Acivity生命周期管理



在三种状态下，activity可以被保留。

Resume：

Activity被放在前台并获取用户焦点，也就是activity的运行状态。

Paused：

另外的activity被放在前台，并获取焦点。但当前activity仍旧能被用户看见。Activity对象礽被保存在内存中，保持和window的连接。当系统内存不足时，随时可以被回收。

Stopped：

完全被另一个activity遮蔽（obscured），不再可见，被放在后台。实例对象仍被保存在内存，失去和window manager的连接，随时可被回收。

生命周期：

整个生命周期：onCreate ---- onDestory

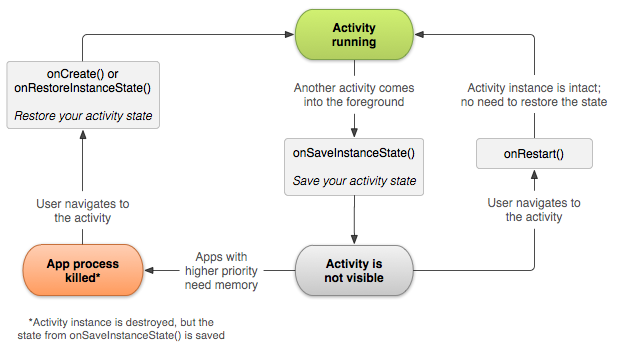
可见生命周期：onStart ---- onStop

在此期间，用户可在屏幕上看见这个activity，当[onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop())被调用时，说明一个新的activity已经被开启，this one is no longer visible.例如，可以在[onStart()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStart())方法中注册[BroadcastReceiver](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\BroadcastReceiver.html) 来监视UI上的变化，并在[onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop())中取消注册。

前台生命周期： onResume() ---- onPause()

在此期间一直持有用户焦点,一个activity可能频繁的在前后台切换，onPause()可能在设备进入睡眠状态或弹出一个dialog时被调用。因为这两个方法被调用的频繁，所以在方法中最好只做些轻量级的操作。

保存activity状态



当activity被pause或是stoped时，其状态仍被保存，所以当activity从新回到前台时，状态保持不变。

系统在activity容易被销毁时调用[onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle))来保存信息，系统传给这个方法一个Bundle对象，用来保存用户信息，Bundle时一个键值对。用put和get存取内容。当系统重新创建activity时，被保存的bundle信息被单做参数传给了onCreate和onRestoreInstanceState。

**Note:** There's no guarantee that [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)) will be called before your activity is destroyed, because there are cases in which it won't be necessary to save the state (such as when the user leaves your activity using the *Back* button, because the user is explicitly closing the activity). If the system calls[onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)), it does so before [onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop()) and possibly before [onPause()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onPause()).

However, even if you do nothing and do not implement [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)), some of the activity state is restored by the [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html) class's default implementation of [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)). Specifically, the default implementation calls the corresponding [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\view\View.html#onSaveInstanceState()) method for every [View](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\view\View.html) in the layout, which allows each view to provide information about itself that should be saved. Almost every widget in the Android framework implements this method as appropriate, such that any visible changes to the UI are automatically saved and restored when your activity is recreated. For example, the [EditText](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\EditText.html) widget saves any text entered by the user and the [CheckBox](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\CheckBox.html) widget saves whether it's checked or not. The only work required by you is to provide a unique ID (with the [android:id](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\resources\layout-resource.html#idvalue) attribute) for each widget you want to save its state. If a widget does not have an ID, then the system cannot save its state.

You can also explicitly stop a view in your layout from saving its state by setting the [android:saveEnabled](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\R.attr.html#saveEnabled)attribute to "false" or by calling the[setSaveEnabled()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\view\View.html#setSaveEnabled(boolean)) method. Usually, you should not disable this, but you might if you want to restore the state of the activity UI differently.

Although the default implementation of [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle))saves useful information about your activity's UI, you still might need to override it to save additional information. For example, you might need to save member values that changed during the activity's life (which might correlate to values restored in the UI, but the members that hold those UI values are not restored, by default).

Because the default implementation of [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)) helps save the state of the UI, if you override the method in order to save additional state information, you should always call the superclass implementation of [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)) before doing any work. Likewise, you should also call the superclass implementation of [onRestoreInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onRestoreInstanceState(android.os.Bundle)) if you override it, so the default implementation can restore view states.

**Note:** Because [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)) is not guaranteed to be called, you should use it only to record the transient state of the activity (the state of the UI)—you should never use it to store persistent data. Instead, you should use [onPause()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onPause()) to store persistent data (such as data that should be saved to a database) when the user leaves the activity.

A good way to test your application's ability to restore its state is to simply rotate the device so that the screen orientation changes. When the screen orientation changes, the system destroys and recreates the activity in order to apply alternative resources that might be available for the new screen configuration. For this reason alone, it's very important that your activity completely restores its state when it is recreated, because users regularly rotate the screen while using applications.

**Handling configuration changes**

Some device configurations can change during runtime (such as screen orientation, keyboard availability, and language). When such a change occurs, Android recreates the running activity (the system calls [onDestroy()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onDestroy()), then immediately calls [onCreate()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle))). This behavior is designed to help your application adapt to new configurations by automatically reloading your application with alternative resources that you've provided (such as different layouts for different screen orientations and sizes).

If you properly design your activity to handle a restart due to a screen orientation change and restore the activity state as described above, your application will be more resilient to other unexpected events in the activity lifecycle.

The best way to handle such a restart is to save and restore the state of your activity using [onSaveInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle))and [onRestoreInstanceState()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onRestoreInstanceState(android.os.Bundle)) (or [onCreate()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle))), as discussed in the previous section.

For more information about configuration changes that happen at runtime and how you can handle them, read the guide to [Handling Runtime Changes](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\resources\runtime-changes.html).

**Coordinating activities**

When one activity starts another, they both experience lifecycle transitions. The first activity pauses and stops (though, it won't stop if it's still visible in the background), while the other activity is created. In case these activities share data saved to disc or elsewhere, it's important to understand that the first activity is not completely stopped before the second one is created. Rather, the process of starting the second one overlaps with the process of stopping the first one.

The order of lifecycle callbacks is well defined, particularly when the two activities are in the same process and one is starting the other. Here's the order of operations that occur when Activity A starts Acivity B:

1. Activity A's [onPause()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onPause()) method executes.
2. Activity B's [onCreate()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)), [onStart()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStart()), and [onResume()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onResume()) methods execute in sequence. (Activity B now has user focus.)
3. Then, if Activity A is no longer visible on screen, its [onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop()) method executes.

This predictable sequence of lifecycle callbacks allows you to manage the transition of information from one activity to another. For example, if you must write to a database when the first activity stops so that the following activity can read it, then you should write to the database during [onPause()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onPause()) instead of during [onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop()).

| Method | | | Description | Killable after? | Next |
| --- | --- | --- | --- | --- | --- |
| [onCreate()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)) | | | 第一次创建时调用. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see [Saving Activity State](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\activities.html#actstate), later).  Always followed by onStart(). | No | onStart() |
|  | [onRestart()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onRestart()) | | Called after the activity has been stopped, just prior to it being started again.  Always followed by onStart() | No | onStart() |
| [onStart()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStart()) | | Called just before the activity becomes visible to the user.  Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden. | No | onResume()  or onStop() |
|  | [onResume()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onResume()) | Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it.  Always followed by onPause(). | No | onPause() |
| [onPause()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onPause()) | Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns.  Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user. | Yes | onResume()  or onStop() |
| [onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop()) | | Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.  Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away. | Yes | onRestart() or onDestroy() |
| [onDestroy()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onDestroy()) | | | Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called[finish()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#finish()) on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the [isFinishing()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#isFinishing()) method. | Yes | *nothing* |