**Android官方培训课程中文版(v0.9.7)**

<http://hukai.me/android-training-course-in-chinese/index.html>

**从命令行安装运行应用程序**

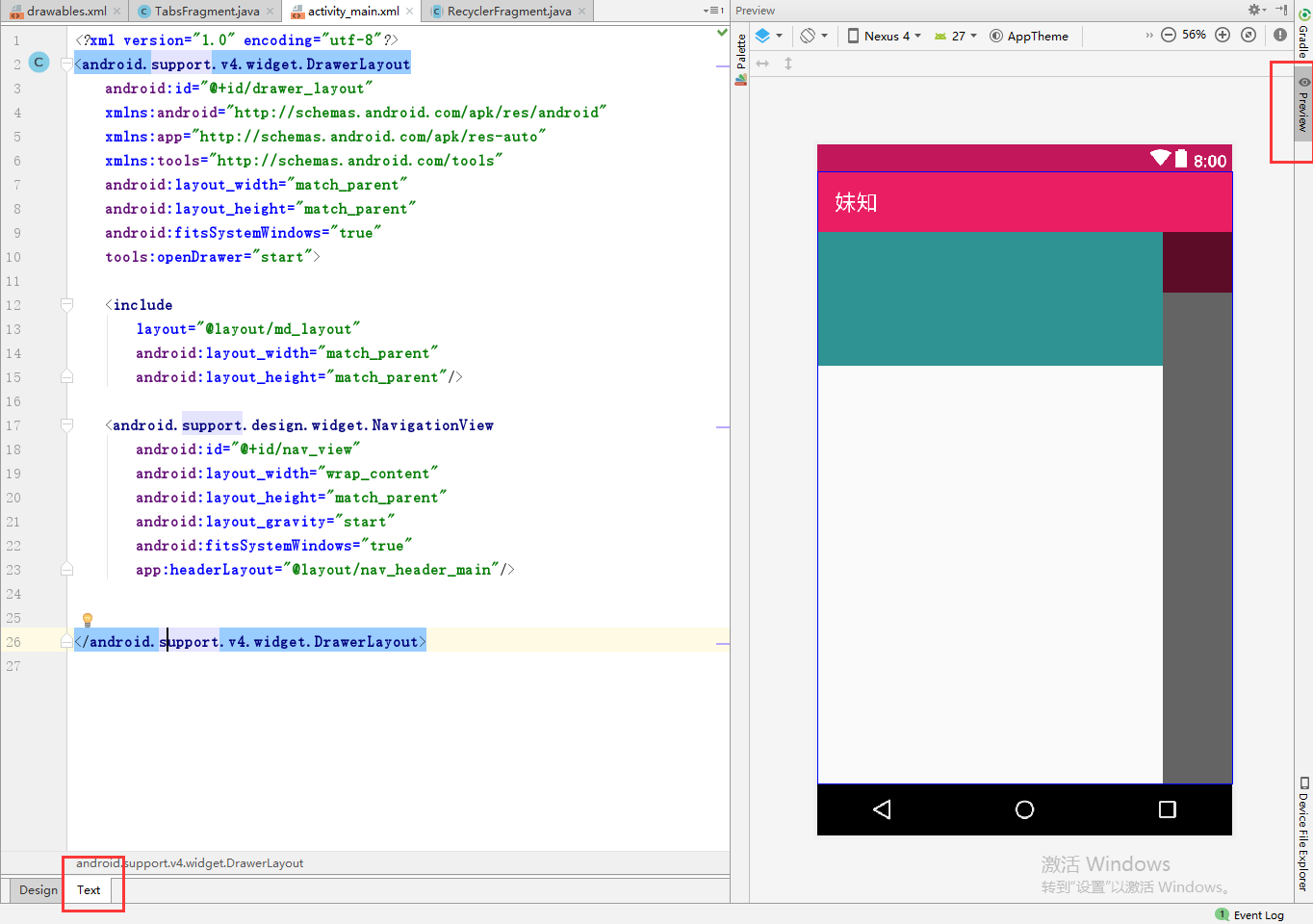
打开命令行并切换当前目录到Andriod项目的根目录，在debug模式下使用Gradle编译项目，使用gradle脚本执行assembleDebug编译项目，执行后会在build/目录下生成MyFirstApp-debug.apk。

Windows操作系统下，执行：

gradlew.bat assembleDebug

 在 **Preview** 面板点击 图标：隐藏关闭右侧 Preview 面板。

在 Android Studio 中打开布局文件时，可以看到一个 Preview 面板。点击这个面板中的标签，可利用 WYSIWYG（所见即所得）工具在 Design 面板看到对应的图形化效果。但在本节中，我们将学习如何直接修改 XML 文件。



在Android Studio中，按Alt + Enter 可以导入缺失的类

public class MyActivity extends ActionBarActivity {

public final static String EXTRA\_MESSAGE = "com.mycompany.myfirstapp.MESSAGE";

...

}

为让新启动的activity能查询extra数据。定义key为一个public型的常量，通常使用应用程序包名作为前缀来定义键是很好的做法，这样在应用程序与其他应用程序进行交互时仍可以确保键是唯一的。

将TextView加入之前被标记为R.id.content的RelativeLayout中

RelativeLayout layout = (RelativeLayout) findViewById(R.id.content);

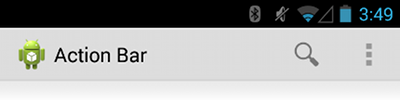
TextView textView = new TextView(this);

layout.addView(textView);

**Action Bar**

。核心的功能包括：

* 一个专门的空间用来显示你的app的标识，以及指出目前所处在app的哪个页面。
* 以一种可预见的方式访问重要的操作（比如搜索）。菜单按钮
* 支持导航和视图切换（通过Tabs和下拉列表）



设置一个基本的 action bar，需要 app 使用一个 activity 主题，该主题必须是 action bar 可用的。

菜单被添加到action bar 中

要为 action bar 布局菜单条目，就要在 activity 中实现 [onCreateOptionsMenu()](https://developer.android.com/reference/android/app/Activity.html#onCreateOptionsMenu(android.view.Menu)) 回调方法来 inflate 菜单资源从而获取 [Menu](https://developer.android.com/reference/android/view/Menu.html) 对象。

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()](https://developer.android.com/reference/android/app/Activity.html" \l "onOptionsItemSelected(android.view.MenuItem))的回调方法。

**为下级 Activity 添加向上按钮**

在不是程序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**up button(向上按钮)**。





然后，通过调用[setDisplayHomeAsUpEnabled()](https://developer.android.com/reference/android/app/ActionBar.html" \l "setDisplayHomeAsUpEnabled(boolean)) 来把 app icon 设置成可用的向上按钮：

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

// 如果你的minSdkVersion属性是11活更高, 应该这么用:

// 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。

# 自定义ActionBar的风格

当使用 Support 库时，必须使用 [Theme.AppCompat](https://developer.android.com/reference/android/support/v7/appcompat/R.style.html#Theme_AppCompat) 主题替代：

* [Theme.AppCompat](https://developer.android.com/reference/android/support/v7/appcompat/R.style.html#Theme_AppCompat)，一个“dark”的主题
* [Theme.AppCompat.Light](https://developer.android.com/reference/android/support/v7/appcompat/R.style.html#Theme_AppCompat_Light)，一个“light”的主题
* [Theme.AppCompat.Light.DarkActionBar](https://developer.android.com/reference/android/support/v7/appcompat/R.style.html#Theme_AppCompat_Light_DarkActionBar)，一个带有“dark” action bar 的“light”主题



然后，将主题应用到 app 全局或单个的 activity 之中：

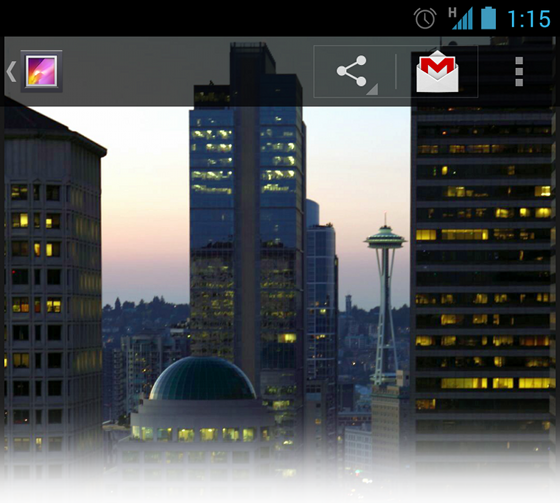
<application android:theme="@style/CustomActionBarTheme" ... />

# ActionBar的覆盖叠加

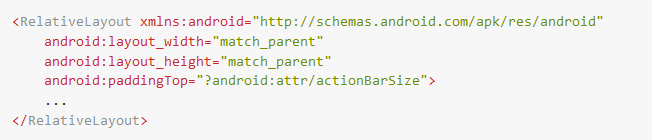
默认情况下，action bar 显示在 activity 窗口的顶部，会稍微地减少其他布局的有效空间。如果在用户交互过程中要隐藏和显示 action bar，可以通过调用 [ActionBar](https://developer.android.com/reference/android/app/ActionBar.html) 中的 [hide()](https://developer.android.com/reference/android/app/ActionBar.html#hide())和[show()](https://developer.android.com/reference/android/app/ActionBar.html" \l "show())来实现。但是，这将导致 activity 基于新尺寸重新计算与绘制布局。

为避免在 action bar 隐藏和显示过程中调整布局的大小，可以为 action bar 启用叠加模式(**overlay mode**)。在叠加模式下，所有可用的空间都会被用来布局就像ActionBar不存在一样，并且 action bar 会叠加在布局之上。这样布局顶部就会有点被遮挡，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。

**Note**：如果希望 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。关于如何定义 action bar 的背景，请查看 [自定义ActionBar的风格](http://hukai.me/android-training-course-in-chinese/basics/actionbar/styling.html)。



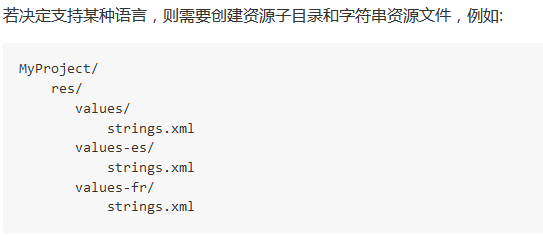
当 action bar 启用叠加模式时，它可能会遮挡住本应保持可见状态的布局。为了确保这些布局始终位于 action bar 下部，可以使用 [actionBarSize](https://developer.android.com/reference/android/R.attr.html#actionBarSize) 属性来指定顶部margin或padding的高度来到达。例如：





# 适配不同的语言

为支持多国语言，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



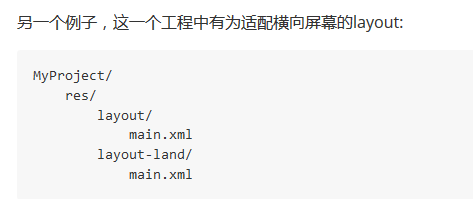


# 适配不同的屏幕

Android用尺寸和分辨率这两种常规属性对不同的设备屏幕加以分类

* 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)





## 创建不同的bitmap

我们应该为4种普遍分辨率:低，中，高，超高精度，都提供相适配的bitmap资源

要生成这些图像，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



## 运行时检查系统版本

Android在[Build](https://developer.android.com/reference/android/os/Build.html" \t "_blank)常量类中提供了对每一个版本的唯一代号，在我们的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() {

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

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.HONEYCOMB) {

ActionBar actionBar = getActionBar();

actionBar.setDisplayHomeAsUpEnabled(true);

}

}

**Note**:当解析XML资源时，Android会忽略当前设备不支持的XML属性。所以我们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。

## 使用平台风格和主题

Android提供了用户体验（内置）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app中。通过使用内置的风格和主题，我们的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activity看起来像对话框:

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

使activity有一个透明背景:

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

## Activity生命周期回调



Activity只能在三种状态之一下存在很长时间。

* **Resumed**：在这种状态下，Activity处于前台，且用户可以与其交互。（有时也称为“运行”状态。）
* **Paused**：在这种状态下，Activity被在前台中处于半透明状态或者未覆盖整个屏幕的另一个Activity—部分阻挡。暂停的Activity不会接收用户输入并且无法执行任何代码。
* **Stopped**：在这种状态下，Activity被完全隐藏并且对用户不可见；它被视为处于后台。停止时，Activity实例及其诸如成员变量等所有状态信息将保留，但它无法执行任何代码。

其他状态（“创建”和“开始”）是瞬态，

我们的activity应该在onPause()与onStop()中执行清除activity资源的操作。然而，如果activity含有在onCreate调用时创建的后台线程，或者是其他有可能导致内存泄漏的资源，则应该在OnDestroy()时进行资源清理，杀死后台线程。

除非程序在onCreate()方法里面就调用了finish()方法，系统通常是在执行了onPause()与onStop() 之后再调用onDestroy()

如果用户在这个activity还是在Paused 状态时回到这个activity，系统则会调用它的onResume() (2).

## 暂停Activity

当系统调用activity中的onPause()，从技术上讲，意味着activity仍然处于部分可见的状态.但更多时候意味着用户正在离开这个activity，并马上会进入Stopped state. 通常应该在onPause()回调方法里面做以下事情:

* 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。

我们应该避免在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(应该把那些heavy-load的工作放到[onStop()](http://developer.android.com/reference/android/app/Activity.html" \l "onStop())去做)。

## 恢复activity

请注意，系统每次调用这个方法时，activity都处于前台，包括第一次创建的时候。所以，应该实现onResume()来初始化那些在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

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的菜单并从我们的app切换到另外一个app，这个时候我们的app是被停止的。如果用户通过手机主界面的启动程序图标或者最近使用程序的窗口回到我们的app，那么我们的activity会重启。
* 用户在我们的app里面执行启动一个新activity的操作，当前activity会在第二个activity被创建后stop。如果用户点击back按钮，第一个activtiy会被重启。
* 用户在使用我们的app时接收到一个来电通话.

## 停止activity

一旦activity停止了，系统会在需要内存空间时摧毁它的实例(和栈结构有关，通常back操作会导致前一个activity被销毁)。极端情况下，系统会直接杀死我们的app进程，并不执行activity的[onDestroy()](http://developer.android.com/reference/android/app/Activity.html" \l "onDestroy())回调方法, 因此我们需要使用onStop()来释放资源，从而避免内存泄漏。(这点需要注意)

即使系统会在activity stop时停止这个activity，它仍然会保存[View](http://developer.android.com/reference/android/view/View.html" \t "_blank)对象的状态(比如[EditText](http://developer.android.com/reference/android/widget/EditText.html" \t "_blank)中的文字) 到一个[Bundle](http://developer.android.com/reference/android/os/Bundle.html" \t "_blank)中，并且在用户返回这个activity时恢复它们（控件的默认实现）

因为onStop()方法应该做清除所有activity资源的操作，我们需要在重启activtiy时重新实例化那些被清除的资源，同样, 我们也需要在activity第一次创建时实例化那些资源。因为系统会在创建activity与从停止状态重启activity时都会调用onStart()，应该使用onStart()作为onStop()所对应方法。也就是说，我们在onStop里面做了哪些清除的操作，就该在onStart里面重新把那些清除掉的资源重新创建出来。

你的Activity会在每次旋转屏幕时被destroyed与recreated

当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，系统就丢失了对Activity实例的引用，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Activity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](http://developer.android.com/reference/android/os/Bundle.html" \t "_blank)对象中的key-value pairs。

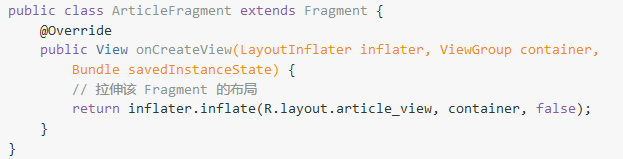
Activity被destroy时，系统可以选择保存数据。默认情况下, 系统使用 Bundle 实例来保存每一个View(视图)对象中的信息(例如输入EditText 中的文本内容)。如果希望保存额外的数据，可以在[onSaveInstanceState()](http://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle)) 中把数据写到bundle实例中

当activity开始Stop时，只有在Activity实例会需要重新创建的情况下才会调用到[onSaveInstanceState()](http://developer.android.com/reference/android/app/Activity.html" \l "onSaveInstanceState(android.os.Bundle)) ，这个方法会默认保存Activity视图的状态信息

若系统在Activity被Destory之后想重新创建这个Activity实例时，之前的Bundle对象会(系统)被传递到你我们activity的[onRestoreInstanceState()](http://developer.android.com/reference/android/app/Activity.html" \l "onRestoreInstanceState(android.os.Bundle))方法与 onCreate() 方法中。

我们也可以选择实现 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() ，因此不需要检查 Bundle 是否为null。**

创建 [Fragment](https://developer.android.com/reference/android/support/v4/app/Fragment.html) 时，必须重写 [onCreateView()](https://developer.android.com/reference/android/support/v4/app/Fragment.html#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle)) 回调方法来定义布局。事实上，这是唯一一个为使 Fragment 运行起来需要重写的回调方法



要用一个 Fragment 替换另一个 Fragment，Activity 的布局中需要包含一个作为 Fragment 容器的空 [FrameLayout](https://developer.android.com/reference/android/widget/FrameLayout.html)。

getSupportFragmentManager().beginTransaction()

.add(R.id.fragment\_container, firstFragment).commit();

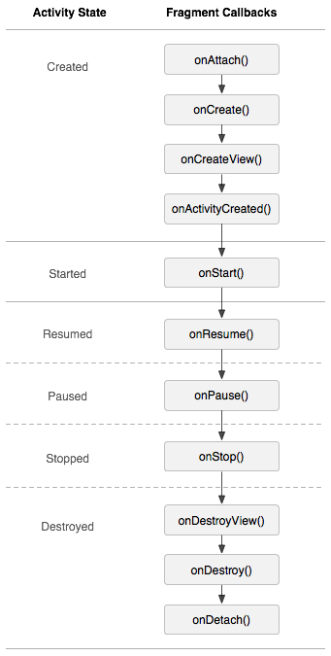
// 将 fragment\_container View 中的内容替换为此 Fragment，

// 然后将该事务添加到返回堆栈，以便用户可以向后导航

transaction.replace(R.id.fragment\_container, newFragment);

transaction.addToBackStack(null);

通常 Fragment 之间可能会需要交互，比如基于用户事件的内容变更。所有 Fragment 之间的交互应通过与之关联的 Activity 来完成。两个 Fragment 之间不应直接交互。

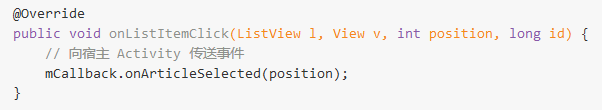


## 定义接口

为了让 Fragment 与包含它的 Activity 进行交互，可以在 Fragment 类中定义一个接口，并在 Activity 中实现。该 Fragment 在它的 onAttach() 方法生命周期中获取该接口的实现，然后调用接口的方法，以便与 Activity 进行交互。（译注：意即，若该 Fragment 中实现了 onAttach() 方法，则会被自动调用。）



例如，当用户点击列表条目时，Fragment 中的下面的方法将被调用。Fragment 用回调接口将事件传递给父 Activity。



## 向 Fragment 传递消息

宿主 Activity 通过 [findFragmentById()](http://developer.android.com/reference/android/support/v4/app/FragmentManager.html#findFragmentById(int)) 获取 [Fragment](http://developer.android.com/reference/android/support/v4/app/Fragment.html) 的实例，然后直接调用 Fragment 的 public 方法向 Fragment 传递消息。

* **Internal storage:**
  + 总是可用的
  + 这里的文件默认只能被我们的app所访问。
  + 当用户卸载app的时候，系统会把internal内该app相关的文件都清除干净。
  + Internal是我们在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，因为用户有时会通过USB存储模式挂载外部存储器，当取下挂载的这部分后，就无法对其进行访问了。
  + 是大家都可以访问的，因此保存在这里的文件可能被其他程序访问。
  + 当用户卸载我们的app时，系统仅仅会删除external根目录（[getExternalFilesDir()](http://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String))）下的相关文件。
  + External是在不需要严格的访问权限并且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。

## 保存到Internal Storage

当保存文件到internal storage时，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](http://developer.android.com/reference/java/io/File.html) 的对象：

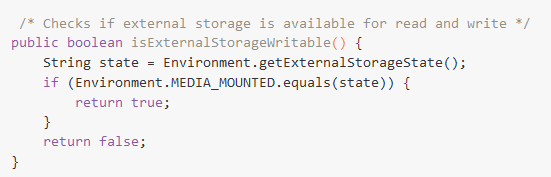
* [getFilesDir()](http://developer.android.com/reference/android/content/Context.html#getFilesDir()) : 返回一个[File](http://developer.android.com/reference/java/io/File.html" \t "_blank)，代表了我们app的internal目录。
* [getCacheDir()](http://developer.android.com/reference/android/content/Context.html#getCacheDir()) : 返回一个[File](http://developer.android.com/reference/java/io/File.html" \t "_blank)，代表了我们app的internal缓存目录。

可以使用[File()](http://developer.android.com/reference/java/io/File.html#File(java.io.File, java.lang.String)) 构造器在那些目录下创建一个新的文件，如下：

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

## 保存文件到External Storage

因为external storage可能是不可用的，比如遇到SD卡被拔出等情况时。因此在访问之前应对其可用性进行检查。我们可以通过执行 [getExternalStorageState()](http://developer.android.com/reference/android/os/Environment.html#getExternalStorageState())来查询external storage的状态。若返回状态为[MEDIA\_MOUNTED](http://developer.android.com/reference/android/os/Environment.html" \l "MEDIA_MOUNTED" \t "_blank), 则可以读写。示例如下：



可以利用一个带有action的intent使当前app能够跳转到其他app。

## 隐式的Intent

Intent.ACTION\_DIAL 是action

ntent callIntent = new Intent(Intent.ACTION\_DIAL, number);

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。如果我们没有在intent中包含一个Uri, 则通常需要使用 [setType()](http://developer.android.com/reference/android/content/Intent.html#setType(java.lang.String)) 方法来指定intent附带的数据类型。设置MIME type 是为了指定应该接受这个intent的activity。例如：

* 发送一个带附件的email:

Intent emailIntent = new Intent(Intent.ACTION\_SEND);

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

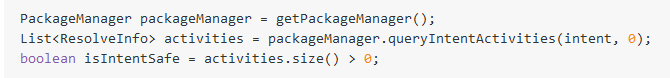
emailIntent.setType(HTTP.PLAIN\_TEXT\_TYPE);

第二个参数是uri?

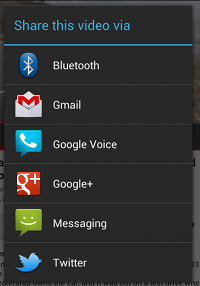
我们还是应该在触发一个intent之前做验证是否有App接受这个intent的步骤

**如果触发了一个intent，而且没有任何一个app会去接收这个intent，则app会crash**

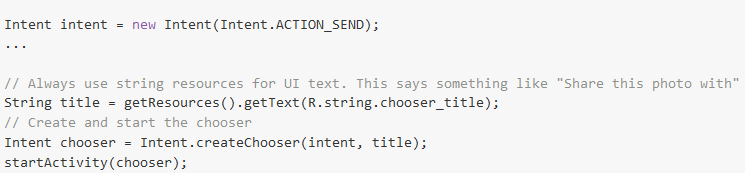
需要执行[queryIntentActivities()](http://developer.android.com/reference/android/content/pm/PackageManager.html" \l "queryIntentActivities(android.content.Intent, int)) 来获取到能够接收这个intent的所有activity的list。若返回的[List](http://developer.android.com/reference/java/util/List.html" \t "_blank)非空，那么我们才可以安全的使用这个intent。例如：



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）。

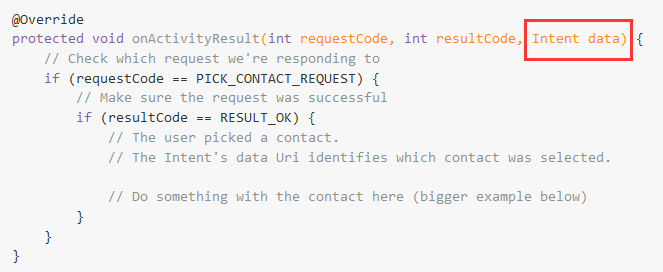


为了显示chooser, 需要使用[createChooser()](http://developer.android.com/reference/android/content/Intent.html" \l "createChooser(android.content.Intent, java.lang.CharSequence))来创建Intent

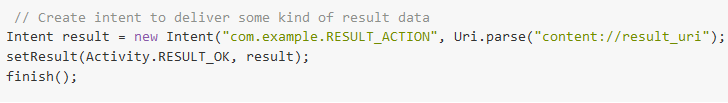


startActivityForResult(pickContactIntent, PICK\_CONTACT\_REQUEST);

Intent 参数是上个activity返回的结果



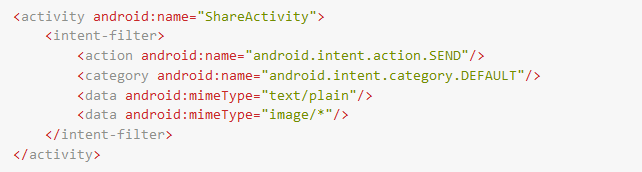
## 返回Result



## 添加Intent Filter

**Data**:Intent附带数据的描述。在intent filter中通过[<data>](http://developer.android.com/guide/topics/manifest/data-element.html" \t "_blank)指定它的值，可以使用一个或者多个属性，我们可以只定义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。

例如，这个有intent filter的activity，当数据类型为文本或图像时会处理ACTION\_SEND的intent。

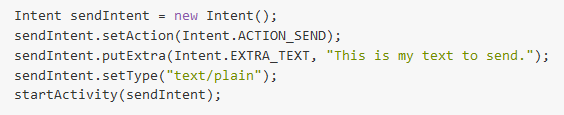


每一个发送出来的intent只会包含一个action与data类型，但handle这个intent的activity的 <intent-filter>可以声明多个<action>, <category>与<data> 。

Android定义了一些actions，比如ACTION\_SEND，该action表明该intent用于从一个activity发送数据到另外一个activity的，甚至可以是跨进程之间的数据发送。

为了发送数据到另外一个activity，我们只需要指定数据与数据的类型

## 分享文本内容(Send Text Content)



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

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类型会根据分享的混合内容而不同。例如，如果分享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我们的数据。下面是一个例子：

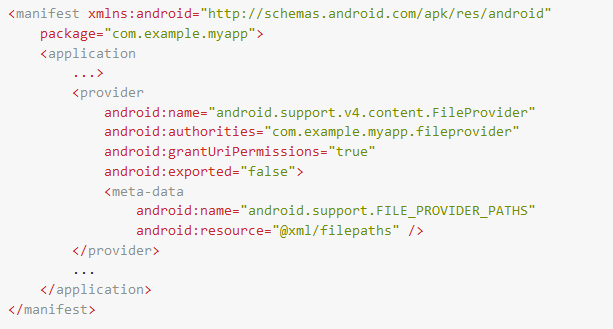


# 添加一个简便的分享功能

<http://hukai.me/android-training-course-in-chinese/content-sharing/sharing/shareaction.html>

介绍了如何使用Android的[FileProvider](http://developer.android.com/reference/android/support/v4/content/FileProvider.html" \t "_blank)组件所创建的content URI在应用之间安全的共享文件。当然，要做到这一点，还需要给接收文件的应用程序访问的这些content URI授予临时访问权限。

## 在Manifest清单文件中指定FileProvider



这里，[android:authorities](http://developer.android.com/guide/topics/manifest/provider-element.html" \l "auth" \t "_blank)字段指定了希望使用的Authority，该Authority针对于[FileProvider](http://developer.android.com/reference/android/support/v4/content/FileProvider.html" \t "_blank)所生成的content URI。本例中的Authority是“com.example.myapp.fileprovider”。对于自己的应用，要在我们的应用程序包名（[android:package](http://developer.android.com/guide/topics/manifest/manifest-element.html" \l "package" \t "_blank)的值）之后继续追加“fileprovider”来指定Authority。要更多关于Authority的知识，请参考：[Content URIs](http://developer.android.com/guide/topics/providers/content-provider-basics.html" \l "ContentURIs" \t "_blank)，以及[android:authorities](http://developer.android.com/guide/topics/manifest/provider-element.html" \l "auth" \t "_blank)。

[<provider>](http://developer.android.com/guide/topics/manifest/provider-element.html)下的[<meta-data>](http://developer.android.com/guide/topics/manifest/meta-data-element.html" \t "_blank)指向了一个XML文件，该文件指定了我们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。首先要在“res/xml/”下创建文件“filepaths.xml”



在这个例子中，<files-path>标签共享的是在我们应用的内部存储中“files/”目录下的目录。“path”属性字段指出了该子目录为“files/”目录下的子目录“images/”。

在这个例子中，<files-path>标签共享的是在我们应用的内部存储中“files/”目录下的目录。“path”属性字段指出了该子目录为“files/”目录下的子目录“images/”。

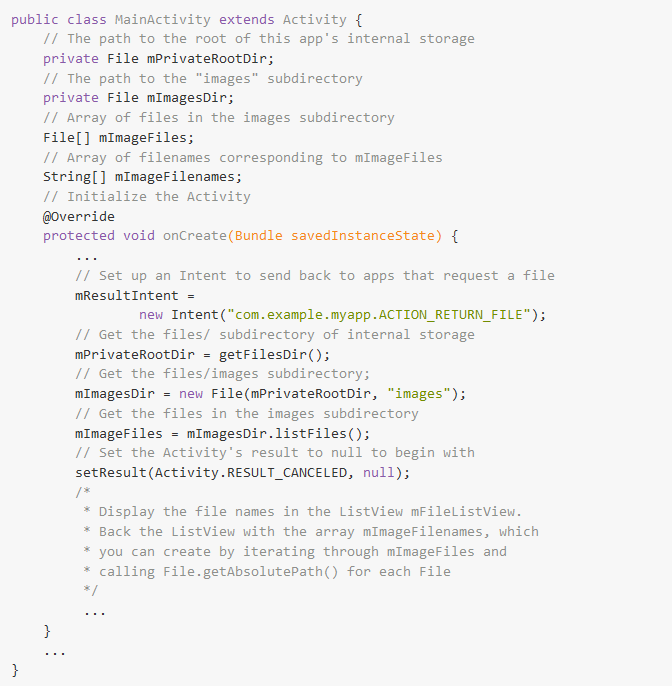
## 接收文件请求

为了从客户端应用程序接收一个文件获取请求并以Content URI的形式进行响应，我们的应用程序应该提供一个选择文件的[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)。客户端应用程序通过调用[startActivityForResult()](http://developer.android.com/reference/android/app/Activity.html" \l "startActivityForResult(android.content.Intent, int))方法启动这一[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)。

在Manifest清单文件中定义[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)，在其Intent过滤器中，匹配[ACTION\_PICK](http://developer.android.com/reference/android/content/Intent.html#ACTION_PICK) Action及[CATEGORY\_DEFAULT](http://developer.android.com/reference/android/content/Intent.html" \l "CATEGORY_DEFAULT" \t "_blank)和[CATEGORY\_OPENABLE](http://developer.android.com/reference/android/content/Intent.html" \l "CATEGORY_OPENABLE" \t "_blank)这两种Category。

### 在代码中定义文件选择Activity

下面，定义一个[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)子类，用于显示在内部存储的“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。下面代码展示了如何定义该[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)，并令其响应用户的选择：



## 响应一个文件选择

根据被选中文件的文件名获取一个[File](http://developer.android.com/reference/java/io/File.html" \t "_blank)对象，然后将其作为参数传递给[getUriForFile()](http://developer.android.com/reference/android/support/v4/content/FileProvider.html" \l "getUriForFile(android.content.Context, java.lang.String, java.io.File))，另外还需传入的参数是在[<provider>](http://developer.android.com/guide/topics/manifest/provider-element.html" \t "_blank)标签中为[FileProvider](http://developer.android.com/reference/android/support/v4/content/FileProvider.html" \t "_blank)所指定的Authority

fileUri = FileProvider.getUriForFile(

MainActivity.this,

"com.example.myapp.fileprovider",

requestFile);

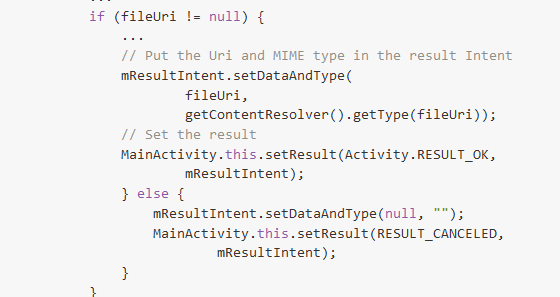
现在已经有了想要共享给其他应用程序的文件所对应的Content URI，我们需要允许客户端应用程序访问这个文件。为了达到这一目的，可以通过将Content URI添加至一个[Intent](http://developer.android.com/reference/android/content/Intent.html" \t "_blank)中，然后为该[Intent](http://developer.android.com/reference/android/content/Intent.html" \t "_blank)设置权限标记。所授予的权限是临时的，并且当接收文件的应用程序的任务栈终止后，会自动过期。

mResultIntent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

mResultIntent 是要SetResult返回的intent

## 与请求应用共享文件

为了向请求文件的应用程序提供其需要的文件，我们将包含了Content URI和相应权限的[Intent](http://developer.android.com/reference/android/content/Intent.html" \t "_blank)传递给[setResult()](http://developer.android.com/reference/android/app/Activity.html" \l "setResult(int))。当定义的[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)结束后，系统会把这个包含了Content URI的[Intent](http://developer.android.com/reference/android/content/Intent.html" \t "_blank)传递给客户端应用程序。下例展示了其中的核心步骤：



# 获取文件信息

<http://hukai.me/android-training-course-in-chinese/content-sharing/secure-file-sharing/retrieve-info.html>

当一个客户端应用程序拥有了文件的Content URI之后。可以向服务端应用程序获取关于文件的信息，包括文件的数据类型和文件大小等等

/\*

\* 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);

## 获取文件名及文件大小

[FileProvider](http://developer.android.com/reference/android/support/v4/content/FileProvider.html)类有一个[query()](http://developer.android.com/reference/android/support/v4/content/FileProvider.html" \l "query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String))方法的默认实现，它返回一个[Cursor](http://developer.android.com/reference/android/database/Cursor.html" \t "_blank)对象，该Cursor对象包含了Content URI所关联的文件的名称和大小。默认的实现返回下面两列信息：

## 鉴别使用的是哪个音频流

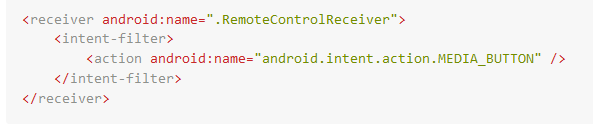
为了创建一个良好的音频体验，我们首先需要知道应用会使用到哪些音频流。Android为播放音乐，闹铃，通知铃，来电声音，系统声音，打电话声音与拨号声音分别维护了一个独立的音频流。这样做的主要目的是让用户能够单独地控制不同的种类的音频

## 使用硬件音量键来控制应用的音量

可能希望通过监听音量键被按下的事件，来调节音频流的音量。其实我们不必这样做。Android提供了[setVolumeControlStream()](http://developer.android.com/reference/android/app/Activity.html" \l "setVolumeControlStream(int))方法来直接控制指定的音频流。

因为该方法只需要在Activity整个生命周期中调用一次，通常，我们可以在负责控制多媒体的[Activity](http://developer.android.com/reference/android/app/Activity.html" \t "_blank)或者[Fragment](http://developer.android.com/reference/android/app/Fragment.html" \t "_blank)的onCreate()方法中调用它。这样能确保不管应用当前是否可见，音频控制的功能都能符合用户的预期。

许多线控或者无线耳机都会有许多媒体播放控制按钮，例如：播放，停止，暂停，跳过，以及回放等。无论用户按下设备上任意一个控制按钮，系统都会广播一个带有[ACTION\_MEDIA\_BUTTON](http://developer.android.com/reference/android/content/Intent.html" \l "ACTION_MEDIA_BUTTON" \t "_blank)的Intent。为了正确地响应这些操作，需要在Manifest文件中注册一个针对于该Action的[BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html" \t "_blank)



在Receiver的实现中，需要判断这个广播来自于哪一个按钮，Intent通过[EXTRA\_KEY\_EVENT](http://developer.android.com/reference/android/content/Intent.html" \l "EXTRA_KEY_EVENT" \t "_blank)这一Key包含了该信息，

因为可能会有多个程序在监听与媒体按钮相关的事件，所以我们必须在代码中控制应用接收相关事件的时机。下面的例子显示了如何使用[AudioManager](http://developer.android.com/reference/android/media/AudioManager.html)来为我们的应用注册监听与取消监听媒体按钮事件，当Receiver被注册上时，它将是唯一一个能够响应媒体按钮广播的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，应用需要在他们失去焦点或者不可见的时候（比如在[onStop()](http://developer.android.com/reference/android/app/Activity.html" \l "onStop())方法里面）取消注册监听。但是对于媒体播放应用来说并没有那么简单，实际上，在应用不可见（不能通过可见的UI控件进行控制）的时候，仍然能够响应媒体播放按钮事件是极其重要的。为了实现这一点，有一个更好的方法，我们可以在程序获取与失去音频焦点的时候注册与取消对音频按钮事件的监听。这个内容会在后面的课程中详细讲解。

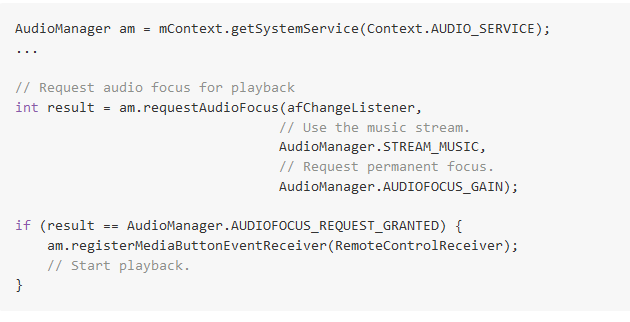
即只有获取到音频焦点的应用才能够播放音频。

## 请求获取音频焦点

如果请求成功，该方法会返回[AUDIOFOCUS\_REQUEST\_GRANTED](http://developer.android.com/reference/android/media/AudioManager.html#AUDIOFOCUS_REQUEST_GRANTED)。

另外我们必须指定正在使用的音频流，而且需要确定所请求的音频焦点是短暂的（Transient）还是永久的（Permanent）。

下面的代码片段是一个在播放音乐时请求永久音频焦点的例子



一旦结束了播放，需要确保调用了[abandonAudioFocus()](http://developer.android.com/reference/android/media/AudioManager.html" \l "abandonAudioFocus(android.media.AudioManager.OnAudioFocusChangeListener))方法。这样相当于告知系统我们不再需要获取焦点并且注销所关联的[AudioManager.OnAudioFocusChangeListener](http://developer.android.com/reference/android/media/AudioManager.OnAudioFocusChangeListener.html)监听器

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”。通常情况下，一个应用在失去音频焦点时会立即关闭它的播放声音。如果我们选择在请求短暂音频焦点的时候开启了Ducking，那意味着其它应用可以继续播放，仅仅是在这一刻降低自己的音量，直到重新获取到音频焦点后恢复正常音量

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

如果应用A请求获取了音频焦点，那么在应用B请求获取音频焦点的时候，A获取到的焦点就会失去。如何响应失去焦点事件，取决于失去焦点的方式。

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()](http://developer.android.com/reference/android/media/AudioManager.OnAudioFocusChangeListener.html" \l "onAudioFocusChange(int))回调方法。如之前提到的，获取焦点有三种类型，我们同样会有三种失去焦点的类型：永久失去，短暂失去，允许Ducking的短暂失去。

在Android应用中加载Bitmaps的操作是需要特别小心处理的

1 Bitmap会消耗很多内存，

2Android应用的UI通常会在一次操作中立即加载许多张bitmaps，而且需要预先加载一些没有在屏幕上显示的内容，为用户滑动的显示做准备。

# 高效加载大图

考虑到应用是在有限的内存下工作的，理想情况是我们只需要在内存中加载一个低分辨率的照片即可。为了更便于显示，这个低分辨率的照片应该是与其对应的UI控件大小相匹配的。

## 读取位图的尺寸与类型

设置 [inJustDecodeBounds](http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html#inJustDecodeBounds) 属性为true可以在解码的时候避免内存的分配，它会返回一个null的Bitmap，但是可以获取到 outWidth, outHeight 与 outMimeType。该技术可以允许你在构造Bitmap之前优先读图片的尺寸与类型。



为了告诉解码器去加载一个缩小版本的图片到内存中，需要在[BitmapFactory.Options](http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html" \t "_blank) 中设置 inSampleSize 的值。例如, 一个分辨率为2048x1536的图片，如果设置 inSampleSize 为4，那么会产出一个大约512x384大小的Bitmap。

动画

淡入淡出

使用[Animator.AnimatorListener](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html)中的 [onAnimationEnd()](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html#onAnimationEnd(android.animation.Animator))，设置淡出View的visibility为[GONE](http://developer.android.com/reference/android/view/View.html#GONE)

