**Android中重启程序自己的几种方法**

2017年05月10日 20:11:06 [又双叒叕](https://me.csdn.net/u014471160) 阅读数：11284

1. 

Intent intent = new Intent(this, Object.class);

startActivity(intent);  
  
2. 通过ActivityManager来重新启动应用程序：  
  
ActivityManager manager = （ActivityManager）this.getSystemService（Context.ACTIVITY\_SERVICE）;  
manager.restartPackage("com.example.test");  
  
  
3. 通过flag来实现：  
private void restartApplication() {  
        final Intent intent = getPackageManager().getLaunchIntentForPackage(getPackageName());  
        intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);  
        startActivity(intent);  
  
}  
  
  
  
4. 利用PendingIntent  
  
Intent mStartActivity = new Intent(context, StartActivity.class);  
int mPendingIntentId = 123456;  
PendingIntent mPendingIntent = PendingIntent.getActivity(context, mPendingIntentId,    mStartActivity, PendingIntent.FLAG\_CANCEL\_CURRENT);  
AlarmManager mgr = (AlarmManager)context.getSystemService(Context.ALARM\_SERVICE);  
mgr.set(AlarmManager.RTC, System.currentTimeMillis() + 100, mPendingIntent);  
System.exit(0);

**Android 重启应用程序 的两种方式**

2017年04月12日 09:51:21 [总Li丶](https://me.csdn.net/wxd_beijing) 阅读数：7025

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/wxd\_beijing/article/details/70139239

/\*\*

\* 重新启动App -> 杀进程,会短暂黑屏,启动慢

\*/

public void restartApp() {

*//启动页*

Intent intent = new Intent(BaseApplication.instance(), SplashActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

BaseApplication.instance().startActivity(intent);

android.os.Process.killProcess(android.os.Process.myPid());

}

/\*\*

\* 重新启动App -> 不杀进程,缓存的东西不清除,启动快

\*/

public void restartApp2() {

final Intent intent = BaseApplication.instance().getPackageManager()

.getLaunchIntentForPackage(BaseApplication.instance().getPackageName());

intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);

BaseApplication.instance().startActivity(intent);

}

# 如何让你的 app 在后台被干掉后优雅的重新启动

阅读 6127

收藏 377

2017-03-30

原文链接：[www.jianshu.com](https://link.juejin.im/?target=http%3A%2F%2Fwww.jianshu.com%2Fp%2F1946bd4f3bb5)

##### 背景介绍

作为一个Android开发者，肯定会遇到这样的一种情况，用户在玩着你开发的app时，突然有微信来消息了，切换到了微信，然后还在微信逗留看视频啊，聊天啊，刷朋友圈啊等等的，你所开发的app就出于后台了，这个时候就很容易出现手机内存不足，app被内存回收干掉的情况了，等用户终于聊完天，刷完朋友圈，回来app的时候，就会进行app的自我恢复了，如果开发者处理不好，就会出现崩溃的情况了，而且肯定会出现返回的时候一瞬间白屏，然后再显示出来，这样的用户体验非常的不好。那我们应该怎样去解决这样的状况呢？扯了那么多，我们的文章就正式开始啦！

##### 开始

首先要介绍下Android中activity的四种启动模式（就当作复习一下旧知识吧，资料来源于网络总结）：

Standard：是默认的也是标准的Task模式，在没有其他因素的影响下，使用此模式的Activity，会构造一个Activity的实例，加入到调用者的Task栈中去，对于使用频度一般开销一般什么都一般的Activity而言，standard模式无疑是最合适的，因为它逻辑简单条理清晰，所以是默认的选择。

\*

singleTop：基本上于standard一致，仅在请求的Activity正好位于栈顶时，有所区别。此时，配置成singleTop的Activity，不再会构造新的实例加入到Task栈中，而是将新来的Intent发送到栈顶Activity中，栈顶的Activity可以通过重载onNewIntent来处理新的Intent（当然，也可以无视...）。这个模式，降低了位于栈顶时的一些重复开销，更避免了一些奇异的行为（想象一下，如果在栈顶连续几个都是同样的Activity，再一级级退出的时候，这是怎么样的用户体验...），很适合一些会有更新的列表Activity展示。一个活生生的实例是，在Android默认提供的应用中，浏览器（Browser）的书签Activity（BrowserBookmarkPage），就用的是singleTop。

\*

singleTask：配置了这个属性的activity，最多仅有一个实例存在，而且，它在根的task中，在之后的被杀死重启的过程中我们会利用到这个配置，也就是我们的主界面MainActivity。

\*

singleInstance：跟上面的singleTask基本上是一样的，但是，singleInstance的Activity，是它所在栈中仅有的一个Activity，如果涉及到的其他Activity，都移交到其他Task中进行，在实际开发中这个是用得比较少的。

这个是activity的生命周期：

咱们就不多介绍这个生命周期了，相信都熟悉不过了，有想了解的自行Google或者百度吧。

###### 重点

接下来是我们的重点：程序如果在后台被杀死之后，我们怎么去处理？是立刻恢复还是重新启动？哪个方法更适合我们？

首先，我们得知道，为什么程序会在后台被干掉的？我们又没有手动关闭程序。

app在后台被强杀，是在内存不足的情况下被强制释放了，也有一些恶心的rom会强制杀掉那些后台进程以释放缓存以提高所谓的用户体验。（注：当你的代码写得混乱、冗余，而且非常消耗内存的时候，那你的app在后台运行时将会比较容易被系统给干掉的，所以从现在开始要约束自己要养成良好的编码习惯和注意内存泄漏的问题）

我们都觉得android rom很恶心，但同时还是用些更恶心的手法去绕开这些瓶颈。乱，是因为在最上层没有一个很好的约束，这也是开源的弊端。anyway。我们还是得想破脑袋来解决这些问题，否则饭碗就没了。

我们现在来重现这个熟悉的一幕：

假设： App A -> B -> C

在C activity中点Home键后台运行，打开ddms，选中该App进程，强杀。

然后从“最近打开的应用”中选中该App，回到的界面是C activity，假设App中没有静态变量，这个时候是不会crash的，点击返回到B，这个时候也只是短暂白屏后显示B界面。但如果B中有引用静态变量，并想要获取静态变量中的某个值时，就NullPointer了。

以上复现的流程就几个点，我们展开说下：

当应用被强杀，整个App进程都是被杀掉了，所有变量全都被清空了。包括Application实例。更别提那些静态变量了。

虽然变量被清空了，但Android给了一些补救措施。activity栈没有被清空，也就是说A -> B -> C这个栈还保存了，只是ABC这几个activity实例没有了。所以回到App时，显示的还是C页面。另外当activity被强杀时，系统会调用onSaveInstance去让你保存一些变量，但我个人觉得面对海量的静态变量，这个根本不够用。返回到B会白屏，是因为B要重绘，重走onCreate流程，渲染上需要点时间，所以会白屏了。

大概是以上这些点。如果App中没有静态变量的引用，那就不用出现NullPointer这个crash，也就不需要解决。一旦你有静态变量，或者有些Application的全局变量，那就很危险了。比如登录状态，user profile等等。这些值都是空了。

肯定会有人说，这没关系啊，所有的静态变量都改到单例去不就好了吗？然后附加上一些持久化cache，空了再取缓存就ok了嘛。嗯，这肯定也是一个办法，但是这样的束手束脚对开发来说也是痛苦，至少需要多50%的编码时间才能全部cover。另外，还有那么多帮你挖坑的队友，难省心啊。

既然App都被强杀了，干嘛不重新走第一次启动的流程呢，别让App回到D而是启动A，这样所有的变量都是按正常的流程去初始化，也就不会空指针了，对吧？有人说这方案用户体验一点都不好呀。但哪有十全十美的事呢，是重走流程好，还是一点一个NullPointer好？好好去沟通，相信产品也不会为难你的。当然你也可以拿来举例，iOS在最近打开的应用里杀了某个App，重新点击那个App，还是会重走流程的啊。

如果你说用户已经打开了C界面，所以重新打开的是是恢复到C界面，这样的用户体验会更好啊，如果你是这样认为的，那你很多时间都是在防止恢复的时候不让你的app crash了，与其这样，还不如让整个app重新走整个流程呢，这样更省时间，而且这样也不用担心随时都会崩溃的情况，难道这样的用户体验不会更好吗？

那且想想如何让它不回到C而是重走流程呢？也就是说中断C的初始化而回到A，并且按back键，不会回到C，B。考虑一下。

我们先实例化这个场景吧。  
A 为App的启动页  
B 为首页  
C 为二级页面

把首页launchMode设置为singleTask，具体为什么上面介绍activity的启动模式的时候已经介绍了singleTask的作用了。

在BaseActivity中onCreate中判断App是否被强杀，强杀就不往下走，直接重走App流程。

首页起一个承接或者中转的作用，所有跨级跳转都需要通过首页来完成。

再给个提示，以上场景的解决方案也可以用于解决其它相关问题：  
在任意页面退出App  
在任意页面返回到首页  
其实最重要的知识点就是launchMode

##### 具体实现

AppStatusConstant

**public** **static** **final** int STATUS\_FORCE\_KILLED = -1; //应用放在后台被强杀了

**public** **static** **final** int STATUS\_NORMAL = 2; //APP正常态//intent到MainActivity 区分跳转目的

**public** **static** **final** **String** KEY\_HOME\_ACTION = "key\_home\_action";//返回到主页面

**public** **static** **final** int ACTION\_BACK\_TO\_HOME = 0; //默认值

**public** **static** **final** int ACTION\_RESTART\_APP = 1;//被强杀

AppStatusManager

**public** **int** appStatus= AppStatusConstant.STATUS\_FORCE\_KILLED; //APP状态 初始值为没启动 不在前台状态

**public** **static** AppStatusManager appStatusManager;

**private** **AppStatusManager**() {

}

**public** **static** AppStatusManager **getInstance**() {

**if** (appStatusManager == **null**{

appStatusManager = **new** AppStatusManager();

}

**return** appStatusManager;

}

**public** **int** **getAppStatus**() {

**return** appStatus;

}

**public** **void** **setAppStatus**(**int** appStatus) {

**this**.appStatus = appStatus;

}

BaseActivity（大致内容）

**switch** (AppStatusManager.getInstance().getAppStatus()) {

**case** AppStatusConstant.STATUS\_FORCE\_KILLED:

restartApp();

**break**;

**case** AppStatusConstant.STATUS\_NORMAL:

setUpViewAndData();

**break**;

}

**protected** **abstract** **void** **setUpViewAndData**();

**protected** **void** **restartApp**() {

Intent intent = **new** Intent(**this**, MainActivity.class);

intent.putExtra(AppStatusConstant.KEY\_HOME\_ACTION,AppStatusConstant.ACTION\_RESTART\_APP);

startActivity(intent);

}

###### 每一个继承于父activity的都不要在oncreat中实现界面初始化和数据的初始化，因为如果被杀死之后，回来会走一次正常的生命流程的。

StartPageActivity配置（在oncreat()方法配置，并且在super()前）：

**AppStatusManager**.getInstance().setAppStatus(AppStatusConstant.STATUS\_NORMAL); //进入应用初始化设置成未登录状态

MainActivity（配置了singleTask的主界面）

**@Override**

**protected** **void** **restartApp**() {

**super**.restartApp();

Toast.makeText(getApplicationContext(),"应用被回收重启",Toast.LENGTH\_LONG).show();

startActivity(**new** Intent(**this**, StartPageActivity.class));

finish();

}

**@Override**

**protected** **void** **onNewIntent**(Intent intent) {

**super**.onNewIntent(intent);

**int** action = intent.getIntExtra(AppStatusConstant.KEY\_HOME\_ACTION,AppStatusConstant.ACTION\_BACK\_TO\_HOME);

**switch** (action) {

**case** AppStatusConstant.ACTION\_RESTART\_APP:

restartApp();

**break**;

}

}

当应用打开的时候，启动StartPageActivity，然后设置app的status为normal状态，记住，一定要设置，因为默认的是被杀死的状态的。

当应用被杀死之后，所有数据都会被回收，所以之前设置的app status也会置于默认状态，即杀死状态，所以再次打开app的时候，status为杀死状态，就会走重启的流程，这里为什么要先跳转到MainActivity呢？就是因为MainActivity配置为了Sing了Task，当跳转到这个界面时，MainActivity就会置于Activity Task的最上层，其他的Activity将会被默认销毁掉，利用这种技巧去销毁其他的Activity，最后才是重新启动StartPageActivity。整个流程就是这样了。

大致的实现就如上所述了，我所倡导的宗旨就是花最少的时间，写最好的代码，实现最好的体验！之前也参考过很多网上大神们的实现方式，但是我觉得以上实现的应该是比较完整的一种了

**android重启应用（应用重新启动自身）**

2013年02月15日 23:12:20 [赵小刚](https://me.csdn.net/androidzhaoxiaogang) 阅读数：29070

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/androidzhaoxiaogang/article/details/8582013

1. private void restartApplication() {
2. final Intent intent = getPackageManager().getLaunchIntentForPackage(getPackageName());
3. intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);
4. startActivity(intent);
5. }