间接指针，但这并不意味着很多JNI bug在老版本上不存在。更多细节见[JNI Local Reference Changes in ICS](#)。
- 使用GetObjectRefType获得引用类型 在Android 4.0 (Ice Cream Sandwich)之前，使用直接指针（见上面）的后果就是正确地实现GetObjectRefType是不可能的。我们可以使用依次检测全局弱引用表，参数，局部表，全局表的方式来代替。第一次匹配到你的直接指针时，就表明你的引用类型是当前正在检测的类型。这意味着，例如，如果你在一个全局jclass上使用GetObjectRefType，而这个全局jclass碰巧与作为静态本地方法的隐式参数传入的jclass一样的，你得到的结果是JNILocalRefType而不是JNIGlobalRefType。

# FAQ: 为什么出现了UnsatisfiedLinkError?

当使用本地代码开发时经常会见到像下面的错误：

```
java.lang.UnsatisfiedLinkError: Library foo not found
```

有时候这表示和它提示的一样---未找到库。但有些时候库确实存在但不能被dlopen(3)打开，更多的失败信息可以参见异常详细说明。

你遇到“library not found”异常的常见原因可能有这些：

- 库文件不存在或者不能被app访问到。使用adb shell ls -l 检查它的存在性和权限。
- 库文件不是用NDK构建的。这就导致设备上并不存在它所依赖的函数或者库。

另一种UnsatisfiedLinkError错误像下面这样：

```
java.lang.UnsatisfiedLinkError: myfunc
    at Foo.myfunc(Native Method)
    at Foo.main(Foo.java:10)
```

在日志中，你会发现：

```
W/dalvikvm( 880): No implementation found for native LFoo;.myfunc
```

这意味着运行时尝试匹配一个方法但是没有成功，这种情况常见的原因有：

- 库文件没有得到加载。检查日志输出中关于库文件加载的信息。
- 由于名称或者签名错误，方法不能匹配成功。这通常是由于：
  - 对于方法的懒查寻，使用 extern "C" 和对应的可见性 (JNIEXPORT) 来声明 C++ 函数没有成功。注意Ice Cream Sandwich之前的版本，JNIEXPORT宏是不正确的，因此对新版本的GCC使用旧的jni.h头文件将不会有效。你可以使用arm-eabi-nm查看它们出现在库文件里的符号。如果它们看上去比较凌乱（像\_Z15Java\_Foo\_myfuncP7\_JNIEnvP7\_jclass 这样而不是 Java\_Foo\_myfunc），或者符号类型是小写的“t”而不是一个大写的“T”，这时你就需要调整声明了。
  - 对于显式注册，在进行方法签名时可能犯了些小错误。确保你传入到注册函数的签名能够完全匹配上日志文件里提示的。记住“B”是byte，“Z”是boolean。在签名中类名组件是以“L”开头的，以“;”结束的，使用“/”来分隔包名/类名，使用“Entry;）。

使用javah来自动生成JNI头文件也许能帮助你避免这些问题。

# FAQ: 为什么FindClass不能找到我的类?

确保类名字符串有正确的格式。JNI类名称以包名开始，然后使用左斜杠来分隔，比如`java/lang/String`。如果你正在查找一个数组类，你需要以对应数目的括号开头，使用“L”和“;”将类名两头包起来，所以一个一维字符串数组应该写成`[Ljava/lang/String;`。

如果类名称看上去正确，你可能运行时遇到了类加载器的问题。FindClass想在与你代码相关的类加载器中开始查找指定的类。检查调用堆栈，可能看起来像：

```
Foo.myfunc (Native Method)
Foo.main(Foo.java:10)
dalvik.system.NativeStart.main(Native Method)
```

最顶层的方法是`Foo.myfunc`。FindClass找到与类`Foo`相关的`ClassLoader`对象然后使用它。

这通常正是你所想的。如果你创建了自己的线程那么就会遇到麻烦（也许是调用了`pthread_create`然后使用`AttachCurrentThread`进行了连接）。现在跟踪堆栈可能像下面这样：

```
dalvik.system.NativeStart.run(Native Method)
```

最顶层的方法是`NativeStart.run`，它不是你应用内的方法。如果你从这个线程中调用`FindClass`，JavaVM将会启动“系统（system）”的而不是与你应用相关的加载器，因此试图查找应用内定义的类都将会失败。

下面有几种方法可以解决这个问题：

- 在`JNI_OnLoad`中使用`FindClass`查寻一次，然后为后面的使用缓存这些类引用。任何在`JNI_OnLoad`当中执行的`FindClass`调用都使用与执行`System.loadLibrary`的函数相关的类加载器（这个特例，让库的初始化更加的方便了）。如果你的app代码正在加载库文件，`FindClass`将会使用正确的类加载器。
- 传入类实例到一个需要它的函数，你的本地方法声明必须带有一个`Class`参数，然后传入`Foo.class`。
- 在合适的地方缓存一个`ClassLoader`对象的引用，然后直接发起`loadClass`调用。这需要额外些工作。

# FAQ: 使用本地代码怎样共享原始数据?

也许你会遇到这样一种情况，想从你的托管代码或者本地代码访问一大块原始数据的缓冲区。常见例子包括对bitmap或者声音文件的处理。这里有两种基本实现方式。

你可以将数据存储到byte[]。这允许你从托管代码中快速地访问。然而，在本地代码端不能保证你不去拷贝一份就直接能够访问数据。在某些实现中，GetByteArrayElements和GetPrimitiveArrayCritical将会返回指向在维护堆中的原始数据的真实指针，但是在另外一些实现中将在本地堆空间分配一块缓冲区然后拷贝数据过去。

还有一种选择是将数据存储在一块直接字节缓冲区（direct byte buffer），可以使用java.nio.ByteBuffer.allocateDirect或者NewDirectByteBuffer JNI函数创建buffer。不像常规的byte缓冲区，它的存储空间将不会分配在程序维护的堆空间上，总是可以从本地代码直接访问（使用GetDirectBufferAddress得到地址）。依赖于直接字节缓冲区访问的实现方式，从托管代码访问原始数据将会非常慢。

选择使用哪种方式取决于两个方面：

- 1.大部分的数据访问是在Java代码还是C/C++代码中发生？
- 2.如果数据最终被传到系统API，那它必须是怎样的形式（例如，如果数据最终被传到一个使用byte[]作为参数的函数，在直接的ByteBuffer中处理或许是不明智的）？

如果通过上面两种情况仍然不能明确区分的，就使用直接字节缓冲区（direct byte buffer）形式。它们的支持是直接构建到JNI中的，在未来的版本中性能可能会得到提升。

编写:[kesenhoo](#)

原文:<http://developer.android.com/training/articles/smp.html>

# SMP(Symmetric Multi-Processor) Primer for Android

从Android 3.0开始，系统针对多核CPU架构的机器做了优化支持。这份文档介绍了针对多核系统应该如何编写C, C++以及Java程序。这里只是作为Android应用开发者的入门教程，并不会深入讨论这个话题，并且我们会把讨论范围集中在ARM架构的CPU上。

如果你并没有时间学习整篇文章，你可以跳过前面的理论部分，直接查看实践部分。但是我们并不建议这样做。

## 0) 简要介绍

SMP 的全称是“**Symmetric Multi-Processor**”。它表示的是一种双核或者多核CPU的设计架构。在几年前，所有的Android设备都还是单核的。

大多数的Android设备已经有了多个CPU，但是通常来说，其中一个CPU负责执行程序，其他的CPU则处理设备硬件的相关事务（例如，音频）。这些CPU可能有着不同的架构，运行在上面的程序无法在内存中彼此进行沟通交互。

目前大多数售卖的Android设备都是SMP架构的，这使得软件开发者处理问题更加复杂。对于多线程的程序，如果多个线程执行在不同的内核上，这会使得程序更加容易发生**race conditions**。更糟糕的是，基于ARM架构的SMP比起x86架构来说，更加复杂，更难进行处理。那些在x86上测试通过的程序可能会在ARM上崩溃。

下面我们会介绍为何会这样以及如何做才能够使得你的代码行为正常。

# 1)理论篇

这里会快速并且简要的介绍这个复杂的主题。其中一些部分并不完整，但是并没有出现错误或者误导。

查看文章末尾的[进一步阅读](#)可以了解这个主题的更多知识。

## 1.1)内存一致性模型(Memory consistency models)

内存一致性模型(Memory consistency models)通常也被叫做“memory models”，描述了硬件架构如何确保内存访问的一致性。例如，如果你对地址A进行了一个赋值，然后对地址B也进行了赋值，那么内存一致性模型就需要确保每一个CPU都需要知道刚才的操作赋值与操作顺序。

这个模型通常被程序员称为：**顺序一致性(sequential consistency)**，请从文章末尾的进一步阅读查看**Adve & Gharachorloo**这篇文章。

- 所有的内存操作每次只能执行一个。
- 所有的操作，在单核CPU上，都是顺序执行的。

如果你关注一段代码在内存中的读写操作，在sequentially-consistent的CPU架构上，是按照期待的顺序执行的。It's possible that the CPU is actually reordering instructions and delaying reads and writes, but there is no way for code running on the device to tell that the CPU is doing anything other than execute instructions in a straightforward manner. (We're ignoring memory-mapped device driver I/O for the moment.)

To illustrate these points it's useful to consider small snippets of code, commonly referred to as litmus tests. These are assumed to execute in program order, that is, the order in which the instructions appear here is the order in which the CPU will execute them. We don't want to consider instruction reordering performed by compilers just yet.

Here's a simple example, with code running on two threads:

Thread 1 Thread 2 A = 3 B = 5 reg0 = B reg1 = A

**Thread 1      Thread 2**

A = 3 B = 5 reg0 = B reg1 = A

In this and all future litmus examples, memory locations are represented by capital letters (A, B, C) and CPU registers start with “reg”. All memory is initially zero. Instructions are executed from top to bottom. Here, thread 1 stores the value 3 at location A, and then the value 5 at location B. Thread 2 loads the value from location B into reg0, and then loads the value from location A into reg1. (Note that we're writing in one order and reading in another.)

Thread 1 and thread 2 are assumed to execute on different CPU cores. You should always make this assumption when thinking about multi-threaded code.

Sequential consistency guarantees that, after both threads have finished executing, the registers will be in one of the following states:

| Registers      | States                        |
|----------------|-------------------------------|
| reg0=5, reg1=3 | possible (thread 1 ran first) |

reg0=0, reg1=0 possible (thread 2 ran first)  
reg0=0, reg1=3 possible (concurrent execution)  
reg0=5, reg1=0 never

To get into a situation where we see B=5 before we see the store to A, either the reads or the writes would have to happen out of order. On a sequentially-consistent machine, that can't happen.

Most uni-processors, including x86 and ARM, are sequentially consistent. Most SMP systems, including x86 and ARM, are not.

### **1.1.1)Processor consistency**

### **1.1.2)CPU cache behavior**

### **1.1.3)Observability**

### **1.1.4)ARM's weak ordering**

## **1.2)Data memory barriers**

### **1.2.1)Store/store and load/load**

### **1.2.2)Load/store and store/load**

### **1.2.3)Barrier instructions**

### **1.2.4)Address dependencies and causal consistency**

### **1.2.5)Memory barrier summary**

## **1.3)Atomic operations**

### **1.3.1)Atomic essentials**

### **1.3.2)Atomic + barrier pairing**

### **1.3.3)Acquire and release**

## 2)实践篇

调试内存一致性(memory consistency)的问题非常困难。如果内存栅栏(memory barrier)导致一些代码读取到陈旧的数据，你将无法通过调试器检查内存dumps文件来找出原因。By the time you can issue a debugger query, the CPU cores will have all observed the full set of accesses, and the contents of memory and the CPU registers will appear to be in an “impossible” state.

### 2.1)What not to do in C

#### 2.1.1)C/C++ and “volatile”

#### 2.1.2)Examples

### 2.2)在Java中不应该做的事

我们没有讨论过Java语言的一些相关特性，因此我们首先来简要的看下那些特性。

#### 2.2.1)Java中的“synchronized”与“volatile”关键字

“synchronized”关键字提供了Java一种内置的锁机制。每一个对象都有一个相对应的“monitor”，这个监听器可以提供互斥的访问。

“synchronized”代码段的实现机制与自旋锁(spin lock)有着相同的基础结构：他们都是从获取到CAS开始，以释放CAS结束。这意味着编译器(compilers)与代码优化器(code optimizers)可以轻松的迁移代码到“synchronized”代码段中。一个实践结果是：你不能判定synchronized代码段是执行在这段代码下面一部分的前面，还是这段代码上面一部分的后面。更进一步，如果一个方法有两个synchronized代码段并且锁住的是同一个对象，那么在这两个操作的中间代码都无法被其他的线程所检测到，编译器可能会执行“锁粗化lock coarsening”并且把这两者绑定到同一个代码块上。

另外一个相关的关键字是“volatile”。在Java 1.4以及之前的文档中是这样定义的：volatile声明和对应的C语言中的一样可不靠。从Java 1.5开始，提供了更有力的保障，甚至和synchronization一样具备强同步的机制。

volatile的访问效果可以用下面这个例子来说明。如果线程1给volatile字段做了赋值操作，线程2紧接着读取那个字段的值，那么线程2是被确保能够查看到之前线程1的任何写操作。更通常的情况是，任何线程对那个字段的写操作对于线程2来说都是可见的。实际上，写volatile就像是释放件监听器，读volatile就像是获取监听器。

非volatile的访问有可能因为照顾volatile的访问而需要做顺序的调整。例如编译器可能会往上移动一个非volatile加载操作，但是不会往下移动。Volatile之间的访问不会因为彼此而做出顺序的调整。虚拟机会注意处理如何的内存栅栏(memory barriers)。

当加载与保存大多数的基础数据类型，他们都是原子的atomic，对于long以及double类型的数据则不具备原子型，除非他们被声明为volatile。即使是在单核处理器上，并发多线程更新非volatile字段值也还是不确定的。

#### 2.2.2)Examples

下面是一个错误实现的单步计数器(monotonic counter)的示例：[\(Java theory and practice: Managing volatility\)](#).

```

class Counter {
    private int mValue;

    public int get() {
        return mValue;
    }
    public void incr() {
        mValue++;
    }
}

```

假设get()与incr()方法是被多线程调用的。然后我们想确保当get()方法被调用时，每一个线程都能够看到当前的数量。最引人注目的问题是mValue++实际上包含了下面三个操作。

1. reg = mValue
2. reg = reg + 1
3. mValue = reg

如果两个线程同时在执行incr()方法，其中的一个更新操作会丢失。为了确保正确的执行++的操作，我们需要把incr()方法声明为“synchronized”。这样修改之后，这段代码才能够在单核多线程的环境中正确的执行。

然而，在SMP的系统下还是会执行失败。不同的线程通过get()方法获取到得值可能是不一样的。因为我们是使用通常的加载方式来读取这个值的。我们可以通过声明get()方法为synchronized的方式来修正这个错误。通过这些修改，这样的代码才是正确的了。

不幸的是，我们有介绍过有可能发生的锁竞争(lock contention)，这有可能会伤害到程序的性能。除了声明get()方法为synchronized之外，我们可以声明mValue为“volatile”。(请注意incr()必须使用synchronize)现在我们知道volatile的mValue的写操作对于后续的读操作都是可见的。incr()将会稍稍有点变慢，但是get()方法将会变得更加快速。因此读操作多于写操作时，这会是一个比较好的方案。(请参考AtomicInteger.)

下面是另外一个示例，和之前的C示例有点类似：

```

class MyGoodies {
    public int x, y;
}

class MyClass {
    static MyGoodies sGoodies;

    void initGoodies() {      // runs in thread 1
        MyGoodies goods = new MyGoodies();
        goods.x = 5;
        goods.y = 10;
        sGoodies = goods;
    }

    void useGoodies() {      // runs in thread 2
        if (sGoodies != null) {
            int i = sGoodies.x;      // could be 5 or 0
            ...
        }
    }
}

```

```
}
```

这段代码同样存在着问题，`sGoodies = goods`的赋值操作有可能在`goods`成员变量赋值之前被察觉到。如果你使用`volatile`声明`sGoodies`变量，你可以认为`load`操作为`atomic_acquire_load()`，并且把`store`操作认为是`atomic_release_store()`。

(请注意仅仅是`sGoodies`的引用本身为`volatile`，访问它的内部字段并不是这样的。赋值语句`z = sGoodies.x`会执行一个`volatile load` `MyClass.sGoodies`的操作，其后会伴随一个`non-volatile`的`load`操作：：`sGoodies.x`。如果你设置了一个本地引用`MyGoodies localGoods = sGoodies, z = localGoods.x`，这将不会执行任何`volatile loads`。)

另外一个在Java程序中更加常用的范式就是臭名昭著的“**double-checked locking**”：

```
class MyClass {
    private Helper helper = null;

    public Helper getHelper() {
        if (helper == null) {
            synchronized (this) {
                if (helper == null) {
                    helper = new Helper();
                }
            }
        }
        return helper;
    }
}
```

上面的写法是为了获得一个`MyClass`的单例。我们只需要创建一次这个实例，通过`getHelper()`这个方法。为了避免两个线程会同时创建这个实例。我们需要对创建的操作加`synchronize`机制。然而，我们不想要为了每次执行这段代码的时候都为“`synchronized`”付出额外的代价，因此我们仅仅在`helper`对象为空的时候加锁。

在单核系统上，这是不能正常工作的。JIT编译器会破坏这件事情。请查看[4\)Appendix](#)的“‘Double Checked Locking is Broken’ Declaration”获取更多的信息，或者是Josh Bloch’s Effective Java书中的Item 71 (“Use lazy initialization judiciously”)。

在SMP系统上执行这段代码，引入了一个额外的方式会导致失败。把上面那段代码换成C的语言实现如下：

```
if (helper == null) {
    // acquire monitor using spinlock
    while (atomic_acquire_cas(&this.lock, 0, 1) != success)
        ;
    if (helper == null) {
        newHelper = malloc(sizeof(Helper));
        newHelper->x = 5;
        newHelper->y = 10;
        helper = newHelper;
    }
    atomic_release_store(&this.lock, 0);
}
```

此时问题就更加明显了：helper的store操作发生在memory barrier之前，这意味着其他的线程能够在store x/y之前观察到非空的值。

你应该尝试确保store helper执行在atomic\_release\_store()方法之后。通过重新排序代码进行加锁，但是这是无效的，因为往上移动的代码，编译器可以把它移动回原来的位置：在atomic\_release\_store()前面。（这里没有读懂，下次再回读）

有2个方法可以解决这个问题：

- 删除外层的检查。这确保了我们不会在synchronized代码段之外做任何的检查。
- 声明helper为volatile。仅仅这样一个小小的修改，在前面示例中的代码就能够在Java 1.5及其以后的版本中正常工作。

下面的示例演示了使用volatile的2各重要问题：

```
class MyClass {  
    int data1, data2;  
    volatile int vol1, vol2;  
  
    void setValues() {      // runs in thread 1  
        data1 = 1;  
        vol1 = 2;  
        data2 = 3;  
    }  
  
    void useValues1() {      // runs in thread 2  
        if (vol1 == 2) {  
            int l1 = data1;      // okay  
            int l2 = data2;      // wrong  
        }  
    }  
    void useValues2() {      // runs in thread 2  
        int dummy = vol2;  
        int l1 = data1;      // wrong  
        int l2 = data2;      // wrong  
    }  
}
```

请注意useValues1()，如果thread 2还没有察觉到vol1的更新操作，那么它也无法知道data1或者data2被设置的操作。一旦它观察到了vol1的更新操作，那么它也能够知道data1的更新操作。然而，对于data2则无法做任何猜测，因为store操作是在volatile store之后发生的。

useValues2()使用了第2个volatile字段：vol2，这会强制VM生成一个memory barrier。这通常不会发生。为了建立一个恰当的“happens-before”关系，2个线程都需要使用同一个volatile字段。在thread 1中你需要知道vol2是在data1/data2之后被设置的。（The fact that this doesn't work is probably obvious from looking at the code; the caution here is against trying to cleverly “cause” a memory barrier instead of creating an ordered series of accesses.）

## 2.3)What to do

### 2.3.1)General advice

在C/C++中，使用pthread操作，例如mutexes与semaphores。他们会使用合适的memory barriers，在所有的Android平台上提供正确有效的行为。请确保正确这些技术，例如在没有获得对应的mutex的情况下赋值操作需要很谨慎。

避免直接使用atomic方法。如果locking与unlocking之间没有竞争，locking与unlocking一个pthread mutex 分别需要一个单独的atomic操作。如果你需要一个lock-free的设计，你必须在开始写代码之前了解整篇文档的要点。（或者是寻找一个已经为SMP ARM设计好的库文件）。

Be extremely circumspect with "volatile" in C/C++. It often indicates a concurrency problem waiting to happen.

In Java, the best answer is usually to use an appropriate utility class from the java.util.concurrent package. The code is well written and well tested on SMP.

Perhaps the safest thing you can do is make your class immutable. Objects from classes like String and Integer hold data that cannot be changed once the class is created, avoiding all synchronization issues. The book Effective Java, 2nd Ed. has specific instructions in “Item 15: Minimize Mutability”. Note in particular the importance of declaring fields “final” (Bloch).

If neither of these options is viable, the Java “synchronized” statement should be used to guard any field that can be accessed by more than one thread. If mutexes won’t work for your situation, you should declare shared fields “volatile”，but you must take great care to understand the interactions between threads. The volatile declaration won’t save you from common concurrent programming mistakes, but it will help you avoid the mysterious failures associated with optimizing compilers and SMP mishaps.

The Java Memory Model guarantees that assignments to final fields are visible to all threads once the constructor has finished — this is what ensures proper synchronization of fields in immutable classes. This guarantee does not hold if a partially-constructed object is allowed to become visible to other threads. It is necessary to follow safe construction practices.(Safe Construction Techniques in Java).

### 2.3.2)Synchronization primitive guarantees

The pthread library and VM make a couple of useful guarantees: all accesses previously performed by a thread that creates a new thread are observable by that new thread as soon as it starts, and all accesses performed by a thread that is exiting are observable when a join() on that thread returns. This means you don’t need any additional synchronization when preparing data for a new thread or examining the results of a joined thread.

Whether or not these guarantees apply to interactions with pooled threads depends on the thread pool implementation.

In C/C++, the pthread library guarantees that any accesses made by a thread before it unlocks a mutex will be observable by another thread after it locks that same mutex. It also guarantees that any accesses made before calling signal() or broadcast() on a condition variable will be observable by the woken thread.

Java language threads and monitors make similar guarantees for the comparable operations.

### 2.3.3)Upcoming changes to C/C++

The C and C++ language standards are evolving to include a sophisticated collection of atomic operations. A full matrix of calls for common data types is defined, with selectable memory barrier semantics (choose from relaxed, consume, acquire, release, acq\_rel, seq\_cst).

See the Further Reading section for pointers to the specifications.

### 3) Closing Notes

While this document does more than merely scratch the surface, it doesn't manage more than a shallow gouge. This is a very broad and deep topic. Some areas for further exploration:

- Learn the definitions of **happens-before**, **synchronizes-with**, and other essential concepts from the Java Memory Model. (It's hard to understand what "volatile" really means without getting into this.)
- Explore what compilers are and aren't allowed to do when reordering code. (The JSR-133 spec has some great examples of legal transformations that lead to unexpected results.)
- Find out how to write immutable classes in Java and C++. (There's more to it than just "don't change anything after construction".)
- Internalize the recommendations in the Concurrency section of **Effective Java, 2nd Edition**. (For example, you should avoid calling methods that are meant to be overridden while inside a synchronized block.)
- Understand what sorts of barriers you can use on x86 and ARM. (And other CPUs for that matter, for example Itanium's acquire/release instruction modifiers.)
- Read through the **java.util.concurrent** and **java.util.concurrent.atomic** APIs to see what's available.
- Consider using concurrency annotations like `@ThreadSafe` and `@GuardedBy` (from `net.jcip.annotations`).

The Further Reading section in the appendix has links to documents and web sites that will better illuminate these topics.

## **4)Appendix**

**4.1)SMP failure example**

**4.2)Implementing synchronization stores**

**4.3)Further reading**

编写:[craftsmanBai - http://z1ng.net](http://z1ng.net)

原文:<http://developer.android.com/training/best-security.html>

# 安全与隐私

这些课程文章告诉你怎样让app数据得到安全保护。

## 安全要点

怎样执行在执行多个任务的同时保护app数据和用户数据安全。

## HTTPS和SSL的安全

怎样确保app进行网络传输时是安全的。

## 企业级开发

怎样为企业级的app实施设备管理策略。

编写:[craftsmanBai](http://craftsmanBai.com) - <http://z1ng.net> - 校正:

参考:<http://su1216.iteye.com/blog/>

原文:<http://developer.android.com/training/articles/security-tips.html>

# 安全要点

Android的安全特性体现在操作系统显著地减少了应用程序的安全问题带来的影响。你可以在默认的系统设置和文件权限设置中建立app，避免了安全带来的一些头疼问题。帮助你建立app的一些核心安全特性如下：

- Android应用程序沙盒，将你的app数据和代码执行同其他程序隔开。
- 鲁棒性实现常见安全功能的应用框架，例如密码学应用，权限控制，安全IPC
- ASLR, NX, ProPolice, safe\_iop, OpenBSD dmalloc, OpenBSD calloc, Linux mmap\_min\_addr等技术减少了常见内存管理错误。
- 加密文件系统可以保护丢失的或被盗走的设备的数据。
- 用户权限控制限制了访问系统详细情况和用户数据。
- 应用程序权限以单个app为基础控制了应用程序的数据。

尽管如此，熟悉Android安全特性是很重要的。遵守这些习惯将其作为好的代码风格，将会减少不经意间给用户带来的安全问题。

# 数据存储

对于一个Android的应用程序来讲，最为常见的安全问题是存放在设备上的数据能否被其他app获取。在设备上存放保存数据有三种基本的方式：

## 使用内存储器

默认情况下，你在[内存储器](#)中创建的文件只有你的app可以访问。这种机制被Android加强了并且对于大多数应用程序都是有效的。你应该避免在IPC文件中使用[MODE\\_WORLD\\_WRITEABLE](#)或者[MODE\\_WORLD\\_READABLE](#)模式，因为它们不对特殊程序提供限制数据访问的功能，它们也不对数据格式提供任何控制。如果你想同其他app的进程共享数据，你可以使用一个[content provider](#)，它给其他apps提供了可读可写的权限并且可以逐项动态的获取权限。

如果想对一些敏感数据提供特别的保护，你可以选择使用应用程序无法直接获取的密钥来加密本地文件。例如，密钥可以存放在[密钥库](#)，使用用户密码保护的密钥，而不存储在设备上。尽管这种方式不能在具有root权限时监视用户输入的密码的情况下保护数据，但是它可以提供对没有进行[文件系统加密](#)的丢失的设备的保护。

## 使用外部存储器

建立在[外部存储](#)的文件，比如sd卡，是全局可读写的。因为外部存储可以被用户移除并且可被任何应用修改，应用不应该使用外部存储存储敏感信息。当处理从外部存储来的数据时应用应该[执行输入验证](#)（参看输入验证章节）我们强烈建议应用在动态加载之前不把可执行或者是class文件存储到外部存储中。如果一个应用从外部存储检索可执行文件，在动态加载之前，他们应该被签名和加密验证。

## 使用content providers

[ContentProviders](#)提供一个结构存储机制，可以限制你自己的应用，或者导出给其他应用程序允许访问。如果你不打算为其他应用提供访问你的[ContentProvider](#)功能，在manifest中标记他们为[android:exported=false](#)即可。当建立一个由其他应用为使用而导出的[ContentProvider](#)，你可以为读写指定一个单一的[许可](#)，或者在manifest中为读写指定确切的许可。我们强烈建议你把你的许可限制在那些必要的事情上来完成临近的任务。记住，通常显示新功能稍后加入许可要比把许可拿开并且打断已经存在的用户要容易。

如果你正在使用ContentProvider在相同开发者的应用间来分享数据，使用签名级别[android:protectionLevel](#)的许可是更可取的。签名许可不需要用户确认，所以这提供一个更好的用户体验并且更能控制ContentProvider访问。ContentProviders也可以通过声明[android:grantUriPermissions](#)元素并且在触发组件的Intent对象中使用[FLAG\\_GRANT\\_READ\\_URI\\_PERMISSION](#)和[FLAG\\_GRANT\\_WRITE\\_URI\\_PERMISSION](#)来提供更颗粒状的访问。这些许可的作用域可以通过[grant-uri-permission](#)元素进一步的限制。当访问一个ContentProvider时，使用参数化的查询方法，比如[query\(\)](#), [update\(\)](#), 和[delete\(\)](#)来避免来自不被新人的数据潜在的SQL注入。注意，如果提交到方法之前的连接是通过连接用户数据建立的，使用参数化的方法是不够的。不要对“写”的许可安全有一个错误的观念 考虑“写”的许可允许sql语句使得一些数据被确认使用创造性的WHERE从句并且分析结果变为可能。例如：一个入侵者可能在通话记录中通过修改一条记录来侦察一个存在的特定的电话号码，只要那个电话号码已经存在。如果content provider数据有可预见的结构，“写”许可也许与提供了“读写”等效了。

# 使用权限

因为安卓沙盒使应用程序隔开，程序必须显式地共享资源和数据。为了拥有附加的功能，他们通过声明他们需要的权限，而基本的沙盒不提供这些功能，包括访问设备比如相机的特性。

## 请求许可

我们建议一个应用请求的许可数量最小化，不具有访问敏感的许可可以减少无意中滥用那些许可的风险，可以让用户更能接受，并且使得攻击者对应用减少兴趣。

如果你的应用有一种可以设计出不需要任何许可的方法，那最好不过。例如：与其请求访问设备信息来建立一个标识，不如建立一个[GUID](#)（这个例子也在[Handling User Data](#)中有讨论）。

除了请求许可之外，你的应用可以使用[许可](#)来保护安全敏感的IPC并且会暴露给其他应用：比如[ContentProvider](#)。总的来说，我们建议使用访问控制而不是在可能的地方让用户确认许可，因为许可会是用户困惑。例如，考虑在许可上为应用间的IPC通信使用单一开发者提供的[签名保护级别](#)

不要产生许可再次授权，这只有当一个应用通过IPC暴露数据才会发生，因为它有一个指定的许可，但是并不要求它的IPC接口的任何客户端许可。潜在影响的更多细节，和这种问题发生的频率在USENIX: [http://www.cs.berkeley.edu/~afelt/felt\\_usenixsec2011.pdf](http://www.cs.berkeley.edu/~afelt/felt_usenixsec2011.pdf)研究论文中都有提供。

## 创建许可

一般来说，你应该力求建立拥有尽量少许可的应用，直至满足你的安全需要。建立一个新的许可对于大多数应用是相对不常见，因为[系统定义的许可](#)覆盖很多情况。在适当的地方使用已经存在的许可执行访问检查。

如果你必须建立一个新的许可，考虑是否你能使用[签名许可](#)完成你的任务。签名许可对用户是透明的并且只允许相同开发者签名的应用访问同应用执行许可检查一样。如果你建立一个[危险的许可](#)，那么用户需要决定是否安装这个应用。这会使其他开发者困惑，也使用户困惑。

如果你建立一个危险的许可，那么会有非常多的复杂情况需要你考虑：

- 许可必须有一个字符串简短的表述给用户他们将要要求做出的安全策略
- 许可字符必须做很多语言的国际化
- 用户也许由于对一个许可风险的困惑或者知晓而选择不安装应用

上面每一个因素都都应用开发者提出一个重要的非技术的挑战，这也是我们劝阻使用危险许可的原因。

## 使用网络

网络交易具有很高的安全风险，因为它涉及到传送私人的数据。人们对移动设备的隐私关注日益加深，特别是当设备进行网络交易时，因此app采用最好的方式保护用户诗句是非常重要的。

### 使用IP网络

android上面的网络与Linux环境上的差别不是很大。主要考虑的是保证对敏感数据使用适当的协议，比如使用[HTTPS进行网络传输](#)。我们在任何服务器支持HTTPS的地方更愿意使用HTTPS而不是HTTP，因为移动设备频繁连接不安全的网络，比如公共的WiFi热点。

认证的、加密的socket级别的通信可以使用[SSLSocket](#)类轻松的实现。根据Android设备使用WiFi连接不安全网络的频率，对于所有应用来说，使用安全网络是强烈被支持的。

我们见过一些应用使用[本地网络](#)端口处理敏感的IPC。我们不鼓励这种方法因为这些接口是可以被设备上的其他应用访问的。取而代之，在可以认证的地方使用一个android IPC机制，例如[Service](#)（比使用回环还糟的是绑定INADDR\_ANY，因为你的应用也许收到任何地方来的请求。我们也已经见识过了）。

一个有必要重复的常见的议题是，保证你不信任从HTTP或者其他不安全协议下载的数据。这包括在[WebView](#)中的输入验证和相对于http的任何响应。

### 使用电话网络

SMS是Android开发者使用最频繁的电话协议。开发者应该记住这个协议主要是设计为用户与用户之间的交流，它并不适用一些应用的目的。由于SMS的限制，我们强烈建议使用[Google Cloud Messaging](#) (GCM) 和IP网络发送数据消息给设备。

很多开发者没有意识到SMS在网络上或者设备上不是加密的或者牢固验证的。尤其是，任何SMS接收者应该预料到恶意用户也许已经给你的应用发送了SMS：不要指望未验证的SMS数据执行敏感操作。你也应该注意到SMS在网络上也许会遭到冒名顶替并且/或者拦截，在Android设备本身上面，SMS消息是通过广播intent传递的，所以他们也许会被其他拥有[READ\\_SMS](#)许可的应用截获。

## 输入验证

不管应用运行在什么平台上，功能不完善的输入验证是最常见的影响应用安全问题之一。Android有平台级别的对策，用于减少应用的公开输入验证问题，你应该在可能的地方使用这些功能。同样需要注意的是，安全类型语言的选择倾向去减少输入验证问题的可能。我们强烈建议使用Android SDK建立你的应用。

如果你使用native代码，那么任何从文件读取的数据，通过网络接收的，或者通过IPC接收的都有可能引入安全问题。最常见的问题是缓存溢出，释放后使用，和off-by-one错误。Android提供一些技术比如ASLR和DEP减少这些错误的可利用性，但是他们没有解决基本的问题。小心处理指针和管理缓存可以预防这些问题。

动态，基于语言的字符串，比如JavaScript和SQL，都常遭受由转义字符和[脚本注入](#)带来的输入验证问题。

如果你使用提交到SQL Database或者Content Provider查询中数据，SQL也许会是个问题。最好的防御是使用参数化的查询，同ContentProviders中讨论的那样。限制权限为只读或者只写可以减少SQL注入潜在危害。

如果你不能使用上面提到的安全功能，我们强烈建议使用结构严谨的数据格式并且验证符合期望的格式。黑名单策略与替换危险字符是一种有效的策略，这些技术在实践中是易错并且当错误可能发生的时候应该尽量避免。

## 处理用户数据

一般来说，最好的处理方法是最小化反问敏感或个人数据的API使用。如果你有对数据的访问并且可以避免存储或者传输信息，那就不要存储或者传输数据。最后，考虑如果有一种你的应用逻辑可能被实现为使用hash或者不可逆形式的数据的方法。例如，你的应用也许使用一个email地址的hash作为主键，避免传输或存储email地址，这减少无意暴露数据的机会，并且它也能减少攻击者尝试利用你的应用的机会。

如果你的应用访问私人数据，比如密码或者用户名，记住司法权也许要求你提供一个你使用和存储这些数据的隐私策略的解释。所以采用最小化访问用户数据的安全最佳实践也许也是单纯的顺从。

你也应该考虑你应用是否会疏忽的暴露个人信息给其他方，比如广告第三方组件或者你应用使用的第三方服务。如果你不知道为什么一个组件或者服务请求个人信息，那么就不要提供给它。一般来说说，通过减少你应用中访问个人信息，将会减少这个区域潜在的问题。

如果必须访问敏感数据，评估这个信息是否必须要传到服务器，或者是否可以被客户端操作。考虑客户端上使用敏感数据运行的任何代码避免传输用户数据 保证你不会无意中通过过渡自由的IPC，world writable文件，或者网络socket暴露用户数据给其他设备上的应用。这是一个再次授权的特别的例子，在[请求权限](#)章节中讨论。

如果请求全球唯一标识符，建立一个大的，唯一的数字并保存它。不要使用电话标识，比如与个人信息相关的电话号码或者IMEI。这个话题在[Android Developer Blog](#)中有更详细的讨论，应用开发这应该谨慎的把log写到机器上。

在Android中，log是共享资源，一个带有[READ\\_LOGS](#)许可的应用可以访问。即使电话log数据是临时的并且在重启之后会擦除，不恰当的记录用户信息也会无意的泄漏用户数据给其他应用。

## 使用WebView

因为[WebView](#)能包含HTML和JavaScript浏览网络内容，不恰当的使用会引入常见的web安全问题，比如[跨站脚本攻击](#)（JavaScript注入）。Android包含一些机制通过限制WebView的能力到你应用请求的功能最小化来减少这些潜在问题的范围。

如果你的应用没有在WebView内直接使用JavaScript，不要调用[setJavaScriptEnabled\(\)](#)。我们见过这个方法在简单的代码中执行，也许会导致在产品应用中改变用途：所以如果必要的化移除它。默认的，WebView不执行JavaScript，所以跨站脚本攻击不可能产生。

使用[addJavaScriptInterface\(\)](#)要特别小心，因为它允许JavaScript执行通常保留给Android应用的操作。只把addJavaScriptInterface()暴露给可靠的输入源。如果不受信任的输入是被允许的，不受信任的JavaScript也许会执行Android方法。总的来说，我们建议只把addJavaScriptInterface()暴露给你应用apk内包含的JavaScript。

如果你的应用通过WebView访问敏感数据，你也许想要使用[clearCache\(\)](#)方法来删除任何存储到本地的文件。服务端的header，比如no-cache，能用于指示应用不应该缓存特定的内容。

## 处理证书

一般来说，我们建议请求用户证书频率最小化：使得钓鱼攻击更明显，并且降低其成功的可能。取而代之使用授权令牌然后刷新它。

可能的情况下，用户名和密码不应该存储到设备上，取而代之，使用用户提供的用户名和密码执行初始认证，然后使用一个短暂的，特定服务的授权令牌。可以被多个应用访问的service应该使用[AccountManager](#)访问。如果可能的话，使用AccountManager类来执行基于云的服务并且不把密码存储到设备上。

使用AccountManager获取[Account](#)之后，进入任何证书前检查[CREATOR](#)，这样你就不会因为疏忽而把证书传递给错误的应用。

如果证书只是用于你建立的应用，那么你能使用[checkSignature\(\)](#)验证访问AccountManager的应用。另一种选择，如果一个应用要使用证书，你可以使用一个[KeyStore](#)来储存。

## 使用密码学

除了提供数据隔离之外，支持完整的文件系统加密，并且提供安全交流通道。Android提供大量加密算法来保护数据。

一般来说，尝试使用最高级别的以存在framework的实现能支持你的用例，如果你需要安全的从一个已知的位置取回一个文件，一个简单的HTTPS URI也许就足够了，并且这部分不要求任何加密知识。如果你需要一个安全隧道，考虑使用[HttpsURLConnection](#)或者[SSLSocket](#)，要比使用你自己的协议好。

如果你发现你的确需要实现你自己的协议，我们强烈建议你不要实现你自己的加密算法。使用已经存在的加密算法，比如[Cipher](#)类中提供的AES或者RSA的实现。

使用一个安全的随机数生成器（[SecureRandom](#)）来初始化加密的key（[KeyGenerator](#)）。使用一个不受由安全随机数生成器生成的key严重削弱算法的优点，而且有能允许离线攻击。

如果你需要存储一个key来重复使用，使用类似于[KeyStore](#)的机制，提供一种机制长期储存和检索加密的key。

# 使用进程间通信

一些Android应用试图使用传统的Linux技术实现IPC，比如网络socket和共享文件。我们强烈鼓励使用Android系统IPC功能，比如[Intent](#), [Binder](#), [Messenger](#)和[BroadcastReceiver](#)。Android IPC机制允许你为每一个IPC机制验证连接到你的IPC和设置安全策略的应用的身份。

很多安全元素通过IPC机制共享。Broadcast Receiver, Activity和Service都在应用的manifest中声明。如果你的IPC机制不打算给其他应用使用，设置`android:exported`属性为`false`。这对由同一个UID内多个进程应用，或者你在开发后期决定不想通过IPC暴露功能但是你又不想重写代码是很有用的。

如果你的IPC打算让别的应用访问，你可以通过使用Permission标记设置一个安全策略。如果IPC是相同开发者应用间的，使用[签名级的许可](#)更好一些。

## 使用意图

[Intent](#)是Android中异步IPC机制的首选。根据你应用的需求，你也许使用[sendBroadcast\(\)](#),[sendOrderedBroadcast\(\)](#)或者直接的intent来指定一个应用组件。

注意，有序广播可以被接收者“消费”，所以他们也许不会被发送到所有的应用中。如果你打算在必须发送给一个指定的receiver的地方发送一个intent，这个intent必须被直接的发送给这个receiver。

intent的发送者能在发送的时候验证接受者是否有一个许可指定了一个non-Null Permission。只有有那个许可的应用才会收到这个intent。如果在广播intent内的数据是敏感的，你应该考虑使用一个许可来保证恶意应用没有恰当的许可无法注册接收那些消息。那种环境下，你也许也考虑直接执行这个receiver而不是发起一个广播。

## 使用服务

[Service](#)经常被用于为其他应用提供功能供其使用。每一个service类必须在它的包的AndroidManifest.xml中有相应的声明。

默认的，Service被导出并且可以被其他应用执行。可以在manifest文件中的标记使用[android:permission](#)保护Service。这样做，其他应用在他们自己的manifest文件中将需要声明一个相应的元素来启动，停止或者绑定到这个service上。

一个Service可以保护单独的具有许可的IPC调用它，在执行那个调用的实现之前，通过调用[checkCallingPermission\(\)](#)实现保护。我们一般建议使用manifest中声明的许可，因为那些是不容易监管的

## 使用binder和AIDL接口

在Android中，[Binders](#)是RPC-style IPC的首选机制。必要的话，他们提供一个定义明确的接口，促进彼此的端点认证。我们强烈鼓励在一定程度上设计不要求接口指定许可检查的接口。Binder不在应用的manifest中声明，并且因此你不能直接在Binder上应用声明的许可。Binder继承在应用在manifest中[Service](#)或者[Activity](#)声明的，Service或者Activity内实现了的许可。如果你打算建立一个接口，在一个指定binder接口上要求验证并且（或者）要求访问控制，这些控制必须在接口中清楚的在代码中添加。

如果提供一个需要访问控制的接口，使用[checkCallingPermission\(\)](#)来验证Binder的主叫者是否拥有必要的许可。因为你的应用的id已经被传递到别的interface，所以代表访问一个

Service之前这尤其重要。如果执行一个service提供的接口，如果你没有对给定的service的访问许可，[bindService\(\)](#)请求也许会失败。如果调用一个你自己应用提供的本地的接口，使用[clearCallingIdentity\(\)](#)来消除内部的安全检查也行是有用的。

## 利用广播接收机

[Broadcast receivers](#)是用来处理通过[intent](#)发起的异步请求。

默认的，receiver是导出的并且可以被其他任何应用执行。如果你的BroadcastReceiver打算让其他应用使用，你也许想要在应用的manifest文件中使用元素对receiver应用安全许可。这将阻止没有恰当许可的应用发送intent给这个BroadcastReceiver。

## 动态加载代码

我们强烈不鼓励从应用apk文件外加载代码。这样做显著增加了应用泄漏的可能，取决于代码注入或者代码篡改，也增加了版本管理和应用测试的复杂性。最终，它会使得不可验证一个应用的行为，所以它也许在一些环境下被进制。

如果你的应用确实动态加载了代码，最重要的事情是记住运行动态加载的代码与应用apk具有相同的安全许可。用户决定安装你的应用是基于你的id，他们期望你提供任何在应用内运行的代码，包括动态加载的代码与动态加载代码结合的重要的安全风险是代码需要代码需要来自可信资源。

如果这个模块是之间在你的apk中包含，那么他们不能被其他应用修改，不论代码是本地库或者是使用DexClassLoader加载的类这都是事实。我们见过很多应用实例尝试从不安全的位置加载代码，比如从网络上通过非加密的协议下载或者从world writable位置（比如外部存储）。这些位置会允许网络上的人在传输过程中修改其内容，或者允许用户设备上的另一个应用修改其内容。

## 在虚拟机器安全性

编写运行在虚拟机的安全代码是一个精心研究的话题，很多问题并不特指在Android上。相比尝试重新讲解这些话题，我们推荐你熟悉已有的文献。

- <http://www.securingjava.com/toc.html>
- [https://www.owasp.org/index.php/Java\\_Security\\_Resources](https://www.owasp.org/index.php/Java_Security_Resources)

这个文档集中于Android专有的并/或者与其他环境不同地方。对于有在其他环境上的VM编程经验开发者，这有这两个普遍的问题也许对于编写Android应用来说有些不同：

- 一些虚拟机，比如JVM或者.net，担任一个安全的边界作用，代码与底层操作系统能力相隔离。在Android上，Dalvik VM不是一个安全边界：应用沙箱是在系统级别实现的，所以Dalvik可以在同一个应用与native代码相互操作没有任何约束。
- 已知的手机上的存储限制，对开发者来说，想要建立模块化应用和使用动态类加载是很常见的。当这么做的时候，要考虑两个资源：一个是你在哪里恢复你的应用逻辑，另一个是你在哪里存储它们。不要从未验证的资源使用动态类加载器，比如不安全的网络资源或者外部存储，因为那些代码可能被修改为包含恶意的行为。

## 在本地代码的安全

一般来说，对于大多数应用开发，我们鼓励开发者使用Android SDK而不是使用[Android NDK] (<http://developer.android.com/tools/sdk/ndk/index.html>) 的native代码。编译native代码的应用更为复杂，移植性差，更容易包含常见的内存崩溃错误，比如缓冲区溢出。

Android使用Linux内核编译并且与Linux开发相似，如果你打算使用native代码，安全最佳实践尤其有用。这篇文档讨论这些所有的最佳实践实在太短了，但是最受欢迎的资源之一是“Secure Programming for Linux and Unix HOWTO”，在这里可以找到<http://www.dwheeler.com/secure-programs.Android>。

和大多数Linux环境之前的一个重要区别是应用沙箱。在Android中，所有的应用运行在应用沙箱中，包括那些用native代码编写的应用。在最基本的级别中，对于开发者来说，一种考虑它的好办法与Linux相似，知道每一个应用被分配一个具有非常有限权限的唯一UID。这里讨论的比[Android Security Overview](#)中更细节化，你应该熟悉应用许可，即使你使用的是native代码。

下一篇：[使用HTTPS与SSL](#)

编写:[craftsmanBai](http://craftsmanBai.com) - <http://z1ng.net> - 校正:

原文: <http://developer.android.com/training/articles/security-ssl.html>

# 使用HTTPS与SSL

SSL，传输层安全([TSL](#))，是一个常见的用来加密客户端和服务器通信的模块。但是应用程序错误地使用SSL可能会导致应用程序的数据在网络中被恶意攻击者拦截。为了帮助你确保这种情况不在你的应用程序中发生，这篇文章突出讲解了使用网络安全协议常见的陷阱和使用[Public-Key Infrastructure\(PKI\)](#)时一些值得关注的问题。

## 概念

一个典型的SSL使用场景是，服务器配置中包含了一个证书，并且有匹配的公钥和私钥。作为SSL客户端和服务端握手的一部分，服务端通过使用[public-key cryptography\(公钥加密算法\)](#)进行证书签名来证明它有私钥。

然而，任何人都可以生成他们自己的证书和私钥，因此一次简单的握手不能证明服务端具有匹配证书公钥的私钥。一种解决这个问题的方法是让客户端拥有一套或者更多的可信赖的证书。如果服务端提供的证书不在其中，那么它将不能得到客户端的信任。

这种简单的方法有一些缺陷。服务端应该根据时间升级到强壮的密钥(key rotation)，更新证书中的公钥。不幸的是，现在客户端app需要根据服务端配置的变化来进行更新。如果服务端不在应用程序开发者的控制下，问题将变得更加麻烦，比如它是一个第三方网络服务。如果程序需要和任意的服务器进行对话，例如web浏览器或者email应用，这种方法也会带来问题。

为了解决这个问题，服务端通常配置了知名的的发行者证书(称为[Certificate Authorities\(CAs\)](#))提供的平台通常包含了一系列知名可信赖的CAs。在Android4.2(Jelly Bean)，Android现在包含了超过100CAs在每个发行版中更新。和服务端相似的是，一个CA拥有一个证书和一个私钥。当为一个服务端发布颁发证书的时候，CA用它的私钥为服务端签名。客户端可以通过服务端拥有被已知平台CA签名的证书来确认服务端。

然而，使用CAs又带来了其他的问题。因为CA为许多服务端证书签名，你仍然需要其他的方法来确保你对话的是你想要的服务器。为了解决这个问题，使用CA签名的的证书通过特殊的名字如 gmail.com 或者带有通配符的域名如 \*.google.com 来确认服务端。下面这个例子会使这些概念具体化一些。在这个片段中，[openssl](#)工具的客户端命令关注Wikipedia服务端证书信息。端口为443因为默认为HTTPS。这条命令将open s\_client的输出发送给 openssl x509，根据[X.509 standard](#)格式化证书中的内容。特别的是，这条命令需要subject参数，包含服务端名字和issuer来确认CA。

```
$ openssl s_client -connect wikipedia.org:443 | openssl x509 -noout  
subject= /serialNumber=sOrr2rKpMVP70Z6E9BT5reY008SJEdYv/C=US/O=*.w  
issuer= /C=US/O=GeoTrust, Inc./CN=RapidSSL CA
```

可以看到是RapidSSL CA颁发给匹配\*.wikipedia.org服务端的证书。

## 一个HTTP的例子

假设你有一个知名CA颁发证书的web服务器，你可以使用下面的代码发送一个安全请求：

```
URL url = new URL("https://wikipedia.org");
URLConnection urlConnection = url.openConnection();
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

是的，它就是这么简单。如果你想要剪裁HTTP的请求，你可以把它投到[HttpURLConnection](#)。Android关于[HttpURLConnection](#)文档中还有进一步的例子关于怎样去处理请求，响应头，posting的内容，管理cookies，使用代理，抓responses等等。但是就这些确认证书和域名的细节而言，Android框架已经通过API来为你考虑这些细节。下面是其他的需要关注的问题。

## 服务器普通问题的验证

假设替代从[getInputStream\(\)](#)接受内容，它抛出了一个异常：

```
javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathVa  
at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl  
at libcore.net.http.HttpConnection.setupSecureSocket (HttpC  
at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.mak  
at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.con  
at libcore.net.http.HttpEngine.sendSocketRequest (HttpEngin  
at libcore.net.http.HttpEngine.sendRequest (HttpEngine.java  
at libcore.net.http.HttpURLConnectionImpl.getResponse (Http  
at libcore.net.http.HttpURLConnectionImpl.getInputStream (H  
at libcore.net.http.HttpsURLConnectionImpl.getInputStream (
```

这种情况发生的原因包括： 1.[颁布证书给服务器的CA不是知名的。](#) 2.[服务器证书不是CA签名的而是自己签名的。](#) 3.[服务器配置缺失了中间CA](#)

下面将会分别讨论当你保持和服务器的安全连接时如何去解决这些问题。

## 无法识别证书机构

在这种情况下，[SSLHandshakeException](#)异常产生的原因是你有一个不被系统信任的CA。可能是你的证书来源于新CA而不被安卓信任，也可能是你的app运行版本较老没有CA。更多的时候，一个CA不知名是因为它不是公开的CA，而是政府，公司，教育机构等组织私有的。

幸运的是，你可以教会[HttpsURLConnection](#)学会信任特殊的CA。过程可能会让人感到有一些费解，下面这个例子是从[InputStream](#)中获得特殊的CA，使用它去创建一个密钥库，用来创建和初始化[TrustManager](#)。[TrustManager](#)是系统用来验证服务器证书的，这些证书通过使用[TrustManager](#)信任的CA和密钥库中的密钥创建。在新的[TrustManager](#)中，下面这个例子初始化了一个新的[SSLContext](#)，提供了一个[SSLSocketFactory](#)，你可以从[HttpsURLConnection](#)中覆盖[SSLSocketFactory](#)。这样连接中会使用你的CA来进行证书验证。

下面是一个华盛顿的大学的CA使用例子

```
// Load CAs from an InputStream
// (could be from a resource or ByteArrayInputStream or ...)
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// From https://www.washington.edu/itconnect/security/ca/load-der.
InputStream caInput = new BufferedInputStream(new FileInputStream(
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}

// Create a KeyStore containing our trusted CAs
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Create a TrustManager that trusts the CAs in our KeyStore
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// Create an SSLContext that uses our TrustManager
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

// Tell the URLConnection to use a SocketFactory from our SSLContext
URL url = new URL("https://certs.cac.washington.edu/CATest/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection)url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

使用一个常用的知道你的CA的TrustManager，系统可以确认你的服务器证书来自于一个可信任的发行者。

注意:许多网站描述了一种简陋的二选一方案是否安装TrustManager。如果这样你最好不要加密通讯过程，因为任何人都可以在公共wifi热点下，使用他们伪装成你的服务器的代理发送你的用户流量，进行DNS欺骗，来攻击你的用户。然后攻击者便可记录用户密码和其他个人资料。这种方式奏效是因为攻击者可以生成一个证书，并且缺少可以验证该证书是否来自受信任的来源的TrustManager。你的app可以同任何人会话。所以不这样做，暂时的也不行。如果你能总是让你的app信任服务器证书的发行者，那么你可以这么做。

## 自签名服务器证书

第二种[SSLHandshakeException](#)取决于自签名证书，意味着服务器就是它自己的CA。这同未知证书权威机构类似，因此你同样可以用前面部分中提到的方法。你可以创建你自己的TrustManager，这一次直接信任服务器证书。这有之前提到的将你的app直接捆绑证书的所有缺点，但是可以安全的执行。然而，你应该小心确保你的自签名证书拥有合适的强密钥。到2012年，一个2048位65537指数位一年到期的RSA签名是合理的。当轮换密钥时，你应该查看权威机构(比如[NIST](#))的建议([recommendation](#))来了解哪种密钥是合适的。

## 缺少中间证书颁发机构

第三种SSLHandshakeException情况的产生于缺少中间CA。大多数公开的CA不直接给服务器签名。相反，他们使用它们主要的机构（简称根认证机构）证书来给中间认证机构签名，他们这样做，因此根认证机构可以离线存储减少危险。然而，操作系统典型的比如安卓只信任直接地根认证机构，在服务器证书(由中间证书颁发机构签名)和证书验证者(只知道根认证机构)之间留下了一个缺口。为了解决这个问题，服务器并不SSL握手的过程中向客户端发送它的证书，而是一系列从服务器到必经的任何中间机构到达根认证机构的证书。

下面是一个 mail.google.com证书链，以openssl\_client命令显示：

```
$ openssl s_client -connect mail.google.com:443
---
Certificate chain
0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=mail.goog
i:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
1 s:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certificatio
---
```

这里显示了一台服务器发送了一个Thawte SGC CA为mail.google.com颁发的证书，Thawte SGC CA是一个中间证书颁发机构，Thawte SGC CA的证书由被安卓信任的Verisign CA颁发。然而，配置一台服务器不包括中间证书机构是不常见的。例如，一台服务器导致安卓浏览器的错误和app的异常：

```
$ openssl s_client -connect egov.uscis.gov:443
---
Certificate chain
0 s:/C=US/ST=District Of Columbia/L=Washington/O=U.S. Department
i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms of
---
```

更有趣的是，用大所属桌面浏览器访问这台服务器不会导致类似于完全未知CA的或者自签名的服务器证书导致的错误。这是因为大多数桌面浏览器缓存随着时间的推移信任中间证书机构。一旦浏览器访问并且从一个网站了解到的一个中间证书机构，下一次它将不需要中间证书机构包含证书链。

一些站点故意这样做为的是让二级服务器用来提供资源服务。比如，他们可能会让他们的主HTML页面用一台拥有全部证书链的服务器来提供，但是像图片，CSS,或者JavaScript等这样的资源用不包含CA的服务器来提供，以此节省带宽。不幸的是，有时这些服务器可能会提供一个在app中呼叫的web服务。这里有两种解决这些问题的方法：

- 配置服务器使它包含服务器链中的中间证书颁发机构
- 或者，像对待不知名的CA一样对待中间CA，并且创建一个TrustManager来直接信任它，就像在前两节中做的那样。

## 验证主机名常见问题

就像在文章开头提到的那样，有两个关键的部分来确认SSL的连接。第一个是确认证书来源于信任的源，这也是前一个部分关注的焦点。这一部分关注第二部分：确保你当前对话的服务器有正确的证书。当情况不是这样时，你可能会看到这样的典型错误：

```
java.io.IOException: Hostname 'example.com' was not verified
    at libcore.net.http.HttpConnection.verifySecureSocketHostn
    at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.con
    at libcore.net.http.HttpEngine.sendSocketRequest (HttpEngin
    at libcore.net.http.HttpEngine.sendRequest (HttpEngine.java
    at libcore.net.http.HttpURLConnectionImpl.getResponse (Http
    at libcore.net.http.HttpURLConnectionImpl.getInputStream (H
    at libcore.net.http.HttpsURLConnectionImpl.getInputStream (
```

服务器配置错误可能会导致这种情况发生。服务器配置了一个证书，这个证书没有匹配的你想连接的服务器的subject或者命名空间中二选一的subject。一个证书被许多不同的服务器使用是可能的。比如，使用 [openssl s\\_client -connect google.com:443 |openssl x509 -text](#) 查看google证书，你可以看到一个subject支持 `google.com`、`youtube.com`、`*.android.com` 或者其他的。这种错误只会发生在你在连接的服务器名称没有被证书列为可接受。

不幸的是另外一种原因也会导致这种情况发生：[虚拟化服务](#)。当用HTTP同拥有一个以上主机名的服务器共享时，web服务器可以从 HTTP/1.1 请求分别出客户端需要的目标主机名。不行的是，使用HTTPS会使情况变得复杂，因为服务器必须知道在发现HTTP请求前返回哪一个证书。为了解决这个问题，新版本的SSL，特别是TLSV1.0和之后的版本，支持[服务器名指示\(SNI\)](#)，允许SSL客户端为服务端指定目标主机名，从而返回正确的证书。幸运的是，从安卓2.3开始，[HttpsURLConnection](#) 支持SNI。不幸的是，Apache HTTP客户端不这样，这也是我们不鼓励用它的原因之一。如果你需要支持安卓2.2或者更老的版本或者Apache HTTP客户端的一个解决方法是，建立一个可选的虚拟化服务并且使用特别的端口，这样服务端就能够清楚该返回哪一个证书。

用不使用你的虚拟服务的主机名更换[HostnameVerifier](#)而不是用服务器默认的来替换，是很重要的选择。

注意：替换[HostnameVerifier](#)可能会非常危险，如果另外一个虚拟服务不在你的控制下，因为中间人攻击可能会直接使流量到达另外一台服务器而超出你的考虑。如果你仍然确定你想要覆盖主机名验证，这里有一个为了单[URLConnection](#)替换验证过程的例子

```
// Create an HostnameVerifier that hardwires the expected hostname
// Note that is different than the URL's hostname:
// example.com versus example.org
HostnameVerifier hostnameVerifier = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        HostnameVerifier hv =
            HttpsURLConnection.getDefaultHostnameVerifier();
        return hv.verify("example.com", session);
    }
};

// Tell the URLConnection to use our HostnameVerifier
```

```
URL url = new URL("https://example.org/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection)url.openConnection();
urlConnection.setHostnameVerifier(hostnameVerifier);
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

但是请记住，如果你发现你在替换主机名验证，特别是虚拟服务，另外一个虚拟主机不在你的控制的情况是非常危险的，你应该找到一个避免这种问题产生的托管管理。

## 关于直接使用SSL Socket的警告

到目前为止，这些例子聚焦于使用HttpsURLConnection上。有时一些app需要让SSL和HTTP分开。举个例子，一个email应用可能会使用SSL的变种，SMTP,POP3,IMAP等。在那些例子中，应用程序会想使用[SSL Socket](#)直接连接，与HttpsURLConnection做的方法相似。这种技术到目前为止处理了证书验证问题，也应用于SSLSocket中。事实上，当使用常规的TrustManager时，传递给HttpsURLConnection的是SSLSocketFactory。如果你需要一个带常规的SSLSocket的TrustManager，跟随下面的步骤使用SSLSocketFactory来创建你的SSLSocket。注意：SSLSocket不具有主机名验证功能。它取决于它自己的主机名验证，通过传入预期的主机名调用[getDefauleHostNameVerifier\(\)](#)。进一步需要注意的是，当发生错误时，[HostnameVerifier.verify\(\)](#)不知道抛出异常，而是返回一个布尔值，你需要进一步明确的检查。下面是一个演示的方法。这个例子演示了当它连接gmail.com 443端口并且没有SNI支持的时候，你将会接收到一个mail.google.com的证书。你需要确保证书的确是mail.google.com的。

```
// Open SSLSocket directly to gmail.com
SocketFactory sf = SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) sf.createSocket("gmail.com", 443);
HostnameVerifier hv = HttpsURLConnection.getDefaultHostnameVerifier();
SSLSession s = socket.getSession();

// Verify that the certificate hostname is for mail.google.com
// This is due to lack of SNI support in the current SSLSocket.
if (!hv.verify("mail.google.com", s)) {
    throw new SSLHandshakeException("Expected mail.google.com, "
                                    "found " + s.getPeerPrincipal()
}

// At this point SSLSocket performed certificate verificaiton and
// we have performed hostname verification, so it is safe to proce

// ... use socket ...
socket.close();
```

## 黑名单

SSL 主要依靠CA来确认证书来自正确无误服务器和域名的所有者。在较少的情况下，CA被欺骗，或者在[Comodo](#)和[DigiNotar](#)的例子中，一个主机名的证书被颁发给了除了服务器和域名的拥有者之外的人，导致被破坏。

为了减少这着危险，安卓可以将一些黑名单或者整个CA列入黑名单。尽管名单是以前是嵌入操作系统的，从安卓4.2开始，这个名单在以后的方案中可以远程更新。

## 阻塞

一个app可以通过阻塞技术保护它自己免于受虚假证书的欺骗。这是简单运用使用未知的CA的例子，限制app信任的CA的仅来自被app使用的服务器。阻止了来自系统中另外一百多个CA的欺骗而导致的app安全通道的破坏。

## 客户端验证

这篇文章聚焦在SSL的使用者安全的同服务器对话上。SSL也支持服务端通过验证客户端的证书来确认客户端的身份。文外，这种技术也与TrustManager的特性相似。可以参考在[HttpsURLConnection](#)文档中关于创建一个常规的[KeyManager](#)的讨论。

编写:[craftsmanBai](http://craftsmanBai.com) - <http://z1ng.net> - 校正:

原文:<http://developer.android.com/training/enterprise/index.html>

# 为企业开发

在这堂课中，你将会学到API和为企业开发时应用程序的技术。

## Lessons

- [使用设备管理策略加强安全](#)

在这堂课中，你将学会如何去创建一个通过使用设备管理策略管理内容访问权限的具有安全意识的应用程序。

编写:[craftsmanBai](http://craftsmanBai.com) - <http://z1ng.net> - 校正:

原文: <http://developer.android.com/training/enterprise/device-management-policy.html>

在Android 2.2(API Level 8)之后，Android平台通过设备管理API提供了系统级的设备管理能力。

在这一课中，你将学到如何通过使用设备管理策略创建一个对安全敏感的应用程序。比如某应用可被配置为：在给用户显示受保护的内容之前，确保已设置一个足够强度的锁屏密码。

## 定义声明你的策略

首先，你需要定义多种在功能层面提供支持的策略。这些策略可以包括屏幕锁密码强度、密码过期时间以及加密等等方面。

你须在res/xml/device\_admin.xml中声明选择的策略集，它将被应用强制实行。在Android manifest也需要引用声明的策略集。每个声明的策略对应[DevicePolicyManager](#)中一些相关设备的策略方法（例如定义最小密码长度或最少大写字母字符数）。如果一个应用尝试调用在XML中没有对应策略的方法，程序在运行时便会抛出一个[SecurityException](#)异常。

如果应用程序试图管理其他策略，那么强制锁force-lock之类的其他权限就会发挥作用。正如你将看到的，作为设备管理权限激活过程的一部分，声明策略的列表会在系统屏幕上显示给用户。如下代码片段在res/xml/device\_admin.xml中声明了密码限制策略：

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-policies>
        <limit-password />
    </uses-policies>
</device-admin>
```

在Android manifest引用XML策略声明：

```
<receiver android:name=".Policy$PolicyAdmin"
    android:permission="android.permission.BIND_DEVICE_ADMIN">
    <meta-data android:name="android.app.device_admin"
        android:resource="@xml/device_admin" />
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLE_REQUEST" />
    </intent-filter>
</receiver>
```

## 创建一个设备管理接受端

创建一个设备管理广播接收端（broadcast receiver），可以接收到与你声明的策略有关的事件通知。也可以对应用程序有选择地重写回调函数。

在同样的应用程序（Device Admin）中，当设备管理（device administrator）权限被用户设为禁用时，已配置好的策略就会从共享偏好设置（shared preference）中擦除。

你应该考虑实现与你的应用业务逻辑相关的策略。例如，你的应用可以采取一些措施来降低安全风险，如：删除设备上的敏感数据，禁用远程同步，对管理员的通知提醒等等。

为了让广播接收端能够正常工作，请务必在Android manifest中注册上面代码片段所示内容。

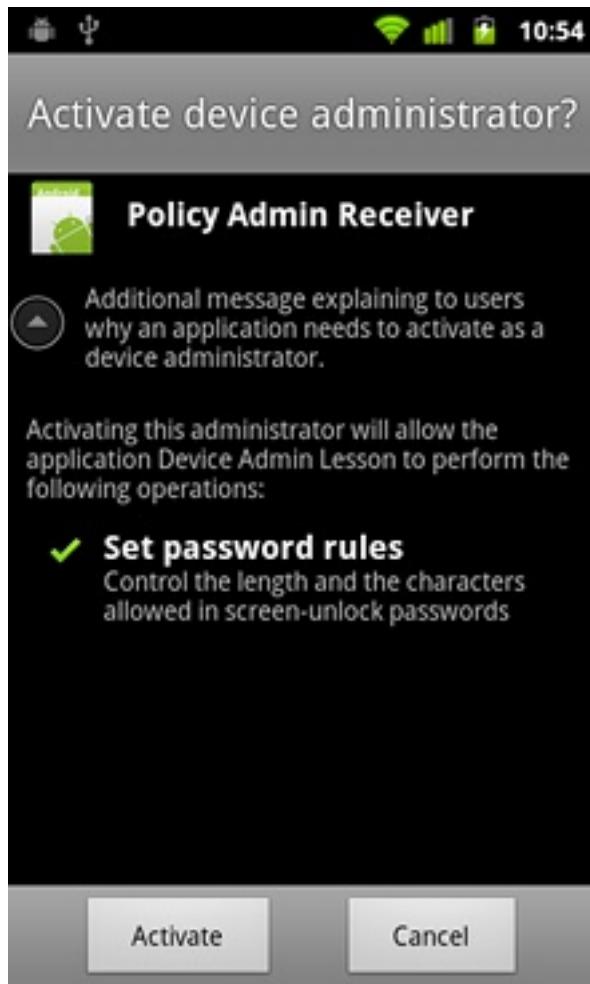
```
<receiver android:name=".Policy$PolicyAdmin">
    android:permission="android.permission.BIND_DEVICE_ADMIN">
        <meta-data android:name="android.app.device_admin"
            android:resource="@xml/device_admin" />
        <intent-filter>
            <action android:name="android.app.action.DEVICE_ADMIN_ENABLE_REQUEST" />
        </intent-filter>
</receiver>
```

## 激活设备管理器

在执行任何策略之前，用户需要手动将程序激活为设备管理权限，下面的程序片段显示了如何触发设置框以便让用户为你的程序激活权限。

通过指定[EXTRA\\_ADD\\_EXPLANATION](#)来给出明显的说明信息以告知用户应用程序为何申请激活设备管理权限是个很好的做法。

```
if (!mPolicy.isAdminActive()) {  
  
    Intent activateDeviceAdminIntent =  
        new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);  
  
    activateDeviceAdminIntent.putExtra(  
        DevicePolicyManager.EXTRA_DEVICE_ADMIN,  
        mPolicy.getPolicyAdmin());  
  
    // It is good practice to include the optional explanation tex  
    // explain to user why the application is requesting to be a d  
    // administrator. The system will display this message on the  
    // screen.  
    activateDeviceAdminIntent.putExtra(  
        DevicePolicyManager.EXTRA_ADD_EXPLANATION,  
        getResources().getString(R.string.device_admin_activation_  
  
    startActivityForResult(activateDeviceAdminIntent,  
        REQ_ACTIVATE_DEVICE_ADMIN);  
}
```



如果用户选择"Activate"，程序就会获取设备管理员权限并可以开始配置与执行策略。当然，程序也需要做好处理用户选择放弃激活的准备，比如用户点击了“取消”按钮，返回键或者HOME键的情况。因此，如果有必要的话，策略设置中的[onResume\(\)](#)方法需要加入重新评估的逻辑判断代码，以便将设备管理激活选项展示给用户。

## 实施设备策略控制

在设备管理权限成功激活后，程序就会根据请求的策略来配置设备策略管理器。要牢记，新策略会被添加到每个版本的Android中。所以你需要在程序中做好平台版本的检测，以便新策略能被老版本平台很好的支持。

例如，“密码最少大写字符数”这个安全策略只有在高于API Level 11（Honeycomb）的平台才被支持，以下代码则演示了如何在运行时检查版本：

```
DevicePolicyManager mDPM = (DevicePolicyManager)
    context.getSystemService(Context.DEVICE_POLICY_SERVICE);
ComponentName mPolicyAdmin = new ComponentName(context, PolicyAdmi
...
mDPM.setPasswordQuality(mPolicyAdmin, PASSWORD_QUALITY_VALUES[mPas
mDPM.setPasswordMinimumLength(mPolicyAdmin, mPasswordLength);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    mDPM.setPasswordMinimumUpperCase(mPolicyAdmin, mPasswordMinUpp
}
```

这样程序就可以执行策略了。当程序无法访问使用的正确的锁屏密码的时候，通过设备策略管理器（Device Policy Manager）API可以判断当前密码是否适用于请求的策略。如果当前锁屏密码满足策略，设备管理API不会采取纠正措施。明确地启动设置程序中的系统密码更改界面是应用程序的责任。例如：

```
if (!mDPM.isActivePasswordSufficient()) {
    ...
    // Triggers password change screen in Settings.
    Intent intent =
        new Intent(DevicePolicyManager.ACTION_SET_NEW_PASSWORD);
    startActivity(intent);
}
```

一般来说，用户可以从可用的锁屏机制中任选一个，例如“无”、“图案”、“PIN码”（数字）或密码（字母数字）。当一个密码策略配置好后，那些比已定义密码策略弱的密码会被禁用。比如，如果配置了密码级别为“数字”，那么用户只可以选择PIN码（数字）或者密码（字母数字）。一旦设备通过设置适当的锁屏密码并保护好后，应用程序便被允许访问受保护的内容。

```
if (!mDPM.isAdminActive(..)) {
    // Activates device administrator.
    ...
} else if (!mDPM.isActivePasswordSufficient()) {
    // Launches password set-up screen in Settings.
    ...
} else {
    // Grants access to secure content.
    ...
    startActivityForResult(new Intent(context, SecureActivity.class));
}
```

编写:[kesenhoo](#) - 校对:

原文:<http://developer.android.com/training/testing.html>

# 测试程序

These classes and articles provide information about how to test your Android application.

## [Testing Your Activity](#)

How to test Activities in your Android applications.

编写:[huanglizhuo](#), 校对:

原文:<http://developer.android.com/training/activity-testing/index.html>

# 测试你的Activity

你应该把编写和运行测试作为你Android应用开发周期的一部分.编写好的测试可以帮助你在开发过程中尽早发现漏洞,并让你对自己的代码更有信心.

测试用例定义了一系列对象和方法从而独立进行多个测试.测试用例可以编写成测试组并按计划的运行,由测试框架组织成一个可以重复运行的测试者

这节内容将会教你如何使用Android基于最流行的JUnit框架来自定义测试框架.你可以编写测试用例来测试你应用程序的特定行为,并在不同的Android设备上检测一致性.你的测试用例也可以通过描述应用组件的预期行为来作为内部代码注释文档.

# Lessons

- [建立测试环境](#)

Learn how to create your test project.

- [创建与执行测试用例](#)

Learn how to write test cases to verify the expected properties of your Activity, and run the test cases with the Instrumentation test runner provided by the Android framework.

- [测试UI组件](#)

Learn how to test the behavior of specific UI components in your Activity.

- [创建单元测试](#)

Learn how to perform unit testing to verify the behavior of an Activity in isolation.

- [创建功能测试](#)

Learn how to perform functional testing to verify the interaction of multiple Activities.

编写:[huanglizhuo](#)

校对:

# 建立测试环境

再开始编写,运行你的测试之前,你应该建立你的测试开发环境。本小节将会教你怎样建立Eclipse IDE来构建和运行你的测试,以及怎样用Gradle框架在命令行下构建和运行你的测试。

注意 :为了帮助你开始,这小节是基于Eclipse及ADT插件。然而,你在自己测试开发时可以自由选用IDE或命令行。

## 用Eclipse建立测试

安装了Android Developer Tools (ADT) 插件的Eclipse将为你创建,构建,以及运行Android程序提供一个基于图形界面的集成开发环境。Eclipse的一项方便的特性是可以自动为你的Android应用项目创建一个对应的测试项目。

开始在Eclipse中创建测试环境:

1. 要是还没安装Eclipse [ADT](#)插件,请先下载安装。
2. 导入或创建你想再次测试的Android应用项目。
3. 生成一个对应于应用程序项目测试的测试项目。为你的导入项目生成一个测试项目:

a.在项目浏览器里,右击你的应用项目,然后选择**Android Tools > New Test Project**

b.在新建Android测试项目面板,为你的测试项目设置合适的参数,然后点击**Finish**

你现在应该可以在Eclipse环境中创建, 构建和运行测试项目了。想要继续学习如何在Eclipse中进行这些任务,请转到[Creating and Running a Test Case](#)

## 用命令行建立测试

如果正在使用Gradle version 1.6或者更高的版本作为构建环境,可以用Gradle Wrapper创建。构建和运行Android应用测试。确保你的gradle.build文件中默认[minSdkVersion](#)属性是8或更高。可以参考包含在这下载包中的示例文件gradle.build

用Gradle Wrapper运行你的测试:

1. 连接你的Android真机或开启你的Android模拟器。
2. 在你的项目目录运行如下命令:

```
./gradlew build connectedCheck
```

进一步学习Gradle关于Android测试的内容,参看[Gradle Plugin User Guide](#)。

进一步学习使用Gradle及其它命令行工具,参看[Testing from Other IDEs.](#)。

本节示例代码[AndroidTestingFun.zip](#)

编写:[huanglizhuo](#)

校对:

# 创建与执行测试用例

为了验证布局设计和功能行为在你的应用程序有没有发生变化(不符合预期), 为应用的每个Activity建立测试是很重要。对于每一个测试, 你需要在测试用例中创建一个个独立的部分, 包括固定测试, 前提测试方法和[Activity](#)的测试方法。然后你就可以运行测试并得到测试报告。如果任何测试方法失败, 这表明在你的代码中有可能潜在的缺陷。

## 注意

在测试驱动开发 (TDD) 方法中, 你应该编写足够的有效代码, 以满足你的测试依赖关系, 更新你的测试案例并以反映新的功能需求, 并反复重复这样的, 你而不是在你写大部分或全部前期代码后再开始测试周期。

## 创建一个测试用例

[Activity](#)都是通过结构化的方式编写的。请务必把你的测试代码放在一个单独的包内,从而与其它的正在测试的代码分开。

按照惯例, 你的测试包的名称应该遵循与应用包名相同的命名方式, 在应用包名后接“.tests”。在你创建的测试包, 为你的测试用例添加Java类。按照惯例, 你的测试用例名称也应遵循你要测试的Java或Android的类相同的名称, 但后缀为“Test”。

要在Eclipse中创建一个新的测试用例:

- a. 在Package Explorer中,右键点击你要测试工程的src/文件夹**New > Package**。
- b. 设置文件夹名称(比如,com.example.android.testingfun.tests)并点击**Finish**。
- c. 右键点击你创的测试包,并选择**New>Class**。
- d. 设置文件名称(比如,MyFirstTestActivityTest),然后点击\*\**Finish*\*\*。

# 建立你的测试夹具(Fixture)

(夹具是用来快速,安全的测试组件功能的工具)

测试夹具由对象必须由一个或多个正在运行测试来初始化。要建立测试夹具,你可以在你的测试中重写[setUp\(\)](#)和[tearDown\(\)](#)方法。测试会在运行任何其它测试方法之前自动[setUp\(\)](#)方法。你可以用这些方法来保持代码的测试初始化和清理是分开。

在你的Eclipse中建立夹具:

1. 在 Package Explorer中双击测试打开之前编写的测试用例,然后修改你的测试用例使它扩展[ActivityTestCase](#)的子类。比如这样:

```
public class MyFirstTestActivityTest  
    extends ActivityInstrumentationTestCase2<MyFirstTestActivi
```

1. 下一步,给你的测试用例添加构造函数和[setUp\(\)](#)方法,并你想测试的Activity添加变量声明。比如:

```
public class MyFirstTestActivityTest  
    extends ActivityInstrumentationTestCase2<MyFirstTestActivi  
  
    private MyFirstTestActivity mFirstTestActivity;  
    private TextView mFirstTestText;  
  
    public MyFirstTestActivityTest() {  
        super(MyFirstTestActivity.class);  
    }  
  
    @Override  
    protected void setUp() throws Exception {  
        super.setUp();  
        mFirstTestActivity = getActivity();  
        mFirstTestText =  
            (TextView) mFirstTestActivity  
                .findViewById(R.id.my_first_test_text_view);  
    }  
}
```

构造函数是由测试运行者初始化测试类反射的,而[setUp\(\)](#)方法是由测试运行者运行其它测试类之前反射的。

通常在[setUp\(\)](#)方法中你应该这样:

- \*. 为[setUp\(\)](#)反射父类构造器,这是JUnit所必须的。
- \*. 通过下面这样初始化测试夹具的状态:

```
定义实例变量夹具的状态存储。
```

为正在测试的 [Activity] (<http://developer.android.com/reference/android/app/Activity.html>) 提供一个引用。

在你想测试的 [Activity] (<http://developer.android.com/reference/android/app/Activity.html>) 提供一个引用。

你可以使用 [getActivity\(\)](#) 方法得到正在测试的 [Activity](#) 的引用。

## 增加一个测试前提

作为一个全面的检查,确认测试夹具的设置是正确的是很好的做法,那样你想要测试的对象就会正确地实例化和初始化。这样,你的测试就不会因为有你的测试夹具的设置错误而失败。按照惯例,验证你的测试夹具的方法被称为`testPreconditions()`。

例如,你可能想添加一个像这样的`testPreconditions()`方法:

```
public void testPreconditions() {  
    assertNotNull("mFirstTestActivity is null", mFirstTestActivity);  
    assertNotNull("mFirstTestText is null", mFirstTestText);  
}
```

断言方法是从Junit[Assert](#)类来的。通常,你可以使用断言验证你想测试一个特定的条件是否是真的。

\*. 如果条件为假,断言方法抛出一个`assertionFailedError`异常,这是典型的测试者报告。你可以在你的断言失败时给你的断言方法添加一个字符串作为第一个参数从而给出一些上下文详细信息。

\*. 如果条件为真,测试通过。

在这两种情况下,测试者继续运行其它测试用例的测试方法。

## 添加一个测试方法验证你的Activity

下一步，添加一个或多个测试方法来验证你的[Activity](#)布局和功能性行为。

例如，如果你的活动含有一个[TextView](#)，你可以添加一个像这样的方法来检查它是否有正确的标签文本：

```
public void testMyFirstTestTextView_labelText() {  
    final String expected =  
        mFirstTestActivity.getString(R.string.my_first_test);  
    final String actual = mFirstTestText.getText().toString();  
    assertEquals(expected, actual);  
}
```

该 `testMyFirstTestTextView_labelText()` 方法只是简单的检查程序是[TextView](#)的默认文本是否是由 `strings.xml` 资源中定义的预期文本设定的。

### 注意

当命名测试方法，你可以使用下划线将被测试的内容从正在测试的具体用例中分离出来。这种风格使它更容易看清楚的那部分正在被测试。

做这种类型的字符串比较时，从你的资源文件中读取预期字符串是良好的做法，而不是在你代码中硬性编写字符串做比较。这可以防止当字符串定义在资源文件被修改时轻易的打断你的测试。

为进行比较，预期的和实际的字符串都要做为[assertEquals\(\)](#)方法的参数。如果值是不一样的，断言将抛出一个[AssertionFailedError](#)异常。

如果你添加了一个 `testPreconditions()` 方法，把你的测试方法放在 `testPreconditions` 之后。

要参看一个完整的测试案例，在参看本节示例中的 `MyFirstTestActivityTest.java`。

## 构建和运行你的测试

你可以在Eclipse中的包浏览器(Package Explorer)中运行你的测试。

这样构建和运行你的测试:

1. 连接一个Android设备到你的机器。在设备或模拟器，打开设置菜单，选择开发者选项并确保启用USB调试。
2. 在包浏览器(Package Explorer)中，右键单击测试类，并选择**Run As > Android Junit Test**。
3. 在Android设备选择对话框，选择刚才连接的设备，然后单击“确定”。
4. 在JUnit视图，验证测试是否通过,有无错误或失败。

本节示例代码[AndroidTestingFun.zip](#)

编写:[huanglizhuo](#)

校对:

# 测试UI组件

通常情况下，你的[Activity](#)，包括用户界面组件（如按钮，复选框，可编辑的文本域，和选框）允许你的用户与Android应用程序交互。本节介绍如何用一个简单的按钮的界面交互测试。你可以使用相同的步骤来测试其他的，更复杂的UI组件。

## 注意

这一节叫做白盒测试，因为你拥有要测试应用程序的源码。Android仪表框架适用于创建应用程序中UI部件的白盒测试。用户界面测试的另一种类型是黑盒测试就是那种你可能无法得知应用程序源代码的类型。这种类型的测试可以用来测试你的应用程序如何与其他应用程序，或与系统进行交互。黑盒测试是不包括在本节中的。了解更多关于如何在你的Android应用程序进行黑盒测试，请参看[UI Testing guide](#)。

要参看一个完整的测试案例，可以在本节示例代码中的clickfunactivitytest.java查看。

## 建立你的测试夹具(Fixture)

(夹具是用来快速,安全的测试组件功能的工具)

当为UI测试建立夹具时,你应该在[setUp\(\)](#)方法中指定[touch mode](#)。把touch mode设置为真可以防止UI组件抢夺你编程指定的点击方法的焦点事件(比如,一个按钮会撤销它的点击监听器)。确定你在调用[getActivity\(\)](#)方法前调用了[setActivityInitialTouchMode](#))。

比如:

```
public class ClickFunActivityTest
    extends ActivityInstrumentationTestCase2 {
    ...
    @Override
    protected void setUp() throws Exception {
        super.setUp();

        setActivityInitialTouchMode(true);

        mClickFunActivity = getActivity();
        mClickMeButton = (Button)
            mClickFunActivity
                .findViewById(R.id.launch_next_activity_button);
        mInfoTextView = (TextView)
            mClickFunActivity.findViewById(R.id.info_text_view
    }
}
```

## 添加测试方法确认UI响应表现

你的UI测试目标应包括:

- \*. 检验Activity启动时Button是按照正确布局显示的。
- \*. 检验TextView初始化时是隐藏的。
- \*. 检验TextView在Button点击时显示预期的字符串

接下来的部分会演示怎样实现上述验证方法

## 验证Button布局参数

你应该像这样添加的测试方法验证在你的[Activity](#)中按钮是否正确显示:

```
@MediumTest
public void testClickMeButton_layout() {
    final View decorView = mClickFunActivity.getWindow().getDecorView();

    ViewAsserts.assertOnScreen(decorView, mClickMeButton);

    final ViewGroup.LayoutParams layoutParams =
        mClickMeButton.getLayoutParams();
    assertNotNull(layoutParams);
    assertEquals(layoutParams.width, WindowManager.LayoutParams.MATCH_PARENT);
    assertEquals(layoutParams.height, WindowManager.LayoutParams.WRAP_CONTENT);
}
```

在调用[assertOnScreen\(\)](#)方法时,你传递根视图以及你期望呈现在屏幕上的视图作为参数。如果你想呈现的视图没有在根视图中,该方法会抛出一个[AssertionFailedError](#)异常,除非测试通过。

你也可以通过获取一个[ViewGroup.LayoutParams](#)对象的引用验证[Button](#)布局是否正确,然后调用声明方法验证[Button](#)对象的宽高属性值是否与预期值一致。

`@MediumTest`注释指定测试在于它刚生成时是如何归类的。要了解更多有关使用测试的注释,见本节示例。

## 验证TextView的布局参数

你应该像这样添加一个测试方法来验证一个[TextView](#)最初是隐藏在你的[Activity](#)的:

```
@MediumTest  
public void testInfoTextView_layout() {  
    final View decorView = mClickFunActivity.getWindow().getDecorView();  
    ViewAsserts.assertOnScreen(decorView, mInfoTextView);  
    assertTrue(View.GONE == mInfoTextView.getVisibility());  
}
```

你可以调用[getdecorview\(\)](#)方法得到一个[Activity](#)中要修饰的View的引用。要修饰的View在布局层次视图中是最上层的ViewGroup([FrameLayout](#))

## 验证按钮的行为

你可以使用一个这样的测试方法来验证当按下按钮时[TextView](#)变得可见：

```
@MediumTest
public void testClickMeButton_clickButtonAndExpectInfoText() {
    String expectedInfoText = mClickFunActivity.getString(R.string
        TouchUtils.clickView(this, mClickMeButton);
    assertTrue(View.VISIBLE == mInfoTextView.getVisibility());
    assertEquals(expectedInfoText, mInfoTextView.getText());
}
```

在你的的测试中调用[clickview\(\)](#)以编程方式点击一个按钮。你必须给正在运行的测试用例传递一个引用和要操作按钮的引用。

注意:[Touchutils](#)助手类提供与应用程序交互的方法可以方便进行模拟触摸操作。你可以使用这些方法来模拟点击，轻敲，或应用程序屏幕拖动View。

警告[Touchutils](#)方法的目的是将事件安全地从测试线程发送到UI线程。你不应该用[Touchutils](#)直接在UI线程或任何标注@UiThread的测试方法这样做可能会增加错误线程异常。

## 应用测试的注释

### [@SmallTest](#)

标志着一个测试运行的小型测试的一部分。

### [@MediumTest](#)

标志着一个测试运行的中等测试的一部分。

### [@LargeTest](#)

标志着一个测试运行的大型测试的一部分。

通常情况下，只需要几毫秒的时间的应该被标记为[@SmallTest](#),长时间运行的测试（100毫秒或更多）通常被标记为[@MediumTest](#),[@LargeTest](#),主要取决于测试访问资源在网络上或在本地系统。可以参看[Android Tools Protip](#)更好的指导你使用测试注释

你可以创建其它的测试注释来控制测试的组织和运行。要了解更多关于其他注释的信息，见[Annotation](#)类参考。

本节示例代码[AndroidTestingFun.zip](#)

编写:[huanglizhuo](#)

校对:

# 创建单元测试

一个[Activity](#)测试单元是快速验证一个[Activity](#)的状态以及与其它独立组件(也就是和系统其它部分分离的部分)交互的最优方式。一个测试单元通常是测试代码中可能性最小的代码块(可以是一个方法,类,或者组件),而且也不依赖于系统或网络资源。比如说,你可以写一个测试单元去检查Activity是否有正确的布局或者它的触发器,以及Intent对象的正确性。

测试单元一般不适合测试与系统有复杂交互事件的UI。相反,你应该使用像 `ActivityInstrumentationTestCase2` 的类作为测试UI组建的描述。

这节内容将会教你编写一个测试单元来验证一个Intent是否正确触发了另一个[Activity](#)。由于测试是与环境独立的,所以[Intent](#)被发送给Android系统的,但你可以检查Intent对象的有效数据的正确性。对于一个完备的测试案例,可以参考一下示例代码中的 `LaunchActivityTest.java`。

注意要测试对系统或外部的依赖,你可以使用来自Mocking Framework的Mock类并把它插入你的测试单元。要了解更多关于Android提供的Mocking Framework内容请参看[Mock Object Classes](#)。

## 编写一个Android单元测试例子

ActiviUnitTestCase类提供了单个[Activity](#)测试的支持。要创建测试单元,你的测试类应该继承自ActiviUnitTestCase。继承ActiviUnitTestCase的Activity不会被Android自动启动的。要单独启动Activity,你需要显式的调用startActivity()方法,并传递一个[Intent](#)来启动你的目标[Activity](#)。

For example:

```
public class LaunchActivityTest
    extends ActivityUnitTestCase<LaunchActivity> {
    ...
    @Override
    protected void setUp() throws Exception {
        super.setUp();
        mLaunchIntent = new Intent(getInstrumentation()
            .getTargetContext(), LaunchActivity.class);
        startActivity(mLaunchIntent, null, null);
        final Button launchNextButton =
            (Button) getActivity()
            .findViewById(R.id.launch_next_activity_button);
    }
}
```

## 验证另一个Activity的启动

你的单元测试目标可能包括:

- 验证当Button被按下是启动的LaunchActivity是否正确。
- 验证启动的Intent是否包含有效的数据。

为验证一个[Intent](#)Button触发的事件,你可以使用[getStartedActivityIntent\(\)](#)方法。通过使用断言方法,你可以验证返回的[Intent](#)是否为空,以及是否包含了预期的数据来启动下一个Activity。如果你俩个断言值都是真,那你就成功的验证了你Activity发送的Intent的正确性了。

你应该这样实现你的方法:

```
@MediumTest
public void testNextActivityWasLaunchedWithIntent() {
    startActivity(mLaunchIntent, null, null);
    final Button launchNextButton =
        (Button) getActivity()
            .findViewById(R.id.launch_next_activity_button);
    launchNextButton.performClick();

    final Intent launchIntent = getStartedActivityIntent();
    assertNotNull("Intent was null", launchIntent);
    assertTrue(isFinishCalled());

    final String payload =
        launchIntent.getStringExtra(NextActivity.EXTRA_PAYLOAD);
    assertEquals("Payload is empty", LaunchActivity.STRING_PAYLOAD
}
```

因为LaunchActivity是独立运行的,所以不可以使用[TouchUtils](#)库来操作UI。要正确处理[Button](#)点击,你可以调用[perfoemClick\(\)](#)方法

本节示例代码[AndroidTestingFun.zip](#)

编写:[huanglizhuo](#)

校对:

# 创建功能测试

功能测试包括验证单个应用组件是否与使用者期望的那样(与其它组件)协同工作。比如,你可以创建一个功能测试验证在用户执行UI交互时[Activity](#)是否正确启动目标[Activity](#)。

要为你的[Activity](#)创建功能测试,你的测试类应该扩展[ActivityInstrumentationTestCase2](#)。  
与[ActivityUnitTestCase](#)不同的是在[ActivityInstrumentationTestCase2](#)中可以与Android系统通信  
以及发送键盘输入和点击事件到UI。

要了解一个完整的测试例子,看一下示例应用中的SenderActivityTest.java。

## 添加测试方法验证函数的行为

你的函数测试目标应该包括:

- 验证UI控制是否正确启动了目标Activity。
- 验证目标Activity的表现是否按照发送Activity提供的数据呈现。

你应该这样实现你的方法:

```
@MediumTest
public void testSendMessageToReceiverActivity() {
    final Button sendToReceiverButton = (Button)
        mSenderIdActivity.findViewById(R.id.send_message_button)

    final EditText senderMessageEditText = (EditText)
        mSenderIdActivity.findViewById(R.id.message_input_edit_t

    // Set up an ActivityMonitor
    ...

    // Send string input value
    ...

    // Validate that ReceiverActivity is started
    ...

    // Validate that ReceiverActivity has the correct data
    ...

    // Remove the ActivityMonitor
    ...
}
```

测试等待与屏幕匹配的Activity,否则返回会在超时后返回null。如果ReceiverActivity启动了,那么你先前设立的[ActivityMoniter](#)就会收到一次撞击。你可以使用断言方法验证ReceiverActivity是否的确启动了,并且[ActivityMoniter](#)记录撞击次数会按照预想的那样增加。

# 设立一个ActivityMonitor

为了再你的应用中监视单个[Activity](#),你可以注册一个[ActivityMoniter](#)。[ActivityMoniter](#)是由系统在每当一个Activity与你的要求符合是开启的。如果发现匹配,监视器的撞击计数就会更新。

通常来说要使用[ActivityMoniter](#),你应该这样:

1. 使用[getInstrumentation\(\)](#)方法为你的测试用例实现[Instrumentation](#)。
2. 为当前使用[Instrumentation](#)[addMonitor\(\)](#)方法的instrumentation添加一个[Instrumentation.ActivityMonitor](#)实例。匹配规则可以是通过[IntentFilter](#)或者一个类名。
3. 等待开启一个Activity。
4. 验证监视器撞击次数的增加。
5. 移除监视器。

下面是一个例子:

```
// Set up an ActivityMonitor
ActivityMonitor receiverActivityMonitor =
    getInstrumentation().addMonitor(ReceiverActivity.class.get
        null, false);

// Validate that ReceiverActivity is started
TouchUtils.clickView(this, sendToReceiverButton);
ReceiverActivity receiverActivity = (ReceiverActivity)
    receiverActivityMonitor.waitForActivityWithTimeout(TIMEOUT);
assertNotNull("ReceiverActivity is null", receiverActivity);
assertEquals("Monitor for ReceiverActivity has not been called",
    1, receiverActivityMonitor.getHits());
assertEquals("Activity is of wrong type",
    ReceiverActivity.class, receiverActivity.getClass());

// Remove the ActivityMonitor
getInstrumentation().removeMonitor(receiverActivityMonitor);
```

## 使用Instrumentation发送一个键盘输入

如果你的Activity有一个EditText,你想要测试用户是否可以给EditText对象输入数值。

通常在ActivityInstrumentationTestCase2中给EditText对象发送串字符,你应该这样作:

1. 使用runOnMainSync()方法在一个循环中同步调用requestFocus()。这样,你的UI线程就会在获得焦点前一直被阻塞。
2. 调用waitForIdleSync()方法等待主线程空闲(也就是说,没有更多的运行事件)。
3. 调用sendStringSync()方法给EditText对象发送一个你输入的字符串。

想这样:

```
// Send string input value
getInstrumentation().runOnMainSync(new Runnable() {
    @Override
    public void run() {
        senderMessageEditText.requestFocus();
    }
});
getInstrumentation().waitForIdleSync();
getInstrumentation().sendStringSync("Hello Android!");
getInstrumentation().waitForIdleSync();
```

本节例子[AndroidTestingFun.zip](#)

# Table of Contents

|                   |     |
|-------------------|-----|
| 序言                | 2   |
| Android入门基础：从这里开始 | 8   |
| 建立第一个App          | 8   |
| 创建Android项目       | 12  |
| 执行Android程序       | 17  |
| 建立简单的用户界面         | 22  |
| 启动其他Activity      | 30  |
| 添加ActionBar       | 42  |
| 建立ActionBar       | 45  |
| 添加Action按钮        | 49  |
| ActionBar的风格化     | 57  |
| ActionBar的覆盖层叠    | 68  |
| 兼容不同的设备           | 72  |
| 适配不同的语言           | 75  |
| 适配不同的屏幕           | 80  |
| 适配不同的系统版本         | 85  |
| 管理Activity的生命周期   | 90  |
| 启动与销毁Activity     | 93  |
| 暂停与恢复Activity     | 100 |
| 停止与重启Activity     | 104 |
| 重新创建Activity      | 108 |
| 使用Fragment建立动态的UI | 112 |
| 创建一个Fragment      | 115 |
| 建立灵活动态的UI         | 120 |
| Fragments之间的交互    | 125 |
| 数据保存              | 125 |
| 保存到Preference     | 132 |
| 保存到文件             | 137 |
| 保存到数据库            | 146 |
| 与其他应用的交互          | 155 |
| Intent的发送         | 158 |
| 接收Activity返回的结果   | 165 |
| Intent过滤          | 169 |
| Android分享操作       | 175 |
| 分享简单的数据           | 177 |
| 给其他App发送简单的数据     | 180 |
| 接收从其他App返回的数据     | 186 |
| 给ActionBar增加分享功能  | 191 |

|                   |     |
|-------------------|-----|
| 分享文件              | 195 |
| 建立文件分享            | 198 |
| 分享文件              | 202 |
| 请求分享一个文件          | 211 |
| 获取文件信息            | 216 |
| 使用NFC分享文件         | 220 |
| 发送文件给其他设备         | 223 |
| 接收其他设备的文件         | 231 |
| Android多媒体        | 231 |
| 管理音频播放            | 240 |
| 控制你得应用的音量与播放      | 243 |
| 管理音频焦点            | 248 |
| 兼容音频输出设备          | 254 |
| 拍照                | 258 |
| 简单的拍照             | 261 |
| 简单的录像             | 271 |
| 控制相机硬件            | 276 |
| 打印                | 286 |
| 打印照片              | 289 |
| 打印HTML文档          | 292 |
| 打印自定义文档           | 297 |
| Android图像与动画      | 305 |
| 高效显示Bitmap        | 307 |
| 高效加载大图            | 310 |
| 非UI线程处理Bitmap     | 315 |
| 缓存Bitmap          | 321 |
| 管理Bitmap的内存占用     | 330 |
| 在UI上显示Bitmap      | 337 |
| 使用OpenGL ES显示图像   | 347 |
| 建立OpenGL ES的环境    | 351 |
| 定义Shapes          | 357 |
| 绘制Shapes          | 362 |
| 运用投影与相机视图         | 367 |
| 添加移动              | 372 |
| 响应触摸事件            | 376 |
| 添加动画              | 382 |
| View间渐变           | 385 |
| 使用ViewPager实现屏幕滑动 | 390 |
| 展示卡片翻转动画          | 400 |
| 缩放View            | 408 |

|                          |     |
|--------------------------|-----|
| 布局变更动画                   | 415 |
| Android网络连接与云服务          | 419 |
| 无线连接设备                   | 421 |
| 使得网络服务可发现                | 424 |
| 使用WiFi建立P2P连接            | 432 |
| 使用WiFi P2P服务             | 441 |
| 网络连接操作                   | 448 |
| 连接到网络                    | 451 |
| 管理使用的网络                  | 459 |
| 解析XML数据                  | 468 |
| 高效下载                     | 480 |
| 为网络访问更加高效而优化下载           | 483 |
| 最小化更新操作的影响               | 491 |
| 避免下载多余的数据                | 496 |
| 根据网络类型改变下载模式             | 500 |
| 使用Sync Adapter传输数据       | 505 |
| 创建Stub授权器                | 508 |
| 创建Stub Content Provider  | 515 |
| 创建Sync Adpater           | 520 |
| 执行Sync Adpater           | 533 |
| 使用Volley执行网络数据传输         | 544 |
| 发送简单的网络请求                | 547 |
| 建立请求队列                   | 553 |
| 创建标准的网络请求                | 559 |
| 实现自定义的网络请求               | 566 |
| 云同步                      | 571 |
| 使用备份API                  | 574 |
| 使用Google Cloud Messaging | 582 |
| 解决云同步的保存冲突               | 588 |
| Android联系人与位置信息          | 597 |
| Android联系人信息             | 600 |
| 获取联系人列表                  | 603 |
| 获取联系人详情                  | 618 |
| 修改联系人信息                  | 626 |
| 显示联系人头像                  | 633 |
| Android位置信息              | 646 |
| 获取当前位置                   | 649 |
| 获取位置更新                   | 659 |
| 显示位置地址                   | 672 |
| 创建并监视异常区域                | 679 |
| 识别用户的当下活动                | 705 |

|                          |     |
|--------------------------|-----|
| 使用模拟位置进行测试               | 721 |
| Android可穿戴应用             | 729 |
| 赋予Notification可穿戴特性      | 731 |
| 创建Notification           | 734 |
| 在Notification中接收语音输入     | 745 |
| 为Notification添加显示页面      | 752 |
| 以Stack的方式显示Notifications | 756 |
| 创建可穿戴的应用                 | 762 |
| 创建并执行可穿戴应用               | 765 |
| 创建自定义的布局                 | 772 |
| 添加语音能力                   | 776 |
| 打包可穿戴应用                  | 783 |
| 通过蓝牙进行调试                 | 789 |
| 发送并同步数据                  | 794 |
| 访问可穿戴数据层                 | 797 |
| 同步数据单元                   | 799 |
| 传输资源                     | 803 |
| 发送与接收消息                  | 807 |
| 处理数据层的事件                 | 811 |
| Android交互与界面设计           | 818 |
| 设计高效的导航                  | 821 |
| 规划屏幕界面与他们之间的关系           | 824 |
| 为多种大小的屏幕进行规划             | 829 |
| 提供向下和横向导航                | 835 |
| 提供向上和时间导航                | 843 |
| 综合：设计样例 App              | 848 |
| 实现高效的导航                  | 855 |
| 使用Tabs创建Swipe视图          | 858 |
| 创建抽屉导航                   | 865 |
| 提供向上的导航                  | 874 |
| 提供向后的导航                  | 880 |
| 实现向下的导航                  | 886 |
| 通知提示用户                   | 890 |
| 建立Notification           | 893 |
| 当启动Activity时保留导航         | 899 |
| 更新Notification           | 903 |
| 使用BigView风格              | 907 |
| 显示Notification进度         | 911 |
| 增加搜索功能                   | 916 |
| 建立搜索界面                   | 919 |

|                     |      |
|---------------------|------|
| 保存并搜索数据             | 924  |
| 保持向下兼容              | 931  |
| 使得你的App内容可被Google搜索 | 936  |
| 为App内容开启深度链接        | 939  |
| 为索引指定App内容          | 944  |
| 为多屏幕设计              | 949  |
| 兼容不同的屏幕大小           | 952  |
| 兼容不同的屏幕密度           | 964  |
| 实现可适应的UI            | 968  |
| 为TV进行设计             | 975  |
| 为TV优化Layout         | 978  |
| 为TV优化导航             | 981  |
| 处理不支持TV的功能          | 983  |
| 创建自定义View           | 985  |
| 创建自定义的View类         | 988  |
| 实现自定义View的绘制        | 995  |
| 使得View可交互           | 1002 |
| 优化自定义View           | 1008 |
| 创建向后兼容的UI           | 1008 |
| 抽象新的APIs            | 1014 |
| 代理至新的APIs           | 1016 |
| 使用旧的APIs实现新API的效果   | 1018 |
| 使用版本敏感的组件           | 1020 |
| 实现辅助功能              | 1022 |
| 开发辅助程序              | 1025 |
| 开发辅助服务              | 1031 |
| 管理系统UI              | 1039 |
| 淡化系统Bar             | 1042 |
| 隐藏系统Bar             | 1047 |
| 隐藏导航Bar             | 1054 |
| 全屏沉浸式应用             | 1058 |
| 响应UI可见性的变化          | 1065 |
| Android用户输入         | 1068 |
| 使用触摸手势              | 1070 |
| 检测常用的手势             | 1073 |
| 跟踪手势移动              | 1080 |
| Scroll手势动画          | 1084 |
| 处理多触摸手势             | 1092 |
| 拖拽与缩放               | 1097 |
| 管理ViewGroup中的触摸事件   | 1107 |

|                       |      |
|-----------------------|------|
| 处理键盘输入                | 1114 |
| 指定输入法类型               | 1117 |
| 处理输入法可见性              | 1123 |
| 兼容键盘导航                | 1128 |
| 处理按键动作                | 1133 |
| 兼容游戏控制器               | 1137 |
| 处理控制器输入动作             | 1139 |
| 支持不同的Android系统版本      | 1141 |
| 支持多个控制器               | 1143 |
| Android后台任务           | 1145 |
| 在IntentService中执行后台任务 | 1147 |
| 创建IntentService       | 1150 |
| 发送工作任务到IntentService  | 1154 |
| 报告后台任务执行状态            | 1156 |
| 使用CursorLoader在后台加载数据 | 1161 |
| 使用CursorLoader执行查询任务  | 1165 |
| 处理查询的结果               | 1170 |
| 管理设备的唤醒状态             | 1174 |
| 保持设备的唤醒               | 1177 |
| 制定重复定时的任务             | 1182 |
| Android性能优化           | 1190 |
| 管理应用的内存               | 1192 |
| 性能优化Tips              | 1201 |
| 提升Layout的性能           | 1214 |
| 优化layout的层级           | 1217 |
| 使用include标签重用layouts  | 1223 |
| 按需加载视图                | 1228 |
| 使得ListView滑动顺畅        | 1232 |
| 优化电池寿命                | 1236 |
| 监测电量与充电状态             | 1238 |
| 判断与监测Docking状态        | 1244 |
| 判断与监测网络连接状态           | 1249 |
| 根据需要操作Broadcast接受者    | 1254 |
| 多线程操作                 | 1257 |
| 在一个线程中执行一段特定的代码       | 1260 |
| 为多线程创建线程池             | 1264 |
| 在线程池中的一个线程里执行代码       | 1270 |
| 与UI线程通信               | 1275 |
| 避免出现程序无响应ANR          | 1281 |
| JNI使用指南               | 1287 |

|                         |      |
|-------------------------|------|
| 优化多核处理器(SMP)下的Android程序 | 1305 |
| Android安全与隐私            | 1318 |
| Security Tips           | 1320 |
| 使用HTTPS与SSL             | 1334 |
| 企业版App                  | 1349 |
| 使用设备管理条例增强安全性           | 1352 |
| Android测试程序             | 1352 |
| 测试你的Activity            | 1360 |
| 建立测试环境                  | 1363 |
| 创建与执行测试用例               | 1367 |
| 测试UI组件                  | 1375 |
| 创建单元测试                  | 1383 |
| 创建功能测试                  | 1387 |