1. **Service的定义（官方翻译）：**

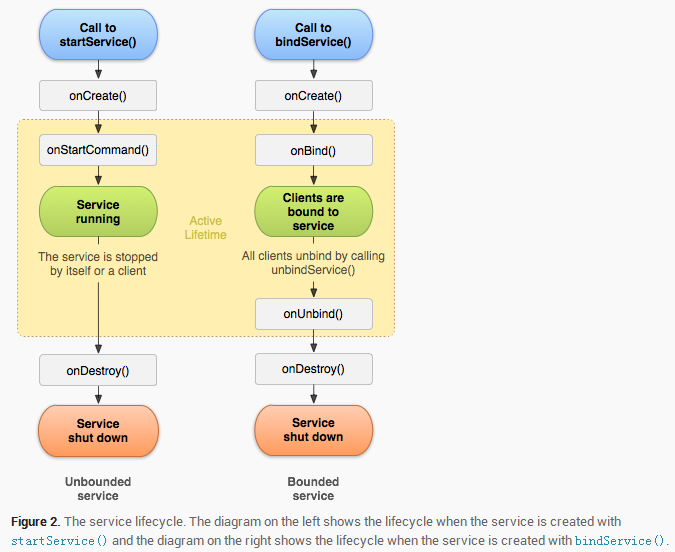
Service（服务）是一个没有用户界面的在后台运行执行耗时操作的应用组件。其他应用组件能够启动Service，并且当用户切换到另外的应用场景，Service将持续在后台运行。另外，一个组件能够绑定到一个service与之交互（IPC机制），例如，一个service可能会处理网络操作，播放音乐，操作文件I/O或者与内容提供者（content provider）交互，所有这些活动都是在后台进行。

1. **Service 分类**

**2.1、Started(启动)** ——> 本地服务：用于应用程序内部。调用者与服务之间没有关连，即使调用者退出了，服务仍然运行。

2.1.1 Started类型的Service的周期：

context.startService() ——> /\*\*onCreate()（只在创建的时候调用一次直到被摧毁） ——> onStartCommand()（取代之前的start()，服务开启后，可多次调用） ——> Service Running\*\*/(这部分是在Service中执行的) ——> context.stopService() ——> onDestory();



（1）如果Service还没有运行，则android先调用onCreate()然后调用onStartcommand()；  
（2）如果Service已经运行，则只调用onStartcommand ()，所以一个Service的onStartcommand方法可能会重复调用多次。

在Service可以调用Context.startService()启动，调用Context.stopService()结束。在内部可以调用Service.stopSelf() 或 Service.stopSelfResult()来停止自己。 无论调用了多少次startService()，都只需调用一次stopService()来停止。

**2.1.2、StartCommand()方法的返回值必须是一个整数，**必须是一下三种的某一种：

#### public int onStartCommand ([Intent](http://android.xsoftlab.net/reference/android/content/Intent.html) intent, int flags, int startId)

（1）**START\_NOT\_STICKY**

如果返回START\_NOT\_STICKY，表示当Service运行的进程被Android系统强制杀掉之后，不会重新创建该Service，当然如果在其被杀掉之后一段时间又调用了startService，那么该Service又将被实例化。那什么情境下返回该值比较恰当呢？如果我们某个Service执行的工作被中断几次无关紧要或者对Android内存紧张的情况下需要被杀掉且不会立即重新创建这种行为也可接受，那么我们便可将 onStartCommand的返回值设置START\_NOT\_STICKY。举个例子，某个Service需要定时从服务器获取最新数据：通过一个定时器每隔指定的N分钟让定时器启动Service去获取服务端的最新数据。当执行到Service的onStartCommand时，在该方法内再规划一个N分钟后的定时器用于再次启动该Service并开辟一个新的线程去执行网络操作。假设Service在从服务器获取最新数据的过程中被Android系统强制杀掉，Service不会再重新创建，这也没关系，因为再过N分钟定时器就会再次启动该Service并重新获取数据。

（2）**START\_STICKY**

如果返回START\_STICKY，表示Service运行的进程被Android系统强制杀掉之后，Android系统会将该Service依然设置为started状态（即运行状态），但是不再保存onStartCommand方法传入的intent对象，然后Android系统会尝试再次重新创建该Service，并执行onStartCommand回调方法，但是onStartCommand回调方法的Intent参数为null，也就是onStartCommand方法虽然会执行但是获取不到intent信息。如果你的Service可以在任意时刻运行或结束都没什么问题，而且不需要intent信息，那么就可以在onStartCommand方法中返回START\_STICKY，比如一个用来播放背景音乐功能的Service就适合返回该值。

（3）**START\_REDELIVER\_INTENT**

如果返回START\_REDELIVER\_INTENT，表示Service运行的进程被Android系统强制杀掉之后，与返回START\_STICKY的情况类似，Android系统会将再次重新创建该Service，并执行onStartCommand回调方法，但是不同的是，Android系统会再次将Service在被杀掉之前最后一次传入onStartCommand方法中的Intent再次保留下来并再次传入到重新创建后的Service的onStartCommand方法中，这样我们就能读取到intent参数。只要返回START\_REDELIVER\_INTENT，那么onStartCommand重的intent一定不是null。如果我们的Service需要依赖具体的Intent才能运行（需要从Intent中读取相关数据信息等），并且在强制销毁后有必要重新创建运行，那么这样的Service就适合返回START\_REDELIVER\_INTENT。

***StartCommand()函数中的flags***

flags有3种取值，flags值和onStartCommand()的返回值有着直接的关系。如果Service被系统意外终止，重启的时候，传入的flags。

#### （1）START\_FLAG\_REDELIVERY（1）

在服务调用stopSelf(int)前，服务已经返回**START\_REDELIVER\_INTENT。那么在重新启动服务时，将会将startCommand()中的flags值设为**START\_FLAG\_REDELIVERY。

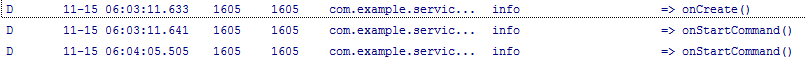
#### （2）START\_FLAG\_RETRY（2）

**由于原来的Intent启动没有成功，或者已经从onStartCommand()返回值，将会将startCommand()中的flags值设为START\_FLAG\_RETRY。**

**//启动Service(**context.startService()**)**



**//再次点击启动Service:只调用onStartCommand()**



**//停止Service（**context.stopService()**）**



**2.2、Bound(绑定) ——> 远程服务：用于android系统内部的应用程序之间**

**简单的说：**使用bindService()方法启用服务，调用者与服务绑定在了一起，调用者一旦退出，服务也就终止，大有“不求同时生，必须同时死”的特点。

实现服务的绑定，必须实现onBind()回调方法，该方法返回一个IBinder对象，它定义一个用于客户端用来与服务交互的程序接口。在客户端要实现ServiceConnection()类的onServiceConnection()回调方法绑定到服务。

**注：**

（1）多个client可以绑定至同一个service，但该 service 的onBind() 方法只会在第一个 client 绑定至其的时候被调用，当其他 client 再次绑定到它的时候，并不会调用  onBind() 方法，而是直接返回第一次被调用时产生的那个 IBinder 对象。也就是说，在其生命周期内，onBind() 只会被调用一次。

（2） Bind Service 不会在后台无限期的一直运行，而是当所有绑定至其的组件都调用了 unbindService() 进行解绑之后，系统就会将其停掉以回收资源。

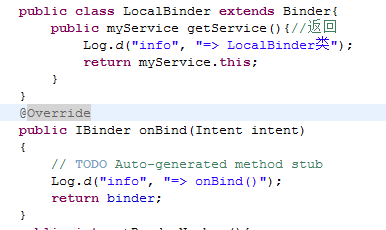
**定义onBind()回调方法返回的接口有一下三种方式：**

（1）继承Binder类

该方法适用于服务是私有的情况（service 与 application 在同一个进程中）。

它分以下几个步骤：

a. 在你的 service 类中声明一个（公共）内部类来继承 Binder 类。在该内部类中，最好提供一个公共方法来返回你的 service 实例。

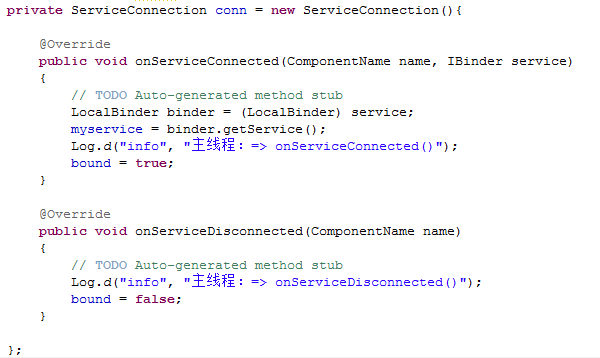


b. 在你的 service 类中需要声明一个这个内部类的实例，以供在onBind()方法中返回

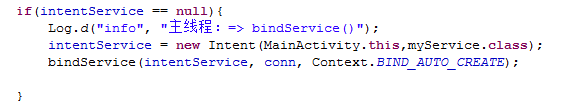
c.  在client端，在onServiceConnected()方法中得到从onBind()方法中返回的 IBinder 对象，然后可以通过该对象中的公共方法得到相应的 service 实例。

d. 在service中提供公共方法， 这样就可以在组件（如 Activity 中）调用这些公共方法了）。

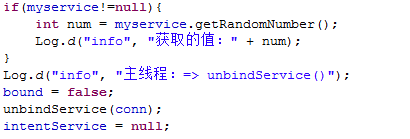
实现ServiceConnection类



绑定服务



调用服务总的公共函数并解除绑定

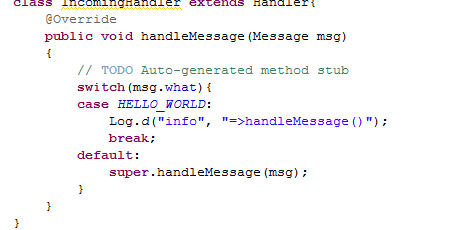




（2）使用Messenger类

如果需要接口跨进程工作，则可以使用Messenger来为服务创建接口。（进程间通信的最简单的方式，因为Messenger类将所有的请求队列化到单独的线程（服务每次收到一个请求）。

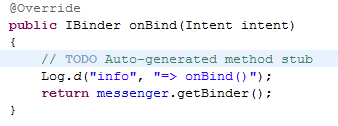
1. 实现Handler的服务因为每次从客户端调用而收到回调。（继承Handler）



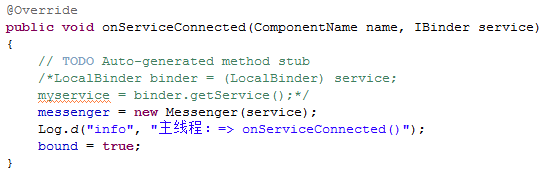
1. Handler用于创建Messenger对象（它是Handler的引用）。

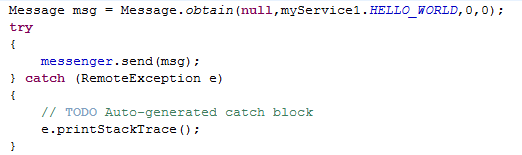


1. Messenger通过getBinder()方法创建IBinder，服务从onBind()方法将其返回到客户端。



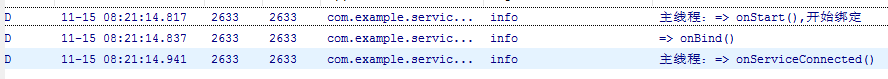
1. 客户端使用IBinder来实例化Messenger，然后使用它来发送Message对象到服务。





1. 服务在其Handler的handleMessage()方法接受Message。

**开始绑定**



**发送信息**



（3）使用AIDL（接口定义语言）

如果希望服务能够同时处理多个请求，则可以直接使用AIDL。

**以上三种方法也是Activity和Service之间建立通信的方法，除此还可以使用Broadcast（广播机制）方式实现Activity和Service的通信：**

**2.2.1、生命周期**

Context.bindService() ——> onCreate() ——> onbind() ——> running ——> unbind() ——> onDestory()

3．IntentService：它每次使用一个工作线程来处理全部启动的请求，开发者只需要实现onHanderIntent()方法，该方法接受每次启动请求的Intent以便完成后台任务。

IntentService必须实现2个：没有参数的构造参数和onHandlerintent()方法。

