### 源码分析相关面试题

* [Volley源码分析](http://www.jianshu.com/p/ec3dc92df581)
* [注解框架实现原理](http://www.jianshu.com/p/20da6d6389e1)
* [okhttp3.0源码分析](http://www.jianshu.com/p/9ed2c2f2a52c)
* [onSaveInstanceState源码分析](http://www.jianshu.com/p/cbf9c3557d64)
* [静默安装和源码编译](http://www.jianshu.com/p/2211a5b3c37f)

### Activity相关面试题

* [保存Activity的状态](http://www.jianshu.com/p/cbf9c3557d64)
* [深刻剖析activity启动模式(一)](http://www.jianshu.com/p/b33fd8c550bf)
* [深刻剖析activity启动模式(二)](http://www.jianshu.com/p/e1ea9e542112)

### 与XMPP相关面试题

* [XMPP协议优缺点](http://www.jianshu.com/p/2c04ac3c526a)
* [极光消息推送原理](http://www.jianshu.com/p/d88dc66908cf)

### 与性能优化相关面试题

* [内存泄漏和内存溢出区别](http://www.jianshu.com/p/5dd645b05c76)
* [UI优化和线程池实现原理](http://www.jianshu.com/p/c22398f8587f)
* [代码优化](http://www.jianshu.com/p/ebd41eab90df)
* [内存性能分析](http://www.jianshu.com/p/2665c31b9c2f)
* [内存泄漏检测](http://www.jianshu.com/p/1514c7804a06)
* [App启动优化](http://www.jianshu.com/p/f0f73fefdd43)
* [与IPC机制相关面试题](http://www.jianshu.com/p/de4793a4c2d0)

### 与登录相关面试题

* [oauth认证协议原理](http://www.jianshu.com/p/2a6ecbf8d49d)
* [token产生的意义](http://www.jianshu.com/p/9b7ce2d6c195)
* [微信扫一扫实现原理](http://www.jianshu.com/p/a9d1f21bd5e0)

### 与开发相关面试题

* [迭代开发的时候如何向前兼容新旧接口](http://www.jianshu.com/p/cbecadec98de)
* [手把手教你如何解决as jar包冲突](http://www.jianshu.com/p/30fdc391289c)
* [context的原理分析](http://www.jianshu.com/p/2706c13a1769)
* [解决ViewPager.setCurrentItem中间很多页面切换方案](http://www.jianshu.com/p/38ab6d856b56)

### 与人事相关面试题

* [人事面试宝典](http://www.jianshu.com/p/d61b553ff8c9)

### **Activity的四种启动模式如下：**

**standard、singleTop、singleTask、singleInstance**

我们一边讲理论一边结合案例来全面学习这四种启动模式。 为了打印方便，定义一个基础BaseActivity，在其onCreate方法和onNewIntent方法中打印出当前Activity的日志信息，主要包括所属的task，当前类的hashcode，之后我们进行测试的Activity都直接继承该BaseActivity

public class BaseActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 Log.i("maweiqi", "\*\*\*\*\*onCreate()方法\*\*\*\*\*\*");  
 Log.i("maweiqi", "onCreate：" + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());  
 dumpTaskAffinity();  
 }  
  
 @Override  
 protected void onNewIntent(Intent intent) {  
 super.onNewIntent(intent);  
 Log.i("maweiqi", "\*\*\*\*\*onNewIntent()方法\*\*\*\*\*");  
 Log.i("maweiqi", "onNewIntent：" + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());  
 dumpTaskAffinity();  
 }  
  
 protected void dumpTaskAffinity(){  
 try {  
 ActivityInfo info = this.getPackageManager()  
 .getActivityInfo(getComponentName(), PackageManager.GET\_META\_DATA);  
 Log.i("maweiqi", "taskAffinity:"+info.taskAffinity);  
 } catch (PackageManager.NameNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

### **standard-默认模式**

这个模式是默认的启动模式，即标准模式，在不指定启动模式的前提下，系统默认使用该模式启动Activity，每次启动一个Activity都会重写创建一个新的实例，不管这个实例存不存在，这种模式下，谁启动了该模式的Activity，该Activity就属于启动它的Activity的任务栈中。

#### **配置形式：**

<activity android:name=".SecondActivity" android:launchMode="standard"/>

#### **使用案例：**

对于standard模式，android:launchMode可以不进行声明，因为默认就是standard。 SecondActivity的代码如下，入口MainActivity中有一个按钮进入该Activity，这个Activity中又有一个按钮启动SecondActivity。

public class MainActivity extends BaseActivity {  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 Button btn\_standard= (Button) findViewById(R.id.btn\_standard);  
 btn\_standard.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(MainActivity.this, SecondActivity.class);  
 startActivity(intent);  
 }  
 });  
 Log.i("maweiqi", "\*\*\*\*\*onCreate()方法\*\*\*\*\*\*");  
 Log.i("maweiqi", "onCreate：" + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());  
 }  
}

public class SecondActivity extends BaseActivity {  
 private Button jump;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_second);  
  
 jump= (Button) findViewById(R.id.button3);  
 jump.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SecondActivity.this, SecondActivity.class);  
 startActivity(intent);  
 }  
 });  
 }  
  
}

#### **测试方式：**

MainActivity --> SecondActivity --> SecondActivity --> SecondActivity --> SecondActivity

#### **输出的日志如下：**

MainActivity TaskId: 2 hasCode:1249333352  
SecondActivity TaskId: 2 hasCode:1249526392  
SecondActivity TaskId: 2 hasCode:1249424816  
SecondActivity TaskId: 2 hasCode:1249439692  
SecondActivity TaskId: 2 hasCode:1249459968

#### **测试结果：**

1）日志输出了四次SecondActivity的和一次MainActivity的，并且所属的任务栈的id都是2，这也验证了谁启动了该模式的Activity，该Activity就属于启动它的Activity的任务栈中这句话.因为启动SecondActivity的是MainActivity，而MainActivity的taskId是2，因此启动的SecondActivity也应该属于id为2的这个task，后续的3个SecondActivity是被SecondActivity这个对象启动的，因此也应该还是2，所以taskId都是2。

2）并且每一个Activity的hashcode都是不一样的，说明他们是不同的实例，即“每次启动一个Activity都会重写创建一个新的实例”

### **singleTop模式**

这个模式下，如果新的activity已经位于栈顶，那么这个Activity不会被重新创建，同时它的onNewIntent方法会被调用，通过此方法的参数我们可以去除当前请求的信息。如果栈顶不存在该Activity的实例，则情况与standard模式相同。需要注意的是这个Activity它的onCreate()，onStart()方法不会被调用，因为它并没有发生改变。

#### **配置形式：**

<activity android:name=".SingleTopActivity" android:launchMode="singleTop"/>  
<activity android:name=".OtherTopActivity" android:launchMode="singleTop"/>

#### **使用案例：**

public class SingleTopActivity extends BaseActivity {  
 private Button jump, jump2;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_singletop);  
  
 jump = (Button) findViewById(R.id.button);  
 jump2 = (Button) findViewById(R.id.button2);  
 jump.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SingleTopActivity.this, SingleTopActivity.class);  
 startActivity(intent);  
 }  
 });  
 jump2.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SingleTopActivity.this, OtherTopActivity.class);  
 startActivity(intent);  
 }  
 });  
 Log.i("maweiqi", "\*\*\*\*\*onCreate()方法\*\*\*\*\*\*");  
 Log.i("maweiqi", "onCreate：" + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());  
 }  
}

public class OtherTopActivity extends AppCompatActivity {  
 private Button jump;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_other);  
  
 jump= (Button) findViewById(R.id.btn\_other);  
 jump.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(OtherTopActivity.this, SingleTopActivity.class);  
 startActivity(intent);  
 }  
 });  
 Log.i("maweiqi", "\*\*\*\*\*onCreate()方法\*\*\*\*\*\*");  
 Log.i("maweiqi", "onCreate：" + getClass().getSimpleName() + " TaskId: " + getTaskId() + " hasCode:" + this.hashCode());  
 }  
}

#### **操作和standard模式类似，直接贴输出日志**

onCreate：MainActivity TaskId: 3 hasCode:1249332216  
onCreate：SingleTopActivity TaskId: 3 hasCode:1249464444  
onNewIntent：SingleTopActivity TaskId: 3 hasCode:1249464444  
onNewIntent：SingleTopActivity TaskId: 3 hasCode:1249464444  
onNewIntent：SingleTopActivity TaskId: 3 hasCode:1249464444

#### **测试结果：**

1）除了第一次进入SingleTopActivity这个Activity时，输出的是onCreate方法中的日志，后续的都是调用了onNewIntent方法，并没有调用onCreate方法，并且四个日志的hashcode都是一样的。

2）说明栈中只有一个实例。这是因为第一次进入的时候，栈中没有该实例，则创建，后续的三次发现栈顶有这个实例，则直接复用，并且调用onNewIntent方法。

3）那么假设栈中有该实例，但是该实例不在栈顶情况又如何呢？ 我们先从MainActivity中进入到SingleTopActivity，然后再跳转到OtherActivity中，再从OtherActivity中跳回SingleTopActivity，再从SingleTopActivity跳到SingleTopActivity中，看看整个过程的日志。

#### **输出的日志如下：**

onCreate：SingleTopActivity TaskId: 4 hasCode:1249520904  
onCreate：OtherTopActivity TaskId: 4 hasCode:1249420244  
onCreate：SingleTopActivity TaskId: 4 hasCode:1249448776  
onCreate：SingleTopActivity TaskId: 4 hasCode:1249448776  
onNewIntent：SingleTopActivity TaskId: 4 hasCode:1249448776

#### **测试结果：**

1）从MainActivity进入到SingleTopActivity时，新建了一个SingleTopActivity对象。

2）从SingleTopActivity跳到OtherActivity时，新建了一个OtherActivity，此时task中存在三个Activity，从栈底到栈顶依次是MainActivity，SingleTopActivity，OtherActivity。

3）此时如果再跳到SingleTopActivity，即使栈中已经有SingleTopActivity实例了，但是依然会创建一个新的SingleTopActivity实例，这一点从上面的日志的hashCode可以看出，此时栈顶是SingleTopActivity，如果再跳到SingleTopActivity，就会复用栈顶的SingleTopActivity，即会调用SingleTopActivity的onNewIntent方法。这就是上述日志的全过程。

#### **对以上内容进行总结**

standard启动模式是默认的启动模式，每次启动一个Activity都会新建一个实例不管栈中是否已有该Activity的实例。

#### **singleTop模式分3种情况**

1）当前栈中已有该Activity的实例并且该实例位于栈顶时，不会新建实例，而是复用栈顶的实例，并且会将Intent对象传入，回调onNewIntent方法

2）当前栈中已有该Activity的实例但是该实例不在栈顶时，其行为和standard启动模式一样，依然会创建一个新的实例

3）当前栈中不存在该Activity的实例时，其行为同standard启动模式standard和singleTop启动模式都是在原任务栈中新建Activity实例，不会启动新的Task

### **singleTask模式**

在这个模式下，如果栈中存在这个Activity的实例就会复用这个Activity，不管它是否位于栈顶，复用时，会将它上面的Activity全部出栈，并且会回调该实例的onNewIntent方法。

#### **配置形式：**

<activity android:name=".SingleTaskActivity" android:launchMode="singleTask"/>  
<activity android:name=".OtherTaskActivity" android:launchMode="singleTask"/>

#### **使用案例：**

public class SingleTaskActivity extends BaseActivity {  
 private Button jump,jump2;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_task);  
 jump = (Button) findViewById(R.id.btn\_task);  
 jump2 = (Button) findViewById(R.id.btn\_other);  
 jump.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SingleTaskActivity.this, SingleTaskActivity.class);  
 startActivity(intent);  
 }  
 });  
 jump2.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SingleTaskActivity.this, OtherTaskActivity.class);  
 startActivity(intent);  
 }  
 });  
}}

public class OtherTaskActivity extends BaseActivity {  
 private Button jump;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_other\_task);  
  
 jump= (Button) findViewById(R.id.btn\_other);  
 jump.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(OtherTaskActivity.this, SingleTaskActivity.class);  
 startActivity(intent);  
 }  
});  
}  
}

#### **日志输出**

onCreate：MainActivity TaskId: 5 hasCode:1249321980  
onCreate：SingleTaskActivity TaskId: 5 hasCode:1249515136  
onCreate：OtherTaskActivity TaskId: 6 hasCode:1249386172  
onNewIntent：SingleTaskActivity TaskId: 6 hasCode:1249513244

#### **测试结果：**

1）从MainActiviyty进入到SingleTaskActivity，再进入到OtherActivity后，此时栈中有3个Activity实例，并且SingleTaskActivity不在栈顶。

2）而在OtherActivity跳到SingleTaskActivity时，并没有创建一个新的SingleTaskActivity，而是复用了该实例，并且回调了onNewIntent方法。并且原来的OtherActivity出栈了，具体见下面的信息，使用命令adb shell dumpsys activity activities可进行查看

Running activities (most recent first):  
 TaskRecord{3c727e #11 A=com.maweiqi.task U=0 sz=2}  
 Run #1: ActivityRecord{5a00d1e u0 com.maweiqi.task/.SingleTaskActivity t11}  
 Run #0: ActivityRecord{2dce0b u0 com.maweiqi.task/.MainActivity t11}

可以看到当前栈中只有两个Activity，即原来栈中位于SingleTaskActivity 之上的Activity都出栈了。

### **singleInstance-全局唯一模式**

该模式具备singleTask模式的所有特性外，与它的区别就是，这种模式下的Activity会单独占用一个Task栈，具有全局唯一性，即整个系统中就这么一个实例，由于栈内复用的特性，后续的请求均不会创建新的Activity实例，除非这个特殊的任务栈被销毁了。以singleInstance模式启动的Activity在整个系统中是单例的，如果在启动这样的Activiyt时，已经存在了一个实例，那么会把它所在的任务调度到前台，重用这个实例。

#### **singleInstance示例一配置形式：**

<activity android:name=".MainActivity" android:launchMode="standard">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
  
 <category android:name="android.intent.category.LAUNCHER"/>  
 </intent-filter>  
</activity>  
 <activity android:name=".SingleInstanceActivity" android:launchMode="singleInstance"/>

public class MainActivity extends BaseActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 Button btn\_standard= (Button) findViewById(R.id.btn\_standard);  
 btn\_standard.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(MainActivity.this, SingleInstanceActivity.class);  
 startActivity(intent);  
 }  
 });  
  
 }  
}

public class SingleInstanceActivity extends BaseActivity {  
 @Override  
 public void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_instance);  
 Button button4 = (Button) findViewById(R.id.button4);  
 button4.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(SingleInstanceActivity.this, MainActivity.class);  
 startActivity(intent);  
 }  
 });  
 }  
}

#### **测试方式：**

MainActivity--> SingleInstanceActivity--> MainActivity--> SingleInstanceActivity

#### **日志输出**

onCreate：MainActivity TaskId: 12 hasCode:201331476  
onCreate：SingleInstanceActivity TaskId: 13 hasCode:57987178  
onCreate：MainActivity TaskId: 12 hasCode:254253633  
onNewIntent：SingleInstanceActivity TaskId: 13 hasCode:57987178

#### **测试结果**

1）两个SingleInstanceActivity是同一个实例。

2） 第一次进入的MainActivity和第一次进入的SingleInstanceActivity位于不同的task中。

3） 两个MainActivity是位于同一个task中的不同实例。

4）这个结论与预期是相同的，即，singleInstance类型的Activity的实例只能有一个，而且它只允许存在于单独的一个task中。

5）至于为什么两个MainActivity是位于同一个task中的不同实例，那是因为它是standard类型的，我们可以将ActivityTest修改为singleTop等其他类型进行测试。

#### **singleInstance示例二配置形式：**

MainActivity的模式改为"singleTop",修改后的manifest如下：

<activity android:name=".MainActivity" android:launchMode="singleTop">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
  
 <category android:name="android.intent.category.LAUNCHER"/>  
 </intent-filter>  
</activity>  
 <activity android:name=".SingleInstanceActivity" android:launchMode="singleInstance"/>

MainActivity进入SingleInstanceActivity,在从SingleInstanceActivity 进入MainActivity,在从MainActivity进入SingleInstanceActivity

#### **日志输出**

onCreate：MainActivity TaskId: 15 hasCode:201331476  
onCreate：SingleInstanceActivity TaskId: 16 hasCode:57987178  
onNewIntent：MainActivity TaskId: 15 hasCode:201331476  
onNewIntent：SingleInstanceActivity TaskId: 16 hasCode:57987178

#### **测试结果**

1）两个SingleInstanceActivity是同一个实例。

2）第一次进入的MainActivity和第一次进入的SingleInstanceActivity位于不同的task中。

3）两个MainActivity是同一个实例。

### **launchMode模式总结**

#### **1. standard**

在该模式下，Activity可以拥有多个实例，并且这些实例既可以位于同一个task，也可以位于不同的task。

#### **2.singleTop**

该模式下，在同一个task中，如果存在该Activity的实例，并且该Activity实例位于栈顶(即，该Activity位于前端)，则调用startActivity()时，不再创建该Activity的示例；而仅仅只是调用Activity的onNewIntent()。否则的话，则新建该Activity的实例，并将其置于栈顶。

#### **3. singleTask**

只容许有一个包含该Activity实例的task存在！ 总的来说：singleTask的结论与android:taskAffinity相关(下章在讲),以A启动B来说

1） 当A和B的taskAffinity相同时：第一次创建B的实例时，并不会启动新的task，而是直接将B添加到A所在的task；否则，将B所在task中位于B之上的全部Activity都删除，然后跳转到B中。 2） 当A和B的taskAffinity不同时：第一次创建B的实例时，会启动新的task，然后将B添加到新建的task中；否则，将B所在task中位于B之上的全部Activity都删除，然后跳转到B中。

#### **4. singleInstance**

顾名思义，是单一实例的意思，即任意时刻只允许存在唯一的Activity实例，而且该Activity所在的task不能容纳除该Activity之外的其他Activity实例！ 它与singleTask有相同之处，也有不同之处。 相同之处：任意时刻，最多只允许存在一个实例。 不同之处： 1） singleTask受android:taskAffinity属性的影响，而singleInstance不受android:taskAffinity的影响。 2） singleTask所在的task中能有其它的Activity，而singleInstance的task中不能有其他Activity。 3） 当跳转到singleTask类型的Activity，并且该Activity实例已经存在时，会删除该Activity所在task中位于该Activity之上的全部Activity实例；而跳转到singleInstance类型的Activity，并且该Activity已经存在时，不需要删除其他Activity，因为它所在的task只有该Activity唯一一个Activity实例。

参考链接：<http://blog.csdn.net/sbsujjbcy/article/details/49360615>

* 欢迎关注微信公众号,长期推荐技术文章和技术视频
* 微信公众号名称：Android干货程序员

img

img