Service分析与使用

## Service概述

Service是Android四大组件之一，官方对Service的定义：A [Service](http://developer.android.com/reference/android/app/Service.html) is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background。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。Service有两种状态，“启动的”和“绑定”

通过startService()启动的服务处于“启动的”状态，一旦启动，service就在后台运行，即使启动它的应用组件已经被销毁了。通常started状态的service执行单任务并且不返回任何结果给启动者。比如当下载或上传一个文件，当这项操作完成时，service应该停止它本身。

还有一种“绑定”状态的service，通过调用bindService()来启动，一个绑定的service提供一个允许组件与service交互的接口，可以发送请求、获取返回结果，还可以通过夸进程通信来交互（IPC）。绑定的service只有当应用组件绑定后才能运行，多个组件可以绑定一个service，当调用unbind()方法时，这个service就会被销毁了。

另外，在官方的说明文档中还有一个警告：意思是service与activity一样都存在与当前进程的主线程中，所以，一些阻塞UI的操作，比如耗时操作不能放在service里进行，比如另外开启一个线程来处理诸如网络请求的耗时操作。如果在service里进行一些耗CPU和耗时操作，可能会引发ANR警告，这时应用会弹出是强制关闭还是等待的对话框。所以，对service的理解就是和activity平级的，只不过是看不见的，在后台运行的一个组件，这也是为什么和activity同被说为Android的基本组件。

## Service问题导入

Service与Activity的不同之处

1、除了没有界面，其他没什么不同。

2、Service的创建也是在主线程

3、Service可以提供context

4、Service如果要进行耗时操作，需要启动新的线程，否则阻塞，不可在onCreate中运行耗时任务

5、Service要配合HandlerThread使用才比较合理

6、Service一般配合后台线程使用，但是如果你想要更新前台如何处理，就必须包装一下，使用一个主线程的Handler来操作。，但是后台线程可以用后台线程的Handler来处理

7、Binder Service的目的：初始化线程，另外可以复用服务，生命周期处理比较好。也是作为提供服务来处理的，一般也是死循环。

8、如果想使用单利模式，最好采用Application引用。注意单利模式只是class本身static，其内部的Field是没必要的。

9、Service可以看成服务的抽象，上层Java实现的时候，不直接采用Binder通信，而是封装了Binder，之所以用Service，就是弄过一个可以无限循环的Thead，作为本地服务，简单的服务，Service是一个入口组件，四大组件之一，为了管理，绑定为了获取服务句柄，其实类似Binder通信。只是抽象层次更高了。其服务内容的实现被封装了。ThreadHanldler与HanlderThread，总之要有个 线程在运行，才叫服务。不然，就是一次Transaction，统一的服务管理接口，或者说服务接口，本地更方便。

10、注意这里服务区分本地Service还是远程进程通信，如果远程通信，获取引用，如果是本地，则直接用，也是说，如果在本地，是地址，如果不是事句柄引用。java层的反应是AIDL

## 3、Service使用场景：

经常复用的重复Task中，最好用HandlerThread，不要忙等待，java自身是没有等待一说的。

Should you use a service or a thread?：使用线程还是Service

Service的使用场景是：即使在没有交互的情况下，也能在后台运行，但是，如果想要在主线程中操作后运行一个外部任务，就应该创建一个线程而不是Service。

## 4、Service用法：



通过这个图可以看到，两种启动service的方式以及他们的生命周期，bind service的不同之处在于当绑定的组件销毁后，对应的service也就被kill了。service的声明周期相比与activity的简单了许多，只要好好理解两种启动service方式的异同就行。

Service、Activity这些都是组件，这些组件都没有采用new的方式来启动，都是通过Android中的“系统调用”来实现的，座椅这些类的实例由Android系统保证，创建在MainThread中，所以不必考虑在子线程中创建组件的实例的问题。

## 5、Service的Bound原理：（主线程）

## 6、IntentService

IntentService是Service的子类，比普通的Service增加了额外的功能。先看Service本身存在两个问题：

* Service不会专门启动一条单独的进程，Service与它所在应用位于同一个进程中；
* Service也不是专门一条新线程，因此不应该在Service中直接处理耗时的任务；

因为大多数情况下，都是单任务在执行，所以可以采用IntentService。

* 会创建独立的worker线程来处理所有发送到[onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) 的Intent请求；
* 创建一个Intent队列，么次都得会向[onHandleIntent()](http://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent))发送一个Intent,无需担心多线程竞争问题
* 所有请求处理完成后，IntentService会自动停止，无需调用stopSelf()方法停止
* 为Service的onBind()提供默认实现，返回null；
* 为Service的onStartCommand提供默认实现，将请求Intent添加到队列中；

使用方法：实现构造方法与 [onHandleIntent()](http://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)).如果要覆盖已经实现的方法，注意调用父类方法。比如 [onStartCommand()](http://developer.android.com/reference/android/app/IntentService.html#onStartCommand(android.content.Intent, int, int)) 必须返回默认的返回。

* @Override  
  public int onStartCommand(Intent intent, int flags, int startId) {  
      Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
      return super.onStartCommand(intent,flags,startId);  
  }

## 6、Service

如果使用场景是需要存在多线程并发，用法是和HandlerThread配合Handler使用

public class HelloService extends Service {  
  private Looper mServiceLooper;  
  private ServiceHandler mServiceHandler;  
  // Handler that receives messages from the thread  
  private final class ServiceHandler extends Handler {  
      public ServiceHandler(Looper looper) {  
          super(looper);  
      }  
      @Override  
      public void handleMessage(Message msg) {  
          // Normally we would do some work here, like download a file.  
          // For our sample, we just sleep for 5 seconds.  
          long endTime = System.currentTimeMillis() + 5\*1000;  
          while (System.currentTimeMillis() < endTime) {  
              synchronized (this) {  
                  try {  
                      wait(endTime - System.currentTimeMillis());  
                  } catch (Exception e) {  
                  }  
              }  
          }  
          // Stop the service using the startId, so that we don't stop  
          // the service in the middle of handling another job  
          stopSelf(msg.arg1);  
      }  
  }  
  @Override  
  public void onCreate() {  
    // Start up the thread running the service.  Note that we create a  
    // separate thread because the service normally runs in the process's  
    // main thread, which we don't want to block.  We also make it  
    // background priority so CPU-intensive work will not disrupt our UI.  
    HandlerThread thread = new HandlerThread("ServiceStartArguments",  
            Process.THREAD\_PRIORITY\_BACKGROUND);  
    thread.start();  
    // Get the HandlerThread's Looper and use it for our Handler  
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
  }  
  
  @Override  
  public int onStartCommand(Intent intent, int flags, int startId) {  
      Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
  
      // For each start request, send a message to start a job and deliver the  
      // start ID so we know which request we're stopping when we finish the job  
      Message msg = mServiceHandler.obtainMessage();  
      msg.arg1 = startId;  
      mServiceHandler.sendMessage(msg);  
  
      // If we get killed, after returning from here, restart  
      return START\_STICKY;  
  }  
  
  @Override  
  public IBinder onBind(Intent intent) {  
      // We don't provide binding, so return null  
      return null;  
  }  
  
  @Override  
  public void onDestroy() {  
    Toast.makeText(this, "service done", Toast.LENGTH\_SHORT).show();  
  }  
}

## 7、Creating a Started Service

只有通过startService，才会调用[onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) ，[IntentService](http://developer.android.com/reference/android/app/IntentService.html) 是为了startService准备的

[START\_NOT\_STICKY](http://developer.android.com/reference/android/app/Service.html#START_NOT_STICKY)

If the system kills the service after [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, do not recreate the service, unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

[START\_STICKY](http://developer.android.com/reference/android/app/Service.html#START_STICKY)

If the system kills the service after [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)), but *do not* redeliver the last intent. Instead, the system calls [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands, but running indefinitely and waiting for a job.

[START\_REDELIVER\_INTENT](http://developer.android.com/reference/android/app/Service.html#START_REDELIVER_INTENT)

If the system kills the service after [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call[onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.