

Android Service基本用法



孟校长 [关注](#)



1 2017.09.02 01:26:45 字数 3,708 阅读 26,470

Service的概念

- Service作为安卓的四大组件之一，固然是每一位安卓开发者必须掌握的一个知识点。虽然它没有Activity的使用频繁，但也是日常开发经常用到的。
- 通过名字我们知道，它是服务的意思。而且通常是"默默"的为我们服务的。为什么说是默默，因为它并不像Activity一样，能够被我们看到。通常，它用于在后台为我们执行一些耗时，或者需要长时间执行的一些操作的。下面让我们来看看它的基本用法。

Service的创建

- 1.任何一个对象，想要发挥其作用，那么就应该首先创建出来。Service的创建和Activity类似，也是通过Intent来实现的。而且既然是安卓四大组件之一，那么它也需要在清单文件中进行注册的。下面，看一下一个简单的创建Service的例子：

```
1 public class SimpleService extends Service {  
2  
3     public static final String TAG = "SimpleService";  
4  
5     @Override  
6     public void onCreate() {
```

```

    public void onCreate() {
6         super.onCreate();
7         Log.d(TAG, "onCreate");
8     }
9
10    @Override
11    public int onStartCommand(Intent intent, int flags, int startId) {
12        Log.d(TAG, "onStartCommand");
13        return super.onStartCommand(intent, flags, startId);
14    }
15
16    @Override
17    public void onDestroy() {
18        super.onDestroy();
19        Log.d(TAG, "onDestroy");
20    }
21
22    @Override
23    public IBinder onBind(Intent intent) {
24        return null;
25    }
26 }
27

```

- 2.首先, 创建一个类SimpleService继承自Service,然后重写它的onCreate, onStartCommand, onDestroy方法, 并分别在它们的方法体中打入Log日志。其中onBind方法是默认实现的, 具体作用后面会讲到。然后呢, 千万不要忘记要在清单文件中注册它, 其实如果你是通过Android Studio直接new了一个Service的话, Android Studio会默认帮助你在清单文件中添加对该Service的注册, 代码如下:

```

1 <service android:name=".ui.main.SimpleService"
2         android:enabled="true"
3         android:exported="true"/>

```

我直接在我之前做的一个项目中新建的，所以不要在意包路径的命名，就是包名点类名。细心的朋友可能看到了下面还有两个属性。其中enabled属性，是指该服务是否能够被实例化。如果设置为true，则能够被实例化，否则不能被实例化，默认值是true。一般情况下，我们都会需要实例化，所以也可以选择不设置。而exported属性用于指示该服务是否能够被其他应用程序组件调用或跟它交互。如果设置为true，则能够被调用或交互(通常如果一个服务需要跨进程使用需要这么设置)，否则不能。设置为false时，只有同一个应用程序的组件或带有相同用户ID的应用程序才能启动或绑定该服务。

- 3.接下来创建一个StartActivity，用于在其中创建SimpleService对象。代码如下：

```
1 public class StartActivity extends AppCompatActivity implements View.OnClickListener {
2
3     private Button startBtn, stopBtn;
4
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         setContentView(R.layout.activity_start);
9
10        startBtn = (Button) findViewById(R.id.btn_start_service);
11        stopBtn = (Button) findViewById(R.id.btn_stop_service);
12
13        startBtn.setOnClickListener(this);
14        stopBtn.setOnClickListener(this);
15    }
16
17    @Override
18    public void onClick(View v) {
19        if (v != null) {
20            switch (v.getId()) {
21                case R.id.btn_start_service:
22                    Intent startIntent = new Intent(this, SimpleService.class);
23                    startService(startIntent);
```

```

23         btn_start_service.setOnClickListener()
24         break;
25     case R.id.btn_stop_service:
26         Intent stopIntent = new Intent(this, SimpleService.class);
27         stopService(stopIntent);
28         break;
29     }
30 }
31 }
32 }

```

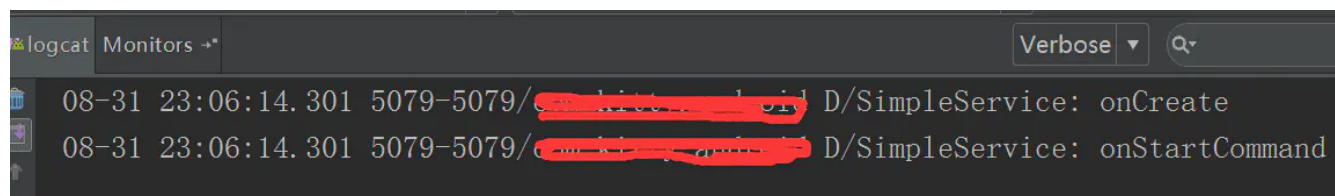
StartActivity的xml文件如下:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:orientation="vertical"
7      tools:context="com.kitty.android.ui.main.StartActivity">
8
9      <Button
10         android:id="@+id/btn_start_service"
11         android:layout_width="match_parent"
12         android:layout_height="50dp"
13         android:text="Start Service"/>
14
15     <Button
16         android:id="@+id/btn_destroy_service"
17         android:layout_width="match_parent"
18         android:layout_height="50dp"
19         android:text="Destroy Service"/>
20 </LinearLayout>

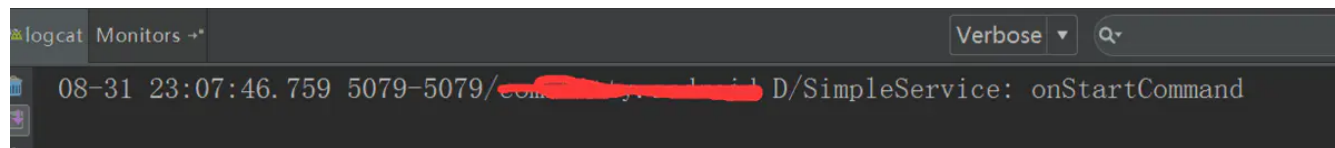
```

- 4.很简单，就是添加两个按钮，分别用于启动Service和停止Service。可以看到，创建一个Service的方法非常简单，就是和创建Activity类似，创建一个Intent对象，然后通过startService方法开启一个服务。然后同样的，通过stopService方法来停止一个服务。为了证明方法的正确性，我们先点击startBtn，看一下Logcat中的日志截图，如下所示：



servicestart.png

由此可以看出，当通过startService方法开启一个服务的时候，会执行Service的onCreate和onStartCommand方法。接下来，我们再次点击一下 start按钮，再来看一下Loacat的日志：



restartservice.png

这一次，只执行了onStartCommand方法，由此我们可以得出结论，当一个Service被创建以后，再次调用startService方法，Service是不会被重新创建的，而是会重新执行onStartCommand方法。无论我们点击多少次start按钮，始终只会执行onStartCommand方法。

以上是服务的创建。接下来，我们点击stop按钮，logcat日志如下：



stopservice.png

可以看到，Service执行了onDestroy方法，这时服务就已经停止了。

以上就是简单的创建一个服务的流程。然而，在我们日常开发中，我们经常需要在服务中做一些逻辑操作，然后将结果返回给一个Activity，即要实现Service和Activity的通信，接下来，让我们看看如何让二者建立起联系。

Service与Activity之间的通信

在上面的介绍中，我们只是实现了在Activity中开启一个服务，然而服务开启了就和这个Activity没什么联系了。其实二者是可以继续保持联络的，还记得前面提到的一个onBind方法吧，其实它就是Service与Activity之间建立通信的桥梁。现在我们修改一下前面的代码，SimpleService代码修改如下：

```
1 public class SimpleService extends Service {
2
3     public static final String TAG = "SimpleService";
4
5     private SimpleBinder mBinder;
6
7     @Override
8     public void onCreate() {
9         super.onCreate();
10        Log.d(TAG, "onCreate");
11        mBinder = new SimpleBinder();
12    }
13
14    @Override
15    public int onStartCommand(Intent intent, int flags, int startId) {
```

```

16         Log.d(TAG, "onStartCommand");
17         return super.onStartCommand(intent, flags, startId);
18     }
19
20     @Override
21     public void onDestroy() {
22         super.onDestroy();
23         Log.d(TAG, "onDestroy");
24     }
25
26     @Override
27     public IBinder onBind(Intent intent) {
28         if (mBinder != null) {
29             return mBinder;
30         }
31         return null;
32     }
33
34     class SimpleBinder extends Binder {
35
36         public void doTask() {
37             Log.d(TAG, "doTask");
38         }
39     }
40 }

```

现在，我们在SimpleService中创建了一个SimpleBinder类，继承自Binder。然后，在里面创建了一个doTask方法，模拟执行一个任务。然后，我们再在StartActivity中加入两个按钮，分别用于绑定服务和解绑服务，XML文件代码我就不贴了，就是添两个按钮，下面是StartActivity的更改后的代码：

```

1 public class StartActivity extends AppCompatActivity implements View.OnClickListener {
2
3     public static final String TAG = "SimpleService";

```

```
4
5     private Button startBtn, stopBtn, bindBtn, unBindBtn;
6
7     private SimpleService.SimpleBinder mBinder;
8
9     private ServiceConnection mConnection = new ServiceConnection() {
10         @Override
11         public void onServiceConnected(ComponentName name, IBinder service) {
12             Log.d(TAG, name.toString());
13             mBinder = (SimpleService.SimpleBinder) service;
14             mBinder.doTask();
15         }
16
17         @Override
18         public void onServiceDisconnected(ComponentName name) {
19             Log.d(TAG, name.toString());
20         }
21     };
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_start);
27
28         startBtn = (Button) findViewById(R.id.btn_start_service);
29         stopBtn = (Button) findViewById(R.id.btn_stop_service);
30         bindBtn = (Button) findViewById(R.id.btn_bind_service);
31         unBindBtn = (Button) findViewById(R.id.btn_un_bind_service);
32
33         startBtn.setOnClickListener(this);
34         stopBtn.setOnClickListener(this);
35         bindBtn.setOnClickListener(this);
36         unBindBtn.setOnClickListener(this);
37     }
38
39     @Override
40     public void onClick(View v) {
41         if (v != null) {
42             switch (v.getId()) {
```



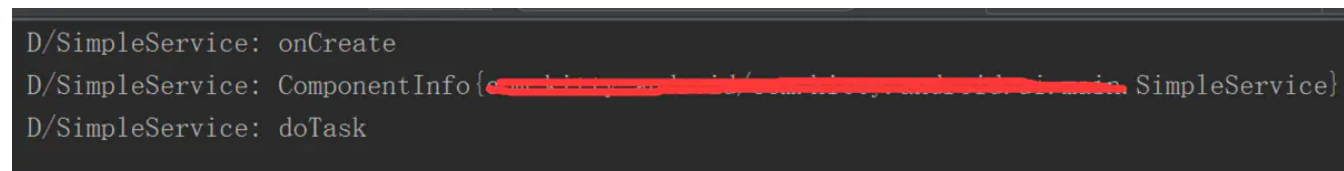
```
43         case R.id.btn_start_service:
44             Intent startIntent = new Intent(this, SimpleService.class);
45             startService(startIntent);
46             break;
47         case R.id.btn_stop_service:
48             Intent stopIntent = new Intent(this, SimpleService.class);
49             stopService(stopIntent);
50             break;
51         case R.id.btn_bind_service:
52             Intent bindIntent = new Intent(this, SimpleService.class);
53             bindService(bindIntent, mConnection, BIND_AUTO_CREATE);
54             break;
55         case R.id.btn_un_bind_service:
56             unbindService(mConnection);
57             break;
58     }
59 }
60 }
61 }
```

这里，我们创建了一个ServiceConnection的匿名内部类，并实现了onServiceConnected和onServiceDisconnected两个方法。ServiceConnection可以看做是一个由Activity操作的代表，负责与Service进行连接，当Activity与Service连接成功时，会执行onServiceConnected方法，相反的，当二者断开连接的时候，会执行onServiceDisconnected方法。当二者连接成功时，在onServiceConnected方法中，我们可以获取到SimpleService中的SimpleBinder的实例对象，然后我们就可以调用其所有的公共方法来实现我们想要做的事了。

现在我们来看一下绑定一个服务的方法，也是先创建一个Intent，然后将其作为bindService的第一个参数，第二个参数就是我们刚才说到的那个代表ServiceConnection，而第三个参数是一个标记，这里我们传入BIND_AUTO_CREATE标记，此标记表示在Activity和Service建立关联后自

动创建Service，并执行Service中的onCreate方法，并不会执行onStartCommand方法。(标记不只这一个，可以自行去查阅其他flag的含义)

为了验证刚刚我所说的，我们点击bind按钮，并查看logcat日志，如下：

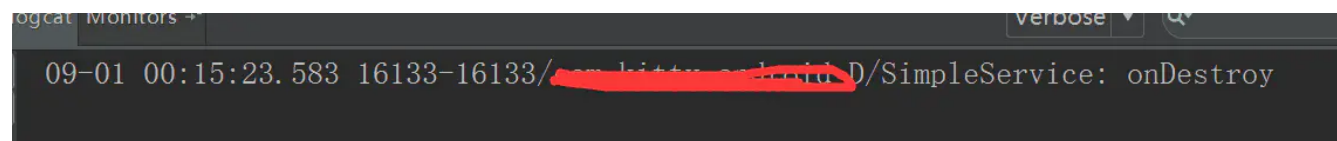


```
D/SimpleService: onCreate
D/SimpleService: ComponentInfo{com.kittu.android.D/SimpleService}
D/SimpleService: doTask
```

bindservice.png

可以看到，确实只执行了onCreate方法，并且在ServiceConnection的onServiceConnected中的第一个参数返回的其实就是SimpleService的具体包名路径。然后接着就执行了SimpleBinder中的doTask方法。当我们再次点击bind按钮，会发现，并没有执行任何的方法，说明了服务如果一旦与一个Activity绑定后，如果没有解绑的话，它是不会重新与这个Activity进行绑定的。

绑定完了，最终我们肯定是需要解绑的，来看一下unBind按钮的操作方法，只有一个unbindService方法，需要传入一个ServiceConnection参数，即我们前面创建的mConnection实例。点击unbind按钮，看到logcat日志如下：



```
09-01 00:15:23.583 16133-16133/com.kittu.android D/SimpleService: onDestroy
```

unbindservice.png

执行了onDestroy方法，这是刚刚通过bindService方法创建的Service就已经被摧毁了。

Service创建与摧毁方式的混淆

通过刚刚上述的描述，我们现在知道创建一个Service的方式有两种，即通过startService和bindService的方式。而二者对Service的摧毁方式分别为stopService和unBindService。那么可能很多人会疑问如果我选择startService的方式创建，然后选择unbindService的方式摧毁。或者说我采用bindService方式创建，stopService方式摧毁呢。那么我们来对每种情况分别验证一下会发生什么样的结果。

我们首先现将SimpleService中的代码还原成如下状态。StartActivity中还是四个按钮分别对应四个方法。

```
1 public class SimpleService extends Service {
2
3     public static final String TAG = "SimpleService";
4
5     private SimpleBinder mBinder;
6
7     @Override
8     public void onCreate() {
9         super.onCreate();
10        Log.d(TAG, "onCreate");
11        mBinder = new SimpleBinder();
12    }
13
14    @Override
15    public int onStartCommand(Intent intent, int flags, int startId) {
16        Log.d(TAG, "onStartCommand");
17        new Thread(new Runnable() {
18            @Override
19            public void run() {
20                // 任务逻辑
21            }
22        }).start();
```

```

23         return super.onStartCommand(intent, flags, startId);
24     }
25
26     @Override
27     public void onDestroy() {
28         super.onDestroy();
29         Log.d(TAG, "onDestroy");
30     }
31
32     @Override
33     public IBinder onBind(Intent intent) {
34         if (mBinder != null) {
35             return mBinder;
36         }
37         return null;
38     }
39
40     class SimpleBinder extends Binder {
41
42         public void doTask() {
43             Log.d(TAG, "doTask");
44         }
45     }
46 }

```

- 1.我们首先点击一下start按钮，然后点击unbind按钮，结果会发现程序崩了。

```

java.lang.IllegalArgumentException: Service not registered: com.kitt
    at android.app.LoadedApk.forgetServiceDispatcher(LoadedApk.java:1461)
    at android.app.ContextImpl.unbindService(ContextImpl.java:1339)
    at android.content.ContextWrapper.unbindService(ContextWrapper.java:672)
    at com.kitty.android.ui.main.MainActivity.onClick(MainActivity.java:100)

```

notregister.png

崩溃日志如下，提示非法状态异常，接着后面又说，Service没有被注册。其实，很容易理解，就是不看崩溃日志，估计很多人也猜到了，Service根本就没有与任何东西绑定，又何谈解绑呢。所以，这里需要注意，在使用unbindService方法关闭一个服务时，为防止出现以上的情况，我们需要在调用比方法前判断一下当前的Service是否已经被绑定。

针对这一判断谷歌并没有提供专门的Api，而我们可以通过在本地创建一个变量，当服务被绑定的时候，在ServiceConnection的onServiceConnected方法中将其设为true，在onServiceDisconnected方法中将其设为false。这样的话，如果服务未被绑定的话，这个值会永远为false，这样我们就直接在调用unbindService方法前做出相应的提示，以防止崩溃产生。

- 2.点击bind按钮后，再点击stop按钮。同样，看一下logcat日志：



```
logcat Monitors →* Verbose Q
09-02 00:34:09.428 12222-12222/com.kitty.android D/SimpleService: onCreate
09-02 00:34:09.438 12222-12222/com.kitty.android D/SimpleService: doTask
```

bindstop.png

请相信我，我真的点击了stop按钮，而且不止点击了一次。可以看到，Service并没有被摧毁。而当我再次点击unbind按钮时，Service才被摧毁。其实这也很好解释，举个例子吧，施瓦星格拍的终结者(第二部)不知道大家有没有看过，施瓦星格扮演的终结者的使命是保护男主，他的程序一旦被启动，就会一直以保护男主(完成使命)为目的，只有消灭了敌人，确保男主可以平安无

事了，他才会停止保护男主的行为。Service就相当于施瓦星格，它一旦被绑定，必须等到一个结果来告诉它"使命"完成了(解绑了)，它才会停止下来。

- 3.点击start按钮后，接着点击bind按钮，然后呢，我们分别单独点击stop和unbind按钮，发现Service都不会被摧毁，只有在我们两个按钮都点击了以后(两个按钮的点击顺序无所谓)，Service才会被摧毁。通过这这样一个结论，我们可以得出，一个Service，只有在即没有和任何Activity绑定又处于停止状态下的时候，才可以被摧毁。
- 4.最后，还要大家清楚一个问题就是。当你通过start按钮，开启一个Service后，再次点击bind按钮，只会执行服务的doTask方法，也就是Service绑定的方法回调，而不会执行onCreate方法再次创建一个Service。而同样的当通过点击bind按钮开启一个服务后，再点击start按钮，也不会执行onCreate方法，只会每次点击都执行onstartCommand方法，这里在前面也提到过。

Service的执行线程

很多对Service了解的不是很透彻的同学，当被问到Service运行在什么线程中，很多人都会第一反应是子线程。原因呢，因为大多数人都知道Service通常用来执行一些比较耗时的后台任务，既然提到了耗时，那么肯定不会运行在主线程啊，因为那样的话会阻塞线程的啊。其实呢，并不是这样的，Service其实是运行在主线程的。为了证明我所说的，我分别在StartActivity的onCreate方法和SimpleService的onCreate方法中获取当前运行的线程的名字和id，代码如下：

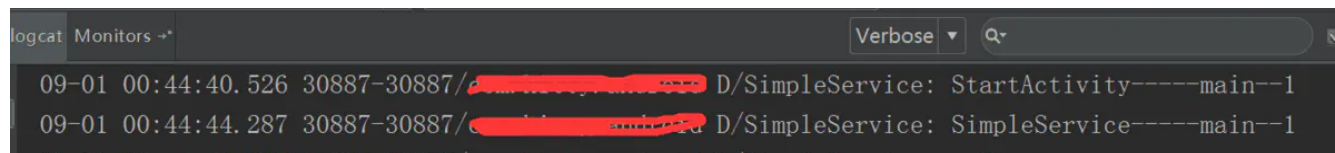
```
1 // Activity的onCreate方法
2 @Override
3 protected void onCreate(Bundle savedInstanceState) {
4     super.onCreate(savedInstanceState);
5     setContentView(R.layout.activity_start);
```

```

6         Log.d(TAG, "StartActivity-----" + Thread.currentThread().getName() + "--" + Thread.curren
7     }
8
9     // Service的onCreate方法
10    @Override
11    public void onCreate() {
12        super.onCreate();
13        Log.d(TAG, "SimpleService-----" + Thread.currentThread().getName() + "--" + Thread.curren
14    }

```

分别启动Activity和服务,Logcat日志如下:



```

logcat Monitors +* Verbose Q-
09-01 00:44:40.526 30887-30887/D/SimpleService: StartActivity-----main--1
09-01 00:44:44.287 30887-30887/D/SimpleService: SimpleService-----main--1

```

servicethread.png

可以看到, Service真的和Activity一样, 是运行在主线程的。

下面我列举出一个例子来演示在服务中开启一个线程来执行操作:

```

1 public class SimpleService extends Service {
2
3     private SimpleBinder mBinder;
4
5     @Override
6     public void onCreate() {
7         super.onCreate();
8         mBinder = new SimpleBinder();
9     }
10

```

```
11     @Override
12     public int onStartCommand(Intent intent, int flags, int startId) {
13         new Thread(new Runnable() {
14             @Override
15             public void run() {
16                 // 任务逻辑
17             }
18         }).start();
19         return super.onStartCommand(intent, flags, startId);
20     }
21
22     @Override
23     public void onDestroy() {
24         super.onDestroy();
25     }
26
27     @Override
28     public IBinder onBind(Intent intent) {
29         if (mBinder != null) {
30             return mBinder;
31         }
32         return null;
33     }
34
35     class SimpleBinder extends Binder {
36
37         public void doTask() {
38             new Thread(new Runnable() {
39                 @Override
40                 public void run() {
41                     // 任务逻辑
42                 }
43             }).start();
44         }
45     }
46 }
```


看到了吧，是不是很简单！

以上就是Service的一些基础用法，只有掌握了这些基本用法以后，我们才能更好的利用Service这一组件来完成我们平时日常中的开发需求。特别提醒的是，要根据场景恰当的选择使用Service，不要拿出一个任务就用Service来实现，以免造成不必要的开销。