New Comer - Android Service 组件

1 什么是服务

服务是一种在后台运行的组件，用于执行长时间运行的操作或为远程进程执行作业。
服务不提供用户界面。

> 例如，当用户位于其他应用中时，服务可能在后台播放音乐或者通过网络获取数据，但不会阻断用户与 Activity 的交互。

服务可由其他应用组件启动，而且即使用户切换到其他应用，服务仍将在后台继续运行。
此外，组件可以绑定到服务，以与之进行交互，甚至是执行进程间通信 (IPC)。

> 例如，服务可以处理网络事务、播放音乐，执行文件 I/O 或与内容提供程序交互，而所有这一切均可在后台进行。

> **CAUTION:** 服务和线程的关系
>
> 服务默认运行在主线程（UI 线程）
>
> 当使用服务处理耗时操作时，需要在服务内创建新线程来完成；否则会发生 ANR

2 服务的启动方式

2.1 startService() 启动

当应用组件（如 Activity）通过调用 `startService()` 启动服务时，服务即处于“启动”状态。

一旦启动，服务即可在后台无限期运行，即使启动服务的组件已被销毁也不受影响。 直到服务使用 `stopSelf()` 自行停止运行，或由其他组件通过调用 `stopService()` 停止它为止。

已启动的服务通常是执行单一操作，而且不会将结果返回给调用方。例如，它可能通过网络下载或上传文件。 操作完成后，服务会自行停止运行。

当服务通过 `startService()` 启动时，服务的 `onStartCommand()` 会被调用。

> **CAUTION:** `startService()` 启动服务后，服务默认运行在后台，在设备 OOM / LMK 回收内存时，后台进程会被系统认为可以安全的杀死。
>
> 所以，对于期望一直保持运行的服务 – 例如音乐播放，在启动服务后，应该再通过 `startForeground` 把服务设置为前台服务。

2.2 bindService() 绑定

当应用组件通过调用 `bindService()` 绑定到服务时，服务即处于“绑定”状态。

绑定服务提供了一个客户端-服务器接口，允许组件与服务进行交互、发送请求、获取结果，甚至是利用进程间通信 (IPC) 跨进程执行这些操作。

仅当与另一个应用组件绑定时，绑定服务才会运行。 多个组件可以同时绑定到该服务，但全部取消绑定后，该服务即会被销毁。

当服务被绑定时，服务的 `onBind` 方法会被调用。

> **NOTE:** 一个服务可以同时支持两种启动方式

3 创建服务的步骤

- 创建 `Service` 子类
- 重写必要的回调方法
- 在清单中声明服务

3.1 服务的主要回调方法

- onCreate

  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。

- onStartCommand

  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。）

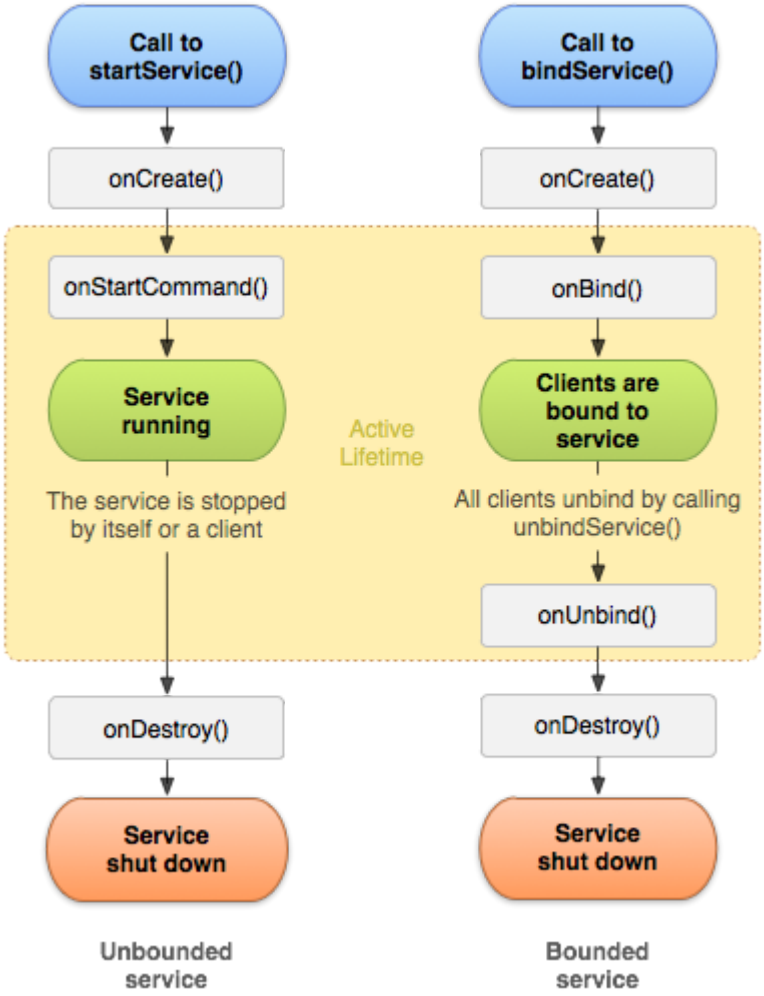
- onBind

  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。
  如果不希望允许绑定，则应返回 null。

- onDestroy

  当服务不再使用且将被销毁时，系统将调用此方法。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。

4 服务的生命周期



5 远端服务

本地服务: 服务端和客户端处于同一个进程。

远端服务: 服务端和客户端处于不同进程。

6 示例

6.1 使用清单文件声明服务

```
<activity android:name=".MainActivity"> <!--- 定义 activity, 作为客户端 -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service android:name=".LocalService"> <!--- 定义 service, 作为服务端 -->
    <intent-filter>
        <action android:name="trainee.intent.action.LocalService"/>
    </intent-filter>
</service>
```

6.2 创建启动服务

6.2.1 扩展 Service 类

> **WARNING:** 在这个演示示例中，LocalService 没有创建新的线程，是直接运行在主线程中的。通常，我们的实际使用，服务应该运行在非 UI 线程，可以选择自己在 Service 中创建新线程，并对并发进行管理；也可以选择继承 Service 的子类 IntentService。

```
public class LocalService extends Service {
    private static final String TAG = "LocalService";

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(TAG, "onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i(TAG, "onStartCommand " + intent + " " + flags + " " + startId
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy");
    }
}
```

6.2.2 启动服务和停止服务

```
btn_startlocalservice.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button Start Local Service clicked");
        Intent i = new Intent("trainee.intent.action.LocalService");
        i.setPackage(getPackageName());
        startService(i);
    }
});
btn_stop_local_service.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button Stop Local Service clicked");
        Intent i = new Intent("trainee.intent.action.LocalService");
        i.setPackage(getPackageName());
        stopService(i);
    }
});
```

6.3 创建绑定服务

> ⚠ **WARNING:** 实际使用中，通常应该按照远端服务的方式创建绑定服务。当然如果特殊情况，服务声明 `exported = false`，限定为进程内部使用时，可以采取下面的简化方式。

6.3.1 扩展 Service 类

```
public class LocalService extends Service {
    private static final String TAG = "LocalService";

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(TAG, "onCreate");
    }

    @Override
    public IBinder onBind(Intent intent) {
        Log.i(TAG, "onBind");
        return new LocalBinder(); /// 注意返回的 IBinder
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy");
    }

    public void whoami() {
        Log.i(TAG, "Hey, I'm LocalService");
    }

    public class LocalBinder extends Binder {
        LocalService getService(){
            return LocalService.this; /// 注意这个返回值
        }
    }
}
```

6.3.2 绑定服务并调用服务接口

```
btn_bind_local_srv.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button bind Local Service clicked");
        Intent i = new Intent("trainee.intent.action.LocalService");
        i.setPackage(getPackageName());
        bindService(i, mLocalConnection, Context.BIND_AUTO_CREATE); /// 注意
    }
});
btn_call_binded_local.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button call bound Local Service clicked");
        if (null != mLocalService) {
            mLocalService.whoami(); /// 注意 mLocalService 这个变量
        } else {
            Log.i(TAG, "mLocalService is null");
        }
    }
});
```

`mLocalConnection` 是一个 `ServiceConnection` 实例

```
private LocalService mLocalService = null;

private ServiceConnection mLocalConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder iBi
        /// 对回调返回的 IBinder 强转，并调用其方法获取服务实例
        mLocalService = ((LocalService.LocalBinder) iBinder).getService();
        Log.i(TAG, "mLocalConnection.onServiceConnected");
    }

    @Override
    public void onServiceDisconnected(ComponentName componentName) { /// 注
        // This is called when the connection with the service has been
        // unexpectedly disconnected -- that is, its process crashed.
        // Because it is running in our same process, we should never
        // see this happen.
        mLocalService = null; /// 注意，应该有其他的释放途径
        Log.i(TAG, "mLocalConnection.onServiceDisconnected " + componentNam
    }
};
```

6.3.3 解绑服务

```
btn_unbind_local_srv.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button unbind Local Service clicked");
        if (null != mLocalConnection) {
            try {
                unbindService(mLocalConnection);
            } catch (RuntimeException e) {
                e.printStackTrace();
            }
        } else {
            Log.i(TAG, "mLocalConnection is null");
        }
    }
});
```

6.4 创建远端绑定服务

> **NOTE:** 启动服务在本地服务和远端服务之间没有区别。

6.4.1 清单声明

```
<service android:name=".RemoteService"
    android:exported="true"
    android:process=":remote"> <!--- 指定了 process -->
    <intent-filter>
        <action android:name="trainee.intent.action.RemoteService"/>
    </intent-filter>
</service>
```

6.4.2 AIDL 定义服务接口

> **CAUTION:**
>
> 只有允许不同应用的客户端用 IPC 方式访问服务，并且想要在服务中处理多线程时，才有必要使用 `AIDL`。如果您不需要执行跨越不同应用的并发 IPC，就应该通过实现一个 `Binder` 创建接口；或者，如果您想执行 IPC，但根本不需要处理多线程，则使用 `Messenger` 类来实现接口。无论如何，在实现 AIDL 之前，请您务必理解绑定服务。

```
// IMyAidlInterface.aidl
package xyz.lego.trainee;

interface IMyAidlInterface {
    void whoami();
}
```

6.4.3 AIDL 接口实现

```
package xyz.lego.trainee;

import android.os.RemoteException;
import android.util.Log;

/// 父类 IMyAidlInterface.Stub
public class MyAidlInterfaceImpl extends xyz.lego.trainee.IMyAidlInterface.
    private static final String TAG = "MyAidlInterfaceImpl";
    @Override
    public void whoami() throws RemoteException {
        Log.i(TAG, "Hey, I'm Remote Service");
    }
}
```

6.4.4 返回 Binder

```
@Override
public IBinder onBind(Intent intent) {
    Log.i(TAG, "onBind");
    return new MyAidlInterfaceImpl();
}
```

6.4.5 绑定远端服务

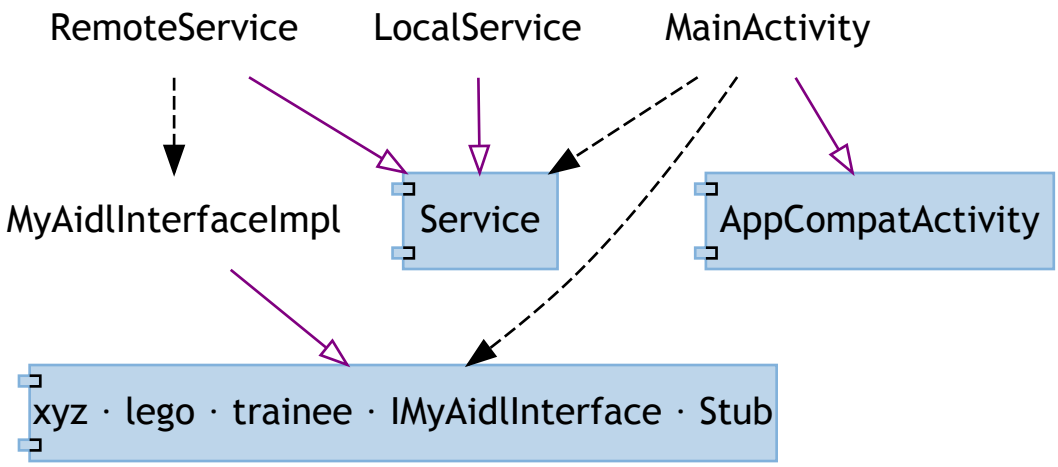
```
private xyz.lego.trainee.IMyAidlInterface mRemoteService = null;

private ServiceConnection mRemoteConn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder iBi
        mRemoteService = xyz.lego.trainee.IMyAidlInterface.Stub.asInterface
        Log.i(TAG, "mRemoteConn.onServiceConnected");
    }

    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        Log.i(TAG, "mRemoteConn.onServiceDisconnected " + componentName);
    }
};
```

```
btn_bind_remote_srv.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.i(TAG, "button bind Remote Service clicked");
        Intent i = new Intent("trainee.intent.action.RemoteService");
        i.setPackage(getPackageName());
        bindService(i, mRemoteConn, Context.BIND_AUTO_CREATE);
    }
});
btn_call_bound_remote.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (null != mRemoteService) {
            try {
                mRemoteService.whoami();
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        } else {
            Log.i(TAG, "mRemoteService is null");
        }
    }
});
```

6.5 类关系图



类关系图由 scanClazz 生成

7 Log 过滤器

```
adb logcat | grep -i -e runtime -e LocalService -e xyz.lego.trainee -e Main
```

8 参考文档和工具

- 服务
- 绑定服务
- Android 接口定义语言 (AIDL)
- scanClazz

9 Revision

2019-08-03

Revision	Author	Modification
190801	JC	create the document
190803	JC	v1.0

更新日期: Sat Aug 03 2019 14:14:10 GMT+0800 (China Standard Time)