彻底弄懂activity 的四大模式

最近有几位朋友给我留言，让我谈一下对Activity启动模式的理解。我觉得对某个知识点的理解必须要动手操作才能印象深刻，所以今天写一篇博文，结合案例理解Activity启动模式。由于之前看过“区长”的一篇博文（文章结尾处有链接）深受启发，因此本文是在那篇文章的基础上更加全面的讲解。众所周知当我们**多次启动同一个Activity时，系统会创建多个实例，并把它们按照先进后出的原则一一放入任务栈中，当我们按back键时，就会有一个activity从任务栈顶移除，重复下去，直到任务栈为空，系统就会回收这个任务栈。**但是这样以来，系统**多次启动同一个Activity时就会重复创建多个实例，这种做法显然不合理，为了能够优化这个问题，Android提供四种启动模式来修改系统这一默认行为。** 进入正题，Activity的四种启动模式如下： **standard**、**singleTop**、**singleTask**、**singleInstance** 接下来，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Activity，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，以及taskAffinity的值。之后我们进行测试的Activity都直接继承该Activity

```
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

/**
 * Created by huangshuai on 2016/5/23.
 * Email：huangshuai@wooyun.org
 * 方便打印的基础Activity
 */
public class BaseActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        Log.i("WooYun", "*****onCreate()方法******");
        Log.i("WooYun", "onCreate：" + getClass().getSimpleName() + "
TaskId: " + getTaskId() + " hasCode:" + this.hashCode());
        dumpTaskAffinity();
    }

@Override
protected void onNewIntent(Intent intent) {
super.onNewIntent(intent);
        Log.i("WooYun", "*****onNewIntent()方法*****");
        Log.i("WooYun", "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("WooYun", "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。**这个Activity它的onCreate()，onStart()，onResume()方法都会被调用。 **配置形式：**

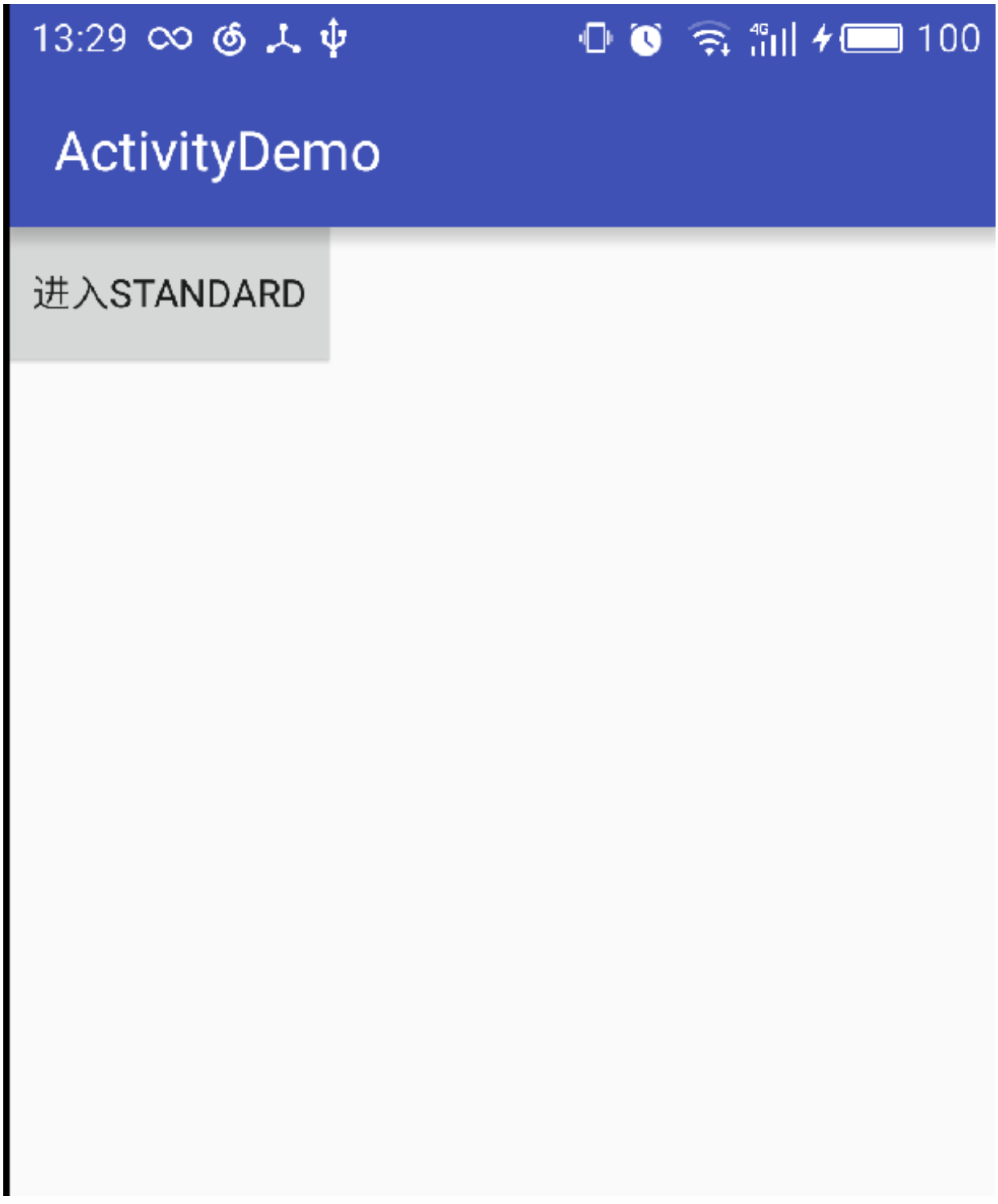
```
<activity android:name=".standard.StandardActivity"
android:launchMode="standard" > 1
```

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。StandardActivity 的代码如下，入口Activity中有一个按钮进入该Activity，这个Activity中又有一个按钮启动StandardActivity。

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

/**
 * Created by huangshuai on 2016/5/23.
 * Email：huangshuai@wooyun.org
 * Standard模式
*/
public class ActivityStandard extends BaseActivity  {
private Buttonjump;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_standard);

jump= (Button) findViewById(R.id.btn_standard);
```

```
jump.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent intent = new Intent(ActivityStandard.this,
ActivityStandard.class);
                startActivity(intent);
            }
        });
    }
}
```

我们首先进入StandardActivity，进入后再点击进入Standard的按钮，再按四次返回键不断返回。



猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai

输出的日志如下：

```
I/WooYun: **********************************************onCreate()方法**********
I/WooYun: onCreate: ActivityStandard TaskId: 2087 hasCode:313797123
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: **********************************************onCreate()方法**********
I/WooYun: onCreate: ActivityStandard TaskId: 2087 hasCode:205006461
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: **********************************************onCreate()方法**********
I/WooYun: onCreate: ActivityStandard TaskId: 2087 hasCode:200916357
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: **********************************************onCreate()方法**********
I/WooYun: onCreate: ActivityStandard TaskId: 2087 hasCode:434539703
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: **********************************************onCreate()方法**********
I/WooYun: onCreate: ActivityStandard TaskId: 2087 hasCode:509835225
I/WooYun: taskAffinity:com.castiel.demo
```

猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai

可以看到日志输出了四次StandardActivity的和一次MainActivity的，从MainActivity进入StandardActivity一次，后来我们又按了三次按钮，总共四次StandardActivity的日志，并且所属的任务栈的id都是2087，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StandardActivity的是MainActivity，而MainActivity的taskId是2087，因此启动的StandardActivity也应该属于id为2087的这个task，后续的3个StandardActivity是被StandardActivity这个对象启动的，因此也应该还是2087，所以taskId都是2087。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-栈顶复用模式

这个模式下，如果新的activity已经位于栈顶，那么这个Activity不会被重写创建，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=".singletop.SingleTopActivity"
android:launchMode="singleTop">1
```

使用案例： ActivitySingleTop.java

```
/**
 * Created by huangshuai on 2016/5/23.
 * Email：huangshuai@wooyun.org
 * SingleTop模式
*/
public class ActivitySingleTop 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.btn_singletop);
jump2 = (Button) findViewById(R.id.btn_other);
jump.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent intent = new Intent(ActivitySingleTop.this,
ActivitySingleTop.class);
                startActivity(intent);
            }
        });
jump2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent intent = new Intent(ActivitySingleTop.this,
OtherTopActivity.class);
                startActivity(intent);
            }
        });
    }
```

OtherTopActivity.java

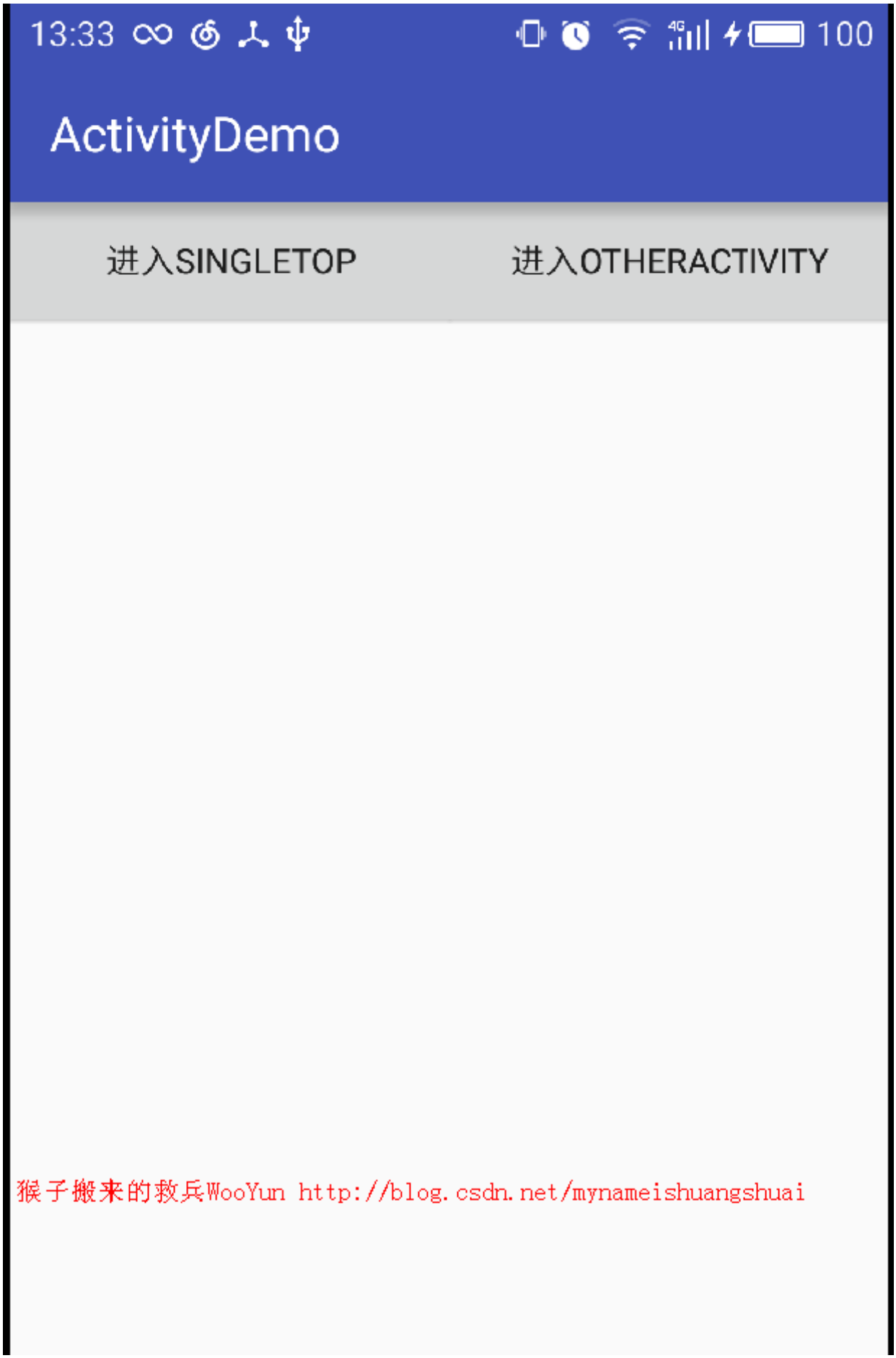
```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;

/**
 * Created by huangshuai on 2016/5/23.
 * Email: huangshuai@wooyun.org
 */
public class OtherTopActivity extends BaseActivity {
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,
ActivitySingleTop.class);
                startActivity(intent);
            }
        });
    }
}1234567891011121314151617181920212223242526
```

操作和standard模式类似，直接贴输出日志

```
I/WooYun: ****************************************onCreate()方法***************
I/WooYun: onCreate: ActivitySingleTop TaskId: 2145 hasCode:897313470
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ****************************************onNewIntent()方法************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2145 hasCode:897313470
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ****************************************onNewIntent()方法************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2145 hasCode:897313470
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ****************************************onNewIntent()方法************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2145 hasCode:897313470
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ****************************************onNewIntent()方法************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2145 hasCode:897313470
I/WooYun: taskAffinity:com.castiel.demo.singletop
```

我们看到，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，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。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。


13:33
100
ActivityDemo
进入SINGLETOP
进入OTHERACTIVITY
猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai


13:33
100
ActivityDemo
返回SINGLETOP
猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai

```
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: MainAcivity TaskId: 2147 hasCode:205006461
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: ActivitySingleTop TaskId: 2147 hasCode:757016486
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: OtherTopActivity TaskId: 2147 hasCode:939397543
I/WooYun: taskAffinity:com.castiel.demo.standard
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: ActivitySingleTop TaskId: 2147 hasCode:406114159
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2147 hasCode:406114159
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2147 hasCode:406114159
I/WooYun: taskAffinity:com.castiel.demo.singletop
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTop TaskId: 2147 hasCode:406114159
I/WooYun: taskAffinity:com.castiel.demo.singletop
```


我们看到从MainActivity进入到SingleTopActivity时，新建了一个SingleTopActivity对象，并且task id与MainActivity是一样的，然后从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，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，即使你指定了taskAffinity属性。 那么什么是**taskAffinity属性呢，可以简单的理解为任务相关性。**

- 这个参数标识了一个Activity所需任务栈的名字，默认情况下，所有Activity所需的任务栈的名字为应用的包名
- 我们可以单独指定每一个Activity的taskAffinity属性覆盖默认值
- 一个任务的affinity决定于这个任务的根activity（root activity）的taskAffinity
- 在概念上，具有相同的affinity的activity（即设置了相同taskAffinity属性的activity）属于同一个任务
- 为一个activity的taskAffinity设置一个空字符串，表明这个activity不属于任何task

很重要的一点taskAffinity属性不对standard和singleTop模式有任何影响，即时你指定了该属性为其他不同的值，这两种启动模式下不会创建新的task（如果不指定即默认值，即包名） 指定方式如下：

```
<activity android:name=".ActivitySingleTop" android:launchMode="singleTop"
android:taskAffinity="com.castiel.demo.singletop"/>1
```

```
<activity android:name=".ActivityStandard" android:launchMode="standard"
android:taskAffinity="com.castiel.demo.standard"/>1
```

singleTask-栈内复用模式

这个模式十分复杂，有各式各样的组合。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。其实这个过程还存在一个任务栈的匹配，因为这个模式启动时，会在自己需要的任务栈中寻找实例，这个任务栈就是通过taskAffinity属性指定。如果这个任务栈不存在，则会创建这个任务栈。 配置形式：**

```
<activity android:name=".singleTask.SingleTaskActivity"
android:launchMode="singleTask" >1
```

使用案例： ActivitySingleTask.java

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

/**
 * Created by huangshuai on 2016/5/23.
 * Email：huangshuai@wooyun.org
 * SingleTask模式
*/
public class ActivitySingleTask 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(ActivitySingleTask.this,
ActivitySingleTask.class);
startActivity(intent);
            }
```

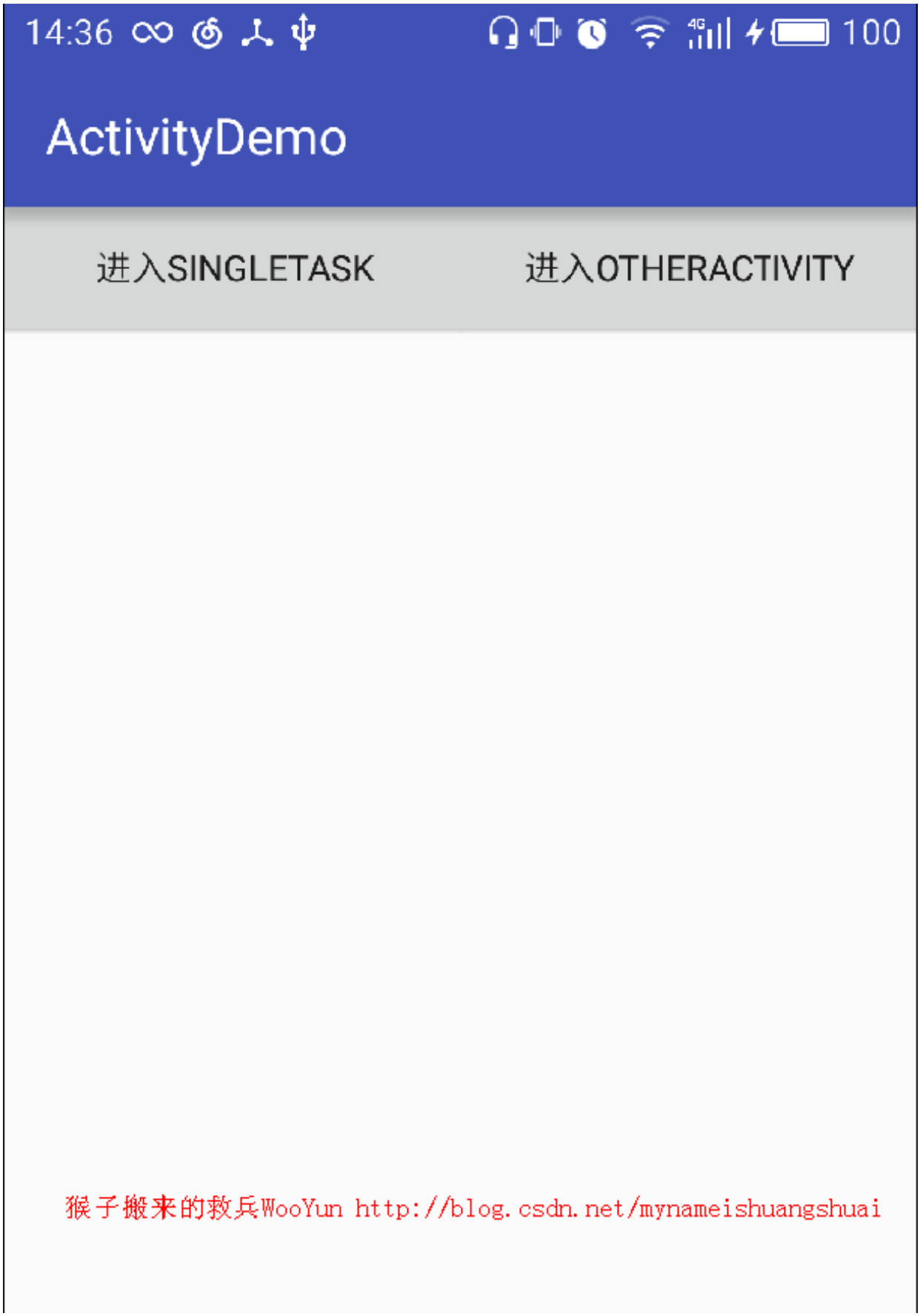
```
        });
jump2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent intent = new Intent(ActivitySingleTask.this,
OtherTaskActivity.class);
startActivity(intent);
            }
        });
    }
}12345678910111213141516171819202122232425262728293031323334351
```

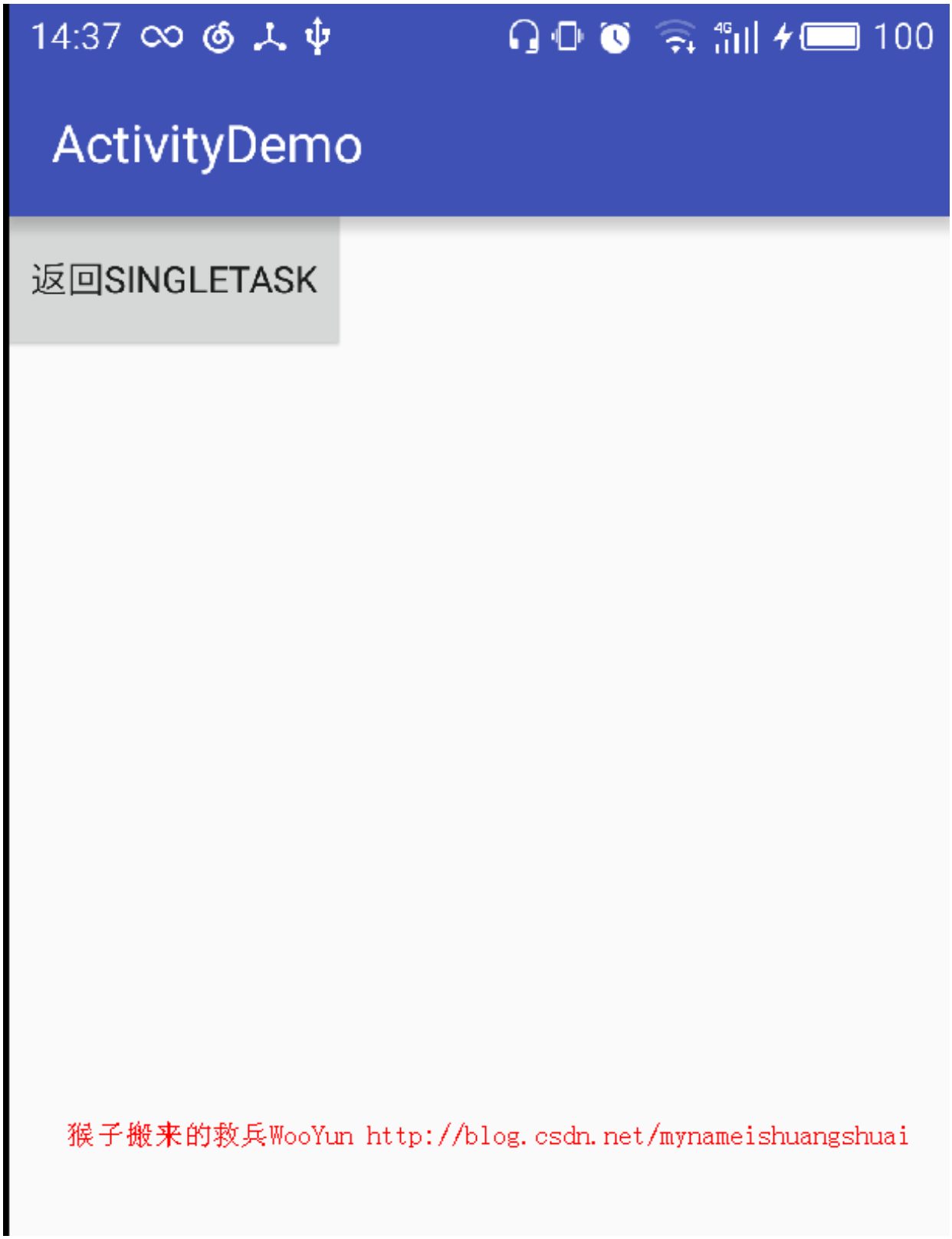
OtherTaskActivity.java

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

/**
 * Created by huangshuai on 2016/5/23.
 * Email：huangshuai@wooyun.org
 */
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,
ActivitySingleTask.class);
                startActivity(intent);
            }
        });
    }
}123456789101112131415161718192021222324252
```

现在我们先不指定任何taskAffinity属性，对它做类似singleTop的操作，即从入口MainActivity进入SingleTaskActivity，然后跳到OtherActivity，再跳回到SingleTaskActivity。看看整个过程的日志。


14:35
100
ActivityDemo
进入SINGLETASK
猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai


14:36
100
ActivityDemo
进入SINGLETASK
进入OTHERACTIVITY
猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai


14:37
100
ActivityDemo
返回SINGLETASK
猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai

```
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: MainAcivity TaskId: 2152 hasCode:313797123
I/WooYun:
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun:
I/WooYun: ********************************************onCreate()方法**********************
I/WooYun: onCreate: OtherTaskActivity TaskId: 2152 hasCode:830602041
I/WooYun:
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun:
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun:
I/WooYun: ********************************************onNewIntent()方法******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun:


                猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai
```

当我们从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，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{39cb8529 #2156 A=com.castiel.demo U=0 sz=2}
    Run #8: ActivityRecord{21675d81 u0 com.castiel.demo/.ActivitySingleTask t2156}
    Run #7: ActivityRecord{244472e7 u0 com.castiel.demo/.MainAcivity t2156}
```

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。 我们看到使用singleTask启动模式启动一个Activity，它还是在原来的task中启动。其实是这样的，我们并没有指定taskAffinity属性，这说明和默认值一样，也就是包名，当MainActivity启动时创建的Task的名字就是包名，因为MainActivity也没有指定taskAffinity，而当我们启动SingleTaskActivity ，首先会寻找需要的任务栈是否存在，也就是taskAffinity指定的值，这里就是包名，发现存在，就不再创建新的task，而是直接使用。当该task中存在该Activity实例时就会复用该实例，这就是栈内复用模式。这时候，如果我们指定SingleTaskActivity 的taskAffinity值。

```
<activity android:name=".ActivitySingleTask"
android:launchMode="singleTask"
android:taskAffinity="com.castiel.demo.singletask"/>1
```

还是之前的操作。但是日志就会变得不一样。

```
'? I/WooYun: *********************************************onCreate()方法*************************************
'? I/WooYun: onCreate: MainAcivity TaskId: 2153 hasCode:642251442
'? I/WooYun: taskAffinity:com.castiel.demo
'com.castiel.demo I/WooYun: *********************************************onCreate()方法*********************
'com.castiel.demo I/WooYun: onCreate: ActivitySingleTask TaskId: 2154 hasCode:205006461
'com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo.singletask
'com.castiel.demo I/WooYun: *********************************************onCreate()方法*********************
'com.castiel.demo I/WooYun: onCreate: OtherTaskActivity TaskId: 2154 hasCode:546182912
'com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo.standard
'com.castiel.demo I/WooYun: *********************************************onNewIntent()方法******************
'com.castiel.demo I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2154 hasCode:205006461
'com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo.singletask
'com.castiel.demo I/WooYun: *********************************************onNewIntent()方法******************
'com.castiel.demo I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2154 hasCode:205006461
'com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo.singletask
'com.castiel.demo I/WooYun: *********************************************onNewIntent()方法******************
'com.castiel.demo I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2154 hasCode:205006461
'com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo.singletask
```


我们看到SingleTaskActivity所属的任务栈的TaskId发生了变换，也就是说开启了一个新的Task，并且之后的OtherActivity也运行在了该Task上 打印出信息也证明了存在两个不同的Task

```
Running activities (most recent first):
  TaskRecord{16afaf3 #2154 A=com.castiel.demo.singletask U=0 sz=1}
    Run #8: ActivityRecord{136d7f23 u0 com.castiel.demo/.ActivitySingleTask t2154}
  TaskRecord{37793229 #2153 A=com.castiel.demo U=0 sz=1}
    Run #7: ActivityRecord{39bd9b1a u0 com.castiel.demo/.MainAcivity t2153}
```

如果我们指定MainActivity的taskAffinity属性和SingleTaskActivity一样，又会出现什么情况呢。

```
I/WooYun: *********************************************onCreate()方法**********************
I/WooYun: onCreate: MainAcivity TaskId: 2152 hasCode:313797123
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: *********************************************onCreate()方法**********************
I/WooYun: onCreate: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: *********************************************onCreate()方法**********************
I/WooYun: onCreate: OtherTaskActivity TaskId: 2152 hasCode:830602041
I/WooYun: taskAffinity:com.castiel.demo.standard
I/WooYun: *********************************************onNewIntent()方法*******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: *********************************************onNewIntent()方法*******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: *********************************************onNewIntent()方法*******************
I/WooYun: onNewIntent: ActivitySingleTask TaskId: 2152 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
```


没错，就是和他们什么都不指定是一样的。 这时候，就有了下面的结论 singleTask启动模式启动Activity时，首先会根据taskAffinity去寻找当前是否存在一个对应名字的任务栈

- 如果不存在，则会创建一个新的Task，并创建新的Activity实例入栈到新创建的Task中去

- 如果存在，则得到该任务栈，查找该任务栈中是否存在该Activity实例
  如果存在实例，则将它上面的Activity实例都出栈，然后回调启动的Activity实例的onNewIntent方法
  如果不存在该实例，则新建Activity，并入栈
  此外，我们可以将两个不同App中的Activity设置为相同的taskAffinity，这样虽然在不同的应用中，但是Activity会被分配到同一个Task中去。
  我们再创建另外一个应用，指定它的taskAffinity和之前的一样，都是com.xingyu.demo.singletask

```
<activity android:name=".MainActivity" android:launchMode="singleTask"
android:taskAffinity="com.castiel.demo.singletask"/>1
```

然后启动一个应用，让他跳转到该Activity后，再按home键后台，启动另一个应用再进入该Activity，看日志

```
I/WooYun: ********************************************onCreate()方法***************
I/WooYun: onCreate: MainAcivity TaskId: 2195 hasCode:642251442
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: ********************************************onCreate()方法***************
I/WooYun: onCreate: ActivitySingleInstance TaskId: 2196 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
I/WooYun: ********************************************onCreate2()方法***************
I/WooYun: onCreate: MainActivity TaskId: 2197 hasCode:642251442
I/WooYun: taskAffinity:com.xingyu.demo
I/WooYun: ********************************************onNewIntent()方法*************
I/WooYun: onNewIntent: ActivitySingleInstance TaskId: 2196 hasCode:290498930
I/WooYun: taskAffinity:com.castiel.demo
```


我们看到，指定了相同的taskAffinity的SingleTaskActivity和OtherActivity被启动到了同一个task中，taskId都为2169。

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。 **配置形式：**

```
<activity android:name=".singleinstance.SingleInstanceActivity"
android:launchMode="singleInstance" >1
```

使用案例： 增加一个Activity如下： ActivitySingleInstance.java

```
import android.os.Bundle;

/**
 * Created by huangshuai on 2016/5/24.
 * Email：huangshuai@wooyun.org
 * SingleInstance模式
 */
public class ActivitySingleInstance extends BaseActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_singleinstance);
    }
}
配置属性如下：
<activity
    android:name=".ActivitySingleInstance"
    android:launchMode="singleInstance">

    <intent-filter>
        <action android:name="com.castiel.demo.singleinstance" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>12345678910111213141516171819202122232 4
```

使用下面的方式分别在两个应用中启动它

```
Intent intent = new Intent();
intent.setAction("com.castiel.demo.singleinstance");
startActivity(intent);123
```

做的操作和上一次是一样的，查看日志

```
05-24 15:40:20.215 22359-22359/com.castiel.demo I/WooYun: **********onCreate()方法**********
05-24 15:40:20.216 22359-22359/com.castiel.demo I/WooYun: onCreate: MainAcivity TaskId: 2195 hasCode:642251442
05-24 15:40:20.217 22359-22359/com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo
05-24 15:40:24.199 22359-22359/com.castiel.demo I/WooYun: **********onCreate()方法**********
05-24 15:40:24.200 22359-22359/com.castiel.demo I/WooYun: onCreate: ActivitySingleInstance TaskId: 2196 hasCode:290498930
05-24 15:40:24.201 22359-22359/com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo
05-24 15:40:29.842 22513-22513/com.xingyu.demo I/WooYun: **********onCreate2()方法**********
05-24 15:40:29.842 22513-22513/com.xingyu.demo I/WooYun: onCreate: MainActivity TaskId: 2197 hasCode:642251442
05-24 15:40:29.843 22513-22513/com.xingyu.demo I/WooYun: taskAffinity:com.xingyu.demo
05-24 15:40:40.560 22359-22359/com.castiel.demo I/WooYun: **********onNewIntent()方法**********
05-24 15:40:40.567 22359-22359/com.castiel.demo I/WooYun: onNewIntent: ActivitySingleInstance TaskId: 2196 hasCode:290498930
05-24 15:40:40.569 22359-22359/com.castiel.demo I/WooYun: taskAffinity:com.castiel.demo

                                猴子搬来的救兵WooYun http://blog.csdn.net/mynameishuangshuai
```

我们看到，第一个应用启动SingleInstanceActivity时，由于系统中不存在该实例，所以新建了一个Task，按home键后，使用另一个App进入该Activity，由于系统中已经存在了一个实例，不会再创建新的Task，直接复用该实例，并且回调onNewIntent方法。可以从他们的hashcode中可以看出这是同一个实例。因此我们可以理解为：SingleInstance模式启动的Activity在系统中具有全局唯一性。

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

原文链接： http://blog.csdn.net/mynameishuangshuai/article/details/51491074

在android应用开发中，打造良好的用户体验是非常重要的。而在用户体验中，界面的引导和跳转是值得深入研究的重要内容。在开发中，与界面跳转联系比较紧密的概念是Task（任务）和Back Stack（回退栈）。activity的启动模式会影响Task和Back Stack的状态，进而影响用户体验。除了启动模式之外，Intent类中定义的一些标志（以FLAG_ACTIVITY_开头）也会影响Task和Back Stack的状态。在这篇文章中主要对四种启动模式进行分析和验证，其中涉及到activity的一个重要属性taskAffinity和Intent中的标志之一FLAG_ACTIVITY_NEW_TASK。关于Intent中其他标志位的具体用法会在另一篇文章中介绍。

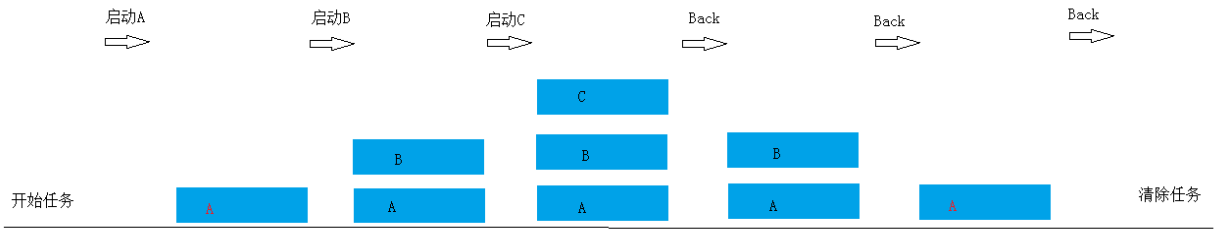
Task是一个存在于Framework层的概念，容易与它混淆的有Application（应用）和Process（进程）。在开始介绍Activity的启动模式的使用之前，首先对这些概念做一个简单的说明和区分。

一 Application，Task和Process的区别与联系

application翻译成中文时一般称为“应用”或“应用程序”，在android中，总体来说一个应用就是一组组件的集合。众所周知，android是在应用层组件化程度非常高的系统，android开发的第一课就是学习android的四大组件。当我们写完了多个组件，并且在manifest文件中注册了这些组件之后，把这些组件和组件使用到的资源打包成apk，我们就可以说完成了一个application。application和组件的关系可以在manifest文件中清晰地体现出来。如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1"
        android:versionName="1"
        xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.android.myapp">

    <application android:label="@string/app_name">
        <activity android:name=".MyActivity"
android:label="@string/app_nam">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    <receiver android:name=".MyReceiver"/>
    <provider android:name=".MyProvider"/>
    <service android:name=".MyService"/>
    </application>
</manifest>
```

而task是在程序运行时，只针对activity的概念。说白了，task是一组相互关联的activity的集合，它是存在于framework层的一个概念，控制界面的跳转和返回。这个task存在于一个称为back stack的数据结构中，也就是说，framework是以栈的形式管理用户开启的activity。这个栈的基本行为是，当用户在多个activity之间跳转时，执行压栈操作，当用户按返回键时，执行出栈操作。举例来说，如果应用程序中存在A,B,C三个activity，当用户在Launcher或Home Screen点击应用程序图标时，启动主Activity A，接着A开启B，B开启C，这时栈中有三个Activity，并且这三个Activity默认在同一个任务（task）中，当用户按返回时，弹出C，栈中只剩A和B，再按返回键，弹出B，栈中只剩A，再继续按返回键，弹出A，任务被移除。如下图所示：



task是可以跨应用的，这正是task存在的一个重要原因。有的Activity，虽然不在同一个app中，但为了保持用户操作的连贯性，把他们放在同一个任务中。例如，在我们的应用中的一个Activity A中点击发送邮件，会启动邮件程序的一个Activity B来发送邮件，这两个activity是存在于不同app中的，但是被系统放在一个任务中，这样当发送完邮件后，用户按back键返回，可以返回到原来的Activity A中，这样就确保了用户体验。

说完了application和task，最后介绍process。process一般翻译成进程，进程是操作系统内核中的一个概念，表示直接受内核调度的执行单位。在应用程序的角度看，我们用java编写的应用程序，运行在dalvik虚拟机中，可以认为一个运行中的dalvik虚拟机实例占有一个进程，所以，在默认情况下，一个应用程序的所有组件运行在同一个进程中。但是这种情况也有例外，即，应用程序中的不同组件可以运行在不同的进程中。只需要在manifest中用process属性指定组件所运行的进程的名字。如下所示：

[html] view plain copy

```
1. <activity android:name=".MyActivity" android:label="@string/app_nam"
2.     android:process=":remote">
3.
```

二 Activity四种启动模式详解

activity有四种启动模式，分别为standard，singleTop，singleTask，singleInstance。如果要使用这四种启动模式，必须在manifest文件中标签中的launchMode属性中配置，如：

[java] view plain copy

```
1. <activity android:name=".app.InterstitialMessageActivity"
2.         android:label="@string/interstitial_label"
3.         android:theme="@style/Theme.Dialog"
4.         android:launchMode="singleTask"
5.
```

同样，在Intent类中定义了很多与Activity启动或调度有关的标志，标签中有一些属性，这些标志，属性和四种启动模式联合使用，会在很大程度上改变activity的行为，进而会改变task和back stask的状态。关于Intent中的标志和标签中有一些属性会在本文后面介绍，在这一节中，先介绍activity的四种启动模式。

standard

> 标准启动模式，也是activity的默认启动模式。在这种模式下启动的activity可以被多次实例化，即在同一个任务中可以存在多个activity的实例，每个实例都会处理一个Intent对象。如果Activity A的启动模式为standard，并且A已经启动，在A中再次启动Activity A，即调用startActivity（new Intent（this，A.class）），会在A的上面再次启动一个A的实例，即当前的栈中的状态为A-->A。

singleTop

> 如果一个以singleTop模式启动的activity的实例已经存在于任务栈的栈顶，那么再启动这个Activity时，不会创建新的实例，而是重用位于栈顶的那个实例，并且会调用该实例的onNewIntent()方法将Intent对象传递到这个实例中。举例来说，如果A的启动模式为singleTop，并且A的一个实例已经存在于栈顶中，那么再调用startActivity（new Intent（this，A.class））启动A时，不会再次创建A的实例，而是重用原来的实例，并且调用原来实例的onNewIntent()方法。这是任务栈中还是这有一个A的实例。

> 如果以singleTop模式启动的activity的一个实例已经存在与任务栈中，但是不在栈顶，那么它的行为和standard模式相同，也会创建多个实例。

singleTask

> 谷歌的官方文档上称，如果一个activity的启动模式为singleTask，那么系统总会在一个新任务的最底部（root）启动这个activity，并且被这个activity启动的其他activity会和该activity同时存在于这个新任务中。如果系统中已经存在这样的一个activity则会重用这个实例，并且调用他的onNewIntent()方法。即，这样的一个activity在系统中只会存在一个实例。
>
> 其实官方文档中的这种说法并不准确，启动模式为singleTask的activity并不会总是开启一个新的任务。详情请参考 解开Android应用程序组件Activity的"singleTask"之谜，在本文后面也会通过示例来进行验证。

singleInstance

> 总是在新的任务中开启，并且这个新的任务中有且只有这一个实例，也就是说被该实例启动的其他activity会自动运行于另一个任务中。当再次启动该activity的实例时，会重用已存在的任务和实例。并且会调用这个实例的onNewIntent()方法，将Intent实例传递到该实例中。和singleTask相同，同一时刻在系统中只会存在一个这样的Activity实例。

[原文链接]http://blog.csdn.net/zhangjg_blog/article/details/10923643