参考链接：

[Android Activity为什么要细化出onCreate、onStart、onResume、onPause、onStop、onDesdroy这么多方法让应用去重载？](http://blog.csdn.net/zhao_3546/article/details/12843477)

[Android中Activity完全解析](http://www.cnblogs.com/shinefy/p/3956137.html)

Dalvik虚拟机是[**Android**](http://lib.csdn.net/base/android)程序的虚拟机，是Android中Java程序的运行基础。其指令集基于寄存器架构，执行其特有的文件格式——dex字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能。它的核心内容是实现库（libdvm.so），大体由[**C语言**](http://lib.csdn.net/base/c)实现。依赖于[**Linux**](http://lib.csdn.net/base/linux)内核的一部分功能——线程机制、内存管理机制，能高效使用内存，并在低速CPU上表现出的高性能。每一个Android应用在底层都会对应一个独立的Dalvik虚拟机实例，其代码在虚拟机的解释下得以执行。

# ****Activity生命周期每个函数的解析****

## 1、onCreate(Bundle saveInstanceState)

该函数是该Activity第一次启动时调用，且启动后就不在执行，除非由于系统内存紧张，把处于Paused和Stopped状态下的Activity回收，然后再重新启动。

它有个形式参数saveInstanceState,它主要保存Activity由于系统内存不足等原因，被系统隐式销毁Activity,在被销毁之前一般的会调用onSaveInstanceState(),保留该Activity此时的状态信息。该方法中传入Bundle参数，系统会保留，当再次启动时，通过onCreate()方法传入。

可以在此函数内调用finish，在这种情况下，将立即调用onDestroy（），而不会执行任何活动生命周期（onStart，onResume，onPause等）的其余任何内容。

## 2、onStart()

一般情况下，该方法是在onCreate()之后调用或者在Stopped状态返回时调用onRestart之后调用。用户重新查看窗体时会执行。

3、onResume()

在onRestoreInstanceState，onRestart或onPause后调用，以便您的活动开始与用户交互。 这是开始动画，开放独家访问设备（如相机）等的好地方。

请记住，onResume不是您的活动对用户可见的最好的指示器; 诸如键盘的系统窗口可以在前面。 使用onWindowFocusChanged可以确定您的活动是否对用户可见（例如，恢复游戏）。

## 4、onPause()

当活动进入后台，但尚未被杀死时，被称为活动生命周期的一部分。 onResume的对应物。

当活动B在活动A前面启动时，将在A上调用此回调。在A的onPause返回之前不会创建B，因此请务必不要在此处做任何冗长的操作（有500ms时长限制，超时ActivityManagerService就会强制关闭这个Activity。）。

这个回调主要用于保存活动正在编辑的任何持久状态，向用户呈现“就地编辑”模型并确保没有什么丢失，如果没有足够的资源来启动新活动，而没有首先杀死这一个。这也是一个好地方，做停止动画和其他事情，消耗显着的CPU数量，以使切换到下一个活动尽可能快，或关闭资源，如独家访问，如相机。

在系统需要更多内存的情况下，它可能会终止已暂停的进程以回收资源。因为这个，你应该确保你的所有状态都保存在你从这个函数返回的时间。一般来说onSaveInstanceState用于在活动中保存每个实例的状态，并且这个方法用于存储全局持久性数据（在内容提供者，文件等中）

收到此呼叫后，您通常会收到以下呼叫onStop（在下一个活动已恢复并显示后），但在某些情况下，将直接回调到onResume，而不经过停止状态。5、onStop()

onPause()-->onStop()。  Activity页面切换，或者用户直接按Back键，将该窗体从当前任务重移除，也会执行onStop()

## 5、onRestart()

onStop()执行之后，该窗体和进程没有被系统销毁，此时用户又重新查看该窗体，则会执行窗体的onRestart()--->onStart()

## 6、onStop

当用户不再可见时调用。 接下来，您将收到{@link #onRestart}，{@link #onDestroy}或什么也不会（系统内存不足，进程直接被杀死），取决于后来的用户活动。所有类都必须调用这个方法的超级类的实现。 如果他们没有，将抛出异常。请注意，在内存不足的情况下，系统没有足够的内存可以在调用onPause方法后保持活动进程运行，因此可能永远不会调用此方法。

## 7、onDestroy()

Activity被销毁时执行onDestroy()。

在活动销毁之前执行任何最终清理。这可以是

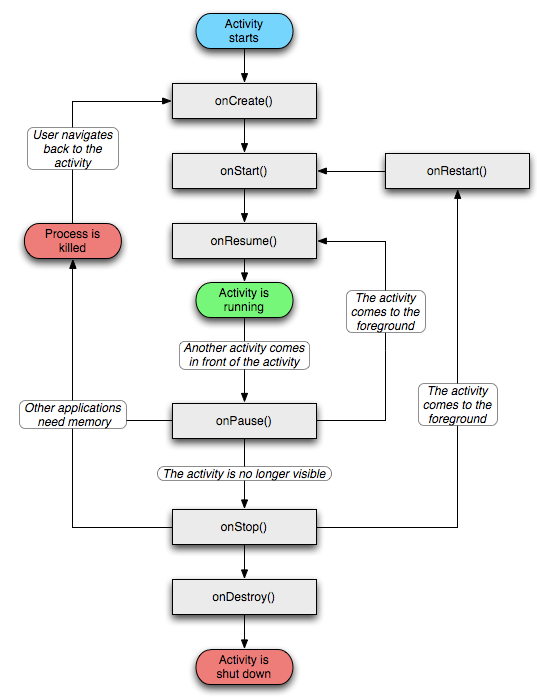
1. 主动调用onDestroy
2. 或是因为活动正在完成（调用finish并正在执行）
3. 或者因为系统正在临时摧毁这个实例的活动为了节省了空间或释放空间

可以调用isFinishing方法区分在这两种情况。

注意：不要将用方法保存数据！例如，如果activity有正在编辑中的内容需要提交，则这些数据的修改提交应在onPause或onSaveInstanceState，不在此处。

此方法通常实现时为了：

释放与activity关联的线程，以便被销毁的activity不会在其应用程序的其余部分仍在运行时留下这个没有与之关联的activity的线程。在某些情况下，系统将简单地杀死活动的主机进程，而不调用该方法（或任何其他方法），因此它不应该用于在进程消失之后继续执行的操作。派生类必须调用到超类的这个方法的实现。如果他们没有，将抛出异常。



# Activity的四种加载模式

## ****1.Activity栈****

Android采用Task来管理多个Activity，当我们启动一个应用时，系统就会为之创建一个Task，然后启动这个应用的入口Activity（即intent-filter中配置为MAIN和LAUNCHER的Activity）。

我们可以把Task理解为Activity栈。Task以栈的形式来管理Activity，根据栈“先进后出”的原则，先启动的Activity被压入栈底，后启动的Activity放在栈顶。如果我们按Back键，则最上面的Activity被弹出栈。

## ****2、四种加载模式****

配置Activity时可以指定android:launchMode属性，用于配置Activity的加载模式

### standard

默认加载模式（始终创建新实例）

启动目标Activity时，总会为该Activity创建一个新的实例，并放入当前的Task的栈顶中。

### singleTop

顶单例模式（**栈顶**有实例则不再创建新实例）

启动目标Activity时，若已经有一个目标Activity位于栈顶，则不会创建新的目标Activity实例，而是复用原有的。

若没有目标Activity位于栈顶，则会创建一个新的目标Activity实例，并添加至当前Task的栈顶。

### singleTask

内单例模式（保持一个Task内只有一个实例）

若当前Task内不存在目标Activity实例，则会创建一个新的目标Activity实例，并添加至当前Task的栈顶。

启动目标Activity时，若已经有一个目标Activity实例**位于栈顶**，则不会创建新的目标Activity实例，而是复用原有的。

启动目标Activity时，若当前Task内已经有一个目标Activity实例，但**不位于**栈顶**，则会移除该实例上面的所有Activity，从而使该实例转入栈顶。**

### singleInstance

**全局**单例模式（保持**所有Task**内只有一个实例，且**用一个单独的Task栈放这个实例**）

若目标Activity的实例不存在，则会创建一个全新的Task，再创建一个该Activity的实例，并将其加入新Task的栈顶。

若已经存在，则会将**该实例所在的Task**转到前台。

# startActivityForResult的使用

A-Activity需要在B-Activtiy中执行一些数据操作，跳转至B-Activity后，B-Activity要将执行操作数据的结果返回给A-Activtiy，此时就需要使用 startActivityForResult()来启动B-Activity了。

使用的三个函数：

startActivityForResult(Intent intent, Int requestCode)   
setResut(int resultCode, Intent intent)   
onActivityResult(int requestCode, int resultCode, Intent intent)

步骤：

1、在A中跳转的时候不是采用startActivity(intent) 这个方法，而是startActivityForResult(intent, Int requestCode)的形式,requestCode可以是大于等于0的任何值。

2、在A中重写onActivityResult方法，用来接收B回传的数据，因为传回来的resultCode不同而做差别处理。

3、在B中采用setResult方法，并且之后要调用finish方法。

Intent intent=new Intent();

intent.putExtra("str1", str\_bookname);

setResult(20, data);

finish(); //关闭掉这个Activity

# 使用Bundle在Activity间传值

Activity使用Bundle包来携带数据，Bundle对象有如下方法：

* 存

putXxx(String key,Xxx data) : 向Bundle中放入int、String等各种类型的数据

putSerializable(String key,Serializable data) : 向Bundle中放入可序列化的对象

* 取

getXxx(String key):取出int、String等各种类型的数据

getSerializable(String key):取出可序列化的对象

当然我们还可以直接调用intent对象的putExtra(String key,Xxx data)方法存入数据，但其本质还是创建或使用了Bundle对象进行传值。