**Service基础**

Service属于Android四大组件之一，它是在后台执行任务的组件，对于用户来说是不可见的。当Service与其他组件不在统一进程中运行时，它还支持跨进程通信。

**服务的启动和销毁**

通过Context启动服务方式有两种，启动服务和绑定服务。也有相应的两种销毁服务方式，停止服务和解绑服务。

* **启动**

**startService启动方式**：在其他组件中（不包括Broadcast Receive）调用startService()方法后，服务即处于启动状态。启动service后该service就处于独立运行状态并且是无限期运行，即使启动service的组件已被销毁也不受其影响，直到其被停止，这种启动方式没有与启动它的组件进行通信的方式。

**bindService绑定方式:**在其他组件中（不包括Broadcast Receive）调用bindService()方法后，服务即处于启动状态。可以通过 ServiceConnection进行通信，组件可以与service进行交互、发送请求、获取结果，甚至是利用IPC跨进程执行这些操作。当所有与其绑定的组件都取消绑定(可能是组件被销毁也有可能是其调用了unbindService()方法)后，service将停止。

* **销毁**

**startService启动:**service中调用stopSelf()方法，或者其他组件调用stopService()方法后，继而会调用服务的onDestory()方法销毁服务。

**bindService绑定:**所有与service绑定的组件都被销毁，或者它们都调用了unbindService()方法后，继而会调用服务的onDestory()方法销毁服务。

注意：startService()与bindService()并不冲突，同一个service可能既有组件调用了startService()启动它，又有组件与它进行了绑定。当同一个服务即被其他组件绑定又被其它组件启动时，这种情况下，要同时调用sopService方法和unbindService方法，onDestory()方法才会执行，服务才会被销毁。

**生命周期**

**onCreate():**只有服务第一次被创建的时候调用，并且只会被调用一次。

**onStartCommand():**当Context通过startService()启动服务时，会调用此方法，这个方法是服务进行主要任务执行的地方，这个方法会接受一个Intent参数，是由启动方传递进来，可以携带参数。Context每调用一次startService()，相应的服务就会调用一次onStartCommand()方法，但是只会有一个服务实例存在。

注意：onStartCommand()返回值的作用是指定系统对当前线程的行为的。有三种值：

\* START\_STICKY：如果系统在onStartCommand()返回后杀死了这个服务，系统就会重新创建这个服务并且调用onStartCommand()方法，但是它不会重新传递最后的Intent对象，系统会用一个null的Intent对象来调用onStartCommand()方法，在这个情况下，除非有一些被发送的Intent对象在等待启动服务。这适用于不执行命令的媒体播放器（或类似的服务），它只是无限期的运行着并等待工作的到来。

\* START\_NOT\_STICKY：如果在执行完onStartCommand后，服务被异常kill掉，系统不会自动重启该服务。这是最安全的选项，用来避免在不需要的时候运行你的服务。

\* START\_REDELIVER\_INTENT：如果系统在onStartCommand()方法返回后，系统就会重新创建了这个服务，并且用发送给这个服务的最后的Intent对象调用了onStartCommand()方法。这适用于那些应该立即恢复正在执行的工作的服务，如下载文件。

**onBind():**当其他组件通过bindService()启动服务的时候会调用这个方法，这个方法需要一个IBinder返回值。所以我们必须要实现一个IBinder实例，一般继承Binder类即可。这样就可以让服务和绑定组件进行通信了。如果不想让这个服务与其他组件绑定，可以返回null。

**onUnbind():**服务被解除时调用。

**onDestory():**Service被销毁的时候调用，可以在这个方法里进行资源回收等操作。

下面是startService时服务的声明周期：

startService

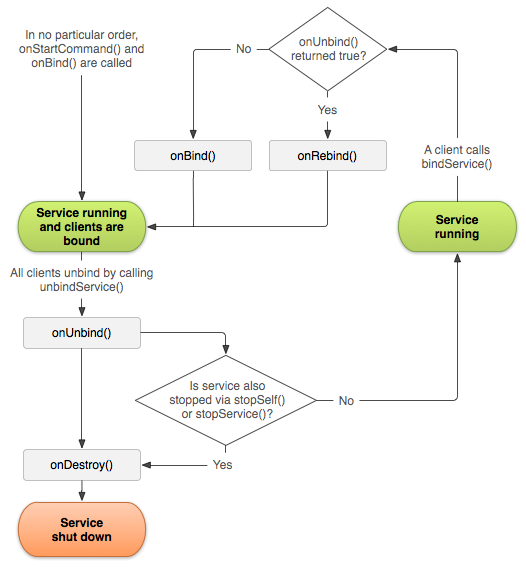
第一次创建会调用onCreate

onStartCommand

调用stopService或者调用stopSelf

onDestory销毁

下面是bindService时服务的生命周期：



绑定服务的生命周期：onCreate()->onBind->onUnbind->onDestory。这当只有一个组件绑定服务的情况，当第二个组件也来绑定这个服务的同时，这些方法都不会被执行，而是在第二个组件中ServiceConnection的onServiceConnected方法中直接返回服务的binder对象。还有，只要某个组件一旦绑定服务不管调用多少次绑定服务的操作，Service的onBind方法和ServiceConnection的onServiceConnected方法只会执行一次。只有当服务的所有绑定者都解除绑定后才会执行onUnbind->onDestory。

**创建一个服务**

下面是两种创建服务的方式。

* **通过startService方式**

****

在Activity中启动这个服务：

1513043981(1)

当服务执行完任务后要调用stopService或者在服务类里面调用stopSelf停止服务。

* **通过bindService方式**

通过bindService方式开启服务，可以完成服务与绑定方的通信。

****

当客户端与服务绑定后会调用onBind方法，这个方法会返回一个IBinder对象，其实IBinder是个接口，它在整个Android系统中起到了重要的作用，它是为高性能而设计轻量级远程调用机制的核心部分，它不仅可以进程之间调用，还可以同一个进程内调用。这里的服务可能会涉及到跨进程交互。一般有三种方式可以Binder对象：继承Binder类，使用Message类，使用AIDL等。

上面获取IBinder的方式是继承Binder类，它实现了IBinder接口，客户端可以通过这个类调用服务端的公共方法。其实Binder是Android中跨进程通信的一种方式，但是一般Service中的Binder是不会涉及到跨进程的。

让一个Activity绑定这个服务：

****

上面绑定服务bindService需要三个参数，这里详细解释第三个参数它有几个不同的取值，如下：

* **0：**如果不想设置任何值，就设置成0
* **Context.BIND\_AUTO\_CREATE：**绑定服务时候，如果服务尚未创建，服务会自动创建，在API LEVEL 14以前的版本不支持这个标志，使用Context.BIND\_WAIVE\_PRIORITY可以达到同样效果
* **Context.BIND\_DEBUG\_UNBIND：**通常用于Debug，在unbindService时候，会将服务信息保存并打印出来，这个标记很容易造成内存泄漏。
* **Context.BIND\_NOT\_FOREGROUND：**不会将被绑定的服务提升到前台优先级，但是这个服务也至少会和客户端在内存中优先级是相同的。
* **Context.BIND\_ABOVE\_CLIENT：**设置服务的进程优先级高于客户端的优先级，只有当需要服务晚于客户端被销毁这种情况才这样设置。
* **Context.BIND\_ALLOW\_OOM\_MANAGEMENT：**保持服务受默认的服务管理器管理，当内存不足时候，会销毁服务
* **Context.BIND\_WAIVE\_PRIORITY：**不会影响服务的进程优先级，像通用的应用进程一样将服务放在一个LRU表中
* **Context.BIND\_IMPORTANT：**标识服务对客户端是非常重要的，会将服务提升至前台进程优先级，通常情况下，即时客户端是前台优先级，服务最多也只能被提升至可见进程优先级，
* **BIND\_ADJUST\_WITH\_ACTIVITY：**如果客户端是Activity，服务优先级的提高取决于Activity的进程优先级，使用这个标识后，会无视其他标识。

**特别注意：**在ServiceConnection中的onServiceConnected方法中是将形参中从服务端传递进来的IBinder对象强制转换成与服务端的实现的Binder类型一致的对象。**这里的强制转换必须在同一个进程内，否则会因为另一个进程内没有相应的Binder实现类而转换失败**。

* **ServiceConnection**

当一个组件进行绑定服务的时候会将一个实现了ServiceConnection接口的类传递给Service，当Service调用完onBind方法后会调用onServiceConnected方法，并将binder对象传递给绑定组件。ServiceConnection接口中一共有两个方法：

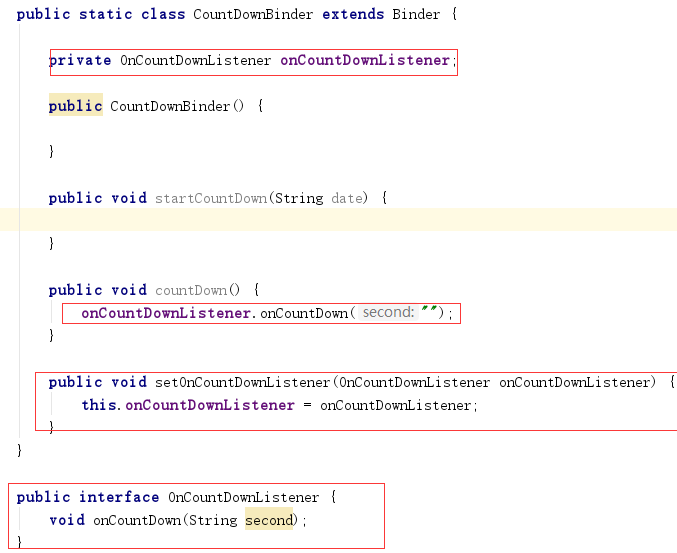
**onServiceConnected(ComponentName name, IBinder service)：**当绑定服务成功后会调用这个方法，并将Service中的binder传递过来。

**onServiceDisconnected(ComponentName name)：**当这个方法被调用时表示你的组件与Service断开了连接，然后需要在这里进行断开连接后的处理工作，例如重新绑定服务等。这里断开连接指的是意外断开的链接，并不是调用unbindService方法断开。

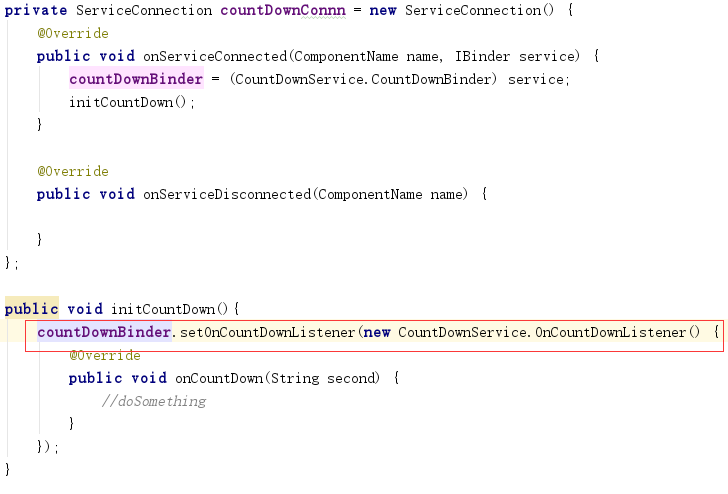
* **Service与其它组件之间通信**

1. 通过广播
2. 使用接口。Activity可以同过丙丁服务的方式获取Service中的binder，进而可以调用Service中的方法。反过来Service调用Acrtivirty中方法，只需要在Service中定义一个接口，让Activity绑定完Service后，去实现这个接口即可，如下图：

**在Service的Binder中：**

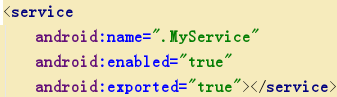


**在Activity中：**



* **在AndroidManifest.xml中进行配置**

创建服务后类后，最后一步就是一定要在AndroidManifest.xml中进行配置：



其中有一些属性需要了解：

* **android:enabled :** 如果为true，则这个service可以被系统实例化，如果为false，则不行。默认为true。
* **android:exported :** 如果为true，则其他应用的组件也可以调用这个service并且可以与它进行互动，如果为false，则只有与service同一个应用或者相同user ID的应用可以开启或绑定此service。它的默认值取决于service是否有intent filters。如果一个filter都没有，就意味着只有指定了service的准确的类名才能调用，也就是说这个service只能应用内部使用——其他的应用不知道它的类名。这种情况下exported的默认值就为false。反之，只要有了一个filter，就意味着service是考虑到外界使用的情况的，这时exported的默认值就为true
* **android:icon :** 一个象征着这个service的icon
* **android:isolatedProcess :** 如果设置为true，这个service将运行在一个从系统中其他部分分离出来的特殊进程中，我们只能通过Service API来与它进行交流。默认为false。
* **android:label :** 显示给用户的这个service的名字。如果不设置，将会默认使用<application>的label属性。
* **android:name :** 这个service的路径名，例如“com.lypeer.demo.MyService”。这个属性是唯一一个必须填的属性。
* **android:permission :** 其他组件必须具有所填的权限才能启动这个service。
* **android:process :** service运行的进程的name。默认启动的service是运行在主进程中的。
* **Service与线程之间的区别**

参考：http://www.maiziedu.com/wiki/component/choose/

**Service进阶**

**使用Messenger通信**

在基础篇的最后讲了同进程内服务端与客户端之间的通信，很简单就是利用Binder。那么不同进程间的通信可以使用Messenger。而Messenger的原理就是对AIDL通信进行的封装。

* **服务端：**



首先在服务端新建一个Handler对象，用来接收客户端发送到服务端的消息。使用Handler新建一个Messenger。在onBind方法中，返回从Messenger获取的IBinder对象，当客户端绑定到服务端的时候就会接收到这个IBinder对象。这里创建Messenger的时候将Handler传递进去，其实Messenger发送消息最终就是利用Hnadler发送的Message。

* **客户端：**

这里的客户端属于另一个进程。



当客户端绑定到服务端的时候，在onServiceConnected方法总初始化一个向服务端发送消息的Messenger（messengerClient）和一个服务端向客户端发型消息的Messenger（messengerClient），并且新建一个Handler来接受服务端的消息。在sendMessageToService方法中向服务器发送消息时，将messengerClient放在message中传递到服务端，服务端收到消息后可以通过消息载体获取到客户端这个Messenger，然后向客户端发送消息。