# Activity总结

目录

[Activity总结 1](#_Toc42718135)

[一、Activity是什么 1](#_Toc42718136)

[二、Activity的基本用法 1](#_Toc42718137)

[2.1 显示界面 1](#_Toc42718138)

[2.2可与用户交互的界面 2](#_Toc42718139)

[三、Activity的跳转以及数据传递 2](#_Toc42718140)

[3.1 Intent跳转 3](#_Toc42718141)

[3.2 Intent传递数据 4](#_Toc42718142)

[启动活动的最佳写法 5](#_Toc42718143)

[四、Activity的生命周期 6](#_Toc42718144)

[4.1 返回栈 6](#_Toc42718145)

[4.2活动状态 6](#_Toc42718146)

[4.3回调方法和生存期 7](#_Toc42718147)

[4.4几种常见的回调总结 8](#_Toc42718148)

[4.5生命周期的实践方案 9](#_Toc42718149)

[4.6对可见、可交互的理解 9](#_Toc42718150)

[五、Activity的启动模式 9](#_Toc42718151)

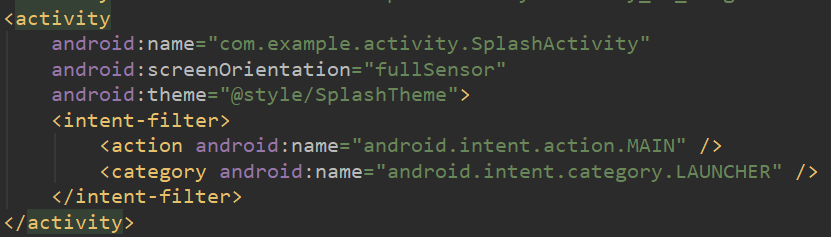
[六、Activity的集中管理 10](#_Toc42718152)

## 一、Activity是什么

Activity是一个包含用户界面、可与用户交互的组件。

## 二、Activity的基本用法

Activity作为四大组件(Activity、Service、Broadcast、ContentProvider)之一，使用的时候需要先在AndroidManifest.xml文件中进行配置。如下：

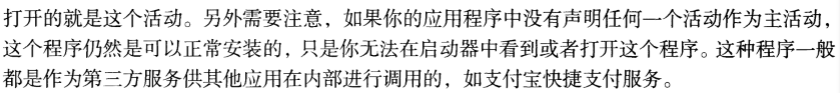


小结：关于配置，需要掌握activity的不同属性，以及子项(intent-filter、meta-data、nav-graph)的属性，合理地使用这些配置，或许可以达到事半功倍的效果。

小知识点：

1、主活动指定的label不仅会成为标题栏中的内容，还会成为启动器(Launcher)中应用程序显示的名称。

2、



### 2.1 显示界面

Activity最重要的功能之一。显示界面的核心方法：setContentView()。下面介绍三种方式：

* 一：控件+xml布局文件+Activity

第零步：如果是自定义控件的话，首先要设计好它的类。(我认为在设计的时候就应该考虑到这个控件将来要采用的一般显示方式)。

第一步：创建布局文件。

第二步：在Activity的onCreate()中，使用setContentView()加载。

第三步：使用findViewById()取到控件，对其进行设置。

* 二：控件+Activity，不适用xml，完全通过代码布局

比如，RadioGroup rg = new RadioGroup(context);

* 三：控件+Activity

这种方式，以Toast为典型。

总结：目前比较普遍的用法都是通过方式一，这样做的好处很明显，可以把布局(View)和控制(Controller)进行分离，开发起来效率高，后期维护更方便。但这种方式是通过xml进行布局的，会多出xml解析的过程，必然消耗一定性能。网上看过一个测试，结论为java代码比xml加载快了100%。(不过我自己写过一个小demo，LinearLayout中放一个ImageView，反而是xml快一点)。具体采用何种方式还是结合实际情况具体分析。

**涉及的知识点：各种控件(View)的使用；自定义控件。**

### 2.2可与用户交互的界面

用户交互指，当用户进行触摸、点击、锁屏、旋转、甚至是看等一系列操作时，当前的activity会做出响应。既然是与用户进行交互的，那设计activity的时候就应该充分考虑到用户的使用情形、可能触发的事件，从而进行有效的、人性化的交互(响应)。

**涉及的知识点：**

**事件分发机制；**软件交互；美学。

对于Android开发者而言，不仅需要知道Activity的各种方法以及它的生命周期，还需要了解Android系统是如何来管理Activity的。当打开一个Android程序的时候，我们并没有去new出一个Activity，那它是如何启动起来的呢？它的生命周期方法是如何回调的呢？

**涉及的知识点：**

**Activity的各种方法的使用； Activity的启动流程；Activity的生命周期。**

## 三、Activity的跳转以及数据传递

跳转的常用核心方法如下：

|  |  |
| --- | --- |
| startActivity(Intent intent) | 无返回值的跳转。 |
| startActivityForResult(Intent intent, int requestCode) | 有返回值的跳转。 |

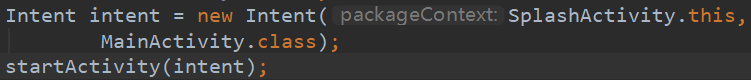
跳转需要一个重要的参数Intent。Intent可以指明要跳转到的Activity，并且可以携带数据。

### 3.1 Intent跳转

Intent是Android程序中各组件之间进行交互的一种重要方式，不仅可以指明当前组件想要执行的动作，还可以在不同组件之间传递数据。Intent一般可被用于启动活动、启动服务、发送广播等场景。

Intent大致可以分为两种：显式Intent、隐式Intent

**3.1.1显式Intent**



**3.1.2隐式Intent**

通过匹配<action>和<category>来找到跳转目标。



只有<action>和<category>中的内容和Intent指定的action和category完全一致，活动才能响应该Intent。

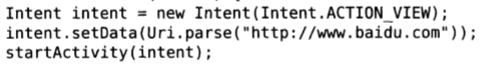


Intent会默认添加：android.intent.categoty.DEFAULT。

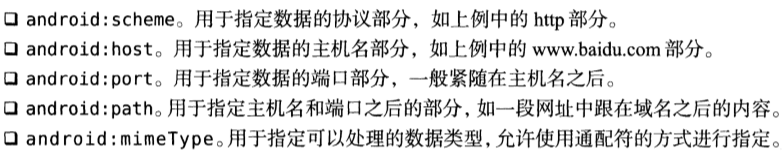
注：每个Intent可以只能指定一个Activity，但可以指定多个category。

**3.1.3一些常用Intent的总结**

* 打开网页



<data>是<intent-filter>中的最后一个标签(其余两个是<action>、<category>)，可以更精确的指定当前activity可以响应什么类型的数据。<data>中的配置主要有：

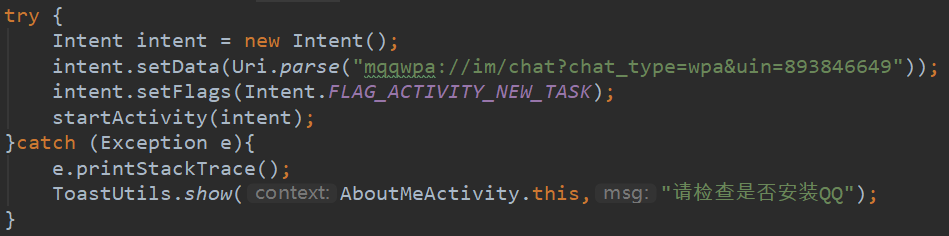


* 打开拨号界面



tel是scheme指定的协议之一，还有geo(地理位置)等。

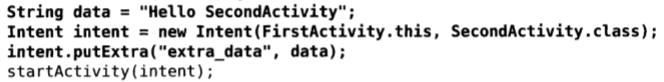
* 启动QQ



### 3.2 Intent传递数据

**3.2.1无返回值的跳转：使用startActivity();**

第一步：当前活动：传入数据



第二步：目标活动：获取数据

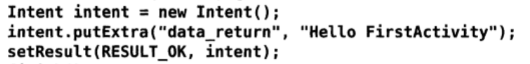


**3.2.2有返回值的跳转：使用startActivityForResult();**

第一步：当前活动，启动活动，传入参数请求码(requestCode)。要根据requestCode判断数据从哪个活动返回。

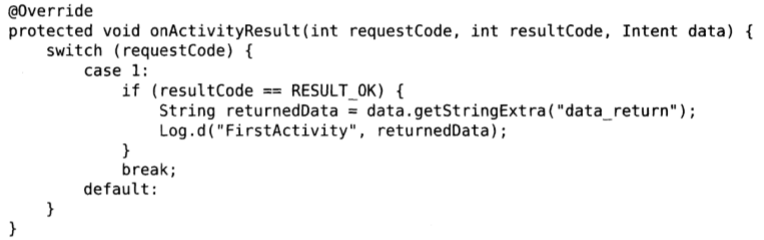


第二步：目标活动，返回数据。



一般在触发点击事件或者按下返回键时添加上面的代码。按下back键会回调onBackPressed()。

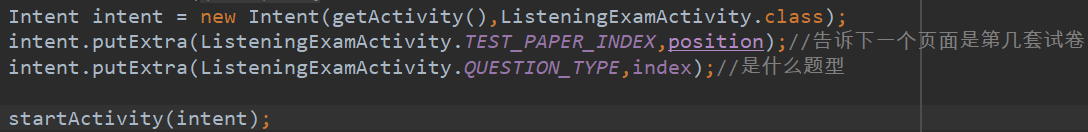
第三步：当前活动，在回调方法onActivityResult()中处理返回的数据。



应用场景：登录成功后更新用户信息界面。

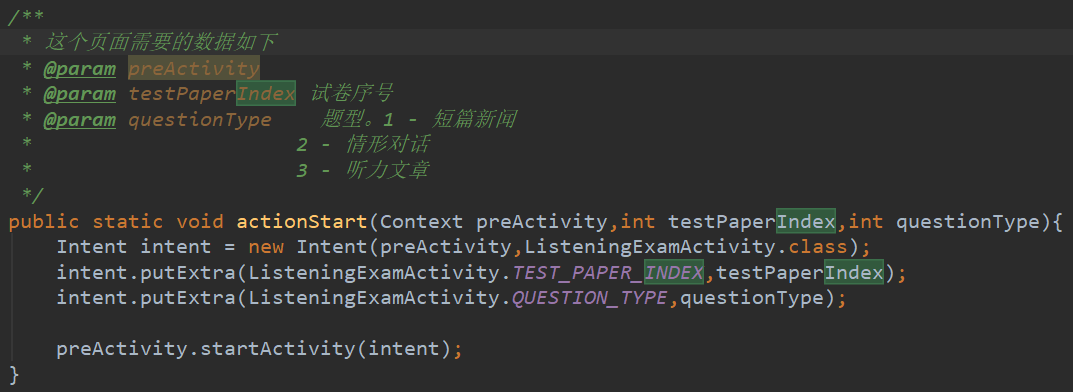
### 启动活动的最佳写法

当启动活动需要使用Intent传值时，一般操作如下：



在实际项目中，如果下一个页面不是我们自己写的，那传值的时候就要自己去询问下一个页面所需要的数据，以及数据的类型。为了提高效率，可以把第二个界面需要的数据进行统一说明。具体实现如下：

在第二个活动(接受数据的活动)中加入方法：

好处：

1. 写上一个页面的同事可以清楚的知道我们所需要的数据是什么，并且类型确定了，当我们修改需求后，上一个页面也会及时的知道(这个方法会报错)。
2. 上一个页面不需要关注intent.putExtra(“testPaperIndex”,1)的key值了。这样也可以避免key值写错。

## 四、Activity的生命周期

### 4.1 返回栈

Android中的activity全都归属于task管理 。task 是多个 activity 的集合， 这些 activity 按照启动顺序排队存入一个栈（即“back stack”）。android默认会为每个App维持一个task来存放该app的所有activity，task的默认name为该app的packagename。

当然我们也可以在AndroidMainfest.xml中申明activity的taskAffinity属性来自定义task，但不建议使用，如果其他app也申明相同的task，它就有可能启动到你的activity，带来各种安全问题（比如拿到你的Intent）。

关于taskAffinity属性和Android的安全问题，后续在探讨。

返回栈和生命周期的联系总结一下：

启动新活动，入栈，位于栈顶。

按下back键或调用finish()，栈顶活动出栈。

系统总是显示位于栈顶的活动。

### 4.2活动状态

每个活动在其生命周期中最多可能有四种状态。

1. **运行状态**

位于栈顶。可交互。

系统最不愿回收。

1. **暂停状态**

不在栈顶，仍然可见。

情形一：跳转

当活动跳转时，第一个活动会先调用onPause()，然后等待第二个活动显示onCreate() –》onStart()，在活动二从创建到显示的过程中，活动一处于暂停状态且可见。

注：目前已知onPause()执行完成后，第二个活动才会onCreate(),所以onPause中执行的操作，要尽量快速，或者放到子线程中，以避免对UI线程的影响。现在还需要探讨onCreate过程中，第一个活动会是怎样的？

情形二：启动一个非全屏的活动。

比如启动了一个对话框形式的活动。暂停状态的活动还完全存活着，系统也不愿回收(它还可见，在内存极低的情况下，系统才会考虑回收)。

1. **停止状态**

不在栈顶，完全不可见。

系统会保存相应的状态和成员变量，但不可靠。因为当其他地方需要内存时，有可能被回收。

1. **销毁状态**

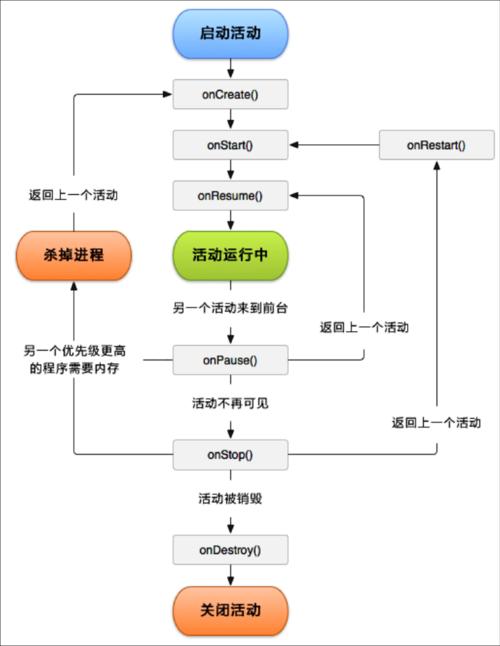
移出返回栈。

最倾向于回收。

注：回收，就是系统把一个对象从内存中去除，以保证内存充足。因为一个活动即使被finish()，回调了onDestory()，也只是把活动移出管理栈，并没有置为null，还存在于内存中。

### 4.3回调方法和生存期

#### 4.3.1 常用生命周期方法



|  |  |
| --- | --- |
| onCreate() | 不可见。加载视图，绑定事件。 |
| onStart() | 不可见🡪可见。 |
| onResume() | 可交互。栈顶，运行状态。 |
| onPause() | 可见。保存数据。释放资源。 |
| onStop() | 不可见。 |
| onDestory() | 销毁，出栈。 |
| onRestart() | 停止🡪运行，之前回调。 |

除了onRestart()，生命周期两两相对，可分为3种生存期。

* 完整生存期

onCreate()🡪onDestory()。

* 可见生存期

onStart🡪onStop()。

* 前台生存期

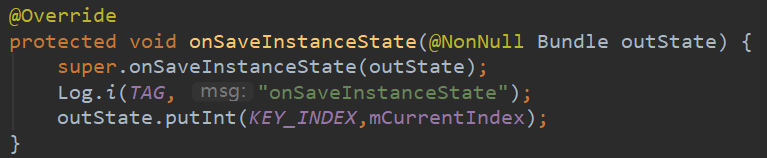
onResume()🡪onPause()。

#### 其他生命周期方法

1. onSaveInstanceState()。

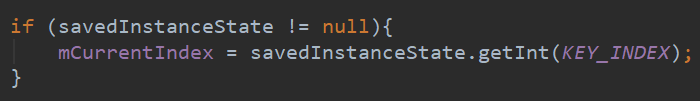
这个方法一定会在活动被回收前调用，因此可以在这里保存临时数据，在下创建活动时，可以恢复数据。

例：在此方法对用户写的内容进行保存

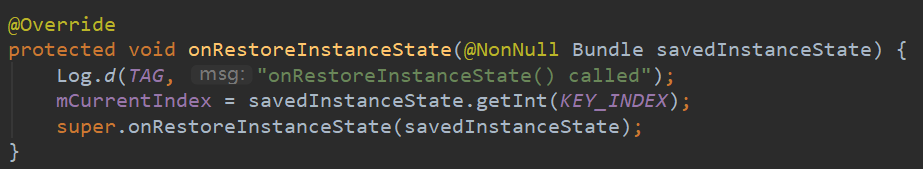


并在再次创建活动时恢复：

在onCreate中恢复：需要判断savedInstanceState是否为空。



或者在onRestoreInstanceState()中恢复，不需要判空。



注：需要注意的是， onRestoreInstanceState()的回调是在onStart()之后的，所以如果保存的数据是希望用来恢复界面的，就不太适合放在这里取出了，而应该放到onCreate()中。

1. onNewIntent()

做过简单的探究，但目前在项目中还没有使用过。是否回调和activity的启动模式有关系，具体的细节还不是很理解。

<https://blog.csdn.net/qq_36478274/article/details/105989145>

### 4.4几种常见的回调总结

1. 跳转

FirstActivity：onPause() 🡪onStop()

SecondActivity： 🡪onCreate()🡪onStart()🡪onResume

1. 返回键

OnPause()🡪onStop()🡪onDestory()

1. Home键

onPause🡪onSaveInstanceState🡪onStop()

再启动：onRestart()🡪onStart()🡪onResume

1. 横竖屏

销毁重建。即onPause🡪…🡪onDestory🡪onCreate…onResume。

网上有说横🡪竖：会执行两次。我用vivoX6Plus-Android5.1.1测试，只会执行一次流程。不同手机、Android版本应该有不同的回调方式。

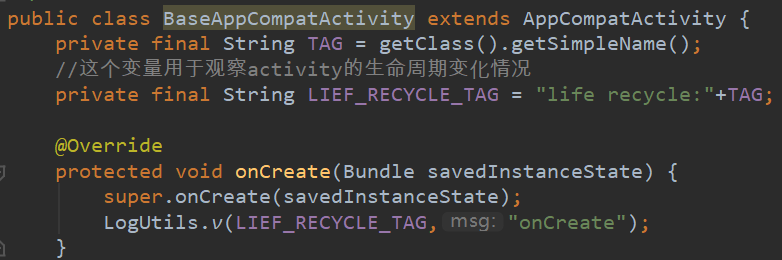
1. 锁屏

onPause🡪onSaveInstanceState🡪onStop()

再启动：onRestart()🡪onStart()🡪onResume

### 4.5生命周期的实践方案

在开发过程中，可以先写一个基类用来统一观察生命周期的变化情况，如下：



### 4.6对可见、可交互的理解

1、当跳转的时候，可见只是意为着当前活动可见，并不是说，下一个活动在onStart执行完就可见，还是要等到onResume执行结束才能看见。

2、可交互，是指在执行onPause🡪onResume之间，不可交互，要到onResume结束后才可以交互。

## 五、Activity的启动模式

启动模式(android:launchMode)有四种：standard、singleTop、singleTask和singleInstance。

1. **standard**

标准模式，是系统的默认模式。每次启动一个Activity都会重新创建一个新的实例，不管这个实例是否已经存在。

应用场景：

默认模式，适用于大多数应用场景。

1. **singleTop**

栈顶复用模式。在此模式下，如果想要跳转的Activity已经位于任务栈的栈顶，那么此Activity不会被重新创建。如果此Activity的实例已经存在但是不在栈顶，那么此Activity会被创建。同时onNewIntent()会被调用，而onCreate()、onStart()、onResume()不会被调用。

应用场景：

多个方式打开的页面。点击通知打开的页面。

1. **singleTask**

栈内复用模式。是一种单实例模式。在此模式下，只要想要跳转的Activity在任务栈中存在，那么启动此Activity都不需要创建新的实例，和singleTop一样，会回调onNewIntent()，onCreate()、onStart()、onResume()不会被调用。

应用场景：

App主页。可以在退出时销毁所有活动。

1. **singleInstance**

单实例模式，是一种加强的singleTask模式。除了具有singleTask所有的特性，还有在此模式下的Activity只能单独的在一个任务栈中。

应用场景：

闹铃、通话界面。

其他说明：

SingleTask指定不同的taskAffinity，也会启动一个新的返回栈。

getTaskId()可以查看当前活动所在栈的id。

OneActivity(standard)🡪TwoActivity(singleInstance)🡪ThreeActivity(standard)，依次点击返回键，会顺序显示one🡪two，而不是two🡪one🡪退出。

启动模式也可通过代码动态设置。(intent.setFlags();)

Intent.FLAG\_ACTIVITY\_NEW\_TASK-------------singleInstance

Intent.FLAG\_ACTIVITY\_SINGLE\_TOP-----------singleTop

Intent.FLAG\_ACTIVITY\_CLEAR\_TOP------------singleTask

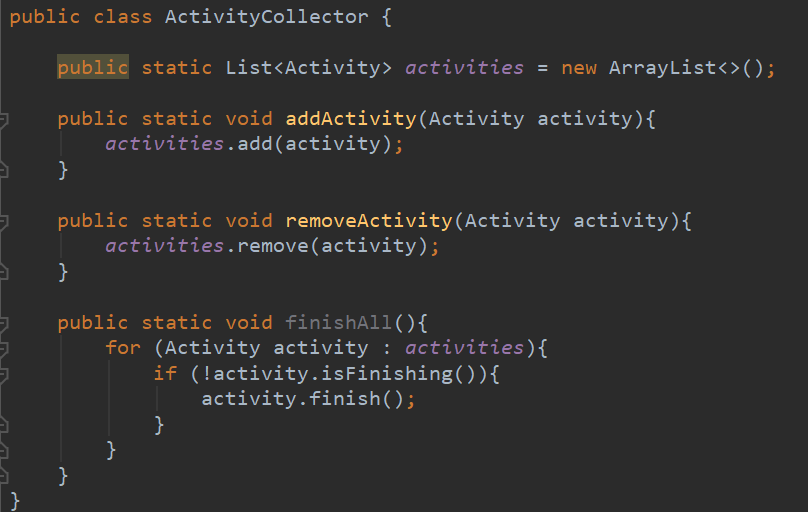
关于启动模式：

1. 一般不要改变activity和task的默认关系，即使用standard就好。
2. 如果有必要修改，要保持谨慎，并确保activity在启动和从其他activity返回时的可用性，多做测试和安全方面的工作。

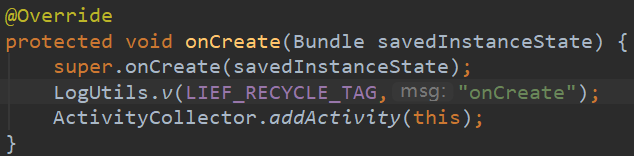
启动模式，现在用的不是很多，关于应用场景和一些可能存在的细节还没很好的理解。还是要真正运用到，理解才会深刻呀。

## 六、Activity的集中管理

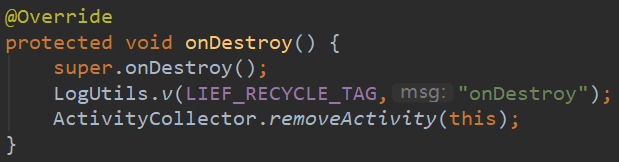
使用List来暂存项目中所有的活动。



在基类的onCreate()中，添加活动。



在基类的onDestory()中移除活动。



这么做有什么好处呢？

1. 要退出程序时，无论在什么地方，都可以快速退出。如果要彻底退出程序，可以在finishAll()后面添加：



killProcess()只能杀掉当前程序的进程。

1. 可以方便查看当前程序存在哪些活动。程序优化时，可以帮助我们思考是否需要把某个活动提前销毁、或者改变活动的启动模式。再者，可以结合Profiler做性能优化。