Service与Activity相似，都是从Context派生而来，因此它们都可以调用Context里定义的如getResources()、getContentResolver()方法。

Service与Activity都代表可执行的程序，其与Activity的区别在于：Service一直在后台运行，没有用户界面。其余方面与Activity几乎一模一样。

开发Service与开发Activity步骤相同：

1. 定义一个继承自Service的子类（开发Activity时是继承自Activity）
2. 在AndroidManifest.xml中配置该Service

有2种方式运行Service：

1. 通过Context的startService()方法

启动者与Service之间没有关联，即使启动者退出，Service也仍然运行。Service与启动者之间也无法进行通信、交换数据。

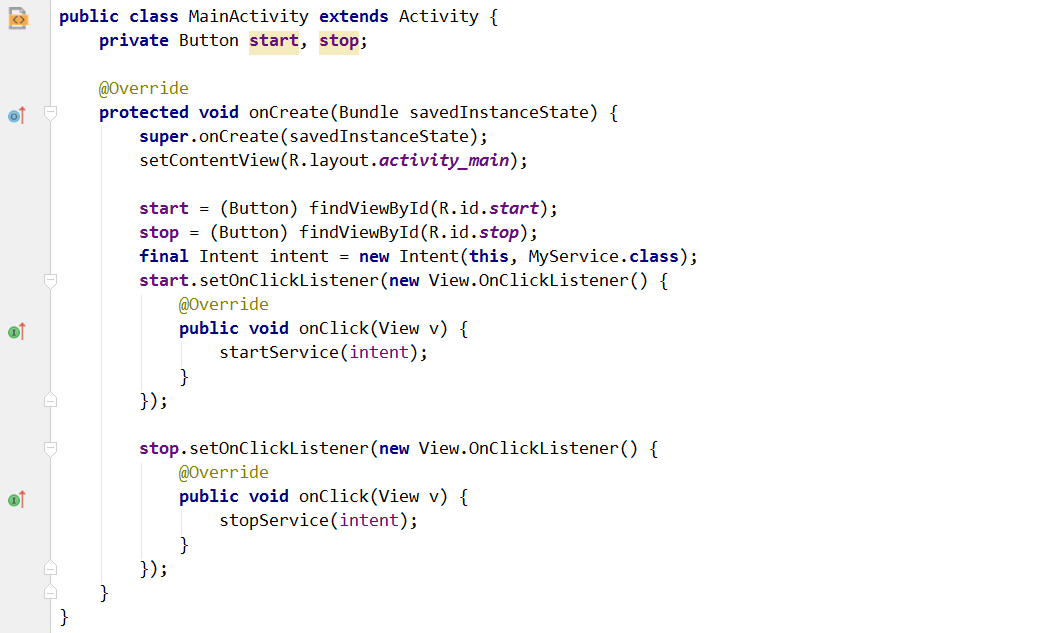
1. 通过Context的bindService()方法

启动者与Service绑定在一起，启动者一旦退出，Service也随之终止。Service与启动者之间可以进行方法调用或交换数据。

通过startService()启动：

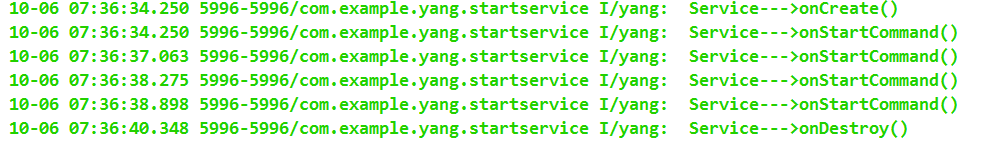
启动Service： startService(intent)

关闭Servcie： stopService(intent)





从Android5.0开始，Google要求必须使用显式Intent启动Service。



从上可以看出：

IBinder onBind(Intent intent)：Service被绑定时回调该方法，此处不会绑定，不会调用

onCreate()：Service第一次被创建时回调.

onStartCommand()：每次启动者调用startService()方法时回调，早期版本是onStart().

onDestroy()：Service被关闭前回调

通过BindService()启动：

启动Service：bindService()

关闭Service：unbindService()

Context的bindService()方法完整签名为：

bindServie(Intent intent, **ServiceConnection conn**, int flags)

参数为；

intent：启动Service的intent

**conn**：是一个ServiceConnection对象，用于监听启动者与Service之间的连接情况，当启动者与Service之间连接成功时将回调conn的**onServiceConnected()**方法；当Service所在的宿主进程由于异常或其他原因终止，导致该Service与启动者之间断开连接时将回调**onServiceDisconnected()**方法.

flags : 指定绑定时是否自动创建Service，0(不自动创建)或BIND\_AUTO\_CREATE

MyService.java



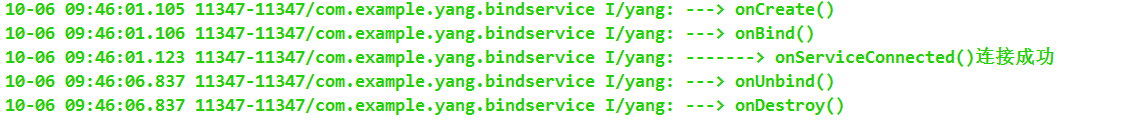
MainActivity.java





对于Service的onBind()方法所返回的IBinder对象来说，其可以被当成该Service所返回的代理对象，Service允许启动者通过IBinder对象来访问Service内部的数据。

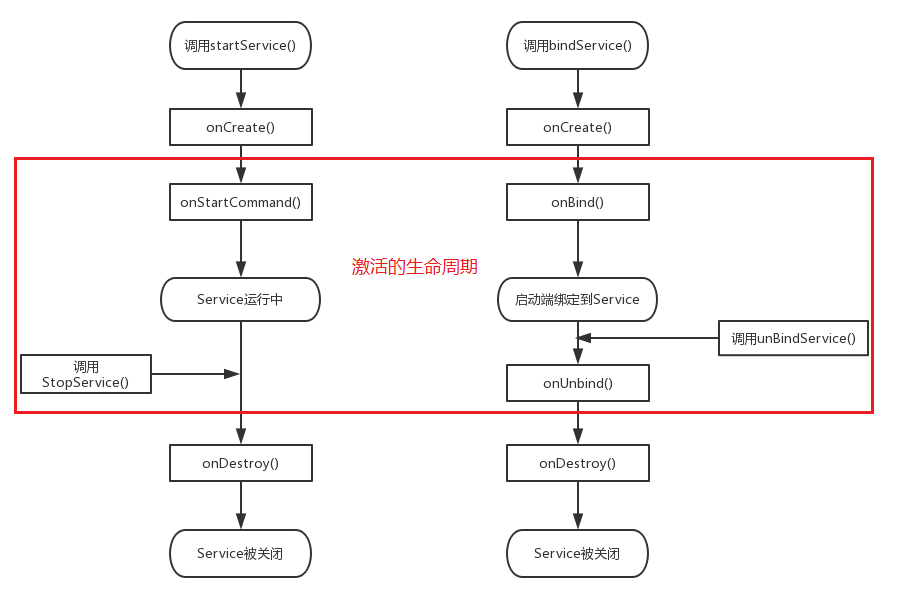
运行结果：



与多次调用startService()会多次回调onStartCommand()方法不同，多次调用bindService()方法并不会执行重复绑定。

当调用unbindService()方法解除对某个Service的绑定时，先回调Service的onUnbind()方法，再回调onDestroy()方法。

两种启动方式的生命周期：



特殊情况：

当Activity调用bindService()绑定一个已经启动的Service时（由其他Activity通过startService()启动），系统只会把Service内部的IBinder对象传给Activity，并不会把Service生命周期完全“绑定”到该Activity。此时，当Activity调用unBindService()解除与Service的绑定时，也只是切断Activity与Service之间的关联，并不能停止该Service组件。

**IntentService**:

普通Service存在两个问题：

1. Service与它所在应用位于同一个进程中
2. Service仍然位于UI线程中，不能在Service中直接处理耗时任务

IntentService是Service的子类，比普通的Service增加了额外的功能。IntentService使用队列来管理请求intent，每当客户端通过intent请求启动IntentService时，IntentService会将该Intent加入到队列中，然后开启一条新的工作线程来处理该Intent。对于异步的startService()请求，intentService会按次序依次处理队列中的Intent，该线程保证同一时刻只处理一个intent。

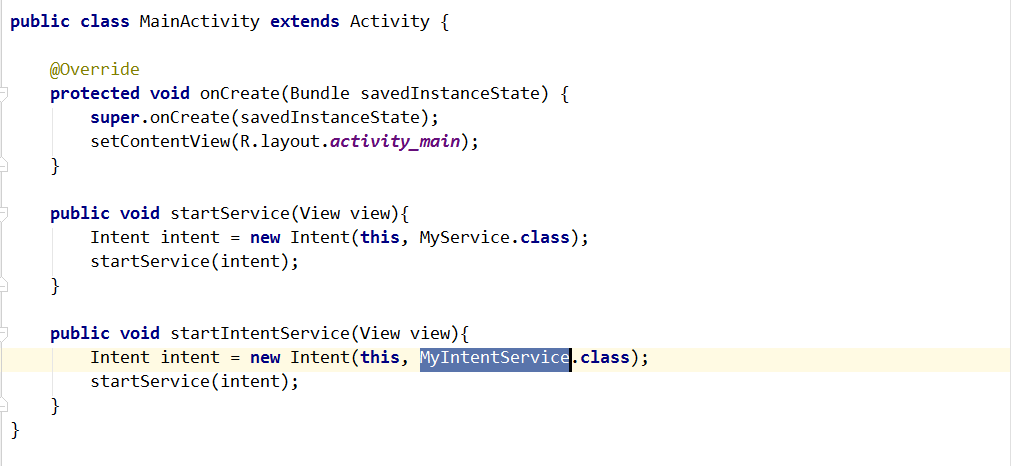
由于IntentService使用新的工作线程处理Intent请求，因此IntentService不会阻塞主线程，故IntentService自己就可以处理耗时任务。

特征：

1. IntentService会创建单独的工作线程来处理onHandleIntent()方法实现的代码
2. 当所有请求处理完成后，IntentService会自动停止，开发者无需手动调用stopSelf()方法来停止该Service.
3. 为Service的onBind()方法提供了默认实现（返回null）.
4. 为Service的onStartCommand()提供了默认实现，该实现会将请求Intent添加到队列中

扩展IntentService实现Service不需要重写onBind()、onStartCommand()方法，只需重写onHandleIntent()方法即可。

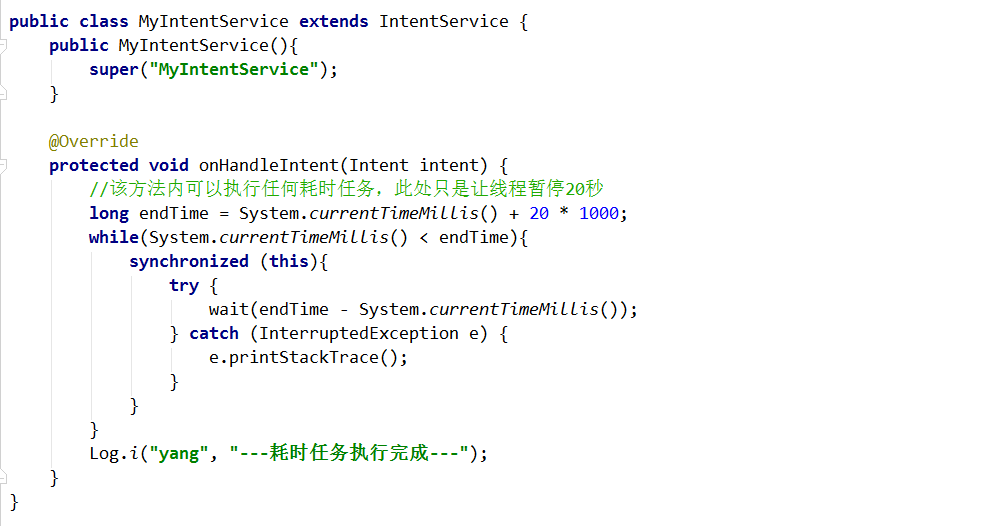
MainActivity.java



MyService.java



MyIntentService.java



运行结果：

普通Service执行耗时任务时会导致ANR异常，而IntentService则不会

