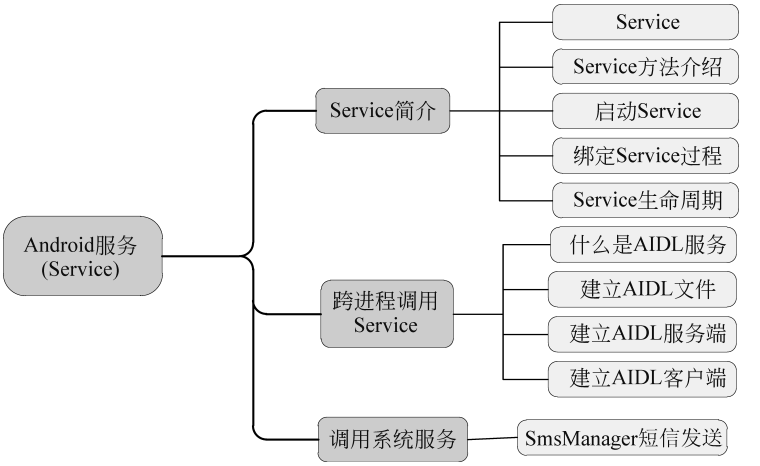
第5章

Android 服务(Service)

本章要点

- Service 组件的作用和意义
- Service 与 Activity 的区别
- 运行 Service 的两种方式
- 绑定 Service 执行的过程
- Service 的生命周期
- 跨进程调用 Service
- 调用系统发短信服务

本章知识结构图



本章示例

根据 Activity 的生命周期,程序中每次只有一个 Activity 处于激活状态,并且 Activity 的执行时间有限,不能做一些比较耗时的操作。如果需要多种工作同时进行,比如一边听音乐,一边浏览网页,使用 Activity 则比较困难。针对这种情况,Android 为我们提供了另一种组件——服务(Service)。

5.1 Service 概述

Service 是一种 Android 应用程序组件,可在后台运行一些耗时但不显示界面的操作。

5.1.1 Service 介绍

Service 与 Activity 类似,都是 Android 中四大组件之一,并且二者都是从 Context 派生而来,最大的区别在于 Service 没有实际的界面,而是一直在 Android 系统的后台运行,相当于一个没有图形界面的 Activity 程序,它不能自己直接运行,需要借助 Activity 或其他 Context 对象启动。

Service 主要有两种用途:后台运行和跨进程访问。通过启动一个 Service,可以在不显示界面的前提下在后台运行指定的任务,这样可以不影响用户做其他事情,如后台播放音乐,前台显示网页信息。AIDL 服务可以实现不同进程之间的通信,这也是 Service 的重要用途之一。

其他的应用程序组件一旦启动 Service,该 Service 将会一直运行,即使启动它的组件跳转到其他页面或销毁。此外,组件还可以与 Service 绑定,从而与之进行交互,甚至执行一些进程内通信。例如,服务可以在后台执行网络连接、播放音乐、执行文件操作或者是与内容提供者交互等。

5.1.2 启动 Service 的两种方式

在 Android 系统中,常采用以下两种方式启动 Service。

(1) 通过 Context 的 startService()启动 Service。启动后,访问者与 Service 之间没有关联,该 Service 将一直在后台执行,即使调用 startService 的进程结束了,Service 仍然还存在,直到有进程调用 stopService(),或者 Service 自杀(stopSelf())。这种情况下,Service 与访问者之间无法进行通信和数据交换,往往用于执行单一操作,并且没有返回结果。例如通过网络上传、下载文件,操作一旦完成,服务应该自动销毁。

(2) 通过 Context 的 bindService()绑定 Service。绑定后,Service 就和调用 bindService()的组件"同生共死"了,也就是说当调用 bindService()的组件销毁了,它绑定的 Service 也要跟着结束,当然期间也可以调用 unbindService()让 Service 提前结束。

注意:一个 Service 可以与多个组件绑定,只有当所有的组件都与之解绑后,该 Service 才会被销毁。

以上两种方式也可以混合使用,即一个 Service 既可以启动也可以绑定,只需要同时实现 onStartCommand()(用于启动)和 onBind()(用于绑定)方法,那么只有当调用 stopService(),并且调用 unbindService()方法后,该 Service 才会被销毁。

注意:Service 运行在它所在进程的主线程。Service 并没有创建它自己的线程,也没有运行在一个独立的进程上(单独指定的除外),这意味着,如果 Service 做一些消耗 CPU 或者阻塞的操作,应该在服务中创建一个新的线程去处理。通过使用独立的线程,就会降低程序出现 ANR(Application No Response,程序没有响应)风险的可能,程序的主线程仍然可以保持与用户的交互。

5.1.3　Service 中常用方法

与开发其他 Android 组件类似,开发 Service 组件需要先开发一个 Service 子类,该类需继承系统提供的 Service 类,系统中 Service 类包含的方法主要有以下几种。

(1) abstractIBinder onBind(Intent intent):该方法是一个抽象方法,因此所有 Service 的子类必须实现该方法。该方法将返回一个 IBinder 对象,应用程序可通过该对象与 Service 组件通信。

(2) void onCreate():当 Service 第一次被创建时,将立即回调该方法。

(3) void onDestroy():当 Service 被关闭之前,将回调该方法。

(4) void onStartCommand(Intent intent, int flags,int startId):该方法的早期版本是 void onStart(Intent intent, int startId),每次客户端调用 startService(Intent intent)方法启动该 Service 时都会回调该方法。

(5) boolean onUnbind(Intent intent):当该 Service 上绑定的所有客户端都断开连接时将会回调该方法。

定义的 Service 子类必须实现 onBind()方法,然后还需在 AndroidManifest.xml 文件中对该 Service 子类进行配置,配置时可通过＜intent-filter.../＞元素指定它可被哪些 Intent 启动。下面具体来创建一个 Service 子类并对它进行配置,代码如下。

程序清单：codes\ch05\ServiceTest\app\src\main\java\iet\jxufe\cn\servicetest\MyService.java

```
1  public class MyService extends Service {                    →自定义服务类
2      private static final String TAG ="MyService";           →定义字符串常量
3      public IBinder onBind(Intent arg0) {                    →重写 onBind()方法
4          Log.i(TAG, "MyService onBind invoked!");            →打印日志信息
5          return null;
6      }
7      public void onCreate() {                                →重写 onCreate()方法
8          super.onCreate();                                   →调用父类方法
9          Log.i(TAG, "MyService onCreate invoked!");          →打印日志信息
10     }
11     public void onDestroy() {                               →重写 onDestroy()方法
12         super.onDestroy();                                  →调用父类方法
13         Log.i(TAG, "MyService onDestroy invoked!");         →打印日志信息
14     }
15     public int onStartCommand(Intent intent, int flags, int startId) {
                                                               →重写方法
16         Log.i(TAG, "MyService onStartCommand invoked!");    →打印日志信息
17         return super.onStartCommand(intent, flags, startId);  →返回结果
18     }
19 }
```

上述代码中创建了自定义的 MyService 类，该类继承了 Android.app.Service 类，并重写了 onBind()、onCreate()、onStartCommand()、onDestroy()等方法，在每个方法中，通过 LOG 语句测试和查看该方法是否被调用。

定义 Service 之后，还需在项目的 AndroidManifest.xml 文件中配置该 Service，增加配置片段如下。

```
1  <service android:name=".MyService">                          →Service 标签
2      <intent-filter>                                          →过滤条件
3          <action android:name="iet.jxufe.cn.MY_SERVICE"/>
4      </intent-filter>
5  </service>
```

虽然目前 MyService 已经创建并注册了，但系统仍然不会启动 MyService，要想启动这个服务，必须显式地调用 startService()方法。如果想停止服务，需要显式地调用 stopService()方法。下面的代码使用 Activity 作为 Service 的启动者，分别定义了“启动服务”和“关闭服务”两个按钮，并为它们添加了事件处理。

程序清单：codes\ch05\FirstService\src\iet\jxufe\cn\android\MainActivity.java

```
1  public class MainActivity extends AppCompatActivity {
2      private Intent intent;                                   →声明 Intent
```

```
3       @Override
4       protected void onCreate(Bundle savedInstanceState) {
5           super.onCreate(savedInstanceState);
6           setContentView(R.layout.activity_main);          →加载布局文件
7           intent=new Intent(this,MyService.class);         →创建 Intent 对象
8       }
9       public void start(View view){                        →按钮事件处理
10          startService(intent);                            →启动服务
11      }
12      public void stop(View view){                         →按钮事件处理
13          stopService(intent);                             →停止服务
14      }
15  }
```

运行本节的例子后,第一次单击“启动服务”按钮后,LogCat 视图下的输出如图 5-1 所示。

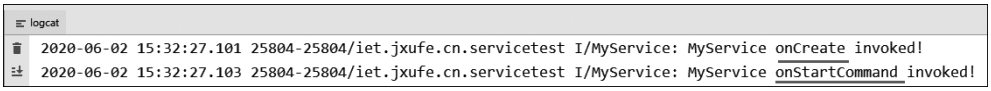


图 5-1　启动 Service LogCat 控制台打印信息

然后单击“停止服务”按钮,LogCat 视图有如图 5-2 所示的输出。

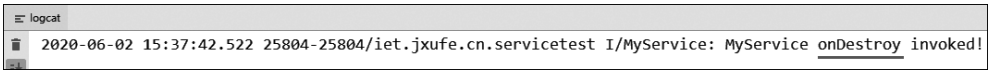


图 5-2　关闭 Service LogCat 控制台打印信息

下面按如下的单击按钮顺序的重新测试一下本例。

“启动服务”→“启动服务”→“启动服务”→“停止服务”

测试程序完成,查看 LogCat 控制台输出信息如图 5-3 所示。系统只在第一次单击“启动服务”按钮时调用 onCreate()和 onStartCommand()方法,再单击该按钮时,系统只会调用 onStartCommand()方法,而不会重复调用 onCreate()方法。

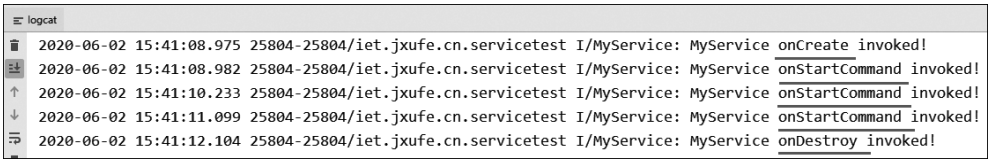


图 5-3　连续启动 Service LogCat 控制台打印信息

启动服务后退出该程序,查看 LogCat 控制台输出信息,发现并没有打印 onDestroy()方法被调用的信息,即服务并没有销毁,可以通过查看系统服务,查看该服务是否正在运行,具体操作为:打开模拟器功能清单中的“设置”,选择“关于模拟设备”,然后**连续单击版本号 7 次进入开发者模式**,接着退出“关于模拟设备”,进入“系统选项”,选择“高级”下面的

“开发者选项”,即可看到正在运行的服务,如图 5-4 所示。

图 5-4　查看系统正在运行的服务

5.1.4　绑定 Service 过程

如果使用 5.1.2 节介绍的方法启动服务,并且未调用 stopService()来停止服务,Service 将会一直驻留在手机的服务之中。Service 会在 Android 系统启动后一直在后台运行,直到 Android 系统关闭后 Service 才停止。但这往往不是我们所需要的结果,我们希望在启动服务的 Activity 关闭后 Service 会自动关闭,这就需要将 Activity 和 Service 绑定。在 Context 类中专门提供了一个用于绑定 Service 的 bindService()方法,Context 的 bindService()方法的完整方法签名为:bindService(Intent service,ServiceConnection conn, int flags),该方法的三个参数解释如下。

(1) service:该参数表示与服务类相关联的 Intent 对象,用于指定所绑定的 Service 应该符合哪些条件。

(2) conn:该参数是一个 ServiceConnection 对象,该对象用于监听访问者与 Service 间的连接。当访问者与 Service 间连接成功时,将回调该 ServiceConnection 对象的 onServiceConnected (ComponentName name,IBinder service)方法;当访问者与 Service 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 BIND_AUTO_CREATE(自动创建)。

当定义 Service 类时,该 Service 类必须提供一个 onBind()方法,在绑定本地 Service 的情况下,onBind()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 IBinder 对象与 Service 进行通信。

在上述示例中,添加两个按钮,一个用于绑定 Service,一个用于解绑,然后分别为其添加事件处理。在绑定 Service 时,需要传递一个 ServiceConnection 对象,所以先创建该对象。在 MainActivity.java 中添加如下代码。

```
1  private ServiceConnection conn=new ServiceConnection() {
2      public void onServiceDisconnected(ComponentName name) {
3          Log.i(TAG,"MainActivity onServiceDisconnected invoked!");
4      }
5      public void onServiceConnected(ComponentName name, IBinder service) {
6          Log.i(TAG,"MainActivity onServiceConnected invoked!");
7      }
8  };
```

然后在 MyService 中添加两个与绑定 Service 相关的方法 onUnbind()和 onRebind(),与其他方法类似,只在方法体中打印出该方法被调用了的信息,代码如下。

```
1  public boolean onUnbind(Intent intent) {
2      Log.i(TAG, "MyService onUnbind invoked!");          →打印日志信息
3      return super.onUnbind(intent);                      →调用父类方法
4  }
5  public void onRebind(Intent intent) {
6      super.onRebind(intent);                             →调用父类方法
7      Log.i(TAG, "MyService onRebind invoked!");          →打印日志信息
8  }
```

最后在 MainActivity 中为“绑定服务”和“解绑服务”按钮添加事件处理,代码如下。

```
1  public void bind(View view){                                →绑定服务事件处理
2      bindService(intent,conn, Service.BIND_AUTO_CREATE);     →绑定服务
3  }
4  public void unbind(View view){                              →解绑服务事件处理
5      unbindService(conn);                                    →解绑服务
6  }
```

程序执行后,单击“绑定服务”按钮,LogCat 控制台打印信息如图 5-5 所示,首先调用 onCreate()方法,然后调用 onBind()方法。

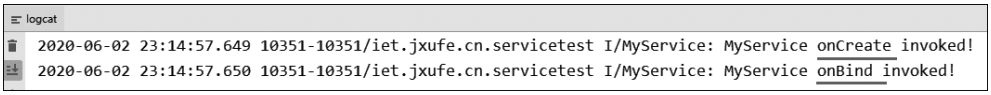


图 5-5　绑定服务 LogCat 控制台打印信息

单击“解绑服务”按钮,LogCat 控制台打印信息如图 5-6 所示,首先调用 onUnbind()方法,然后调用 onDestroy()方法。

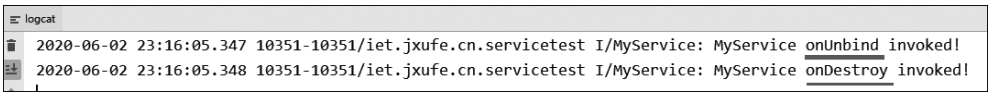


图 5-6　解绑服务 LogCat 控制台打印信息

程序运行后,单击“绑定服务”按钮,然后退出程序,LogCat 控制台打印信息如图 5-7 所示。可以看出,程序退出后,Service 会自动销毁。

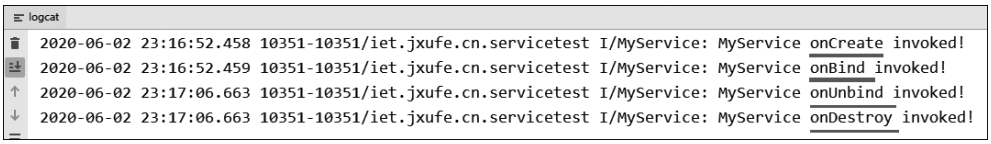


图 5-7　绑定服务直接退出程序 LogCat 控制台打印信息

当单击“绑定服务”按钮后,再重复多次单击“绑定服务”按钮,查看控制台打印信息,发现程序并不会多次调用 onBind()方法。

采用绑定服务的另一个优势是组件可以与 Service 之间进行通信,传递数据。这主要是通过 IBinder 对象进行的,因此需在 Service 中创建一个 IBinder 对象,然后让其作为 onBind()方法的返回值返回,对数据的操作是放在 IBinder 对象中的。修改 MyService 类,添加一个内部类 MyBinder,同时在 onCreate()方法中启动一个线程,模拟后台服务,该线程主要做数据递增操作,在 MyBinder 类中,提供一个方法,可以获取当前递增的值(count 的值),具体代码如下。

```
1  public class MyService extends Service {
2      private static final String TAG ="MyService";
3      private int count=0;
4      private boolean quit=false;                          →是否退出的标志
5      private MyBinder myBinder=new MyBinder();            →创建对象
6      public class MyBinder extends Binder {
7          public MyBinder() {                              →构造方法
8              Log.i(TAG, "MyBinder Constructure invoked!");→打印日志信息
9          }
10         public int getCount() {                          →获取数据方法
11             return count;                                →返回数据
12         }
13     }
14     public IBinder onBind(Intent arg0) {                 →重写 onBind 方法
15         Log.i(TAG, "MyService onBind invoked!");         →打印日志信息
16         return myBinder;                                 →返回对象
17     }
18 public void onCreate() {
19         Log.i(TAG, "MyService onCreate invoked!");
20         super.onCreate();
```

```
21          new Thread(){                                    →创建线程
22              public void run(){
23                  while(!quit){                            →判断是否退出
24                      try{
25                          Thread.sleep(500);               →休眠 0.5s
26                          count++;                         →数据递增
27                      }catch (Exception e) {               →捕获异常
28                          e.printStackTrace();             →打印异常信息
29                      }
30                  }
31              }
32          }.start();                                       →启动线程
33      }
34  public void onDestroy() {                                →重写方法
35          Log.i(TAG, "MyService onDestroy invoked!");      →打印日志信息
36          super.onDestroy();                               →调用父类方法
37          quit=true;                                       →修改退出标志
38  }
39  }
```

接着在 MainActivity 中添加一个“获取数据”的按钮，获取数据的前提是要绑定服务，所以先绑定服务，在 ServiceConnection()对象的 onServiceConnected()方法中，获取绑定服务时返回的 IBinder 对象，然后将该对象强制类型转换成 MyBinder 对象，最后利用 MyBinder 对象获取服务中的数据信息，

首先改写创建 ServiceConnection 对象的方法，将绑定服务中的 IBinder 对象强制类型转换成 MyService.MyBinder 对象，关键代码如下。

```
1   private ServiceConnection conn=new ServiceConnection() {
2       public void onServiceDisconnected(ComponentName name) {
3           Log.i(TAG,"MainActivity onServiceDisconnected invoked!");
4       }
5       public void onServiceConnected(ComponentName name, IBinder service) {
6           Log.i(TAG,"MainActivity onServiceConnected invoked!");
7           myBinder=(MyService.MyBinder)service;
8       }
9   };
```

然后，为“获取数据”按钮添加事件处理方法，关键代码如下。

```
1   public void getData(View view){                          →事件处理方法
2       Toast.makeText(MainActivity.this, "Count="+myBinder.getCount(),
            Toast.LENGTH_SHORT).show();                      →弹出显示数据
3   }
```

此时，单击“绑定服务”后，LogCat 控制台打印信息如图 5-8 所示，首先创建

MyBinder 对象,因为该对象是作为 MyService 的成员变量进行创建的,完成 MyService 的初始化工作,然后调用 onCreate()方法,再调用 onBind()方法,该方法返回一个 IBinder 对象,因为 IBinder 对象不为空,表示有服务连接,所以会调用 ServiceConnection 接口的 onServiceConnected()方法,并将 IBinder 对象作为它的第二个参数。

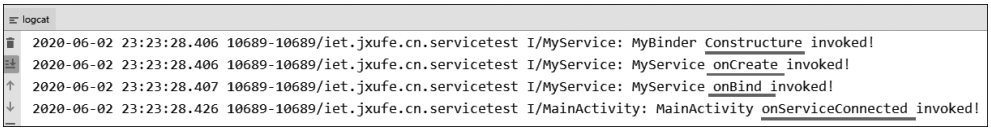


图 5-8　绑定 Service LogCat 控制台打印信息

单击“绑定服务”按钮后,后台服务就开始执行,此时单击“获取数据”按钮,得到如图 5-9 所示结果。多次单击时,得到的数据不一致,从而可以动态获取后台服务状态。

图 5-9　单击“获取数据”按钮的运行效果

当混合使用这两种运行 Service 方式时,它的执行效果又将是怎么样的呢? 下面我们以不同的顺序来运行服务,观察 LogCat 控制台打印的信息。

(1) 先启动 Service,然后绑定 Service。测试步骤为单击“启动服务”→“绑定服务”→“启动服务”→“停止服务”→“绑定服务”→“解绑服务”。LogCat 控制台打印信息如图 5-10 所示。

总结:调用顺序为 onCreate()→[onStartCommand()　1～N 次]→onBind()→onServiceConnected()→onUnbind()[→onServiceConnected()→onRebind()　0～N 次]→ onDestroy()。

(2) 先绑定 Service,后启动 Service。测试步骤为单击“绑定服务”→“启动服务”→“绑定