## Activity生命周期循环



Figure 1 Activity生命周期（完整版）



Figure 2 Activity生命周期（简化版）

Activity的生命周期中存在三种静态，任何Activity只可能在这三种状态下存在较长时间：

Resumed：这种状态下，Activity处于前台，且用户可以与之交互。（所谓的“运行状态”）。

Paused：这种状态下，Activity被在前台中处于半透明状态或者未覆盖整个屏幕的另一个Activity一部分遮挡，此时被暂停的Activity不会接受用户输入并且无法执行任何代码。

Stopped：这种状态下，Activity被完全隐藏并且对用户不可见，被视作处于后台，此时Activity实例及其成员变量等所有状态信息将保留，但无法执行任何代码。

另外两个状态Created与Started都是瞬态，系统会快速调用下一个生命周期方法，从而切换到其他状态。

要点速记：

（1）状态转移顺序：Activity启动——Created——Started——Resumed——Paused——Stopped——Activity销毁。

（2）方法名与调用后进入的状态名统一，如调用onResume()方法即会进入Resumed状态，因而只要记住状态之间的转移关系即可（唯一例外：Stopped状态下回到Started状态需要先调用onRestart()再调用onStart()）。

（3）Resumed状态下被其他Activity部分遮挡会通过调用onPause()方法进入Paused状态，可通过onResume()方法回到Resumed状态。

（4）Paused状态下退回到后台会调用onStop()方法进入Stopped状态，可通过调用onRestart()方法与onStart()方法回到Started状态。

（5）Stopped状态下，Activity可能会因为系统资源紧缺而被杀死，此时回到Activity时会调用onCreate()方法进入Created状态。

（6）Activity销毁时会调用onDestroy()方法。

（7）如果在onCreate()方法中调用finish()方法，Activity会跳过其他生命周期方法，直接调用onDestroy()。

（8）若Activity在Stopped状态下被杀死，onDestroy()方法不会被调用。

当一个 Activity 启动另一个 Activity 时，启动第二个 Activity 的过程与停止第一个 Activity 的过程存在重叠。

生命周期回调的顺序经过明确定义，当两个 Activity 位于同一进程，并且由一个 Activity 启动另一个 Activity 时，其定义尤其明确。 以下是当 Activity A 启动 Activity B 时一系列操作的发生顺序：

1、Activity A 的 onPause() 方法执行。

2、Activity B 的 onCreate()、onStart() 和 onResume() 方法依次执行。（Activity B 现在具有用户焦点。）

3、然后，如果 Activity A 在屏幕上不再可见，则其 onStop() 方法执行。

可以利用这种可预测的生命周期回调顺序管理从一个 Activity 到另一个 Activity 的信息转变。 例如，如果必须在第一个 Activity 停止时向数据库写入数据，以便下一个 Activity 能够读取该数据，则应在 onPause() 而不是 onStop() 执行期间向数据库写入数据。

## 指定程序首次启动的Activity

当用户从主界面点击程序图标时，系统会调用app中被声明为“launcher”或“main”的activity的onCreate()方法。这个Activity也就成了程序的入口。

主Activity在AndroidManifest.xml中定义，示例如下（关键在action与category标签）：

<activity android:name=".MainActivity" android:label="@string/app\_name">

<intent-filter>

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

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

</intent-filter>

</activity>

## 实现生命周期方法

当打开一个半透明activity（如对话框）时，之前的activity会进入Paused状态。

onPause()：一般在onPause()方法中做以下事情：

（1）停止动画或是其他正在进行的操作。

（2）保存在离开状态下期待被保存的的内容，如邮件草稿等。

（3）释放系统资源，例如broadcast receivers,sensors,照相机等。

费时的操作一般来讲应留到onStop()中去做，否则会影响切换到下一个activity的速度。

onResume()：一般在onResume()方法中做以下事情：

（1）开始动画等耗时操作。

（2）获取系统资源。

onResume()与onPause()应配套使用，里面的操作一般是成对出现（获取、释放，暂停、开始等）。

在以下情况下会进入Stopped状态：

（1）用户切换到另外一个App。

（2）在原App中执行了创建新activity的方法。

（3）在使用App的过程中接到一个来电通话。

onStop()：一般在onStop()方法中做以下事情：

（1）耗时操作，如往数据库中写信息等。

（2）清除所有资源。

onStart()：应做与onStop()相对应的操作，如验证资源是否可用等。

onRestart()：特殊情况下使用。

Activity会在每次旋转屏幕时被destroyed与recreated。当屏幕改变方向时，系统会Destory与Recreate前台的activity，因为屏幕配置被改变，你的Activity可能需要加载另一些替代的资源(例如layout)。

如果activity是因为用户点击了back按钮或是activity调用finish()方法结束自己而被销毁的，那么系统就丢失了对这个activity的引用，因为这意味着这个activity不再被需要了。而如果是因为系统资源紧张导致activity的销毁，系统会使用那些保存的记录数据来重新创建activity的实例。那些用于恢复activity的数据叫做instance state，是一些存放在Bundle对象里的key-value pairs。跳转到其他的activity或者是点击Home、锁屏都会导致当前的activity执行onSaveInstanceState，因为这种情况下的activity都是有可能会被destory并且是需要保存状态以便后续恢复使用的，而从跳转的activity点击back回到前一个activity，那么跳转前的activity是执行退栈的操作，所以这种情况下是不会执行onSaveInstanceState的，因为这个activity不可能存在需要重建的操作。

为了可以保存额外更多的数据到saved instance state。在Activity的生命周期里面存在一个额外的回调函数，这个方法是onSaveInstanceState() ，在onPause()之后执行。当用户离开Activity时，系统会调用它。当系统调用这个函数时，系统会在Activity被异常Destory时传递 Bundle 对象，这样我们就可以增加额外的信息到Bundle中并保存到系统中。若系统在Activity被Destory之后想重新创建这个Activity实例时，之前的Bundle对象会(系统)被传递到activity的onRestoreInstanceState()方法与 onCreate() 方法中。

默认情况下，系统利用Bundle实例保存每一个View对象中的信息，因此如果activity被destroyed and recreated，layout中的状态信息会自动恢复。为了使Android系统能够恢复Activity中的View的状态，每个View都必须有一个唯一ID，由android:id定义。必须要调用 onSaveInstanceState() 方法的父类实现，这样默认的父类实现才能保存视图状态的信息。总是需要调用onRestoreInstanceState()方法的父类实现，这样默认的父类实现才能保存视图状态的信息

由于 onCreate() 方法会在第一次创建新的Activity实例与重新创建之前被Destory的实例时都被调用，我们必须在尝试读取 Bundle 对象前检测它是否为null。如果它为null，系统则是创建一个新的Activity实例，而不是恢复之前被Destory的Activity。onRestoreInstanceState()方法会在 onStart() 方法之后执行. 系统仅仅会在存在需要恢复的状态信息时才会调用 onRestoreInstanceState() ，因此不需要检查 Bundle 是否为null。