

Android Service解析解析再解析

学习android有一段时间了，在没有好的师傅带领下，入门时总是碰到这样那样的新概念、新知识，而这些知识虽说能看得明白，却没有深得它的精髓。就好比有一双好看的鞋子，我只知道它很好看，穿起来挺舒服的，但是却不了解为什么穿起来舒服，它是由什么材料组合成的，怎么穿才能更舒服有效。不过开始学习android虽然开头困难重重，但是我坚信大家只要有耐心，有决心不间断地学习下去，总会有大丰收的时候。打个比方，你每天都对着一匹马观察它，时间久了让你画出来，即使没有到庖丁解牛的境界，但至少你能将这匹马的整个外形都牢记在心中，自己慢慢地描绘，到后面有人指导一下或者自己突然的醒悟，很快可以突破这一层表面的隔膜，后面再去深入了解也自然水到渠成。

如今再去看看android 文档，发现以前很多小的知识点都没注意到，但文档上却有写着。想来想去，归根到底这就是看二手鞋（因为是中文所以学习起来很快，我就是贪那个快^-^）的悲哀，二手鞋很多都是抛去小树叶，只留一条光溜溜的枝条，告诉大家有这样的概念怎么用，但是如果要想突破自身的瓶颈，那就要再去看一手鞋了（都是英文的，虽然会耗大家很多时间，但是却收获颇丰）。

这段时间总结了一下Service的概念，下面开始是小细节的深入（不过这也是二手鞋^-^，如果可以的话大家可以去看看英文文档，在Reference——android.app——service）。因为全文好像都是翻译过来的，再加上一点点自己的了解。所以难免有理解错误的时候，所以大家如果发现错误了，请猛写评论吧再猛击提交吧，尽情地向我扔砖头吧，我爱砖头^-^。

Service 作为android组件之一，但在界面上却很难看到它的身影，它负责着后台一些繁重的数据处理，比如音乐播放，单词的查询。当然也有跟activity交互的功能，比如我想跳过这首播放下一首的动作、我想查的这个单词。

什么是Service?

解惑：

- 1、 Service不是分离开的进程，除非其他特殊情况，它不会运行在自己的进程，而是作为启动运行它的进程的一部分。
- 2、 Service不是线程，这意味着它将在主线程里劳作。

启动service有两种方法：

1、 Context.startService()

调用者与服务之间没有关联，即使调用者退出，服务仍可运行

2、 Context.bindService()

调用者与服务绑定在一起，调用者一旦退出，服务也就终止

Service的生命周期

如果使用startService()启动service，系统将通过传入的Intent在底层搜索相关符合Intent里面信息的service。如果服务没有启动则先运行onCreate，然后运行onStartCommand（可在里面处理启动

时传过来的Intent和其他参数），直到明显调用stopService或者stopSelf才将停止Service。无论运行startService多少次，只要调用一次stopService或者stopSelf，Service都会停止。使用stopSelf(int)方法可以保证在处理好intent后再停止。

控制service运行的主要方式有两种，主要是根据onStartCommand方法返回的数值。方法：

1、START_STICKY

2、START_NOT_STICKY or START_REDELIVER_INTENT

这里主要解释这三个变量的意义：

1、START_STICKY

在运行onStartCommand后service进程被kill后，那将保留在开始状态，但是不保留那些传入的intent。不久后service就会再次尝试重新创建，因为保留在开始状态，在创建 service后将保证调用onstartCommand。如果没有传递任何开始命令给service，那将获取到null的intent

2、START_NOT_STICKY

在运行onStartCommand后service进程被kill后，并且没有新的intent传递给它。Service将移出开始状态，并且直到新的明显的方法（startService）调用才重新创建。因为如果没有传递任何未决定的intent那么service是不会启动，也就是期间onstartCommand不会接收到任何null的intent。

3、START_REDELIVER_INTENT

在运行onStartCommand后service进程被kill后，系统将会再次启动service，并传入最后一个intent给onstartCommand。直到调用stopSelf(int)才停止传递intent。如果在被kill后还有未处理好的intent，那被kill后服务还是会启动。因此onstartCommand不会接收到任何null的intent。

客户端也可以使用bindService来保持跟service持久关联。谨记：如果使用这种方法，那么将不会调用onstartCommand（跟startService不一样，下面例子注释也有解析，大家可试试）。客户端将会在onBind回调中接收到IBinder接口返回的对象。通常IBinder作为一个复杂的接口通常是返回aidl数据。

Service也可以混合start和bind一起使用。

权限

要运行service，首先必须在AndroidManifest.xml里申明<service>标签。

Service能够保护个人的IPC调用，所以在执行实现该调用时前先使用checkCallingPermission(String)方法检查是否有这个权限。

进程生命周期

当service运行在低内存的环境时，将会kill掉一下存在的进程。因此进程的优先级将会很重要：

1、 如果service当前正在执行onCreate、 onStartCommand、 onDestroy方法，主进程将会成为前台进程来保证代码可以执行完成避免被kill

2、 如果service已经启动了，那么主进程将会比其他可见的进程的重要性低，但比其他看不见的进程高。因为只有少部分进程始终对用户可见的，因此除非在极度低内存的时候，不然 service是不会被kill的。

3、 如果有客户端关联到service，那么service永远比客户端重要。也就是说客户端可见，那么service也可见（我理解这里的可见并不是可以看到，而是重要性，因为可见往往就表示重要性高）。

4、 Service可以使用startForeground API将service放到前台状态。这样在低内存时被kill的几率更低，但是文档后面又写了，如果在极度极度低内存的压力下，该service理论上还是会被kill掉。但这个情况基本不用考虑。

当然如果service怎么保持还是被kill了，那你可以通过重写onStartCommand返回变量来设置它的启动方式。比如：START_STICKY、START_REDELIVER_INTENT等等，前面已经讨论了它们的作用，这里就不再累赘了

另外：

service 的onCreate和onStartCommand 是运行在主线程的，所以如果里面有处理耗时间的任务。两种处理：

- 1、 请将它们都挪到新的线程里。
- 2、 用系统提供的IntentService，它继承了Service，它处理数据是用自身新开的线程。

好了说了这么多下面就是例子的时刻了。总共有两个例子，第一个是本地调用，第二个是远程调用。

口水快没了，所以下面就直接进入代码环节吧。代码里面已经有详细的注解了，如果真的真的还是不明白，那就是我这篇二手鞋的失败了。:(



```
public class LocalService extends Service {

    private NotificationManager mNM;

    // 通知唯一标示，在通知开始和结束使用
    private int NOTIFICATION = R.string.local_service_started;

    // 与界面交互的类，由于service跟界面总是运行在同一程序里，所以不用处理IPC
    public class LocalBinder extends Binder {
        LocalService getService() {
            return LocalService.this;
        }
    }
}
```

```
@Override
public void onCreate() {
    mNM = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    // 在service开始时, 将icon图标放到通知任务栏
    showNotification();
}

//
private void showNotification() {
    CharSequence text = getText(R.string.local_service_started);

    Notification notification = new Notification(R.drawable.icon, text,
        System.currentTimeMillis());

    // 当点击通知时, 启动该contentIntent关联的activity
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, showActivity.class), 0);

    // 在通知栏上显示标题和内容
    notification.setLatestEventInfo(this,
        getText(R.string.local_service_label), text, contentIntent);

    mNM.notify(NOTIFICATION, notification);
}

// 兼容2.0以前版本
@Override
public void onStart(Intent intent, int startId) {

    // 在2.0以后的版本如果重写了onStartCommand, 那onStart将不会被调用, 注: 在2.0以前是没有
    onStartCommand方法
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        Log.i("Service", "Received start id " + startId + ": " + intent);
        // 如果服务进程在它启动后(从onStartCommand()返回后)被kill掉, 那么让他呆在启动状态但不取传给
        它的intent.
        // 随后系统会重写创建service, 因为在启动时, 会在创建新的service时保证运行onStartCommand
        // 如果没有任何开始指令发送给service, 那将得到null的intent, 因此必须检查它.
        // 该方式可用在开始和在运行中任意时刻停止的情况, 例如一个service执行音乐后台的重放

        return START_STICKY;
    }
}
```

```
@Override
public void onDestroy() {
    mNM.cancel(NOTIFICATION);
    Toast
        .makeText(this, R.string.local_service_stopped,
            Toast.LENGTH_SHORT).show();
}

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

private final IBinder mBinder = new LocalBinder();
}
```



```
public class LocalActivity extends Activity {
    /** Called when the activity is first created. */
    private LocalService mBoundService;
    private boolean mIsBound;

    private ServiceConnection mConnection = new ServiceConnection() {

        @Override
        public void onServiceDisconnected(ComponentName name) {
            // 当进程崩溃时将被调用, 因为运行在同一程序, 如果是崩溃将所以永远不会发生
            // 当解除绑定时也被调用
            mBoundService = null;
            Toast.makeText(LocalActivity.this,
                R.string.local_service_disconnected, Toast.LENGTH_SHORT)
                .show();
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            // service连接建立时将调用该方法
            mBoundService = ((LocalService.LocalBinder) service).getService();
            Toast.makeText(LocalActivity.this,
                R.string.local_service_connected, Toast.LENGTH_SHORT)
                .show();
        }
    };
}
```

```
        .show();
    }
};

void doBindService() {
    // 建立service连接。因为我们知道程序会运行在本地里，因此使用显示的类名来实现service
    // （但是不支持跟其他程序交互）
    // 两种传递，一种是在manifest里写好intent-filter的action，一种是显示传递
    // bindService(new Intent("com.LocalService.LocalService"), mConnection,
    // Context.BIND_AUTO_CREATE);
    // bindService(new Intent(LocalActivity.this, LocalService.class),
    // mConnection, Context.BIND_AUTO_CREATE);

    //如果用这种方法将会调用onStartCommand方法
    startService(new Intent(LocalActivity.this, LocalService.class));
    mIsBound = true;
}

void doUnbindService() {
    if (mIsBound) {
        // Detach our existing connection.
        stopService(new Intent(LocalActivity.this, LocalService.class));
        // unbindService(mConnection);
        mIsBound = false;
    }
}

@Override
protected void onDestroy() {
    doUnbindService();

    super.onDestroy();
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    doBindService();
}
}
```



大家可以试试startService 和bindService这两种区别。

轻松一下, `Toast.makeText(this, "温馨提示: \n代码已经写好了, 如果想测试一下可以去掉注释的喔", 2000).show(); ^-^`

下面是远程调用的例子, 主要是用系统提供的Messenger, 省去自己去写复杂的aidl文件

```
<service android:name="MessengerService" android:process=":remote">
    <intent-filter>
        <action android:name="com.LocalService.MessengerService" />
    </intent-filter>
</service>
```

如果加了`android:process=":remote"`, 那在调试时在`service`断点是不会触发的。



```
public class MessengerService extends Service {

    private NotificationManager mNM;

    // 保存所有跟服务连接的客户端
    ArrayList<Messenger> mClients = new ArrayList<Messenger>();
    // 保存最后一次跟服务连接的客户端的标志
    int mValue = 0;
    // 注册指令, Message's replyTo 字段值必须是client 的Messenger
    static final int MSG_REGISTER_CLIENT = 1;
    // 取消指令, Message's replyTo 字段值必须是先前给MSG_REGISTER_CLIENT的Messenger
    static final int MSG_UNREGISTER_CLIENT = 2;
    // 服务发送指令, 可以在客户端和服务直接交流
    static final int MSG_SET_VALUE = 3;

    // 处理客户端传送过来的消息
    class IncomingHandler extends Handler {

        @Override
        public void handleMessage(Message msg) {

            switch (msg.what) {
                case MSG_REGISTER_CLIENT:
                    // Optional Messenger where replies to this message can be sent.
                    // The semantics of exactly how this is used are up to the
                    // sender and receiver.
                    mClients.add(msg.replyTo);
                    break;
                case MSG_UNREGISTER_CLIENT:
                    mClients.remove(msg.replyTo);
                    break;
```

```
case MSG_SET_VALUE:
    mValue = msg.arg1;
    for (int i = mClients.size() - 1; i >= 0; i--) {
        try {
            mClients.get(i).send(
                Message.obtain(null, MSG_SET_VALUE, mValue, 0));

        } catch (RemoteException e) {
            // 远程客户端出错, 从list中移除
            // 遍历列表以保证内部循环安全运行
            mClients.remove(i);
        }
    }
    break;
default:
    super.handleMessage(msg);
}

Log.i("Service", "有" + mClients.size() + "客户端");
}
}

// 创建一个新的Messenger跟已存在的Handler关联
// 如果有任何消息发送到Messenger, 将交给Handler处理
final Messenger mMessenger = new Messenger(new IncomingHandler());

@Override
public void onCreate() {
    mNM = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    // 在service开始时, 将icon图标放到通知任务栏
    showNotification();
}

//
private void showNotification() {
    CharSequence text = getText(R.string.local_service_started);

    Notification notification = new Notification(R.drawable.icon, text,
        System.currentTimeMillis());

    // 当点击通知时, 启动该contentIntent关联的activity
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, showActivity.class), 0);

    // 在通知栏上显示标题和内容
```



```

        notification.setLatestEventInfo(this,
            getText(R.string.remote_service_label), text, contentIntent);

        mNM.notify(R.string.remote_service_started, notification);
    }
    //
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("Service", "Received start id " + startId + ": " + intent);
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        mNM.cancel(R.string.remote_service_started);
        Toast.makeText(this, R.string.remote_service_stopped,
            Toast.LENGTH_SHORT).show();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
}

```



```

public class MessengerActivity extends Activity {
    /** Called when the activity is first created. */
    private Messenger mService = null;

    private boolean mIsBound;

    private TextView mCallbackText;

    class IncomingHandler extends Handler {

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MessengerService.MSG_SET_VALUE:
                    mCallbackText.setText("Received from service: " + msg.arg1);
                    break;
            }
        }
    }
}

```

```
        default:
            super.handleMessage(msg);
        }
    }
}

final Messenger mMessenger = new Messenger(new IncomingHandler());

private ServiceConnection mConnection = new ServiceConnection() {

    @Override
    public void onServiceDisconnected(ComponentName name) {
        // 当进程崩溃时将被调用，因为运行在同一程序，如果是崩溃将所以永远不会发生
        // 当解除绑定时也被调用
        mService = null;
        mCallbackText.setText("Disconnected.");

        Toast.makeText(MessengerActivity.this,
            R.string.remote_service_disconnected, Toast.LENGTH_SHORT)
            .show();
    }

    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        // service连接建立时将调用该方法
        // 返回IBinder接口以便我们可以跟service关联。
        // 我们可通过IDL接口来交流
        mService = new Messenger(service);
        mCallbackText.setText("Attached.");

        // 只有我们连接着都监听服务
        try {
            // 注册
            Message msg = Message.obtain(null,
                MessengerService.MSG_REGISTER_CLIENT);
            msg.replyTo = mMessenger;
            mService.send(msg);

            // 例子
            msg = Message.obtain(null, MessengerService.MSG_SET_VALUE,
                11111111, 0);
            mService.send(msg);
        } catch (RemoteException e) {
            // In this case the service has crashed before we could even
            // do anything with it; we can count on soon being

```

```
        // disconnected (and then reconnected if it can be restarted)
        // so there is no need to do anything here.
    }
    Toast.makeText(MessengerActivity.this,
        R.string.remote_service_connected, Toast.LENGTH_SHORT)
        .show();
}

};

void doBindService() {
    bindService(new Intent(MessengerActivity.this, MessengerService.class),
        mConnection, Context.BIND_AUTO_CREATE);
    mIsBound = true;
    mCallbackText.setText("Binding.");
}

void doUnbindService() {
    if (mIsBound) {
        if (mService != null) {
            try {
                // 取消注册
                Message msg = Message.obtain(null,
                    MessengerService.MSG_UNREGISTER_CLIENT);
                msg.replyTo = mMessenger;
                mService.send(msg);
            } catch (RemoteException e) {
                // There is nothing special we need to do if the service
                // has crashed.
            }
        }

        // Detach our existing connection.
        unbindService(mConnection);
        mIsBound = false;
        mCallbackText.setText("Unbinding.");
    }
}

@Override
protected void onDestroy() {
    doUnbindService();
    super.onDestroy();
}
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.messenger);
    mCallbackText = (TextView) findViewById(R.id.text);
    doBindService();
}
}
```



测试远程调用，我弄多一份项目来测试，主要是查看是否连接成功和有多少个客户端连接上。

效果图：

Time		pid	tag	Message
05-21 13:09:09.562	I	9101	Service	有1客户端
05-21 13:09:09.567	I	9101	Service	有1客户端
05-21 13:09:15.187	I	9101	Service	有2客户端
05-21 13:09:15.192	I	9101	Service	有2客户端
05-21 13:09:18.077	I	9101	Service	有1客户端

本文为原创，如需转载，请注明作者和出处，谢谢！

出处：<http://www.cnblogs.com/not-code/archive/2011/05/21/2052713.html>

以下是项目代码：

http://files.cnblogs.com/not-code/Sea_LocalService.zip

第一份是包含service源码

http://files.cnblogs.com/not-code/Sea_testLocalService.zip

第二份是为了测试远程调用是否真的有用

分类：[android基础知识不断巩固](#)