Service 与 BroadcaseReceiver

**本章要点**

* Service 组件的作用和意义
* 创建、配置Service
* 启动、停止Service
* 绑定本地Service并与之通信
* Service的生命周期
* 开发远程AIDL Service
* 在客户端程序中调用远程AIDL Service
* TelephonyManager的功能和用法
* 监听手机电话
* SmsManager的功能和用法
* 监听手机短信
* AudioManager的功能和用法
* BroadcastReceiver组件的作用和意义
* 开发、配置BroadcastReceiver组件
* 发送广播、发送有序广播
* 使用BroadcastReceiver接收系统广播

Service 是 Android四大组件中与Activity最相似的组件，它们都代表可执行的程序，Service与Activity 的区别在于：Service一直在后台运行，它没有用户界面，所以约不会到前台来。一早Service被启动起来之后，它就与Activity一样，完全具有自己的生命周期。关于程序中Activity与Service的选择标准是：如果某个程序组件需要在运行时向用户呈现某种界面，或者该程序需要与用户交互，就需要使用Activity，否则就应该考虑使用Service。

开发者开发Service的步骤与Activity的步骤很像，开发Service组件需要先开发一个Service的子类，然后在AndroidManifest.xml文件中配置该Service，配置时可通过<intent-filter …/>元素指定它可被哪些Intent启动。

Android系统本身提供了大量的Service组件，开发者可以通过这些系统Service来操作Android系统本身。

BroadcaseReceiver 组件就像是一个全局的事件监听器，只不过它用于监听系统发出来的Broadcast。通过使用BroadcastReceiver，即可在不同应用程序之间通信。

# 1 Service 简介

Service组件也是可执行的程序，它也有自己的生命周期。创建，配置Service与创建、配置Activity的过程基本相似。

## 1.1 创建、配置 Service

就像开发Activity需要两个步骤：

* 定义一个继承Service的子类
* 在AndroidManifest.xml文件中配置该Service。

Service与Activity还有一点相似之外，它们都是从Context派生出来的，因此它们都可以调用Context里定义的如 getResources(),getContextResolver()等方法。

与Activity相似的是，Service中也定义了一系列的生命周期方法。如下：

* abstract IBinder onBInd(Intent intent) : 该方法是Service子类必须实现的方法。该方法返回一个IBinder对象，应用程序可通过该对象与Service组件通信.
* void onCreate() : 当该Service第一次被创建后将立即架设该方法
* void onDestroy() : 当该Service被关闭之前将会回调该方法。
* void onStartCommand(Intent intent,int flags,int started):该方法的早期版本是void onStart(Intent intent,int started);每次客户端调用startService(intent) 方法启动该Service时都会架设方法。
* Boolean onUnBind(Intent intent) : 当该Service上绑定的所有客户端都断开连接时，将会架设方法方法。

## 1.2 启动和停止Service

## 1.3 绑定本地Service并与之通信

当程序通过startService()和stopService()启动，关闭Service 时，Service与访问者之间基本上不存在太多的关联，因此Service的访问者之间也无法进行通信，数据交换。

如果Service访问者之间需要进行方法调用或数据交换，则应该使用bindService()和unbindService()方法启动，关闭对象。

Context的BindService()方法的完整签名是：bindService(Intent service,ServiceConnection conn,int flags) ，该方法的三个参数的解释如下：

1. Service : 通过Intent指定要启动的Service
2. conn : 是一个ServiceConnection对象，该对象用于监听访问者与Service之间的连接情况，当访问者与Service之间连接成功时将架设该ServiceConnection对象的onServiceConnected(ComponentName,IBinder service) 方法，当访问者与Servic之间断开连接时将回调该ServiceConnection对象的onServiceDisconnected(ComponentName,name)方法。
3. flags : 指定绑定时是否自动创建Service（如果Service还未创建）；该参数可指定为0(不自动创建)或BIND\_AUTO\_CREATE(自动创建)

注意到ServiceConnection对象的onServiceConnected 方法中有一个IBinder对象，该对象即可实现与被绑定Service之间的通信。

当开发Service类时，该Service类必须提供一个IBinder onBind(Intent intent)方法，在绑定本地Service情况下，onBind(Intent intent) 方法所返回的IBinder对象将会传给ServiceCOnnection对象里onServiceConnected(ComponentName name,IBinder service) 方法的Service参数，这样访问者就可以通过该IBinder对象与Service进行通信。

# 2 跨进程调用Service（AIDL服务）

Android系统中，各应用程序都运行在自己的进程中，进程之间一般无法直接进行数据交换。

在过去的技术实现中，Corba可实现跨进程的调用，在Java技术中，RMI也可实现跨进程的调用，在Android的技术实现中，跨进程调用Service的方法与此基本相似。

## 2.1 AIDL服务简介

Android 的远程Service调用与Java的RMI基本相似，一样都先定义一个远程调用接口，然后该接口提供一个实现类即可。

与RMI不同的是，客户端访问Service时，Android并不是直接返回Service对象给客户端 – 这一点绑定本地Service时已经看到，Service只是将一个回调对象（IBinder对象）通过onBind()方法返回给客户端。因此Android的AIDL远程接口的实现类就是那个IBinder实现类。

与绑定本地Service不同的是，本地Service的onBInd()方法会直接把IBinder对象本身传给客户端的ServiceConnection的onServiceConnected 方法的第二个参数。但远程Service的onBind()方法只是将IBinder对象的代码传给客户端的ServiceConnection的onServiceConnected方法的第二个参数。

当客户端获取远程Service的IBinder对象的代码之后，接下来就可通过IBinder对象去架设远程Service属性或方法了。

## 2.2 创建AIDL文件

## 2.3 将接口暴露给客户端

## 2.4 客户端访问AIDLService

正如前面提到的,AIDL接口定义了两个进程之间的通信接口，因此不仅服务器端需要AIDL接口，客户端同样需要前面定义的AIDL接口。因此开发客户端的第一步就是将Service端的AIDL接口文件复制到客户端应用中，复制到客户端后ADT工具会为AIDL接口生成相应的实现类。

客户端绑定远程Service与绑定本地Service的区别并不大，同样只需要两步。

1. 创建ServiceConnection对象
2. 以ServiceConnection对象作为参数，调用Context的bindService()方法绑定远程Service即可。