[Activity的四种LaunchMode(写的真心不错，建议大家都看看)](http://blog.csdn.net/moreevan/article/details/6788048)

标签： [android](http://www.csdn.net/tag/android)[button](http://www.csdn.net/tag/button)

2011-09-18 21:35 10244人阅读 [评论](http://blog.csdn.net/moreevan/article/details/6788048#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/moreevan/article/details/6788048#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

android（12） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

合抱之木，生於毫末；九層之台，起於累土；千里之行，始於足下。《老子》

今天在社区看到有朋友问“如何在半年内成为顶级[**架构**](http://lib.csdn.net/base/16)师”，有网友道“关灯睡觉，不用半年的...”,的确，做梦还来的快一些。作为一个程序员，树立远大的目标是值得欣赏的，但不能只去空想，要一步一步地实践才行。成大事者，须从小事做起；万事起于忽微，量变引起质变。

我们今天要讲的是Activity的四种launchMode。

launchMode在多个Activity跳转的过程中扮演着重要的角色，它可以决定是否生成新的Activity实例，是否重用已存在的Activity实例，是否和其他Activity实例公用一个task里。这里简单介绍一下task的概念，task是一个具有栈结构的对象，一个task可以管理多个Activity，启动一个应用，也就创建一个与之对应的task。

Activity一共有以下四种launchMode：

1.standard

2.singleTop

3.singleTask

4.singleInstance

我们可以在AndroidManifest.xml配置<activity>的[**Android**](http://lib.csdn.net/base/15):launchMode属性为以上四种之一即可。

下面我们结合实例一一介绍这四种lanchMode：

1.standard

standard模式是默认的启动模式，不用为<activity>配置android:launchMode属性即可，当然也可以指定值为standard。

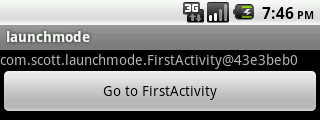
我们将会一个Activity，命名为FirstActivity，来演示一下标准的启动模式。FirstActivity代码如下：

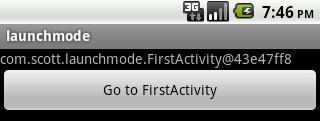
[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

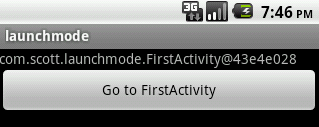
1. **package** com.scott.launchmode;
3. **import** android.app.Activity;
4. **import** android.content.Intent;
5. **import** android.os.Bundle;
6. **import** android.view.View;
7. **import** android.widget.Button;
8. **import** android.widget.TextView;
10. **public** **class** FirstActivity **extends** Activity {
11. @Override
12. **public** **void** onCreate(Bundle savedInstanceState) {
13. **super**.onCreate(savedInstanceState);
14. setContentView(R.layout.first);
15. TextView textView = (TextView) findViewById(R.id.textView);
16. textView.setText(**this**.toString());
17. Button button = (Button) findViewById(R.id.button);
18. button.setOnClickListener(**new** View.OnClickListener() {
19. @Override
20. **public** **void** onClick(View v) {
21. Intent intent = **new** Intent(FirstActivity.**this**, FirstActivity.**class**);
22. startActivity(intent);
23. }
24. });
25. }
26. }

我们FirstActivity界面中的TextView用于显示当前Activity实例的序列号，Button用于跳转到下一个FirstActivity界面。

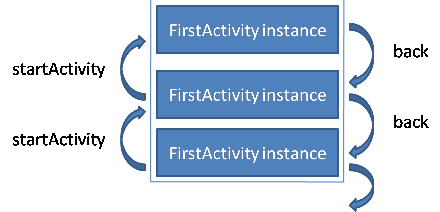
然后我们连续点击几次按钮，将会出现下面的现象：







我们注意到都是FirstActivity的实例，但序列号不同，并且我们需要连续按后退键两次，才能回到第一个FristActivity。standard模式的原理如下图所示：

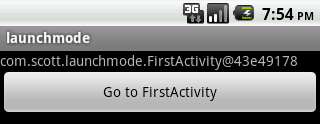


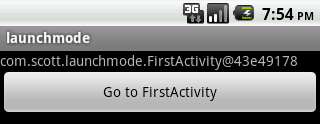
如图所示，每次跳转系统都会在task中生成一个新的FirstActivity实例，并且放于栈结构的顶部，当我们按下后退键时，才能看到原来的FirstActivity实例。

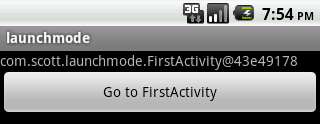
这就是standard启动模式，不管有没有已存在的实例，都生成新的实例。

2.singleTop

我们在上面的基础上为<activity>指定属性android:launchMode="singleTop"，系统就会按照singleTop启动模式处理跳转行为。我们重复上面几个动作，将会出现下面的现象：







我们看到这个结果跟standard有所不同，三个序列号是相同的，也就是说使用的都是同一个FirstActivity实例；如果按一下后退键，程序立即退出，说明当前栈结构中只有一个Activity实例。singleTop模式的原理如下图所示：



正如上图所示，跳转时系统会先在栈结构中寻找是否有一个FirstActivity实例正位于栈顶，如果有则不再生成新的，而是直接使用。也许朋友们会有疑问，我只看到栈内只有一个Activity，如果是多个Activity怎么办，如果不是在栈顶会如何？我们接下来再通过一个示例来证实一下大家的疑问。

我们再新建一个Activity命名为SecondActivity，如下：

[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

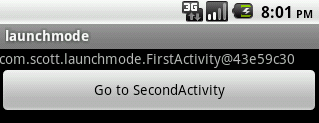
1. **package** com.scott.launchmode;
3. **import** android.app.Activity;
4. **import** android.content.Intent;
5. **import** android.os.Bundle;
6. **import** android.view.View;
7. **import** android.widget.Button;
8. **import** android.widget.TextView;
10. **public** **class** SecondActivity **extends** Activity {
11. @Override
12. **protected** **void** onCreate(Bundle savedInstanceState) {
13. **super**.onCreate(savedInstanceState);
14. setContentView(R.layout.second);
15. TextView textView = (TextView) findViewById(R.id.textView);
16. textView.setText(**this**.toString());
17. Button button = (Button) findViewById(R.id.button);
18. button.setOnClickListener(**new** View.OnClickListener() {
19. @Override
20. **public** **void** onClick(View v) {
21. Intent intent = **new** Intent(SecondActivity.**this**, FirstActivity.**class**);
22. startActivity(intent);
23. }
24. });
25. }
26. }

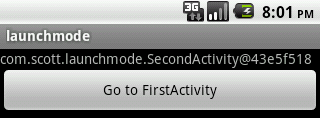
然后将之前的FirstActivity跳转代码改为：

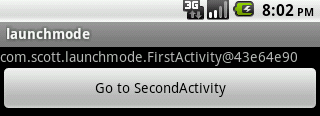
[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

1. Intent intent = **new** Intent(FirstActivity.**this**, SecondActivity.**class**);
2. startActivity(intent);

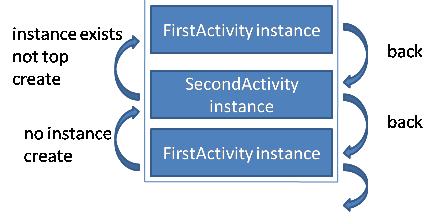
是的，FirstActivity会跳转到SecondActivity，SecondActivity又会跳转到FirstActivity。演示结果如下：







我们看到，两个FirstActivity的序列号是不同的，证明从SecondActivity跳转到FirstActivity时生成了新的FirstActivity实例。原理图如下：

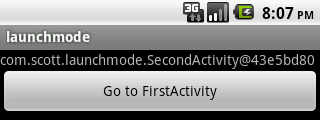
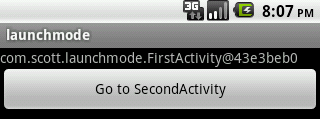


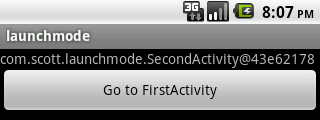
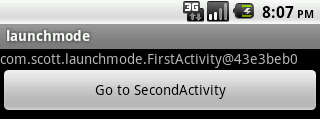
我们看到，当从SecondActivity跳转到FirstActivity时，系统发现存在有FirstActivity实例,但不是位于栈顶，于是重新生成一个实例。

这就是singleTop启动模式，如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例。

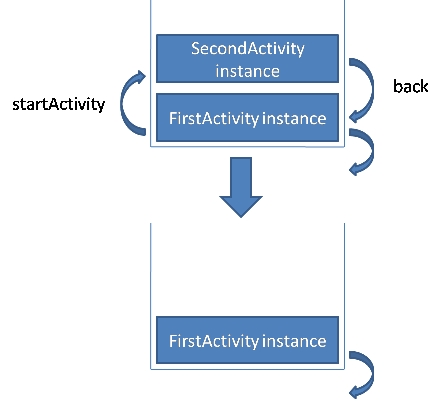
3.singleTask

在上面的基础上我们修改FirstActivity的属性android:launchMode="singleTask"。演示的结果如下：





我们注意到，在上面的过程中，FirstActivity的序列号是不变的，SecondActivity的序列号却不是唯一的，说明从SecondActivity跳转到FirstActivity时，没有生成新的实例，但是从FirstActivity跳转到SecondActivity时生成了新的实例。singleTask模式的原理图如下图所示：



在图中的下半部分是SecondActivity跳转到FirstActivity后的栈结构变化的结果，我们注意到，SecondActivity消失了，没错，在这个跳转过程中系统发现有存在的FirstActivity实例，于是不再生成新的实例，而是将FirstActivity之上的Activity实例统统出栈，将FirstActivity变为栈顶对象，显示到幕前。也许朋友们有疑问，如果将SecondActivity也设置为singleTask模式，那么SecondActivity实例是不是可以唯一呢？在我们这个示例中是不可能的，因为每次从SecondActivity跳转到FirstActivity时，SecondActivity实例都被迫出栈，下次等FirstActivity跳转到SecondActivity时，找不到存在的SecondActivity实例，于是必须生成新的实例。但是如果我们有ThirdActivity，让SecondActivity和ThirdActivity互相跳转，那么SecondActivity实例就可以保证唯一。

这就是singleTask模式，如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前。

4.singleInstance

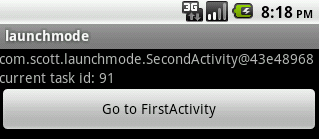
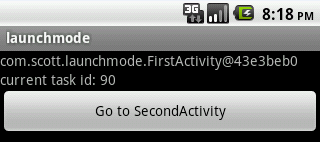
这种启动模式比较特殊，因为它会启用一个新的栈结构，将Acitvity放置于这个新的栈结构中，并保证不再有其他Activity实例进入。

我们修改FirstActivity的launchMode="standard"，SecondActivity的launchMode="singleInstance"，由于涉及到了多个栈结构，我们需要在每个Activity中显示当前栈结构的id，所以我们为每个Activity添加如下代码：

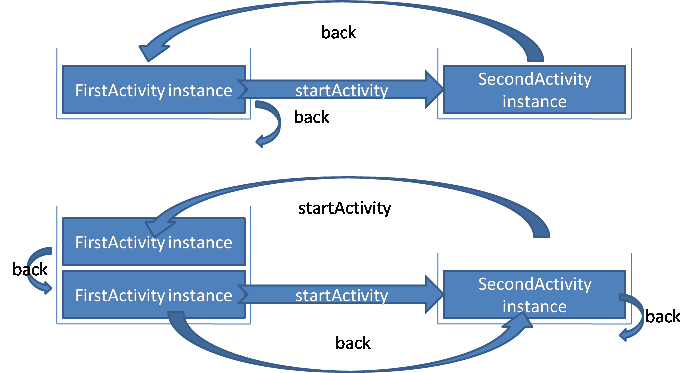
[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

1. TextView taskIdView = (TextView) findViewById(R.id.taskIdView);
2. taskIdView.setText("current task id: " + **this**.getTaskId());

然后我们再演示一下这个流程：

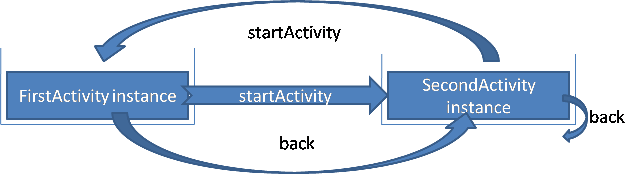


我们发现这两个Activity实例分别被放置在不同的栈结构中，关于singleInstance的原理图如下：



我们看到从FirstActivity跳转到SecondActivity时，重新启用了一个新的栈结构，来放置SecondActivity实例，然后按下后退键，再次回到原始栈结构；图中下半部分显示的在SecondActivity中再次跳转到FirstActivity，这个时候系统会在原始栈结构中生成一个FirstActivity实例，然后回退两次，注意，并没有退出，而是回到了SecondActivity，为什么呢？是因为从SecondActivity跳转到FirstActivity的时候，我们的起点变成了SecondActivity实例所在的栈结构，这样一来，我们需要“回归”到这个栈结构。

如果我们修改FirstActivity的launchMode值为singleTop、singleTask、singleInstance中的任意一个，流程将会如图所示：



singleInstance启动模式可能是最复杂的一种模式，为了帮助大家理解，我举一个例子，假如我们有一个share应用，其中的ShareActivity是入口Activity，也是可供其他应用调用的Activity，我们把这个Activity的启动模式设置为singleInstance，然后在其他应用中调用。我们编辑ShareActivity的配置：

[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

1. **<activity** android:name=".ShareActivity" android:launchMode="singleInstance"**>**
2. **<intent-filter>**
3. **<action** android:name="android.intent.action.MAIN" **/>**
4. **<category** android:name="android.intent.category.LAUNCHER" **/>**
5. **</intent-filter>**
6. **<intent-filter>**
7. **<action** android:name="android.intent.action.SINGLE\_INSTANCE\_SHARE" **/>**
8. **<category** android:name="android.intent.category.DEFAULT" **/>**
9. **</intent-filter>**
10. **</activity>**

然后我们在其他应用中这样启动该Activity：

[view plain](http://blog.csdn.net/liuhe688/article/details/6754323)

1. Intent intent = **new** Intent("android.intent.action.SINGLE\_INSTANCE\_SHARE");
2. startActivity(intent);

当我们打开ShareActivity后再按后退键回到原来界面时，ShareActivity做为一个独立的个体存在，如果这时我们打开share应用，无需创建新的ShareActivity实例即可看到结果，因为系统会自动查找，存在则直接利用。大家可以在ShareActivity中打印一下taskId，看看效果。关于这个过程，原理图如下：

