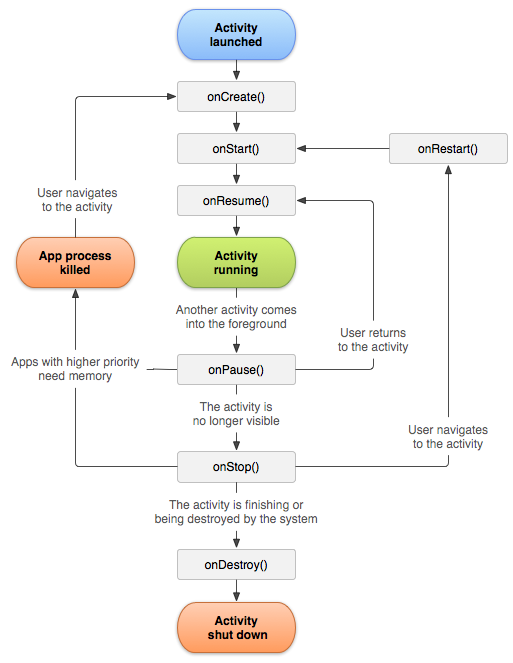
|  |
| --- |
| 1. Android中的activity全都归属于task管理(task是一个具有栈结构的容器),task是多个activity的集合,android默认情况会为每个App维持一个task来存放app的所有activity(当然这只是默认情况),task的默认name为该app的packagename(包名).   2、生命周期 onCreate() -> onStart() -> onResume()  onPause() -> onStop() -> onDestroy()  onReStart()  onSaveInstanceState()  onRestoreInstanceState()  3、开始时，A被实例化，执行的回调有A:onCreate -> A:onStart -> A:onResume。  当用户点击A中按钮来到B时，假设B全部遮挡住了A，将依次执行A:onPause -> B:onCreate -> B:onStart -> B:onResume -> A:onStop。  此时如果点击Back键，将依次执行B:onPause -> A:onRestart -> A:onStart -> A:onResume -> B:onStop -> B:onDestroy。  4、四种启动模式:  标准模式standard、栈顶复用模式singleTop、  栈内复用模式（全栈单例）singleTask、全局唯一模式（全系统单例）singleInstance。 |

## 1、task栈的机制

相信大家都知道Android中的4大组件(Activity活动,Service服务,ContentProvider内容提供者,BroadcastReceiver广播接收器),Activity是我们使用最多的也是最基本的组件,Activity提供窗口和用户进行交互。

Android中的activity全都归属于task管理(task是一个具有栈结构的容器),task是多个activity的集合,android默认情况会为每个App维持一个task来存放app的所有activity(当然这只是默认情况),task的默认name为该app的packagename(包名).

系统是通过task(栈)的方式来管理activity的,当一个新的activity开始的时候,该activity会被放置在堆栈(back stack)的顶部,成为正在运行的活动,之前的activity始终保持低于它在堆栈,而不会出现在前台。



当然我们也可以在AndroidMainfest.xml中申明activity的 taskAffinity 属性来自定义task,但是不建议使用,因为如果其他app也申明了同样的task,那个app可能启动到你的activity,这样会带来各种安全问题(比如拿到你的Intent)。

affinity是指Activity的归属，Activity与Task的吸附关系，也就是该Activity属于哪个Task。一般情况下在同一个应用中，启动的Activity都在同一个Task中，它们在该Task中度过自己的生命。每个Activity都有taskAffinity属性，这个属性指出了它希望进入的Task。如果一个Activity没有显式的指明taskAffinity，那么它的这个属性就等于Application指明的taskAffinity，如果Application也没有指明，那么该taskAffinity的值就等于应用的包名。我们可以通过在元素中增加taskAffinity属性来为某一个Activity指定单独的affinity。这个属性的值是一个字符串，可以指定为任意字符串，但是必须至少包含一个”.”，否则会报错。

## 2、Activity生命周期

当我们打开一个新的activity实例的时候,系统会以此调用

onCreate() -> onStart() -> onResume() 然后开始running

当activity在running的时候如果被覆盖(打开新的activity,或者被锁屏,但是它依然在前台运行,lost focus but is still visible),系统就会调用onPause();

在onpause()方法中,我们通常会提交未保存的更改到持久化数据,停止动画和其他东西.但是我们要知道现在这个activity还是完全活着(它保存所有的状态和成员信息,并且保存到窗口管理器的连接)。

1.启动Activity：系统会先调用onCreate方法，然后调用onStart方法，最后调用onResume，Activity进入运行状态。

2.当前Activity被其他Activity覆盖其上或被锁屏：系统会调用onPause方法，暂停当前Activity的执行。

3.当前Activity由被覆盖状态回到前台或解锁屏：系统会调用onResume方法，再次进入运行状态。

4.当前Activity转到新的Activity界面或按Home键回到主屏，自身退居后台：系统会先调用onPause方法，然后调用onStop方法，进入停滞状态。

5.用户后退回到此Activity：系统会先调用onRestart方法，然后调用onStart方法，最后调用onResume方法，再次进入运行状态。

6.当前Activity处于被覆盖状态或者后台不可见状态，即第2步和第4步，系统内存不足，杀死当前Activity，而后用户退回当前Activity：再次调用onCreate方法、onStart方法、onResume方法，进入运行状态。

7.用户退出当前Activity：系统先调用onPause方法，然后调用onStop方法，最后调用onDestory方法，结束当前Activity。

onSaveInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，此方法会被调用；(2)在用户改变屏幕方向时，此方法会被调用；(3)在当前Activity跳转到其他Activity或者按Home键回到主屏，自身退居后台时，此方法会被调用。第一种情况我们无法保证什么时候发生，系统根据资源紧张程度去调度；第二种是屏幕翻转方向时，系统先销毁当前的Activity，然后再重建一个新的，调用此方法时，我们可以保存一些临时数据；第三种情况系统调用此方法是为了保存当前窗口各个View组件的状态。onSaveInstanceState的调用顺序是在onStop之前。

（疑问：该方法默认保存的哪些数据？屏幕翻转功能设置涉及哪些？如何给APP添加方法）

onRestoreInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，然后用户又回到了此Activity，此方法会被调用；(2)在用户改变屏幕方向时，重建的过程中，此方法会被调用。我们可以重写此方法，以便可以恢复一些临时数据。onRestoreInstanceState的调用顺序是在onStart之后。

activity被回收的状态和信息保存和恢复过程

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

if(savedInstanceState!=null){ //判断是否有以前的保存状态信息

savedInstanceState.get("Key");

}

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

@Overrideprotected void onSaveInstanceState(Bundle outState) {

// TODO Auto-generated method stub

//可能被回收内存前保存状态和信息，

Bundle data = new Bundle();

data.putString("key", "last words before be kill");

outState.putAll(data);

super.onSaveInstanceState(outState);

}

@Overrideprotected void onRestoreInstanceState(Bundle savedInstanceState) {

// TODO Auto-generated method stub

if(savedInstanceState!=null){ //判断是否有以前的保存状态信息

savedInstanceState.get("Key");

}

super.onRestoreInstanceState(savedInstanceState);

}

}

onSaveInstanceState()方法,在activity被回收之前调用,用来保存自己的状态信息,以便回收后重建时恢复数据(在onCreate()或onRestoreInstanceState()中恢复).旋转屏幕重建activity会调用该方法,但其他情况在onRause()和onStop()状态的activity不一定会调用,  
下面是官方文档说明:

One example of when onPause and onStop is called and not this method is when a user navigates back from activity B to activity A: there is no need to call onSaveInstanceState on B because that particular instance will never be restored, so the system avoids calling it. An example when onPause is called and not onSaveInstanceState is when activity B is launched in front of activity A: the system may avoid calling onSaveInstanceState on activity A if it isn't killed during the lifetime of B since the state of the user interface of A will stay intact.

也就是说,系统灵活的来决定会不会调用该方法,但是如果要调用就一定发生在onStop方法之前,但不保证发生在onPase的前面还是后面.

onRestoreInstanceState()该方法在onStart和onPostCreate之间调用,当然我们也可以在onCreat中恢复之前我们在onSaveInstanceState()中保存下来的数据,但是我们有时候需要在初始化布局完成之后再恢复数据,这是就可以在onRestoreInstanceState()中恢复数据.  
 onPostCreate():一般我们都没有实现这个方法,它的作用是在代码开始运行之前,调用系统做最后的初始化工作。

## 3、activity的四种状态

**运行状态、暂停状态、停止状态、销毁状态。**

启动状态：Activity的启动状态很短暂，当Activity启动后便会进入运行状态。

运行状态：当活动位于栈顶时，就处于运行状态，系统最不愿意回收这样的活动，回收会导致很差的用户体验。个人认为就是程序闪退，崩溃等。Activity在此状态时处于屏幕最前端，它是可见、有焦点的，可以与用户进行交互。如单击、长按等事件。

暂停状态：在某些情况下，Activity对用户来说仍然可见，但它无法获取焦点，用户对它操作没有没有响应，此时它处于暂停状态。当活动不再处于栈顶，但是对用户依然可见时。不是说在栈顶的元素才对用户可见吗？这是因为不是每一个活动都是覆盖整个屏幕的，一些对话框形式的活动就只会占用一部分而已，比如Dialog。系统也是不愿意回收处于暂停状态的活动的，除非在万不得已的情况下。

停止状态：当Activity完全不可见时，它处于停止状态，但仍然保留着当前的状态和成员信息。如系统内存不足，那么这种状态下的Activity很容易被销毁。活动不再处于栈顶，并且完全不可见的时候，就会处于停止状态。但系统会保存相应的状态和变量，但是当内存不足时，还是会回收这一类的活动的。

销毁状态：当Activity处于销毁状态时，将被清理出内存。当活动从栈中移除以后就变成了销毁状态。系统最乐意回收这类活动，以此来保证充足的内存。

Activity是由Activity栈进管理，当来到一个新的Activity后，此Activity将被加入到Activity栈顶，之前的Activity位于此Activity底部。**Acitivity一般意义上有四种状态**：

（1）当Activity位于栈顶时，此时正好处于屏幕最前方，此时处于***运行状态***；

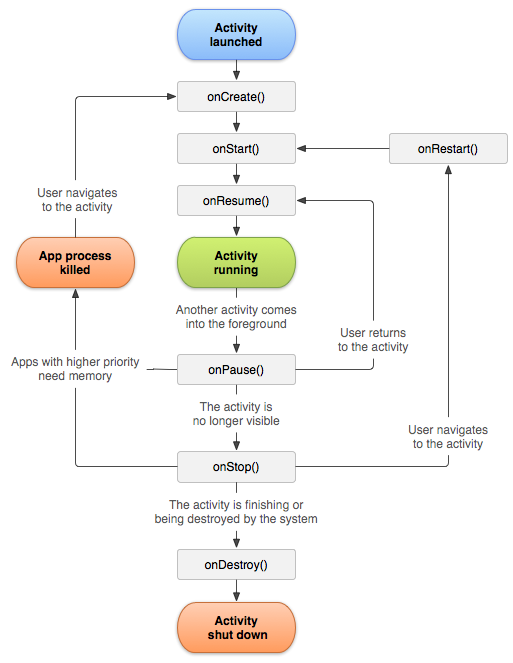
（2）当Activity失去了焦点但仍然对用于可见（如栈顶的Activity是透明的或者栈顶Activity并不是铺满整个手机屏幕），此时处于***暂停状态***；（失去焦点，就是不能响应事件了。焦点，计算机程序语言中所谓的焦点，就是关注的区域,当前光标被激活的位置，是哪个控件被选中，可以被操作。）

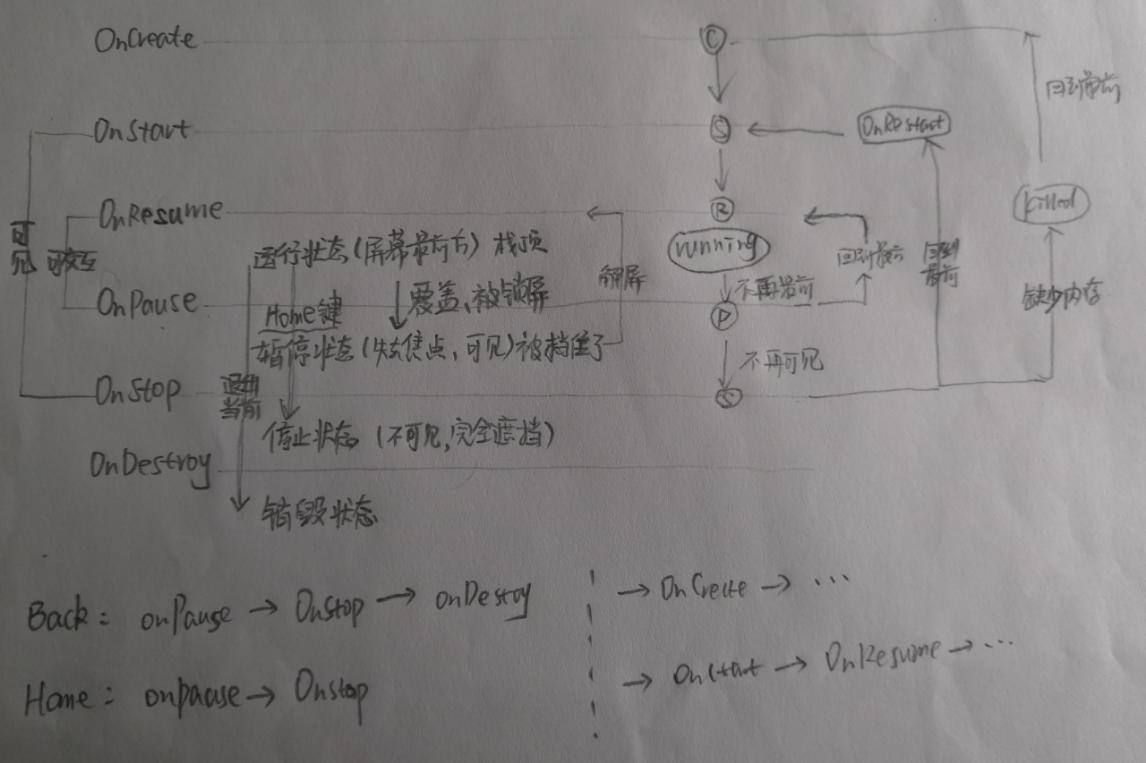
（3）当Activity被其他Activity完全遮挡，此时此Activity***对用户不可见***，此时处于***停止状态***；

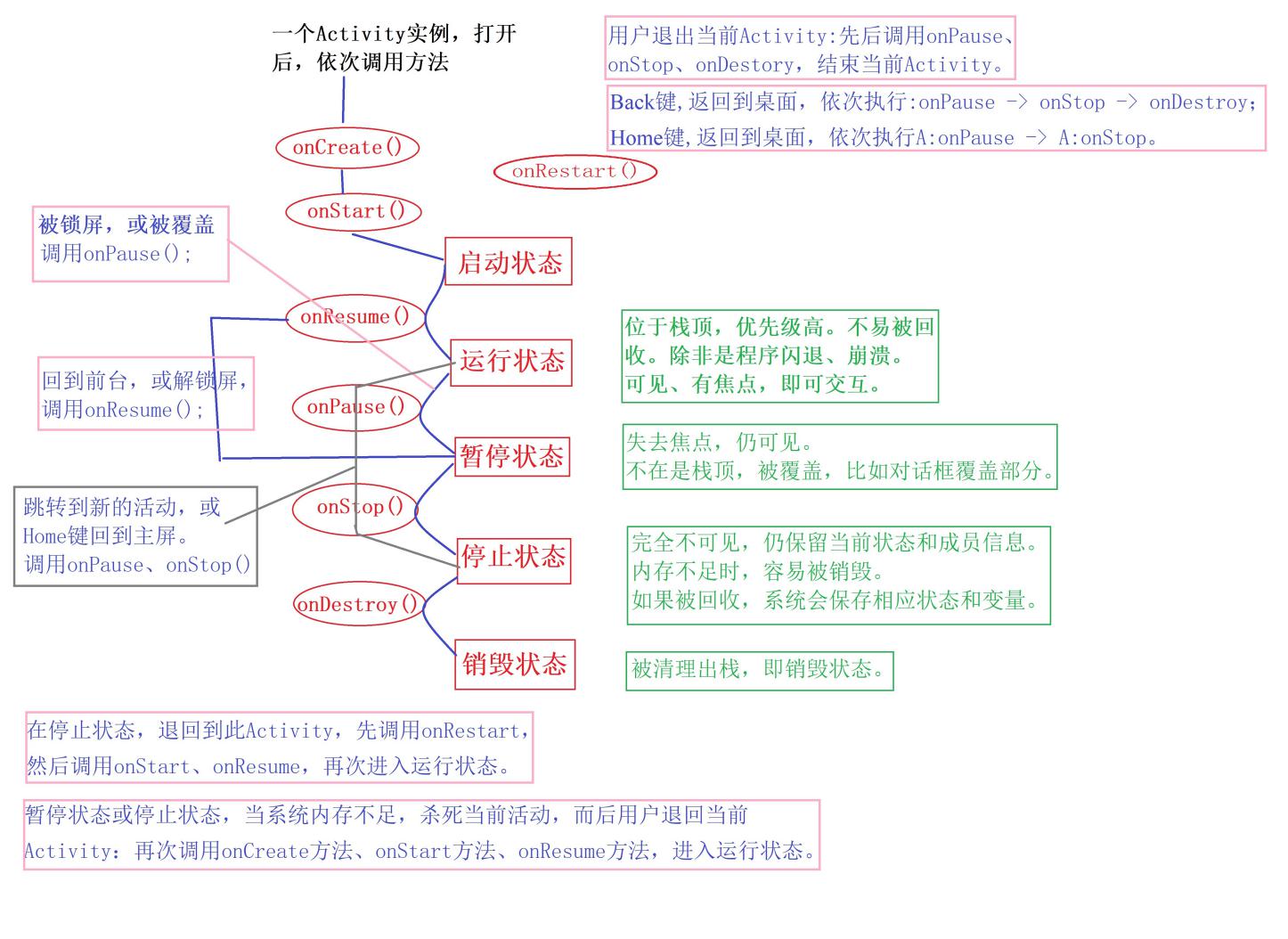
（4）当Activity由于人为或系统原因（如低内存等）被销毁，此时处于***销毁状态***；

在每个不同的状态阶段，**Adnroid系统对Activity内相应的方法进行了回调。**因此，我们在程序中写Activity时，一般都是继承Activity类并重写相应的回调方法。

## 4、不同状态之间的切换，回调方法







（1）Activity实例是由系统自动创建，并在不同的状态期间回调相应的方法。一个最简单的完整的Activity生命周期会按照如下顺序回调：onCreate -> onStart -> onResume -> onPause -> onStop -> onDestroy。称之为entire lifetime。完整生存期

（2）当执行onStart回调方法时，Activity开始被用户所见（也就是说，onCreate时用户是看不到此Activity的，那用户看到的是哪个？当然是此Activity之前的那个Activity），一直到onStop之前，此阶段Activity都是被用户可见，称之为visible lifetime。可见生存期

（3）当执行到onResume回调方法时，Activity可以响应用户交互，一直到onPause方法之前，此阶段Activity称之为foreground lifetime。前台生存期

（4）切换AB两个活动回调的方法

在实际应用场景中，假设A Activity位于栈顶，此时用户操作，从A Activity跳转到B Activity。那么对AB来说，具体会回调哪些生命周期中的方法呢？回调方法的具体回调顺序又是怎么样的呢？

开始时，A被实例化，执行的回调有A:onCreate -> A:onStart -> A:onResume。

当用户点击A中按钮来到B时，假设B全部遮挡住了A，将依次执行A:onPause -> B:onCreate -> B:onStart -> B:onResume -> A:onStop。

此时如果点击Back键，将依次执行B:onPause -> A:onRestart -> A:onStart -> A:onResume -> B:onStop -> B:onDestroy。

（5）Back键和Home键

至此，Activity栈中只有A。在Android中，有两个按键在影响Activity生命周期这块需要格外区分下，即Back键和Home键。我们先直接看下实验结果：

此时如果按下Back键，系统返回到桌面，并依次执行A:onPause -> A:onStop -> A:onDestroy。

此时如果按下Home键（非长按），系统返回到桌面，并依次执行A:onPause -> A:onStop。由此可见，Back键和Home键主要区别在于是否会执行onDestroy。

（6）重写Back键的回调函数，实现Home的功能

由于Android本身的特性，使得现在不少应用都没有直接退出应用程序的功能，按照一般的逻辑，当Activity栈中有且只有一个Activity时，当按下Back键此Activity会执行onDestroy，那么下次点击此应用程图标将从重新启动，因此，当前不少应用程序都是采取如Home键的效果，当点击了Back键，系统返回到桌面，然后点击应用程序图标，直接回到之前的Activity界面，这种效果是怎么实现的呢？

通过重写按下Back键的回调函数，转成Home键的效果即可。

@Override

public void onBackPressed() {

Intent home = new Intent(Intent.ACTION\_MAIN);

home.addCategory(Intent.CATEGORY\_HOME);

startActivity(home);

}

当然，此种方式通过Home键效果强行影响到Back键对Activity生命周期的影响。注意，此方法只是针对按Back键需要退回到桌面时的Activity且达到Home效果才重写。

或者，为达到此类效果，Activity实际上提供了直接的方法。

activity.moveTaskToBack(true);

moveTaskToBack()此方法直接将当前Activity所在的Task移到后台，同时保留activity顺序和状态。

（7）不保留活动设置选择

在之前的项目开发过程中，当时遇到一个很奇怪的问题：手机上的“开发者选项”中有一个“不保留活动”的设置，当开启此设置，手机上的设置提示是“用户离开后即销毁每个活动”，开启后，对于其他的应用程序是从A Acticity到B Activity，然后Back键回到A，此时，其他应用程序只是先白屏（有可能黑屏等，取决于主题设置）一下，然后A开始可见，但是我的应用程序中出现的一个结果却是直接返回到了桌面。一开始百思不得其解。最后终于定位出问题。首先，我们需要明确开启此设置项后对Activity生命周期的影响。开启此设置项后，当A到B时，假设B全部遮挡住了A，将依次执行A:onPause -> B:onCreate -> B:onStart -> B:onResume -> A:onStop -> A:onDestroy。是的，A在系统原本的生命周期回调中增加了onDestroy。此即“用户离开后即销毁每个活动”的含义。但此时需要注意的是，只要没有认为的调用A的finish()方法，虽然A执行了onDestroy，但Activity栈中依然保留有A，此时B处于栈顶。那么在B中按Back键回到A时，将依次执行：B:onPause -> A:onCreate -> A:onStart -> A:onResume -> B:onStop -> B:onDestroy。没错，A从onCreate开始执行了。此处也就解释了为什么A可能会出现白屏（或黑屏等）一下的原因了。

## 5、Home、Back、解锁屏

1. android的手机的back键默认行为是finish处于前台的Activity的即Activity的状态为Destroy状态，再次启动该Activity是从onCreate开始的。

返回键：onPause-->onStop-->onDestroy。

1. 而Home键默认是stop前台的Activity即状态为onStop而不是Destroy,若再次启动它，则是从OnStart-->OnResume开始的，即会保持上次Activityd的状态。

返回桌面键：onPause-->onStop。

1. back键也有例外的，按back键不会关闭Activity的，比如播放音乐，按了back键之后仍可以继续播放音乐，这是Music这支app已经重写了back键的事件处理。
2. 为什么需要Home键和Back键呢？一个使得Activity 为Stop一个使得为Destroy呢？我想原因的原因在于是android也是一个多任务的操作系统，通过Home键切换不同的任务，而通过back关闭任务中的某一个活动。
3. 解锁屏的情况：锁屏是从运行状态running执行onPause()到暂停状态；解锁就是执行OnResume回到运行状态。

## 6、如何实现完全的退出程序呢？

所有的活动我们都destroy掉，要怎么操作？

别的程序设计一般是如何考虑的？不考虑这个问题会有什么办法？

## 回调方法的方法体

在系统内存不足的时候可能不会执行onStop方法，因此程序状态的保存、独占设备和动画的关闭、以及一些数据的保存最好在onPause中进行，但要注意不能太耗时。

1. onCreate(Bundle)

除了初始化成员变量外， onCreate（Bundle）方法还承担着另外两个职责：恢复之前保存的状态并调用setContentView()方法。

（2）onStart()

监听事件注册、功能流、异步功能流、资源分配。

（3）onResume()

不需要覆盖onResume（）方法。除非，用于恢复播放动画和视频。

1. onPause()

不需要重写onPause（）。特殊，相机的使用，由于相机是所有应用程序共享的单一资源，因此通常您会想要在onPause（）方法中释放它。

1. onStop()

注销所有观察者和监听者，并释放onStart（）中分配的所有资源。

在onStop()中操作，而不是在onDestroy()操作的好处是：在destroy操作的坏处，这将是对用户电池寿命的不负责任的浪费，而且在这种情况下，应用消耗的内存会逐渐增多，应用进程被OOM（Out Of Memory）Killer杀死的可能性也会增大。

1. onDestroy()

永远都不要覆盖重写这个方法。

## 四种启动模式

标准模式、栈顶复用模式、栈内复用模式（全栈单例）、全局唯一模式（全系统单例）。

启动模式是什么?

简单点说就是:定义activity实例与task的关联方式

为什么我们要定义启动模式呢?

为了实现默认启动(standard)模式之外的需求:

让某个activity启动一个新的task(而不是被放入当前task)

让activity启动时只调用已有的某个实例(而不是在back stack(之前我们有提到过哟) 顶创建一个新的实例)

当用户离开task时只想保留根activity,而back stack中的其他activity都要清空.

怎样定义启动模式

定义启动模式的方法有2种:

使用manifest文件定义

使用Intent标志定义

（1）使用manifest文件定义启动模式:  
 在 manifest 文件中activity声明时，利用 activity 元素的 launchMode 属性来设定 activity 与 task 的关系。

<activity

．．．．．．

android:launchMode="standard" >

．．．．．．．</activity>

注意： 你用 launchMode 属性为 activity 设置的模式可以被启动 activity 的 intent 标志所覆盖,代码的优先级最高。

现在我们知道了怎么定义启动模式了,但是有哪些启动模式呢?

standard (默认模式，标准模式)  
 当通过这种模式启动activity时,Android总会为目标 Activity创建一个新的实例(之前有过也会重新创建),并将该Activity添加到当前Task栈中。这种方式不会启动新的Task,只是将新的 Activity添加到原有的Task中。

singleTop(单顶，栈顶复用模式)  
 该模式和standard模式基本一致,但有一点不同:当将要被启动的Activity已经位于Task栈顶时,系统不会重新创建目标Activity实例,而是直接复用Task栈顶的Activity。

singleTask（单例，栈内复用模式）  
 Activity在同一个Task内只有一个实例。  
 如果将要启动的Activity不存在,那么系统将会创建该实例,并将其加入Task栈顶；

如果将要启动的Activity已存在,且存在栈顶,直接复用Task栈顶的Activity。

如果Activity存在但是没有位于栈顶,那么此时系统会把位于该Activity上面的所有其他Activity全部移出Task,从而使得该目标Activity位于栈顶。

singleInstance(新开辟一个栈，全局唯一模式，全系统就这唯一的实例)  
 无论从哪个Task中启动目标Activity,只会创建一个目标Activity实例且会用一个全新的Task栈来装载该Activity实例（全局单例）.

如果将要启动的Activity不存在,那么系统将会先创建一个全新的Task,再创建目标Activity实例并将该Activity实例放入此全新的Task中。

如果将要启动的Activity已存在,那么无论它位于哪个应用程序,哪个Task中;系统都会把该Activity所在的Task转到前台,从而使该Activity显示出来。

和"singleTask"类似，唯一不同的是系统不会在这个activity的实例所在的task中启动任何其他activity。

这个activity的实例永远是这个task中的唯一一个成员，这个activity启动的任何其他activity都将在另外的task中打开。

使用Intent标识定义启动模式:

FLAG\_ACTIVITY\_NEW\_TASK  
 和之前讨论过的"singleTask"相同，在新的task中启动activity，如果一个你需要的activity的task已经存在，则将它推向前台，恢复其上一个状态，它通过[onNewIntent()](https://link.juejin.im/?target=http://developer.android.com/reference/android/app/Activity.html%23onNewIntent(android.content.Intent)" \t "https://juejin.im/entry/_blank)收到这个新的intent。

FLAG\_ACTIVITY\_SINGLE\_TOP  
 和"singleTop"行为相同，如果被启动的activity是当前顶部的activity，则已经存在的实例收到 [onNewIntent()](https://link.juejin.im/?target=http://developer.android.com/reference/android/app/Activity.html%23onNewIntent(android.content.Intent)" \t "https://juejin.im/entry/_blank)，而不是新建实例。

FLAG\_ACTIVITY\_CLEAR\_TOP  
 如果被启动的activity已经在当前task运行，不创建它的新实例，而是销毁在它之上的其他所有activities，然后通过 [onNewIntent()](https://link.juejin.im/?target=http://developer.android.com/reference/android/app/Activity.html%23onNewIntent(android.content.Intent)" \t "https://juejin.im/entry/_blank)传递一个新的intent给这个恢复了的activity。这个行为在 [launchMode](https://link.juejin.im/?target=http://developer.android.com/guide/topics/manifest/activity-element.html%23lmode" \t "https://juejin.im/entry/_blank)中没有对应的属性值。  
　　注意，如果activity的启动模式是"standard"，它自己也将被移除，然后一个新的实例将被启动。  
　　这是因为当启动模式是"standard"时，为了接收新的intent必须创建新的实例。

开启一个task

你可以通过给activity一个intent filter（action是"android.intent.action.MAIN"，category是"android.intent.category.LAUNCHER"），让这个activity是一个task的进入点。

　　如下：

<activity ... >

<intent-filter ... >

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

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

...</activity>

一个这样的intent filter会使得这个activity的icon和label显示在程序启动处，提供了一种方法，使得用户可以启动这个activity，当它启动后，用户也可以通过它来返回到这个task。  
　　第二个能力是很重要的：用户必须能够离开一个task，然后通过activity launcher返回到它。  
　　因为这个原因，两个让activity永远实例化一个task的启动模式："singleTask" 和"singleInstance"，应该仅在activity有一个 [ACTION\_MAIN](https://link.juejin.im/?target=http://developer.android.com/reference/android/content/Intent.html%23ACTION_MAIN" \t "https://juejin.im/entry/_blank) 和[CATEGORY\_LAUNCHER](https://link.juejin.im/?target=http://developer.android.com/reference/android/content/Intent.html%23CATEGORY_LAUNCHER" \t "https://juejin.im/entry/_blank) filter的时候用它们。