

## 1. Service 的基本认识

### 1.1 Service 是什么？

Service(服务)是一种可以在后台执行长时间运行操作而没有用户界面的组件。它运行于 UI 线程，因此不能进行耗时的操作。

### 1.2 Service 和 Thread 的区别

Service 的运行是在 UI 线程当中的，是绝对绝对不能进行耗时操作的，而 Thread 开启的子线程则可以进行耗时操作，但是 Thread 开启的子线程是不能直接对 UI 进行操作的，否则极有可能发生直接让程序崩掉，这就是它们的区别。

## 2. 启动 Service 的 2 种方式

### 2.1 startService()方法开启 Service

步骤：

- 定义一个类继承 Service。
- 在 AndroidManifest.xml 文件中配置该 Service。
- 使用 Context 的 startService(Intent)方法启动该 Service。
- 不再使用该 Service 时，调用 Context 的 stopService(Intent)方法停止该 Service。

### 2.2 bindService 方法开启 Service(Activity 与 Service 绑定)

步骤：

- 创建BinderService 服务端，继承自 Service 并在类中创建一个实现 IBinder 接口的实现实例对象并提供公共方法给客户端调用。
- 从 onBind()回调方法返回此 Binder 实例。
- 在客户端中，从 onServiceConnected 回调方法接收 Binder,并使用提供的方法调用绑定服务。

## 3. Service 的生命周期

服务的生命周期有两种，因为服务可以跟 Activity 绑定起来，也可以不绑定，Activity 和服务进行通信的话，是需要把服务和 Activity 进行绑定的。因此服务的生命周期分为未绑定Activity 的和绑定 Activity 的。

没有绑定 Activity 的服务生命周期图：



绑定 Activity 的服务生命周期图:



淘宝搜《闵课通商学院》、小白轻松拿高薪offer

1.通过Intent 和startService()方法启动了一个服务，接下来执行 onCreate()方法,首次创建服务时，系统将调用此方法来执行一次性设置程序（在调用 onStartCommand() 或 onBind()之前）。如果服务已在运行，则不会调用此方法。

2.当另一个组件（如 Activity）通过调用 startService() 请求启动服务时，系统将调用此方法。一旦执行此方法，服务即会启动并可在后台无限期运行。如果您实现此方法，则在服务工作完成后，需要由您通过调用 stopSelf() 或 stopService() 来停止服务。（如果您只想

提供绑定，则无需实现此方法。)

3.服务开始处于运行状态。

4.某个操作导致服务停止，比如执行了方法 `stopService()`，那么服务接下来会执行 `onDestroy()` 销毁。服务应该实现此方法来清理所有资源，如线程、注册的侦听器、接收器等。这是服务接收的最后一个调用。

5.服务被完全销毁，下一步就是等待被垃圾回收器回收了。

6.通过 `Intent` 和 `bindService()` 方法启动了一个服务,接下来会执行 `onCreate()` 方法，首次创建服务时，系统将调用此方法来执行一次性设置程序（在调用 `onStartCommand()` 或 `onBind()` 之前）。如果服务已在运行，则不会调用此方法。

7.当另一个组件想通过调用 `bindService()` 与服务绑定（例如执行 `RPC`）时，系统将调用此方法。在此方法的实现中，您必须通过返回 `IBinder` 提供一个接口，供客户端用来与服务进行通信。请务必实现此方法，但如果您并不希望允许绑定，则应返回 `null`。

8.服务开始处于运行状态。成功与 `Activity` 绑定。

9.某个操作导致服务解除绑定，比如执行了方法 `unbindService()`，那么服务接下来会解除与当前 `Activity` 的绑定。接下来服务将面临销毁。

10.服务执行 `onDestroy()` 方法被销毁。服务应该实现此方法来清理所有资源，如线程、注册的侦听器、接收器等。这是服务接收的最后一个调用。

11.服务被完全销毁，下一步就是等待被垃圾回收器回收了。

关于服务，总结一下：

a 被启动的服务的生命周期：如果一个 `Service` 被某个 `Activity` 调用

`Context.startService` 方法启动，那么不管是否有 `Activity` 使用 `bindService` 绑定或 `unbindService` 解除绑定到该 `Service`，该 `Service` 都在后台运行。如果一个 `Service` 被 `startService` 方法多次启动，那么 `onCreate` 方法只会调用一次，`onStart` 将会被调用多次（对应调用 `startService` 的次数），并且系统只会创建 `Service` 的一个实例（因此你应该知道只需要一次 `stopService` 调用）。该 `Service` 将会一直在后台运行，而不管对应程序的 `Activity` 是否在运行，直到被调用 `stopService`，或自身的 `stopSelf` 方法。当然如果系统资源不足，`android` 系统也可能结束服务。

b 被绑定的服务生命周期：如果一个 `Service` 被某个 `Activity` 调用

`Context.bindService` 方法绑定启动，不管调用 `bindService` 调用几次，`onCreate` 方法都只会调用一次，同时 `onStart` 方法始终不会被调用。当连接建立之后，`Service` 将会一直运行，除非调用 `Context.unbindService` 断开连接或者之前调用

bindService 的 Context 不存在了 (如Activity 被finish 的时候)，系统将会自动停止Service，对应onDestroy 将被调用。

淘宝搜《[闵课通商学院](#)》，小白轻松拿高薪offer

c 被启动又被绑定的服务的生命周期：如果一个 **Service** 又被启动又被绑定，则该 **Service** 将会一直在后台运行。并且不管如何调用，**onCreate** 始终只会调用一次，对应 **startService** 调用多少次，**Service** 的 **onStart** 便会调用多少次。调用 **unbindService** 将不会停止 **Service**，而必须调用 **stopService** 或 **Service** 的 **stopSelf** 来停止服务。

d 当服务被停止时清除服务：当一个 **Service** 被终止（1、调用 **stopService**；2、调用 **stopSelf**；3、不再有绑定的连接（没有被启动））时，**onDestroy** 方法将会被调用，在这里你应当做一些清除工作，如停止在 **Service** 中创建并运行的线程。

特别注意：

1、你应当知道在调用 **bindService** 绑定到 **Service** 的时候，你就应当保证在某处调用 **unbindService** 解除绑定（尽管 **Activity** 被 **finish** 的时候绑定会自动解除，并且 **Service** 会自动停止）；

2、你应当注意使用 **startService** 启动服务之后，一定要使用 **stopService** 停止服务，不管你是否使用 **bindService**；

3、同时使用 **startService** 与 **bindService** 要注意到，**Service** 的终止，需要 **unbindService** 与 **stopService** 同时调用，才能终止 **Service**，不管 **startService** 与 **bindService** 的调用顺序，如果先调用 **unbindService** 此时服务不会自动终止，再调用 **stopService** 之后服务才会停止，如果先调用 **stopService** 此时服务也不会终止，而再调用 **unbindService** 或者之前调用 **bindService** 的 **Context** 不存在了（如 **Activity** 被 **finish** 的时候）之后服务才会自动停止；

4、当在旋转手机屏幕的时候，当手机屏幕在“横”“竖”变换时，此时如果你的 **Activity** 如果会自动旋转的话，旋转其实是 **Activity** 的重新创建，因此旋转之前的使用 **bindService** 建立的连接便会断开（**Context** 不存在了），对应服务的生命周期与上述相同。

5、在 **sdk 2.0** 及其以后的版本中，对应的 **onStart** 已经被否决变为了 **onStartCommand**，不过之前的 **onStart** 任然有效。这意味着，如果你开发的应用程序用的 **sdk** 为 **2.0** 及其以后的版本，那么你应当使用 **onStartCommand** 而不是 **onStart**。