

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](#) 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/zy_jibai/article/details/80587083

嗨大家好，又和大家见面了，上一次我们一起搞清楚了Handler的源码机制（现在回想起来是不是感觉也就那么回事，当时看的头皮发麻-。+！！），今天一谈我们在Android开发中必不可少的一个组件——Activity：

Activity作为四大组件之一，也可以说是四大组件中最重要的一个组件，它负责App的视图，还负责用户交互，而且有时候还经常其他组件绑定使用，可以的重要。

虽然说我们天天都在使用Activity，但是你真的对Activity的生命机制烂熟于心，完全了解了吗？的确，Activity的生命周期方法只有七个（自己数-。+），但那只是最平常的情况，或者说是默认的情况。也就是说在其他情况下，Activity的生命周期可能不会是按照我们以前所知道的流程，这就要说到我们今天的重要——**Activity的启动模式**：我们的Activity会根据自身不同的启动模式，自身的生命周期方法会进行不同的调用。

一、在将启动模式之前必须了解的一些知识：

在正式的介绍Activity的启动模式之前，我们首先要了解一些旁边的知识，这些知识如果说模糊不清，那么在讨论启动模式的时候会一头雾水（笔者亲身感受）。

1.一个应用程序通常会有多个Activity，这些Activity都有一个对应的action（如MainActivity的action），我们可以通过action来启动对应Activity（隐式启动）。

```
<action android:name="android.intent.action.MAIN" />
```

2.一个应用程序可以说由一系列组件组成，这些组件以进程为载体，相互协作实现App功能。

3.任务栈（Task Stack）或者叫退回栈（Back Stack）介绍：

- 3.1.任务栈用来存放用户开启的Activity。
- 3.2.在应用程序创建之初，系统会默认分配给其一个任务栈（默认一个），并存储根Activity。
- 3.3.同一个Task Stack，只要不在栈顶，就是onStop状态：
- 
- 3.4.任务栈的id自增长型，是Integer类型。
- 3.5.新创建Activity会被压入栈顶。点击back会将栈顶Activity弹出，并产生新的栈顶元素作为显示界面（onResume状态）。
- 3.6.当Task最后一个Activity被销毁时，对应的应用程序被关闭，清除Task栈，但是还会保留应用程序进程（狂点Back退出到Home界面后点击Menu会发现这个App的框框。个人理解应该是这个意思），再次点击进入应用会创建新的Task栈。

4.Activity的affinity：

- 4.1.affinity是Activity内的一个属性（在ManiFest中对应属性为taskAffinity）。默认情况下，拥有相同affinity的Activity属于同一个Task中。
- 4.2.Task也有affinity属性，它的affinity属性由根Activity（创建Task时第一个被压入栈的Activity）决定。
- 4.3.在默认情况下（我们什么都不设置），所有的Activity的affinity都从Application继承。也就是说Application同样有taskAffinity属性。

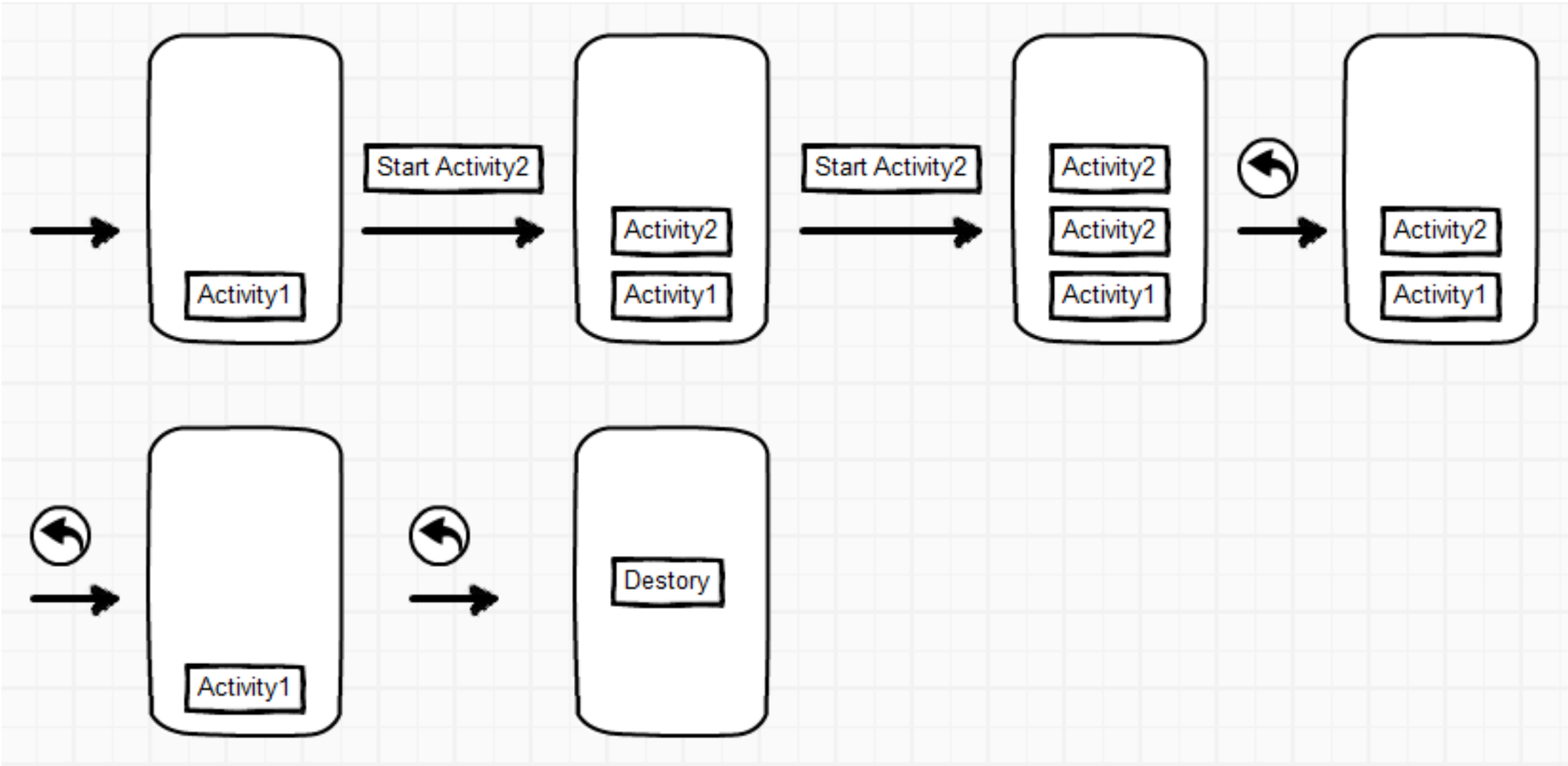
```
1 | <application
2 |     android:taskAffinity="gf.zy"
```

暂时就是这么多了，如果还有不妥的地方我会补充的。接下来我们来正式看Activity的启动模式：

二、Activity启动模式：

1.默认启动模式standard：

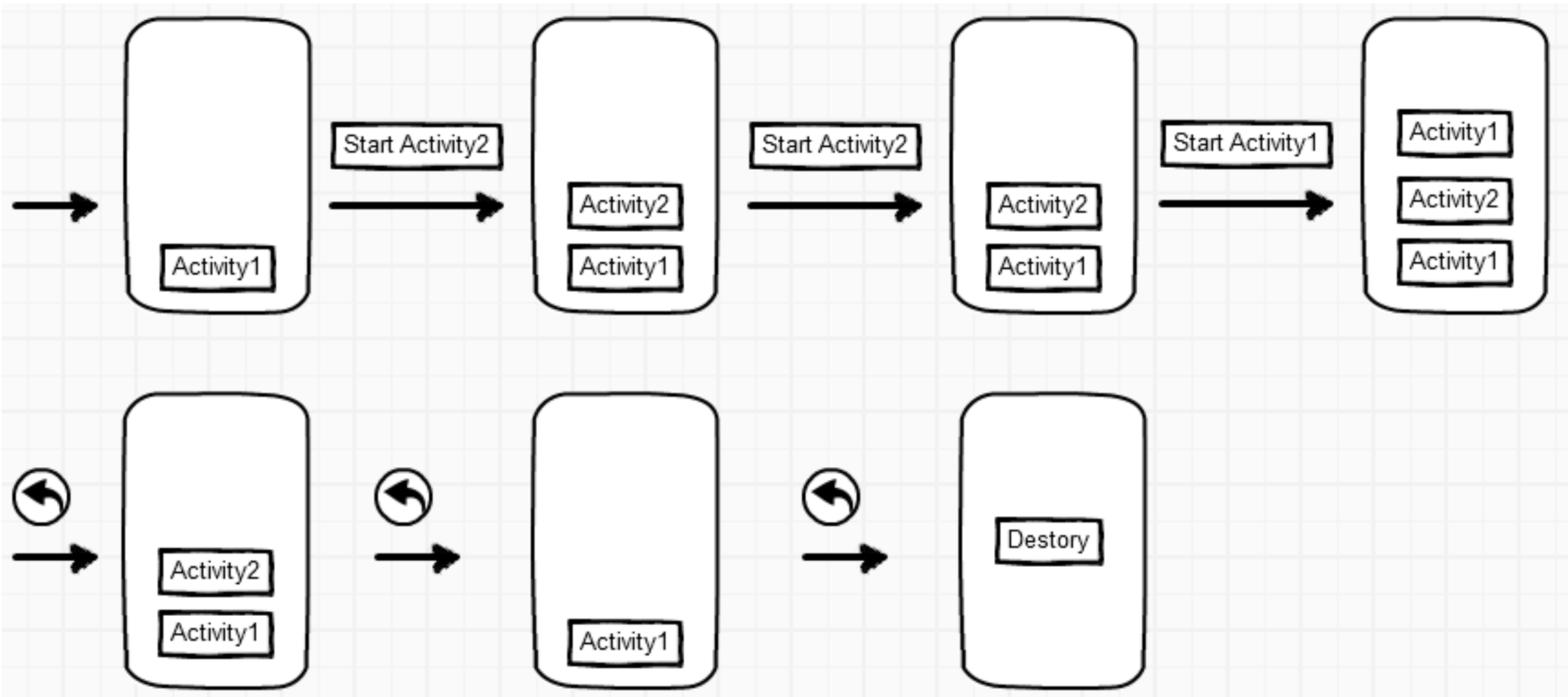
该模式可以被设定，不在manifest设定时候， Activity的默认模式就是standard。在该模式下， 启动的Activity会依照启动顺序被依次压入Task中：



上面这张图讲的已经很清楚，我想应该不用做什么实验来论证了吧，这个是最简单的一个，我们过。

2.栈顶复用模式singleTop：

在该模式下，如果栈顶Activity为我们要新建的Activity（目标Activity），那么就不会重复创建新的Activity。



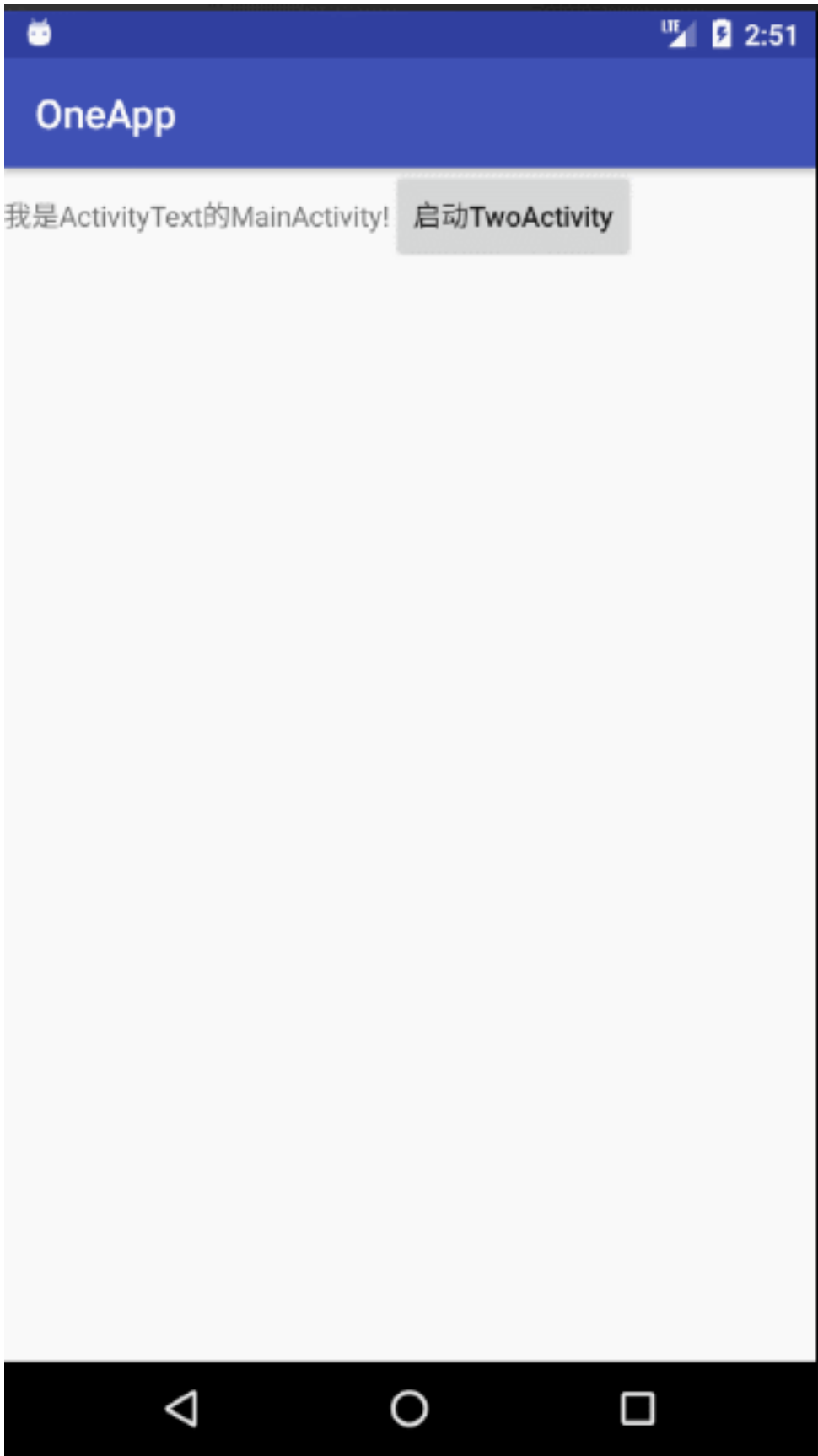
这次我来用代码举例：

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="zy.pers.activitytext">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="@string/app_name"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:taskAffinity="gf.zy"
12        android:theme="@style/AppTheme">
```

```
13 |         <activity android:name=".MainActivity">
14 |             <intent-filter>
15 |                 <action android:name="android.intent.action.MAIN" />
16 |
17 |                 <category android:name="android.intent.category.LAUNCHER" />
18 |             </intent-filter>
19 |         </activity>
20 |         <activity android:name=".TwoActivity"
21 |             android:launchMode="singleTop">
22 |             <intent-filter>
23 |                 <action android:name="ONETEXT_TWOACTIVITY" />
24 |                 <category android:name="android.intent.category.DEFAULT" />
25 |             </intent-filter>
26 |         </activity>
27 |         <activity android:name=".ThreeActivity">
28 |             <intent-filter>
29 |                 <action android:name="ONETEXT_THREEACTIVITY" />
30 |                 <category android:name="android.intent.category.DEFAULT" />
31 |             </intent-filter>
32 |         </activity>
33 |     </application>
34 |
35 | </manifest>
```

这是我的第一个应用OneText的Mainfest结构，里面创建了三个Activity，我们把第二个Activity的模式设置为singleTop。

每个Activity界面都只有一个显示当前界面名称的TextView和一个用来组跳转的Button，所以应用OneText的功能就是从活动1跳转到活动2，活动2继续跳转活
码就不给大家展示了，都能写出来。



我们发现在我们跳转到TwoActivity之后， 点击跳转新的TwoActivity时候， 他没有响应。

为了作对比，我们再把TwoActivity设置为standard， 看一看效果：



我们发现创建了很多的TwoActivity。

同时我们打印上task的Id（我没有把所有周期方法都打印log）：

```
06-06 10:44:44.620 20255-20255/? E/MainActivity: onCreate: 4351
06-06 10:44:45.463 20255-20255/zy.pers.activitytext E/TwoActivity: onCreate: 4351
06-06 10:44:45.464 20255-20255/zy.pers.activitytext E/TwoActivity: onStart:
06-06 10:44:45.465 20255-20255/zy.pers.activitytext E/TwoActivity: onResume:
06-06 10:44:45.945 20255-20255/zy.pers.activitytext E/MainActivity: onStop:
06-06 10:44:46.315 20255-20255/zy.pers.activitytext E/ThreeActivity: onCreate: 4351
06-06 10:44:46.316 20255-20255/zy.pers.activitytext E/ThreeActivity: onStart:
06-06 10:44:46.316 20255-20255/zy.pers.activitytext E/ThreeActivity: onResume:
06-06 10:44:47.122 20255-20255/zy.pers.activitytext E/TwoActivity: onCreate: 4351
06-06 10:44:47.123 20255-20255/zy.pers.activitytext E/TwoActivity: onStart:
06-06 10:44:47.124 20255-20255/zy.pers.activitytext E/TwoActivity: onResume:
06-06 10:44:47.591 20255-20255/zy.pers.activitytext E/ThreeActivity: onStop:
06-06 10:44:48.419 20255-20255/zy.pers.activitytext E/ThreeActivity: onCreate: 4351
06-06 10:44:48.420 20255-20255/zy.pers.activitytext E/ThreeActivity: onStart:
06-06 10:44:48.421 20255-20255/zy.pers.activitytext E/ThreeActivity: onResume:
06-06 10:44:49.288 20255-20255/zy.pers.activitytext E/TwoActivity: onCreate: 4351
    onStart:
06-06 10:44:49.289 20255-20255/zy.pers.activitytext E/TwoActivity: onResume:
06-06 10:44:49.749 20255-20255/zy.pers.activitytext E/ThreeActivity: onStop:
```

发现他们全部都是来自一个Task。这个可以过。

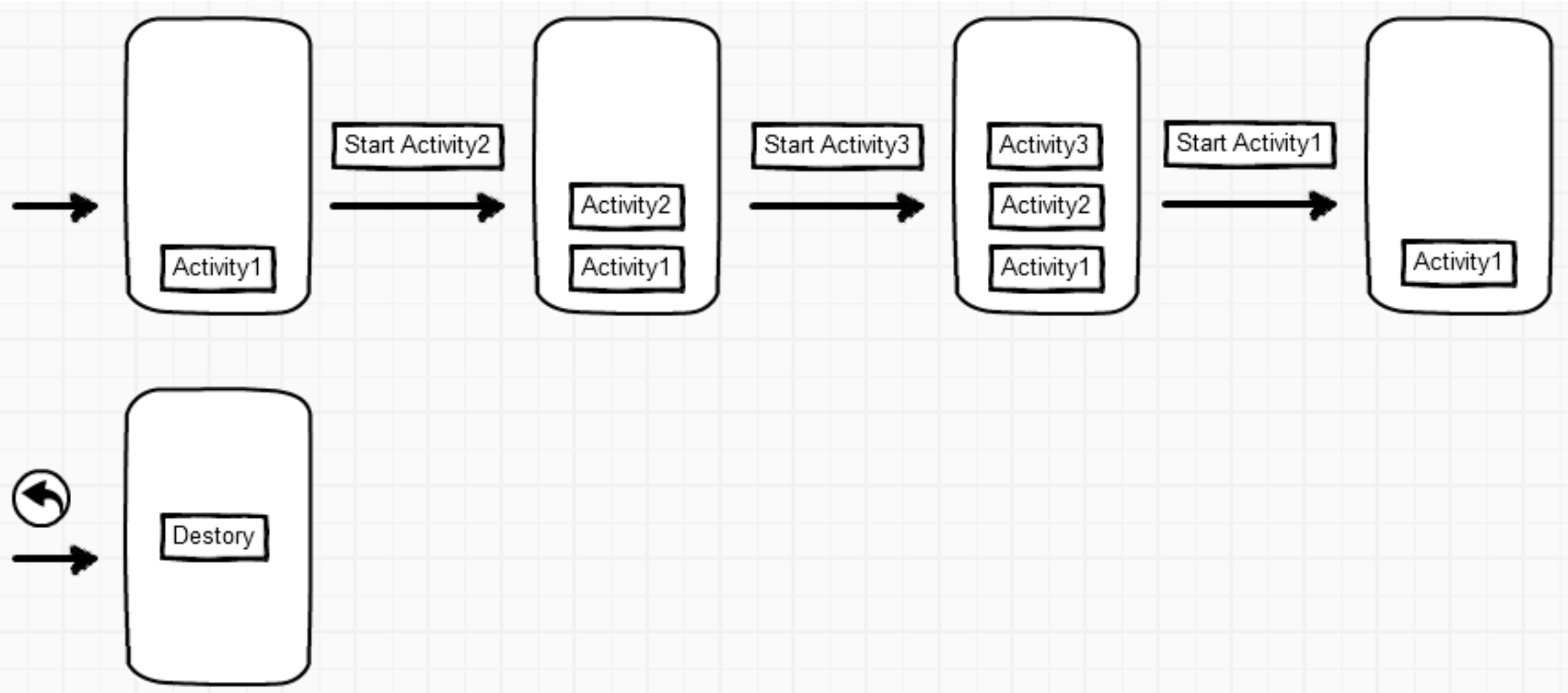
应用场景：

开启渠道多，适合多应用开启调用的Activity：通过这种设置可以避免已经创建过的Activity被重复创建（多数通过动态设置使用，关于动态设置下面会详细介绍）

3.栈内复用模式singleTask：

与singleTop模式相似，只不过singleTop模式是只是针对栈顶的元素，而singleTask模式下，如果task栈内存在目标Activity实例，则：

- 1. 将task内的对应Activity实例之上的所有Activity弹出栈。
- 2. 将对应Activity置于栈顶，获得焦点。



同样我们也用代码来实现一下这个过程：

还是刚才的那一坨代码，只是我们修改一下Activity1的模式为singleTask，然后让Activity2跳转到Activity3，让Activity3跳转到Activity1：



在跳回MainActivity之后点击back键发现直接退出引用了，这说明此时的MainActivity为task内的最后一个Activity。所以这个模式过。

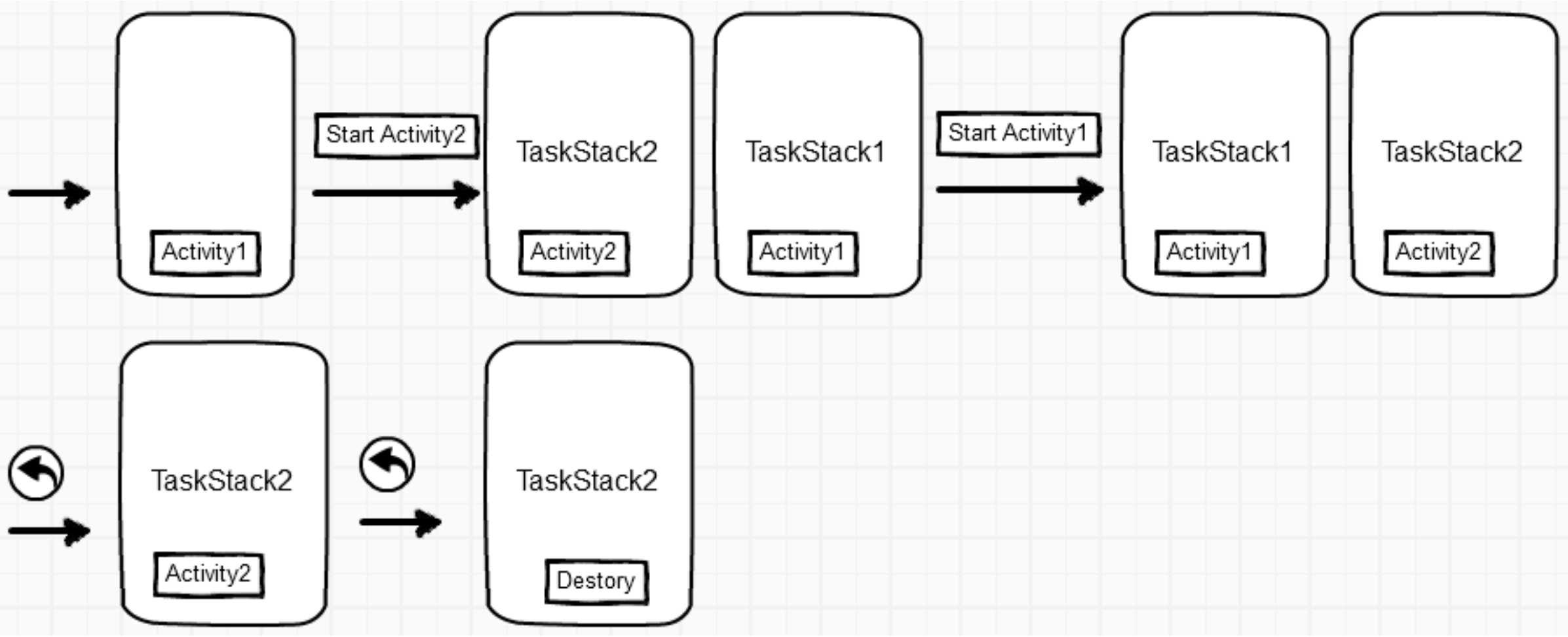
应用场景：

程序主界面，我们肯定不希望主界面被多创建，而且在主界面退出的时候退出整个App是最好的设想。

耗费系统资源的Activity：对于那些及其耗费系统资源的Activity，我们可以考虑将其设为singleTask模式，减少资源耗费（在创建阶段耗费资源的情况，个人理解+）。

4.全局唯一模式singleInstance：

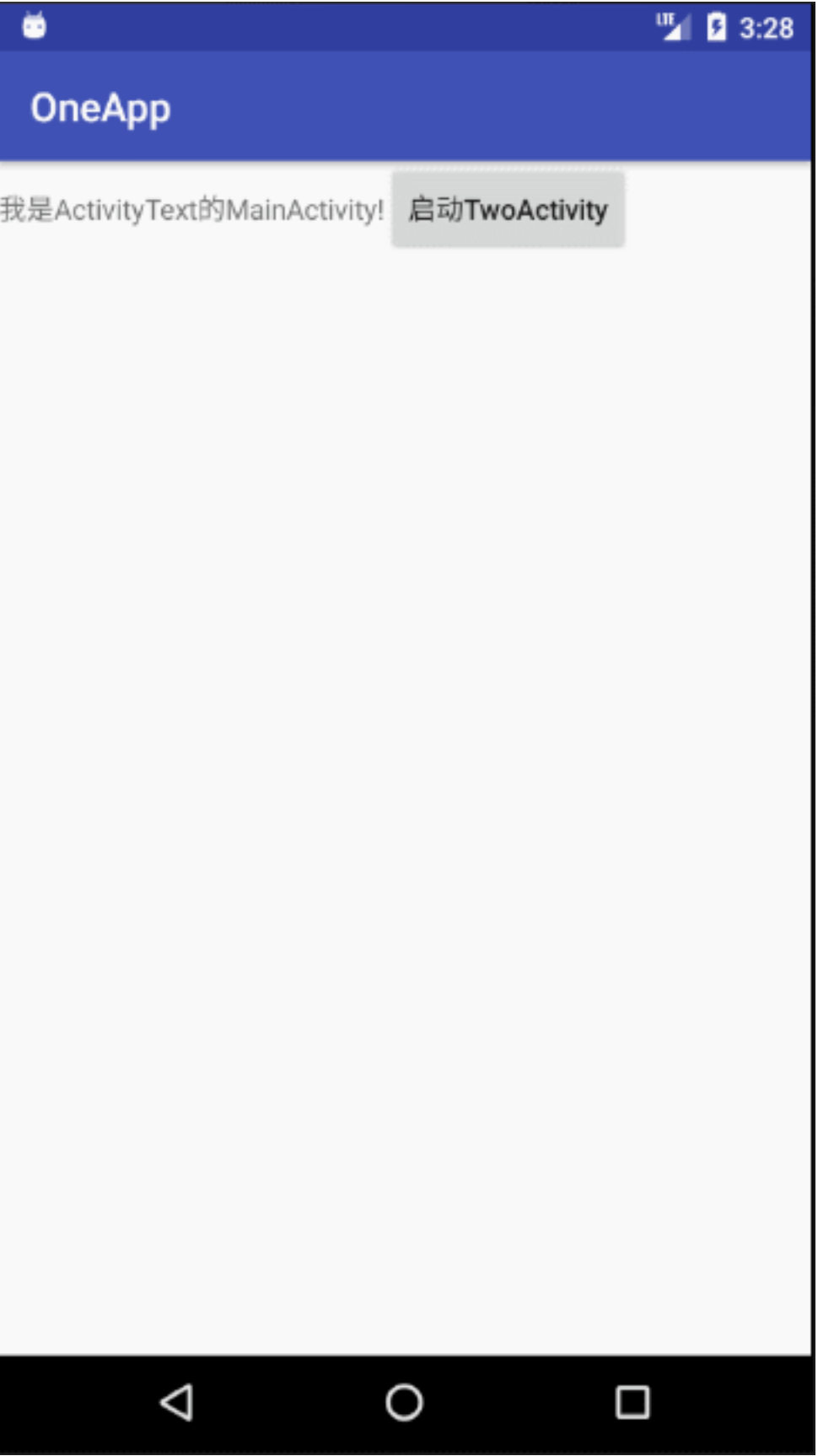
这是我们最后的一种启动模式，也是我们最恶心的一种模式：在该模式下，我们会为目标Activity分配一个新的affinity，并创建一个新的Task栈，将目标Activity放入新的Task，并让目标Activity获得焦点。新的Task有且只有这一个Activity实例。 如果已经创建过目标Activity实例，则是将以前的Activity唤醒（对应Task设为Foreground状态）



我们为了看的更明确，这次不按照上图的步骤设计程序了（没错，这几张图都不是我画的-。+！）。

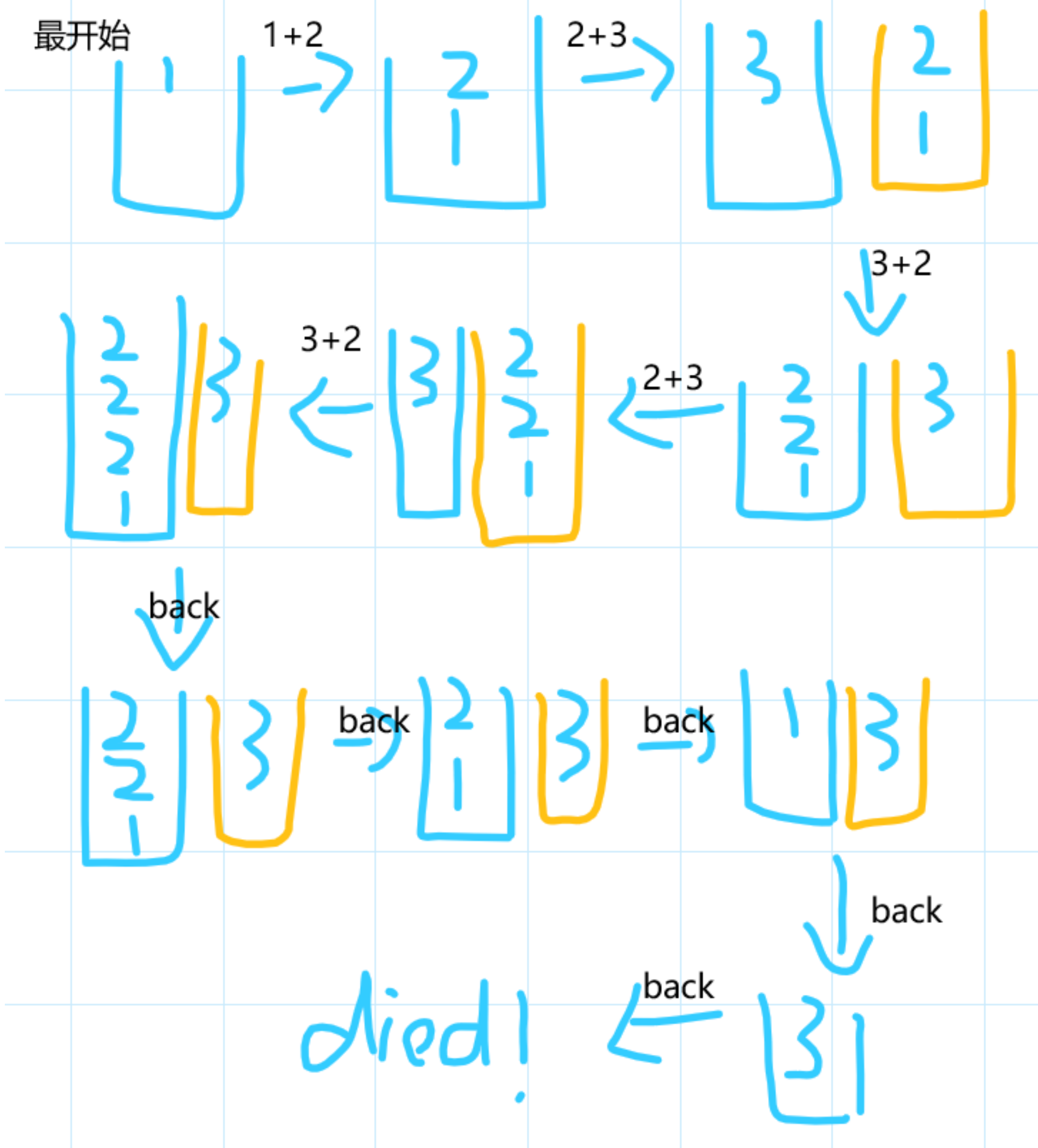
我们先指定一下这次的程序：还是这三个Activity，这次Activity3设置为singleInstance，1和2默认（standard）。

然后我们看一下这个效果：



说一下我们做了什么操作：

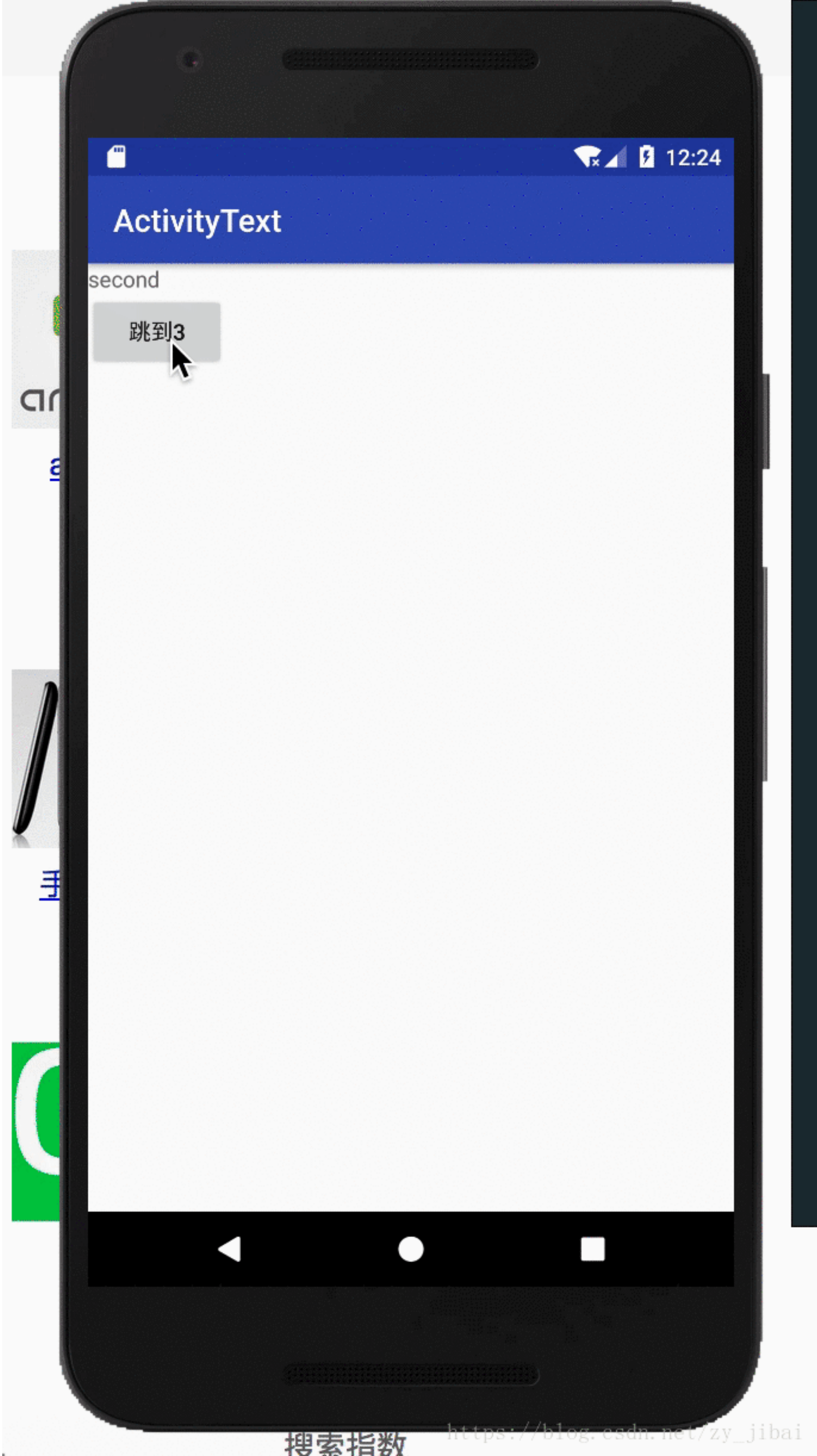
首先由1创建2,2创建3，然后又由3创建2,2创建3,3创建2，然后一直back，图如下：



还请各位别嫌弃我-。+，图虽然不好看，但是很生动形象。。。。具体说一下：这张图对应着我们上面的程序流程，黄色的代表Background的Task，蓝色的Foreground的Task。

我们发现back的时候会先把Foreground的Task中的Activity弹出，直到Task销毁，然后将Background的Task唤到前台，所以最后将Activity3销毁之后，会直接回到应用。

但是有没有想过这样会出现一个问题，什么问题我们直接看图就好：



我简单说一下这个案例：1，2，3三个Activity，2是singleInstance模式，然后1->2,2->3，之后狂点back，在回到Home界面后点击菜单键，发现首先启动的是2Activity。

简单解释一下：1和3是一个task，2是单独的一个task，在我们2->3后，前台的task又从2的回到了1和3的。所以最后退出的task是2的线程，而如果不是重新启动App。上一次最后关闭的Task是2的，所以。。

所以说singleInstance设置的Activity最好不要设置成中间界面。

以上表示我们关于四种模式的最基本理解，其实有了前面的知识了解之后，我们发现这些其实也不是很难对吧。。。真正比较绕的在后面-。+，注意前方高能

三、动态设置启动模式

在上述所有情况，都是我们在Manifest中设置的（通过launchMode属性设置），这个被称为静态设置（我们写程序写多了会发现静态就有动态，而且静态在xml设置，动态在java代码设置-。+），接下来我们来看一下如何动态的设置Activity启动方式。

注）：如果同时有动态和静态设置，那么动态的优先级更高。

1.关于动态设置与静态设置的理解：

关于这个理解我是看过一篇文章，比较认同里面的思想，所以在这里也总结一下：

静态设置，可以理解为通知别人：就是当我被创建的时候，我告诉你我是通过这种模式启动的。

动态设置，可以理解为别人的要求：别人给我设一个Flag，我就以这种Flag的方式启动。

可能这个没什么用哈，但是仔细想一下这种对程序的思想理解应该是正确的。

2.几种常见的Flag：

我们说的动态设置，其实是通过Intent。对与Intent这个类大家应该不陌生吧，我们刚才在启动Activity的时候就用到这个类了。

如果我们要设置要启动的Activity的启动模式的话，只需要这样：

```
intent.setFlags(、、、、);
```

然后在里面添加对应的Flag就好，那么接下来我们介绍几个常见的Flag（他的Flag太多了，头皮发麻。）：

2.1._NEW_TASK

他对应的Flag如下：

```
Intent.FLAG_ACTIVITY_NEW_TASK
```

这个Flag跟我们的singleInstance很相似：在给目标Activity设立此Flag后，会根据目标Activity的affinity进行匹配：如果已经存在与其affinity相同的task，则将Activity压入此Task。反之没有的话，则新建一个task，新建的task的affinity值与目标Activity相同。然后将目标Activity压入此栈。

其实简单来说，就是先看看需不需要创建一个新的Task，根据就是有没有相同的affinity。然后把Activity放进去。

但是此情况和singleInstance有不同，有两点注意的地方：

- 1. 新的Task没有说只能存放一个目标Activity。只是说决定是否新建一个Task。而singleInstance模式下新的Task只能放置一个目标Activity。
- 2. 在同一应用下，如果Activity都是默认的affinity，那么此Flag无效。而singleInstance默认情况也会创建新的Task。

这个东西理解起来可能有一些抽象，我们通过一个实例来证明他：

在之前的一些例子中，我们都是在同一应用之间进行跳转，而现在我们进行不同App的Activity相互跳转（其实就是创造一个不同taskAffinity的情况。。。忘了，一、4）。

首先，我们需要创建一个新的App——TwoApp，这个App目前只需要一个MainActivity就够了，我们在MainActivity放置一个button，让他跳转到OneApp的TwoActivity。

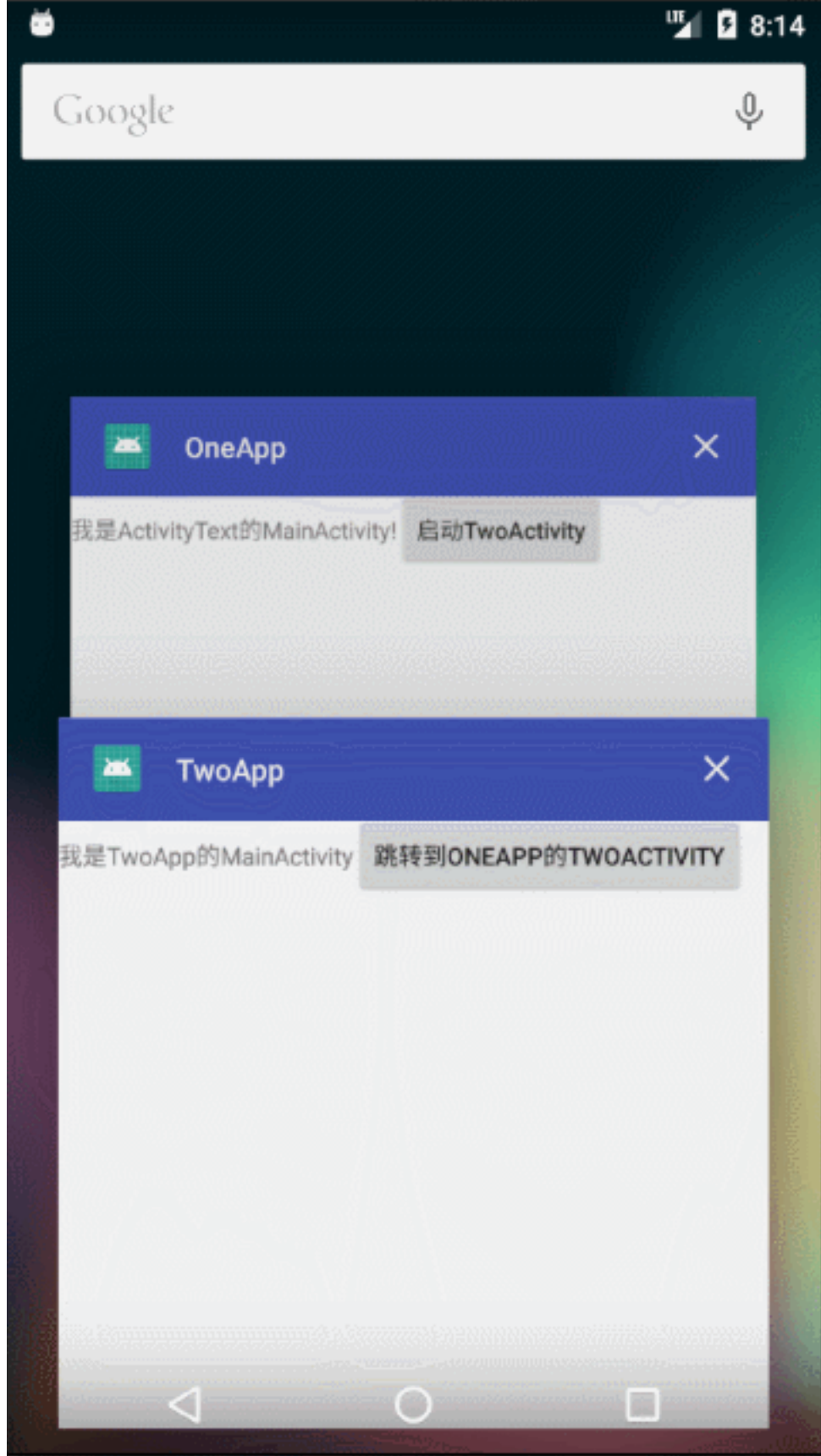
```
1 public void onClick(View v) {
2     Intent intent = new Intent("ONETEXT_TWOACTIVITY");
3
4     startActivity(intent);
5 }
```

这是跳转的代码。

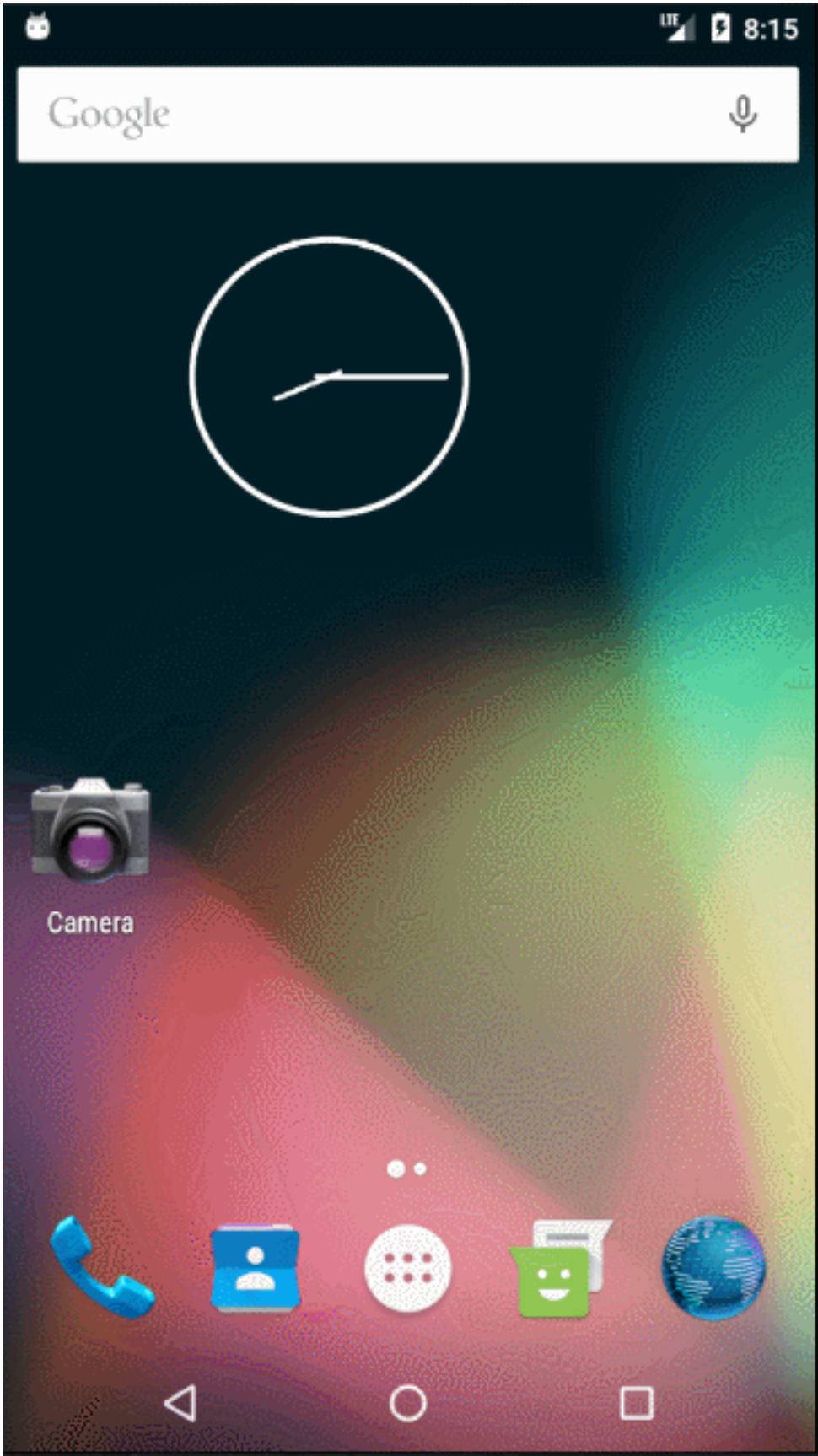
现在我们先概述一下我们的流程：我们先打开TwoApp，然后在TwoApp的MainActivity界面跳转到OneApp的TwoActivity。

对于OneApp的设定，我们已经将三个Activity都设置成了standard模式。还是1->2,2->3,3->2。

代码就不上了，这么简单，大家自己也能写出来。效果如下：



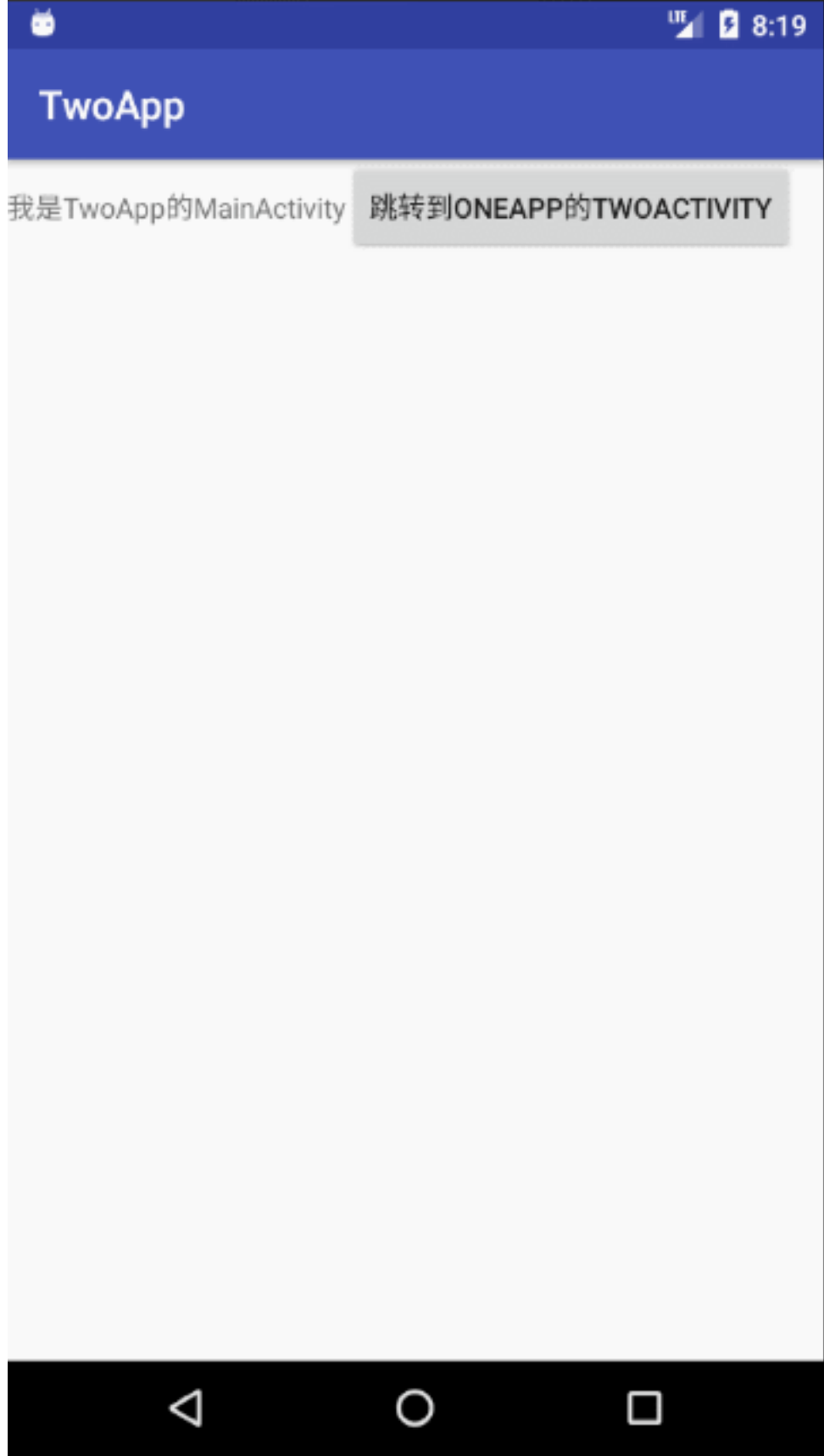
为了看的清清楚楚，最开始清空了所有的进程。



现在我们点开TwoApp，

现在只有TwoApp一个进程，

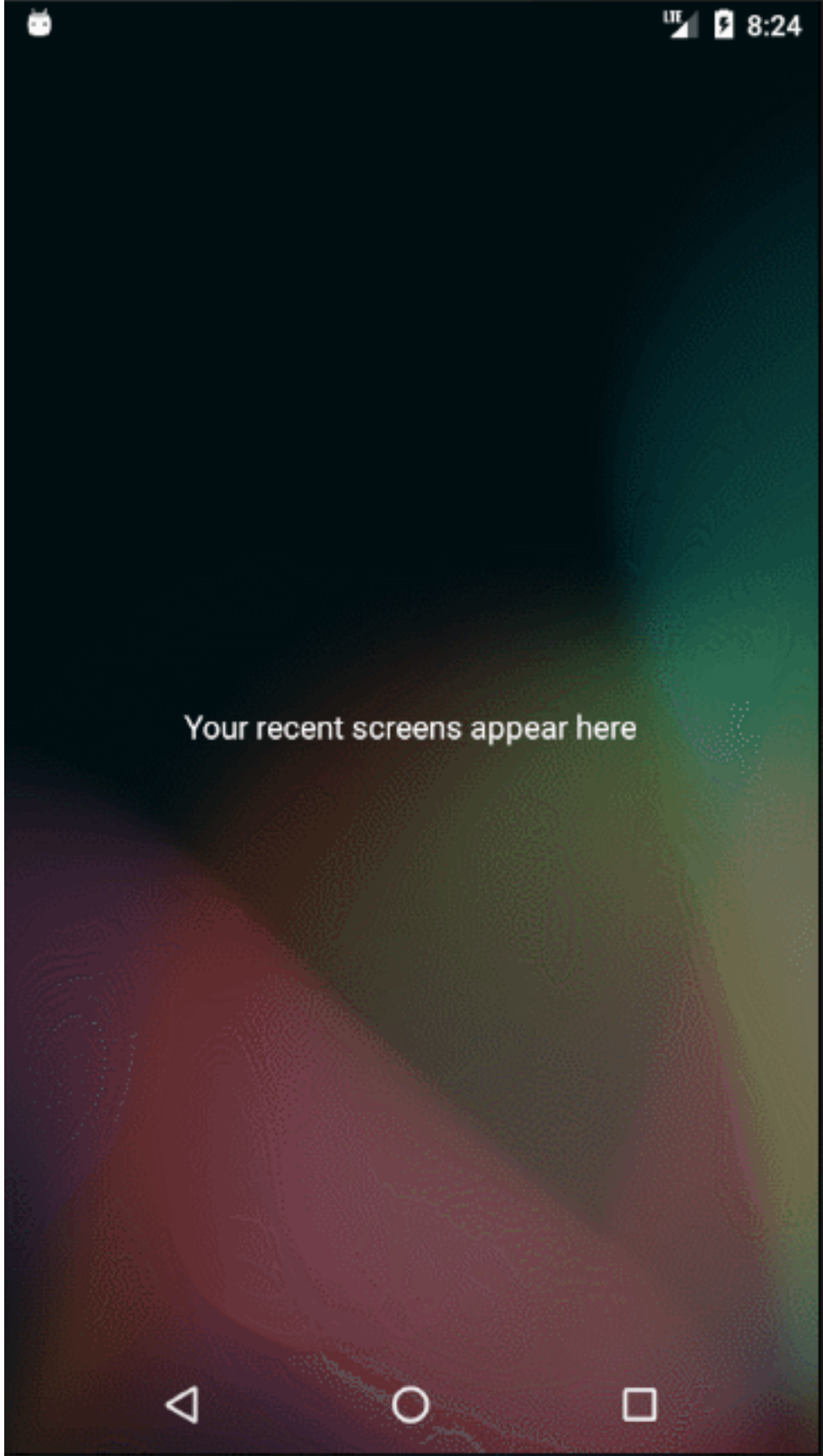
现在我们点开了 OneApp的TwoActivity， 但是我们发现他还是只有一个进程，



现在我们在TwoApp的MainActivity跳转到OneApp的TwoActivity，添加_NEW_TASK的Flag。

```
1 public void onClick(View v) {  
2     Intent intent = new Intent("ONETEXT_TWOACTIVITY");  
3     intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
4     startActivity(intent);  
5 }
```

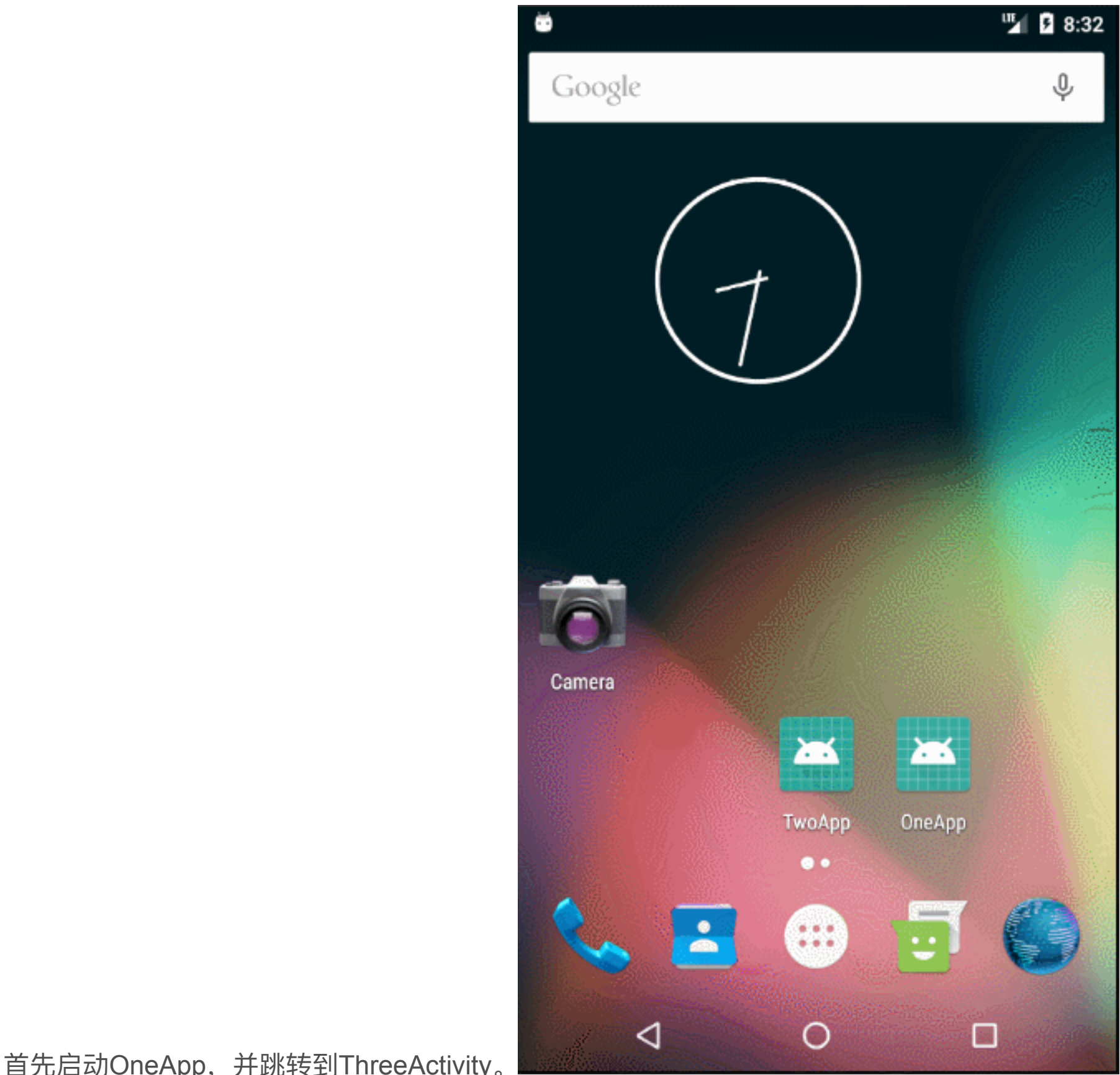
我们再看一下效果，还是跟刚才一样先清空所有的进程，这次效果直接连起来看了：



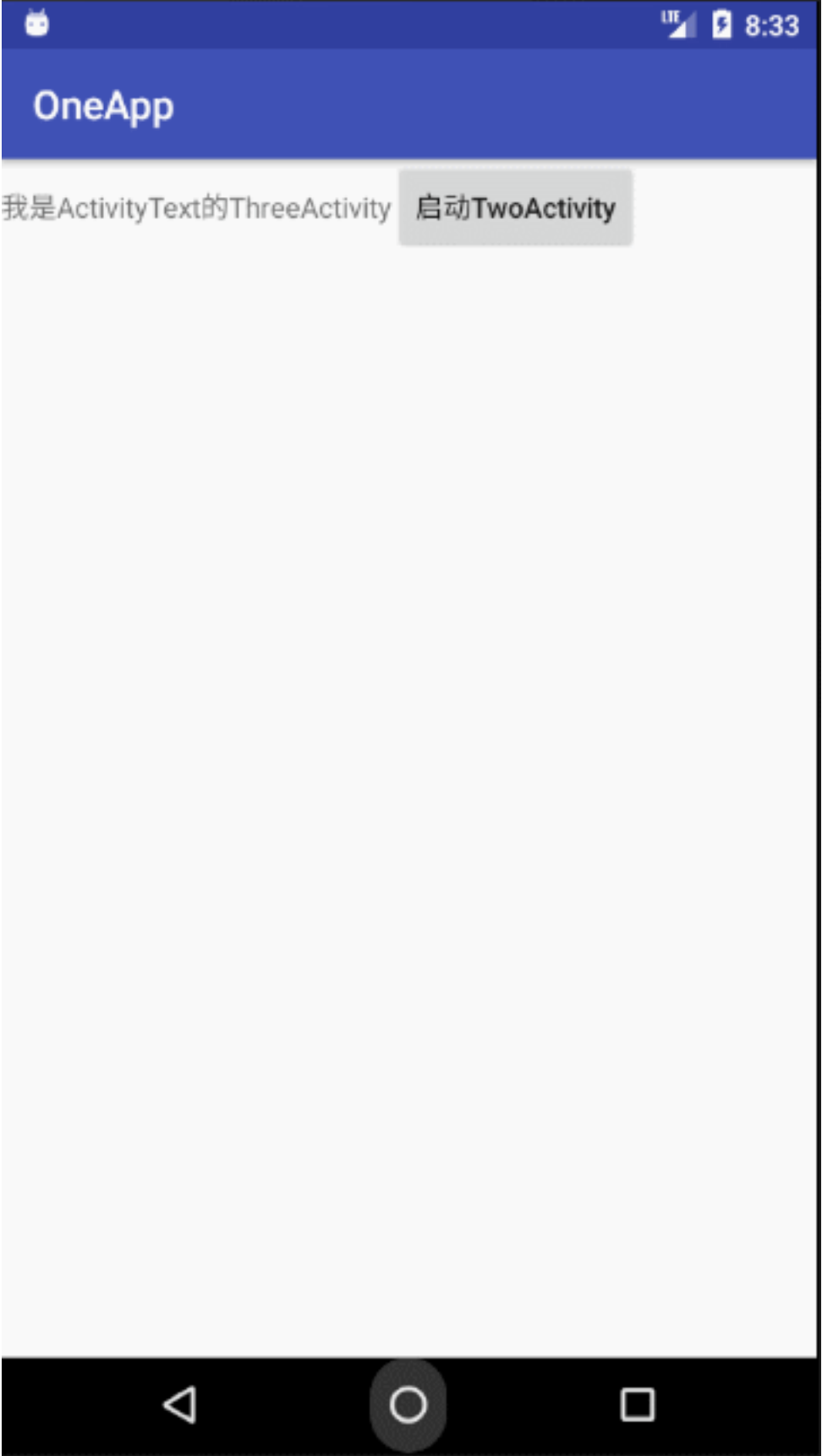
与上面不同的地方在于，我们新的界面创建在了新的进程中——其实就是OneApp被唤醒了，我们来分析一下为什么会这样：

首先我们会想一下我们上面所学过的一个东西，affinity：我们说这个东西在默认情况下就是App的包名packageName，而OneApp中的TwoActivity默认的affinity就是OneApp的包名。这里能够理解吧。

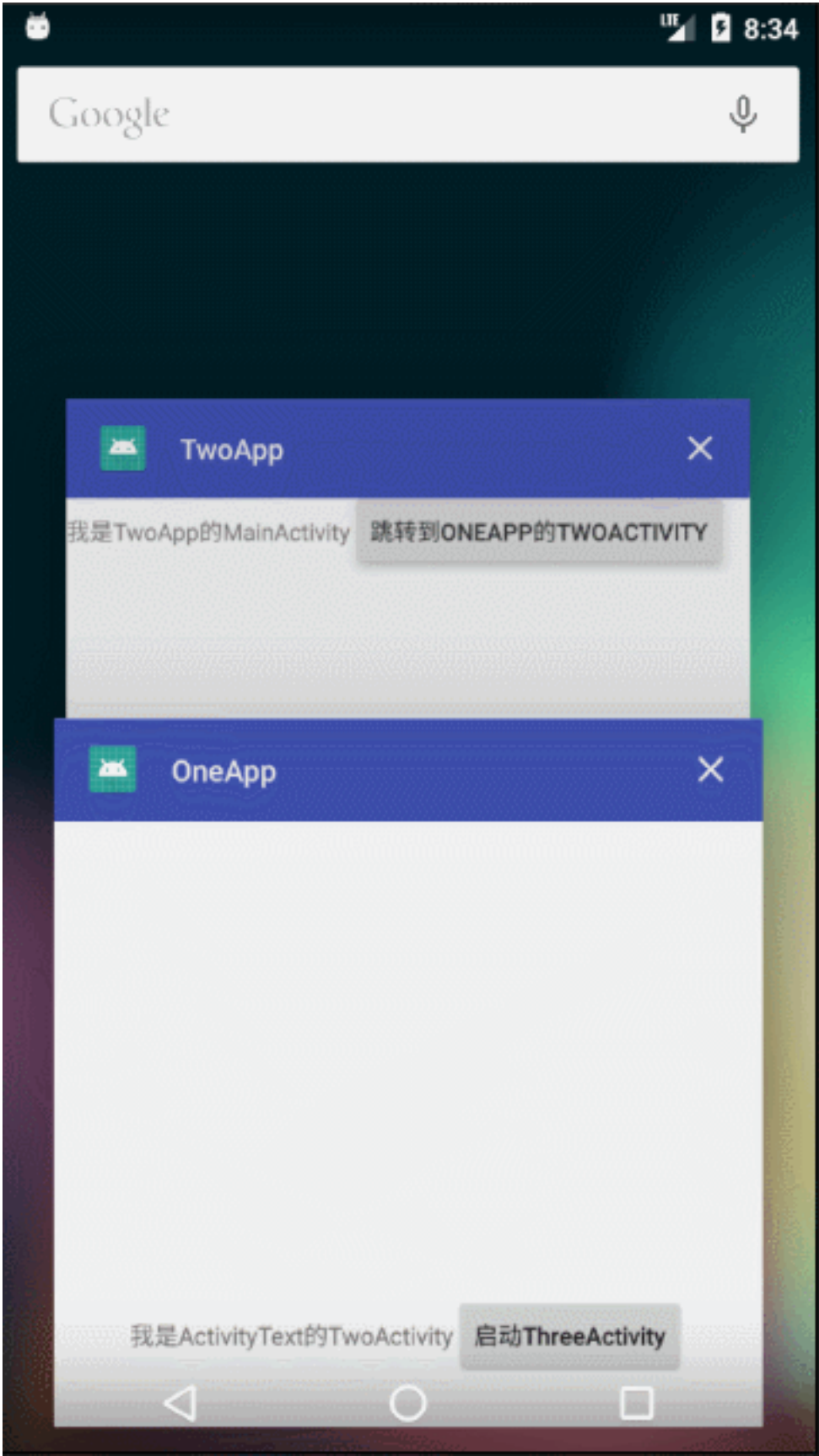
然后我们说_NEW_TASK情况下，会先查找是否有对应的affinity的task，如果有就不在创建，直接将其放入，反之新建task，所以新建的task就是我们的OneApp的task，我们可以再做一个测试，我们先唤醒OneApp，然后再让TwoApp跳转到OneApp的TwoActivity（有点绕啊。。。），我们看是什么情况：



首先启动OneApp，并跳转到ThreeActivity。



然后启动TwoApp，并跳转到OneApp的TwoActivity。



然后一直点击Back，

我们发现在Two中唤醒One的TwoActivity，同样是被放入了OneApp的默认Task中。

关于_NEW_TASK我们就说这么多吧。

2.2._SINGLE_TOP

该模式比较简单，对应Flag如下：

```
Intent.FLAG_ACTIVITY_SINGLE_TOP
```


次Flag与静态设置中的singleTop效果相同，所以请见二、2.

2.3._CLEAR_TOP

这个模式对应的Flag如下：

```
Intent.FLAG_ACTIVITY_CLEAR_TOP
```

当设置此Flag时， 目标Activity会检查Task中是否存在此实例， 如果没有则添加压入栈，

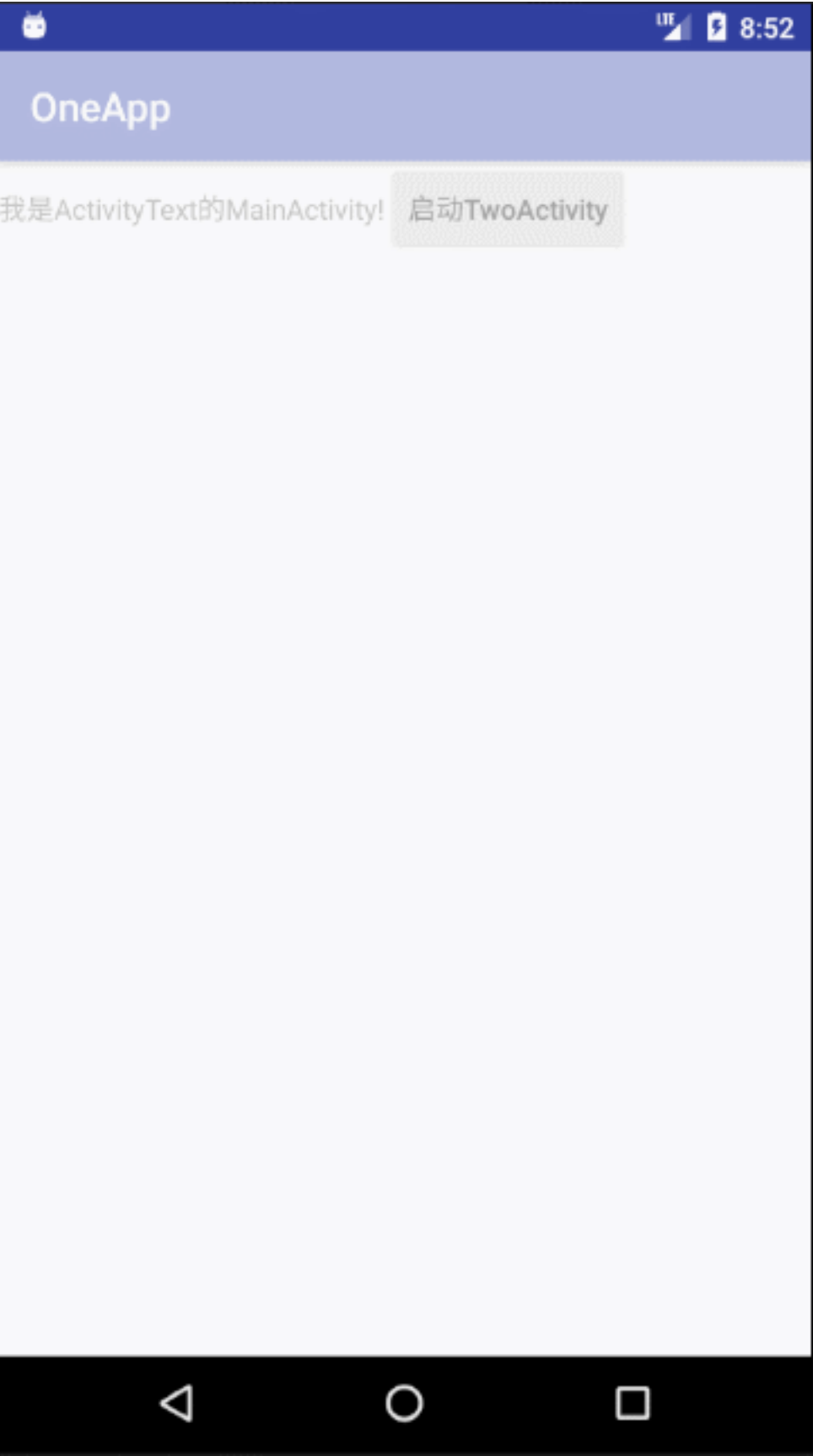
如果有， 就将位于Task中的对应Activity其上的所有Activity弹出栈， 此时有以下两种情况：

- 如果同时设置Flag_ACTIVITY_SINGLE_TOP， 则直接使用栈内的对应Activity，
- 没有设置。。。。。。， 则将栈内的对应Activity销毁重新创建。

关于这个Flag，我们发现他和singleTask很像， 准确的说， 是在_CLEAR_TOP和_SINGLE_TOP同时设置的情况下， 就是singleTask模式。

而唯一不同的一点就在于： 他会销毁已存在的目标实例， 再重新创建。这个我们通过打印一下生命周期就好。

这次我们只用OneApp就好了， 还是1->2,2->3,3->2,这次我们将2的Flag设置为_CLEAR_TOP， 看一下TwoActivity的生命周期。



我们的流程如下： 1->2 2->3 3->2 2->3 3->2 back back 然后就退出了， 这说明在Task内2上面的3的确被弹出栈了。

然后我们再看一下2的日志：

06-06 08:55:36.050 13487-13487/? E/MainActivity: onCreate: 76	
06-06 08:55:43.989 13487-13487/zy.pers.activitytext E/TwoActivity: onCreate: 76	
06-06 08:55:43.993 13487-13487/zy.pers.activitytext E/TwoActivity: onStart:	
onResume:	1->2
06-06 08:55:44.411 13487-13531/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown buffer: 0xaaa89780	
06-06 08:55:44.735 13487-13487/zy.pers.activitytext E/MainActivity: onStop:	
06-06 08:55:46.490 13487-13487/zy.pers.activitytext E/TwoActivity: onPause:	
06-06 08:55:46.568 13487-13487/zy.pers.activitytext E/ThreeActivity: onCreate: 76	
06-06 08:55:46.572 13487-13487/zy.pers.activitytext E/ThreeActivity: onStart:	2->3
06-06 08:55:46.576 13487-13487/zy.pers.activitytext E/ThreeActivity: onResume: my id = 76	
06-06 08:55:46.907 13487-13531/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown buffer: 0xaaa89be0	
06-06 08:55:47.229 13487-13487/zy.pers.activitytext E/TwoActivity: onStop:	
06-06 08:55:50.936 13487-13487/zy.pers.activitytext E/TwoActivity: onDestroy:	
06-06 08:55:50.953 13487-13487/zy.pers.activitytext E/TwoActivity: onCreate: 76	
06-06 08:55:50.960 13487-13487/zy.pers.activitytext E/TwoActivity: onStart:	3->2
06-06 08:55:50.961 13487-13487/zy.pers.activitytext E/TwoActivity: onResume:	
06-06 08:55:51.342 13487-13531/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown buffer: 0xaaa89140	
06-06 08:55:51.543 13487-13487/zy.pers.activitytext E/ThreeActivity: onStop:	
onDestroy:	

我想在日志图片上面标注的很清楚了，我只截取了一部分日志，我们质疑3->2时候先销毁，后创建。

好，现在我们同时加上_SINGLE_TOP的Flag。

效果相同，我们只看log：

06-06 08:59:38.133 13723-13723/? E/MainActivity: onCreate: 77	
06-06 08:59:41.643 13723-13723/zy.pers.activitytext E/TwoActivity: onCreate: 77	
06-06 08:59:41.651 13723-13723/zy.pers.activitytext E/TwoActivity: onStart:	
onResume:	1->2
06-06 08:59:42.036 13723-13765/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown b	
06-06 08:59:42.311 13723-13723/zy.pers.activitytext E/MainActivity: onStop:	
06-06 08:59:47.567 13723-13723/zy.pers.activitytext E/TwoActivity: onPause:	
06-06 08:59:47.622 13723-13723/zy.pers.activitytext E/ThreeActivity: onCreate: 77	
06-06 08:59:47.624 13723-13723/zy.pers.activitytext E/ThreeActivity: onStart:	2->3
06-06 08:59:47.625 13723-13723/zy.pers.activitytext E/ThreeActivity: onResume: my id = 77	
06-06 08:59:47.949 13723-13765/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown b	
06-06 08:59:48.240 13723-13723/zy.pers.activitytext E/TwoActivity: onStop:	
06-06 08:59:50.616 13723-13723/zy.pers.activitytext E/TwoActivity: onRestart:	
06-06 08:59:50.617 13723-13723/zy.pers.activitytext E/TwoActivity: onStart:	
06-06 08:59:50.618 13723-13723/zy.pers.activitytext E/TwoActivity: onResume:	3->2
06-06 08:59:50.865 13723-13765/zy.pers.activitytext E/Surface: getSlotFromBufferLocked: unknown b	
06-06 08:59:51.103 13723-13723/zy.pers.activitytext E/ThreeActivity: onStop:	
06-06 08:59:51.105 13723-13723/zy.pers.activitytext E/ThreeActivity: onDestroy:	

很明显，在3->2的时候，TwoActivity调用了onRestart方法，也就是栈顶复用了。

这个Flag过。

其实还有一点点东西想提一下的，但是感觉 篇幅已经很恶心了。。。再写下去可能更没人看了吧/笑着哭，所以决定找机会穿插到下一篇文章中：[Activity task相关属性](#)

喜欢的话可以关注一波，有建议和不同观点的欢迎下方留言！感谢大家的支持！

文章最后发布于: 2018-06-0

有 0 个人打赏

Android：图解四种启动模式 及 实际应用场景解说 阅读数 42

在一个项目中会包括着多个Activity，系统中使用任务栈来存储创建的Activity实例，任... [博文](#) | 来自: [weixin_3...](#)

想对作者说点什么

yjw1997

1个月前 #4楼

查看回复(

博文写的很好，不过我有两个问题： 1、在设置了taskAffnity的情况下 singleTask启动模式应该与NEW_TASK + CLEAR_TOP + SINGLE_TOP等同吧？ 2、为什么官方文档说NEW_TASK与singleTask启动模式等同呢？

qiushuihhh

2个月前 #3楼

写的挺好的，果然图文结合看的更清楚！！

qq_43216321

5个月前 #2楼

Destroy拼错了

[查看 8 条热评](#)

彻底弄懂Activity四大启动模式 阅读数 8万+

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... [博文](#) | 来自: [Android...](#)

Activity四种启动方式简述 阅读数 1万+

Activity的启动模式分为四种。（standard、singleTop、singTask、singleInstance）;... [博文](#) | 来自: [qq_3821...](#)

Activity四种启动模式 阅读数 18万+

Activity启动方式有四种，分别是：standardsingleTopsingleTasksingleInstance可以根... [博文](#) | 来自: [哇哈哈。...](#)

Android Activity 的四种启动模式 阅读数 1万+

在实际开发中，应根据特定的需求为每个Activity制定恰当的启动模式。Activityde的启... [博文](#) | 来自: [YeeCeeY...](#)

在程序开发中必须了解Activity的四种启动模式 阅读数 1196

Activity的启动模式有四种,分别是standard、singleTop、singleTask和singleInatance... [博文](#) | 来自: [lxn_1221...](#)

Android----四种Activity的启动模式 阅读数 141

一，standard启动模式此模式是默认的启动模式，每次startActivity都是创建一个新的a... [博文](#) | 来自: [A_droid...](#)

史上最详细的IDEA优雅整合Maven+SSM框架（详细思路+附带源码） 阅读数 1万+

网上很多整合SSM博客文章并不能让初探ssm的同学思路完全的清晰，可以试着关掉... [博文](#) | 来自: [程序员宜...](#)

程序员实用工具网站 阅读数 17万+

目录1、搜索引擎2、PPT3、图片操作4、文件共享5、应届生招聘6、程序员面试题库... [博文](#) | 来自: [不脱发的...](#)

从入门到精通，Java学习路线导航 阅读数 5万+

引言最近也有很多人来向我"请教"，他们大都是一些刚入门的新手，还不了解这个行... [博文](#) | 来自: [wangweijun](#)


细谈Activity四种启动模式 - zy_jibai - CSDN博客

Android---四种**Activity**的**启动模式** - A_droid的博客 - CSDN博客


细谈**Activity**四种**启动模式** - zy_jibai - CSDN博客

Android---四种**Activity**的**启动模式** - A_droid的博客 - CSDN博客


Activity启动模式阅读数 1575
一，启动模式分类：Standard(标准模式，默认)SingleTop(栈顶复用模式)SingleTask(...[博文](#) | 来自：[wangxp4...](#)




weixin_33671935
4716篇文章
[关注](#) 排名:千里之外



猴子搬来的救兵Castiel
79篇文章
[关注](#) 排名:5000+



qq_38217237
22篇文章
[关注](#) 排名:千里之外



肚皮会唱歌
17篇文章
[关注](#) 排名:千里之外

Activity四种**启动模式** - zlb_lover的博客 - CSDN博客

Activity的四种**启动模式**及特点 - zhiqiang.com的博客 - CSDN博客

Activity四种**启动模式** - zlb_lover的博客 - CSDN博客

activity的四种**启动模式** - u012675267的博客 - CSDN博客

鸿蒙 OS 的到来，能为我们改变什么？阅读数 3万+
作者|屠敏出品|CSDN（ID：CSDNnews）「鸿蒙初辟原无姓，打破顽空需悟空」，在...[博文](#) | 来自：[CSDN资讯](#)

shell-【技术干货】工作中编写shell脚本实践阅读数 2万+
在公司项目的开发过程中，需要编写shell脚本去处理一个业务，在编写过程中发现自...[博文](#) | 来自：[web洋仔](#)

Android的四种**启动模式** - xytong1991的博客 - CSDN博客

activity四种**启动模式**深解析 - micotale的博客 - CSDN博客

Activity的四种**启动模式**及特点 - zhiqiang.com的博客 - CSDN博客

Android的四种**启动模式** - xytong1991的博客 - CSDN博客

阿里资深工程师教你如何优化 Java 代码！阅读数 2万+
作者|王超责编|伍杏玲明代王阳明先生在《传习录》谈为学之道时说：私欲日生，如地...[博文](#) | 来自：[CSDN资讯](#)

docker学习笔记阅读数 1万+
docker学习笔记常用的镜像:dockerpullanibali/pytorch:cuda-10.0Docker是什么？Dock...[博文](#) | 来自：[pan_jinqu...](#)

Activity四种**启动**方式简述 - qq_38217237的博客 - CSDN博客

activity的四种**启动模式** - u012675267的博客 - CSDN博客

...**Activity**的四状态、七生命周期、和四**启动模式** - Flo..._CSDN博客

activity四种**启动模式**深解析 - micotale的博客 - CSDN博客

C/C++ 最易受攻击、70% 漏洞无效，揭秘全球开源组件安全现状阅读数 5959
开源是一种精神，更是一种合作共赢的模式。不过如今的开源生态虽然得以让诸多的...[博文](#) | 来自：[CSDN资讯](#)

C语言实现推箱子游戏阅读数 7万+
很早就想过做点小游戏了，但是一直没有机会动手。今天闲来无事，动起手来。过程...[博文](#) | 来自：[ZackSoc...](#)

Activity的四种**启动模式** - x_Danding的博客 - CSDN博客

Activity的四种**启动模式** - fjnu_se的博客 - CSDN博客

<div>我花了一夜用数据结构给女朋友写个H5走迷宫游戏</div> <div>起因又到深夜了，我按照以往在csdn和公众号写着数据结构！这占用了我大量的时间...</div>	阅读数 13万+
<div>面试最后一问：你有什么问题想问我？</div> <div>尽管，我们之前分享了这么多关于面试的主题：高薪必备的一些SpringBoot高级面试...</div>	阅读数 2万+
<div>python 程序员进阶之路：从新手到高手的100个模块</div> <div>在知乎和CSDN的圈子里，经常看到、听到一些python初学者说，学完基础语法后， ...</div>	阅读数 3万+
<div>史上最全的中高级JAVA工程师-面试题汇总</div> <div>史上最全的java工程师面试题汇总，纯个人总结，精准无误。适合中高级JAVA工程师...</div>	阅读数 3万+
<div>程序员必须掌握的核心算法有哪些？</div> <div>由于我之前一直强调数据结构以及算法学习的重要性，所以就有一些读者经常问我， ...</div>	阅读数 6万+
<div>程序员真是太太太太太有趣了！！</div> <div>网络上虽然已经有了很多关于程序员的话题，但大部分人对这个群体还是很陌生。我...</div>	阅读数 3万+
<div>武汉为什么进不了互联网第一梯队？</div> <div>作者 盛佳莹、张帆 本文经授权转自猎云网（ID：ilieyun） 从2011年以前双创在武汉...</div>	阅读数 1574
<div>别再翻了，面试二叉树看这 11 个就够了~</div> <div>写在前边 数据结构与算法： 不知道你有没有这种困惑，虽然刷了很多算法题，当我去...</div>	阅读数 6万+
<div>接班马云的为何是张勇？</div> <div>上海人、职业经理人、CFO 背景，集齐马云三大不喜欢的张勇怎么就成了阿里接班人...</div>	阅读数 2万+
<div>让程序员崩溃的瞬间（非程序员勿入）</div> <div>今天给大家带来点快乐，程序员才能看懂。来源：https://zhuanlan.zhihu.com/p/4706...</div>	阅读数 18万+
<div>用Python分析2000款避孕套，得出这些有趣的结论</div> <div>到现在为止，我们的淘宝教程已经写到了第四篇，前三篇分别是： 第一篇：Python模...</div>	阅读数 3万+
<div>IPv6 带来的反欺诈难题，程序员该如何破解？</div> <div>作者 威胁猎人 本文转载自威胁猎人(ID:ThreatHunter) IP是互联网最基础的身份标识...</div>	阅读数 977
<div>分享靠写代码赚钱的一些门路</div> <div>作者 mezod，译者 josephchang10如今，通过自己的代码去赚钱变得越来越简单，不...</div>	阅读数 4万+
<div>技术人员要拿百万年薪，必须要经历这9个段位</div> <div>很多人都问，技术人员如何成长，每个阶段又是怎样的，如何才能走出当前的迷茫， ...</div>	阅读数 9060
<div>面试官：兄弟，说说基本类型和包装类型的区别吧</div> <div>Java 的每个基本类型都对应了一个包装类型，比如说 int 的包装类型为 Integer，doub...</div>	阅读数 3万+
<div>C语言这么厉害，它自身又是用什么语言写的？</div> <div>这是来自我的星球的一个提问：“C语言本身用什么语言写的？”换个角度来问，其实是...</div>	阅读数 3万+
<div>一些实用的GitHub项目</div> <div>最近整理了一些在GitHub上比较热门的开源项目关于GitHub，快速了解请戳这里其中...</div>	阅读数 3万+
<div>八大排序(C语言)</div> <div>void BubbleSort();//冒泡 void SelectSort();//选择 void InsertSort();//直接插入 void She...</div>	阅读数 1894
<div>为什么说 Web 开发永远不会退出历史舞台？</div> <div>早在 PC 崛起之际，Web 从蹒跚学步一路走到了主导市场的地位，但是随着移动互联...</div>	阅读数 9093
<div>动画：用动画给面试官解释 TCP 三次握手过程</div> <div>作者 小鹿 来源 公众号：小鹿动画学编程 写在前边 TCP 三次握手过程对于面试是...</div>	阅读数 2万+



<div>2019诺贝尔经济学奖得主：贫穷的本质是什么？</div> <div>2019年诺贝尔经济学奖，颁给了来自麻省理工学院的 阿巴希·巴纳吉（Abhijit Vinayak...</div>	阅读数 5790	博文
<div>linux：最常见的linux命令（centOS 7.6）</div> <div>最常见，最频繁使用的20个基础命令如下： 皮一下，这都是干货偶，大佬轻喷 一、li...</div>	阅读数 8027	博文
<div>只因写了一段爬虫，公司200多人被抓！</div> <div>“一个程序员写了个爬虫程序，整个公司200多人被端了。”“不可能吧！” 刚从朋友听到...</div>	阅读数 9万+	博文
<div>别在学习框架了，那些让你起飞的计算机基础知识。</div> <div>我之前里的文章，写的大部分都是与计算机基础知识相关的，这些基础知识，就像我...</div>	阅读数 4万+	博文
<div>MySQL数据库—SQL汇总</div> <div>一、准备 下文整理常见SQL语句的用法，使用MySQL5.7测试，参考了尚硅谷MySQL...</div>	阅读数 8272	博文
<div>动画：用动画给女朋友讲解 TCP 四次分手过程</div> <div>作者 小鹿 来源 公众号：小鹿动画学编程 写在前边 大家好，我们又见面了，做为...</div>	阅读数 2万+	博文
<div>Python——画一棵漂亮的樱花树（不同种樱花+玫瑰+圣诞树喔）</div> <div>最近翻到一篇知乎，上面有不少用Python（大多是turtle库）绘制的树图，感觉很漂亮...</div>	阅读数 2万+	博文
<div>程序员不懂浪漫？胡扯！</div> <div>程序员男朋友你的程序员男朋友为你做过什么暖心的事情呢？我的男朋友是一个程序...</div>	阅读数 1万+	博文
<div>Java 8：一文掌握 Lambda 表达式</div> <div>本文将介绍 Java 8 新增的 Lambda 表达式，包括 Lambda 表达式的常见用法以及方...</div>	阅读数 1万+	博文
<div>Python自动化完成tb喵币任务</div> <div>2019双十一，tb推出了新的活动，商店喵币，看了一下每天都有几个任务来领取喵币...</div>	阅读数 1万+	博文
<div>从月薪3K的中专生，到身家千万的CTO！人生最大的对手，就是自己</div> <div>关注“技术领导力”博客，独家大厂干货推送 文/Daniel.W David坐在我对面，窗外是梦...</div>	阅读数 1万+	博文
<div>这应该是把计算机网络五层模型讲的最好是文章了，看不懂你打我</div> <div>帅地：用心写好每一篇文章！ 前言 天各一方的两台计算机是如何通信的呢？在成千上...</div>	阅读数 1万+	博文
<div>单点登录（SSO）</div> <div>一、SSO（单点登录）介绍 SSO英文全称Single SignOn，单点登录。SSO是在多个...</div>	阅读数 9814	博文
<div>漫话：什么是 https ?这应该是全网把 https 讲的最好的一篇文章了</div> <div>今天这篇文章，讲通过对话的形式，让你由浅入深着知道，为什么 Https 是安全的。 ...</div>	阅读数 1万+	博文
<div>史上最全的mysql基础教程</div> <div>启动与停止 启动mysql服务 sudo /usr/local/mysql/support-files/mysql.server start 停...</div>	阅读数 2714	博文
<div>为什么你学不会递归？告别递归，谈谈我的经验</div> <div>可能很多人在大一的时候，就已经接触了递归了，不过，我敢保证很多人初学者刚开...</div>	阅读数 1万+	博文
<div>大学四年，分享看过的优质书籍</div> <div>数据结构与算法是我在大学里第一次接触到的，当时学了很多其他安卓、网页之类的...</div>	阅读数 1万+	博文
<div>有哪些让程序员受益终生的建议</div> <div>从业五年多，辗转两个大厂，出过书，创过业，从技术小白成长为基层管理，联合几...</div>	阅读数 1万+	博文
<div>最近程序员频繁被抓，如何避免面向监狱编程！？</div> <div>最近，有关程序员因为参与某些项目开发导致被起诉，甚至被判刑的事件发生的比较...</div>	阅读数 4万+	博文
<div>一文搞懂什么是TCP/IP协议</div> <div>什么是TCP/IP协议? 计算机与网络设备之间如果要相互通信,双方就必须基于相同的方...</div>	阅读数 1万+	博文

各大公司在GitHub上开源投入排名分析

阅读数 1904

基于GitHub的数据进行分析各个公司在开源上的投入排名

[博文](#)

大学四年自学走来，这些私藏的实用工具/学习网站我贡献出来了

阅读数 6万+

大学四年，看课本是不可能一直看课本的了，对于学习，特别是自学，善于搜索网上...

[博文](#)

学习 Java 应该关注哪些网站？

阅读数 1万+

经常有一些读者问我：“二哥，学习 Java 应该关注哪些网站？”，我之前的态度一直是...

[博文](#)

哪些 Java 知识不需要再学了

阅读数 8940

张无忌在学太极拳的时候，他爹的师父张三丰告诫他一定要把之前所学习的武功全部...

[博文](#)

Vue + Spring Boot 项目实战（十四）： 用户认证方案与完善的访问拦截

阅读数 2400

本篇文章主要讲解 token、session 等用户认证方案的区别并分析常见误区，以及如何...

[博文](#)

c# codedom c#读取cad文件文本 c# 控制全局鼠标移动 c# temp 目录 bytes初始化 c# c#显示无焦点窗口 c# 类是否继承指定接口 c# 检查数据更新 c#连接扫描枪 c# 跟c++ 进程同步

没有更多推荐了，[返回首页](#)



1

YEAR

Zy_JiBai

TA的个人主页 >

私信

关注

原创

36

粉丝

66

获赞

89

评论

24

访问

4万+

等级:

博客 4

周排名:

3万+

积分:

900

总排名:

7万+

勋章:



最新文章

Android Studio debug无法打断点情况大全

jvmGC机制及引用类型详解（一）——java四种引用类型

图解java泛型的协变和逆变

从多态角度，理解kotlin的继承和重写

简易移动端爬虫实现pixabay网站图片搜索

分类专栏

 Android基础14篇

 Android小儿科9篇

 java6篇

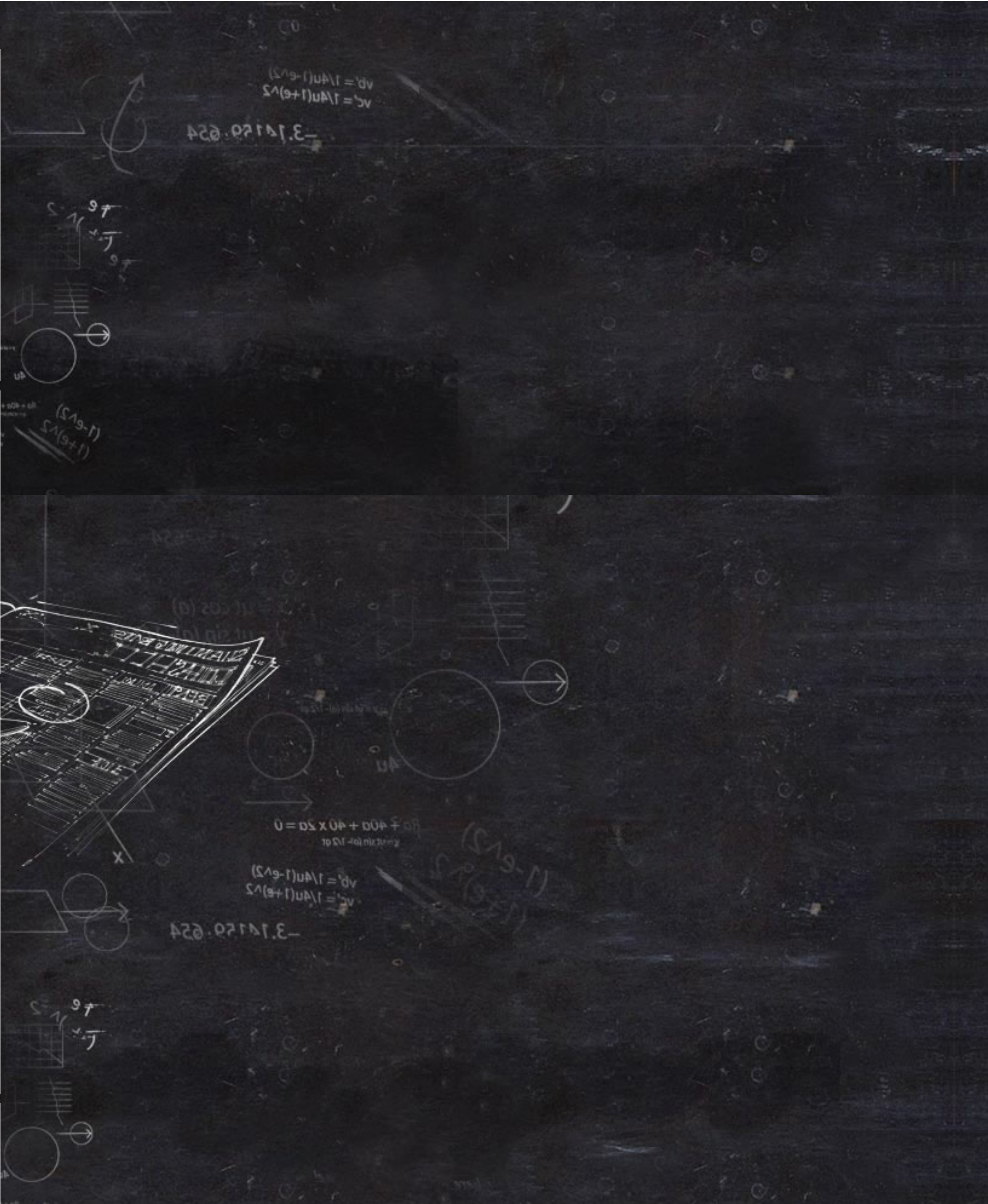
 设计模式2篇

 基础面试题5篇

展开

归档

2019年8月1篇



2019年5月	3篇
2019年4月	1篇
2019年1月	2篇
2018年12月	2篇
2018年11月	1篇
2018年9月	1篇
2018年8月	2篇

展开

热门文章

细谈Activity四种启动模式

阅读数 15270

最新高德地图使用——申请key、显示地图

阅读数 7659

Android自定义动态壁纸开发

阅读数 6836

高德地图使用——定位功能

阅读数 2591

面试题之内存泄漏相关

阅读数 2059

最新评论

细谈Activity四种启动模式

zy_jibai: [reply]qq_19694907[/reply] 这个我记得不太清楚了，好像当时是试过设置NEWTASK和...

细谈Activity四种启动模式

qq_19694907: 博主写的很好，不过我有两个问题： 1、在设置了taskAffnity的情况下 singleTask...

Java中字符串常见问题之Strin...

qq_41525021: 这段意思有点矛盾： 2.通过new创建String对象。这里其实只有一句话，无论常量...

细谈Activity四种启动模式

qiushuihhh: 写的挺好的，果然图文结合看的更清楚！！

细谈Activity四种启动模式

qq_43216321: Destroy拼错了



程序人生



CSDN资讯

👤 QQ客服 ✉ kefu@csdn.net
🗣 客服论坛 ☎ 400-660-0108
⌚ 工作时间 8:30-22:00

关于我们 | 招聘 | 广告服务 | 网站地图
🐾 百度提供站内搜索 京ICP备19004658号
©1999-2019 北京创新乐知网络技术有限公司

网络110报警服务 经营性网站备案信息
北京互联网违法和不良信息举报中心
中国互联网举报中心 家长监护 版权申诉

👤 QQ客服 ✉ kefu@csdn.net
🗣 客服论坛 ☎ 400-660-0108
⌚ 工作时间 8:30-22:00

关于我们 | 招聘 | 广告服务 | 网站地图
🐾 百度提供站内搜索 京ICP备19004658号

