宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Alpha 0.01 版

Android-Service

1.Service 简介（★★★）

很多情况下，一些与用户很少需要产生交互的应用程序，我们一般让它们在后台运行就行了，而且在它们运行期间我们仍然能运行其他的应用。为了处理这种后台进程，Android 引入了 Service 的概念。

- Service 在 Android 中是一种长生命周期的组件，它不实现任何用户界面,是一个没有界面的 Activity
- Service 长期在后台运行，执行不关乎界面的一些操作比如：网易新闻服务,每隔 1 分钟去服务查看是否有最新新闻
- Service 和 Thread 有点相似,但是使用 Thread 不安全，不严谨
- Service 和其他组件一样，都是运行在主线程中，因此不能用它来做耗时的操作

2.Android 中的进程（★★）

2.1 Android 中进程的种类

进程优先级由高到低，依次为：

1. Foreground process 前台进程
2. Visible process 可视进程，可以看见，但不可以交互.
3. Service process 服务进程
4. Background process 后台进程

- 5. Empty process 空进程(当程序退出时，进程没有被销毁，而是变成了空进程)

2.2 进程的回收机制

Android 系统有一套内存回收机制,会根据优先级进行回收。Android 系统会尽可能的维持程序的进程，但是终究还是需要回收一些旧的进程节省内存提供给新的或者重要的进程使用。

- 进程的回收顺序是：从低到高
- 当系统内存不够用时，会把空进程一个一个回收掉
- 当系统回收所有的完空进程不够用时，继续向上回收后台进程，依次类推
- 但是当回收服务，可视，前台这三种进程时，系统非必要情况下不会轻易回收，如果需要回收掉这三种进程，那么在系统内存够用时，会再给重新启动进程;但是服务进程如果用户手动的关闭服务，这时服务不会再重启了。

2.3 为什么用服务而不是线程

进程中运行着线程， Android 应用程序刚启动都会开启一个进程给这个程序来使用。Android 一个应用程序把所有的界面关闭时，进程这时还没有被销毁，现在处于的是空进程状态,Thread 运行在空进程中，很容易的被销毁了。

服务不容易被销毁，如果非法状态下被销毁了，系统会在内存够用时，重新启动。

3.案例-电话窃听器（★★★）

需求：

开启一个服务监听用户电话，当电话被接通时开始录音，电话挂断时停止录音。

1 新建一个 Android 工程《电话窃听器》，包名：com.itheima.listenCall。

2 在 src 目录下新创建一个 MyService 类继承 Service 类，在该类中实现核心业务方法，实现监听电话，以及完成录音的功能。

```
public class MyService extends Service {
    /**
     * 绑定服务时调用
     */
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    /**
     * 服务被创建时调用
     */
    @Override
    public void onCreate() {
        super.onCreate();
        System.out.println("服务已经被创建。");
        TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(TELEPHONY_SERVICE);
        telephonyManager.listen(new MyPhoneListener(),
PhoneStateListener.LISTEN_CALL_STATE);

    }
    class MyPhoneListener extends PhoneStateListener{
        MediaRecorder recorder;
        boolean isCalling = false;
        @Override
        public void onCallStateChanged(int state, String
incomingNumber) {
            super.onCallStateChanged(state, incomingNumber);
            if (TelephonyManager.CALL_STATE_OFFHOOK==state) {
                System.out.println("开始通话。。。");
                isCalling = true;
                //新建一个 MediaRecorder 对象
                recorder = new MediaRecorder();
```

```
            //设置声音来源
            recorder.setAudioSource(AudioSource.MIC);
            //设置输入格式
            recorder.setOutputFormat(OutputFormat.THREE_GPP);
            //格式化日期，作为文件名称
            SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd_hh_mm_ss");
            String date = format.format(new Date());
            //设置输出到的文件
            recorder.setOutputFile(getFilesDir()+"/"+date+".3gp");
            //设置音频编码
            recorder.setAudioEncoder(AudioEncoder.DEFAULT);
            try {
                //录音准备
                recorder.prepare();
                //录音开始
                recorder.start();
            } catch (Exception e) {
                e.printStackTrace();
            }

        }else if
(TelephonyManager.CALL_STATE_IDLE==state&&isCalling) {
            recorder.stop();
            recorder.release();
            isCalling = false;
            System.out.println("录音结束。");
        }
    }
}
```

3 在 MainActivity 类中启动 Service

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent service = new Intent();
        service.setClass(this, MyService.class);
        startService(service);
    }
```

4 在 AndroidManifest.xml 文件中注册该 Service。

Tips：在 Android 中四大组件都需要在清单文件中进行注册。

```
<service android:name="com.itheima.listenCall.MyService"/>
```

5 在 AndroidManifest.xml 中添加权限。

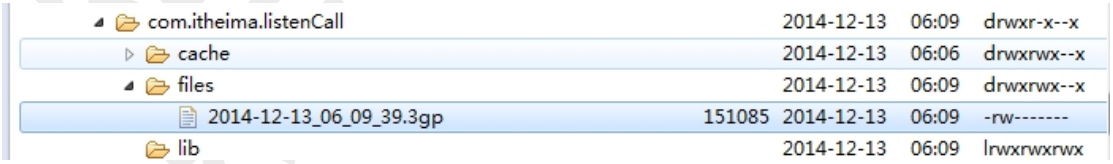
Tips：在该案例中，我们把录音文件存储在 data/data/com.itheima.listenCall/files 目录中，因此不需要声明外部存储的写权限。

```
<uses-permission
android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission
android:name="android.permission.RECORD_AUDIO"/>
```

6 将本工程部署到模拟器中，然后通过 DDMS 给该模拟器拨打电话，当我们接听一段时间并关闭后，发现控制台成功打印出了录音信息。

```
1922    com.itheima.listenCall          System.out                  开始通话。。。
1922    com.itheima.listenCall          System.out                  录音结束。
```

打开 data/data/com.itheima.listenCall/files 目录发现，录音文件被成功保存了。我们将该文件导出到电脑上，发现声音可以正常播放。



4.Service 的生命周期（★★★★）

service 的生命周期，从它被创建开始，到它被销毁为止，可以有两条不同的路径。

A started service（标准开启模式）

被开启的 service 通过其他组件调用 startService()被创建。这种 service 可以无限地运行下去，必须调用 stopSelf()方法或者其他组件调用 stopService()方法来停止它。当 service 被停止时，系统会销毁它。

A bound service（绑定模式）

被绑定的 service 是当其他组件（一个客户）调用 bindService()来创建的。客户可以通过一个 IBinder 接口和 service 进行通信。客户可以通过 unbindService()方法来关闭这种连接。一个 service 可以同时和多个客户绑定，当多个客户都解除绑定之后，系统会销毁 service。

Tips：Service 的这两中生命周期并不是完全分开的。

也就是说，你可以和一个已经调用了 startService()而被开启的 service 进行绑定。比如，一个后台音乐 service 可能因调用 startService()方法而被开启了，稍后，可能用户想要控制播放器或者得到一些当前歌曲的信息，可以通过 bindService()将一个 activity 和 service 绑定。这种情况下，stopService()或 stopSelf()实际上并不能停止这个 service，除非所有的客户都解除绑定。

Service 的生命周期回调函数

和 activity 一样，service 也有一系列的生命周期回调函数，你可以实现它们来监测 service 状态的变化，并且在适当的时候执行适当的工作。

下面的 service 展示了每一个生命周期的方法：

```
public class TestService extends Service {
    int mStartMode; // indicates how to behave if the service is killed
    IBinder mBinder; // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }

    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }

    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }

    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }

    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

Tips：不像是 activity 的生命周期回调函数，我们不需要调用基类的实现。

Service 的生命周期图

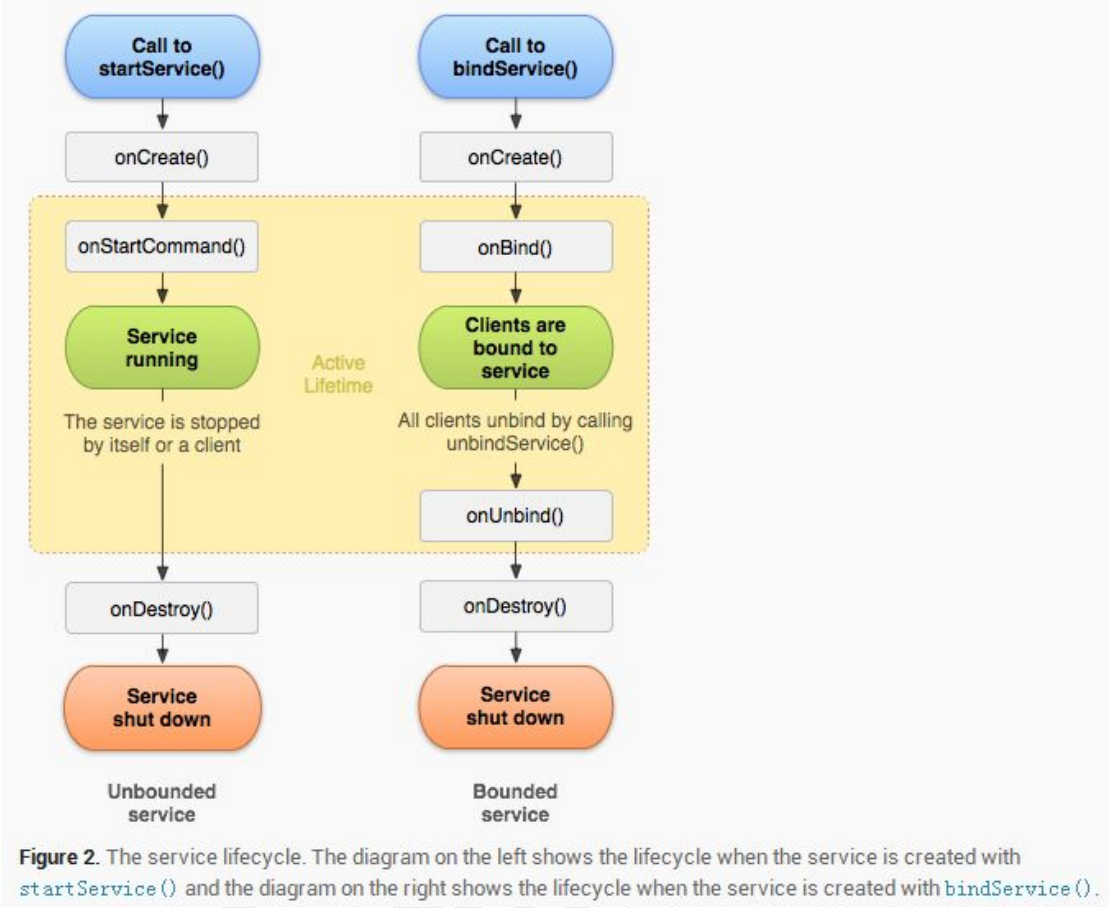


Figure 2. The service lifecycle. The diagram on the left shows the lifecycle when the service is created with `startService()` and the diagram on the right shows the lifecycle when the service is created with `bindService()`.

这个图说明了 service 典型的回调方法，尽管这个图中将开启的 service 和绑定的 service 分开，但是你需要记住，任何 service 都潜在地允许绑定。所以，一个被开启的 service 仍然可能被绑定。实现这些方法，你可以看到两层嵌套的 service 的生命周期（**拓展知识**）：

整体生命周期（The entire lifetime）

service 整体的生命时间是从 onCreate()被调用开始，到 onDestroy()方法返回为止。

和 activity 一样，service 在 onCreate()中进行它的初始化工作，在 onDestroy()中释放残留的资源。

比如，一个音乐播放 service 可以在 onCreate()中创建播放音乐的线程，在 onDestory()中停止这个线程。

onCreate() 和 onDestroy()会被所有的 service 调用，不论 service 是通过 startService()还是 bindService()建立。

积极活动的生命时间(The active lifetime)

service 积极活动的生命时间（active lifetime）是从 onStartCommand() **或** onBind()被调用开始，它们各自处理由 startService()或 bindService()方法传过来的 Intent 对象。

如果 service 是被开启的，那么它的活动生命周期和整个生命周期一同结束。

如果 service 是被绑定的，它们它的活动生命周期是在 onUnbind()方法返回后结束。

Tips：尽管一个被开启的 service 是通过调用 stopSelf() 或 stopService()来停止的，没有一个对应的回调函数与之对应，即没有 onStop()回调方法。所以，当调用了停止的方法，除非这个 service 和客户组件绑定，否则系统将会直接销毁它，onDestory()方法会被调用，并且是这个时候唯一会被调用的回调方法。

管理生命周期

当绑定 service 和所有客户端解除绑定之后，Android 系统将会销毁它，（除非它同时被 onStartCommand()方法开启）。

因此，如果你的 service 是一个纯粹的绑定 service，那么你不需要管理它的生命周期。然而，如果你选择实现 onStartCommand()回调方法，那么你必须显式地停止 service，因为 service 此时被看做是开启的。这种情况下，service 会一直运行到它自己调用 stopSelf()或另一个组件调用 stopService()，不论它是否和客户端绑定。

另外，如果你的 service 被开启并且接受绑定，那么当系统调用你的 onUnbind()方法时，如果你想要在下次客户端绑定的时候接受一个 onRebind()的调用（而不是调用 onBind()），你可以选择在 onUnbind()中返回 true。

onRebind()的返回值为 void，但是客户端仍然在它的 onServiceConnected()回调方法中得到 IBinder 对象。

下图展示了这种 service（被开启，还允许绑定）的生命周期：

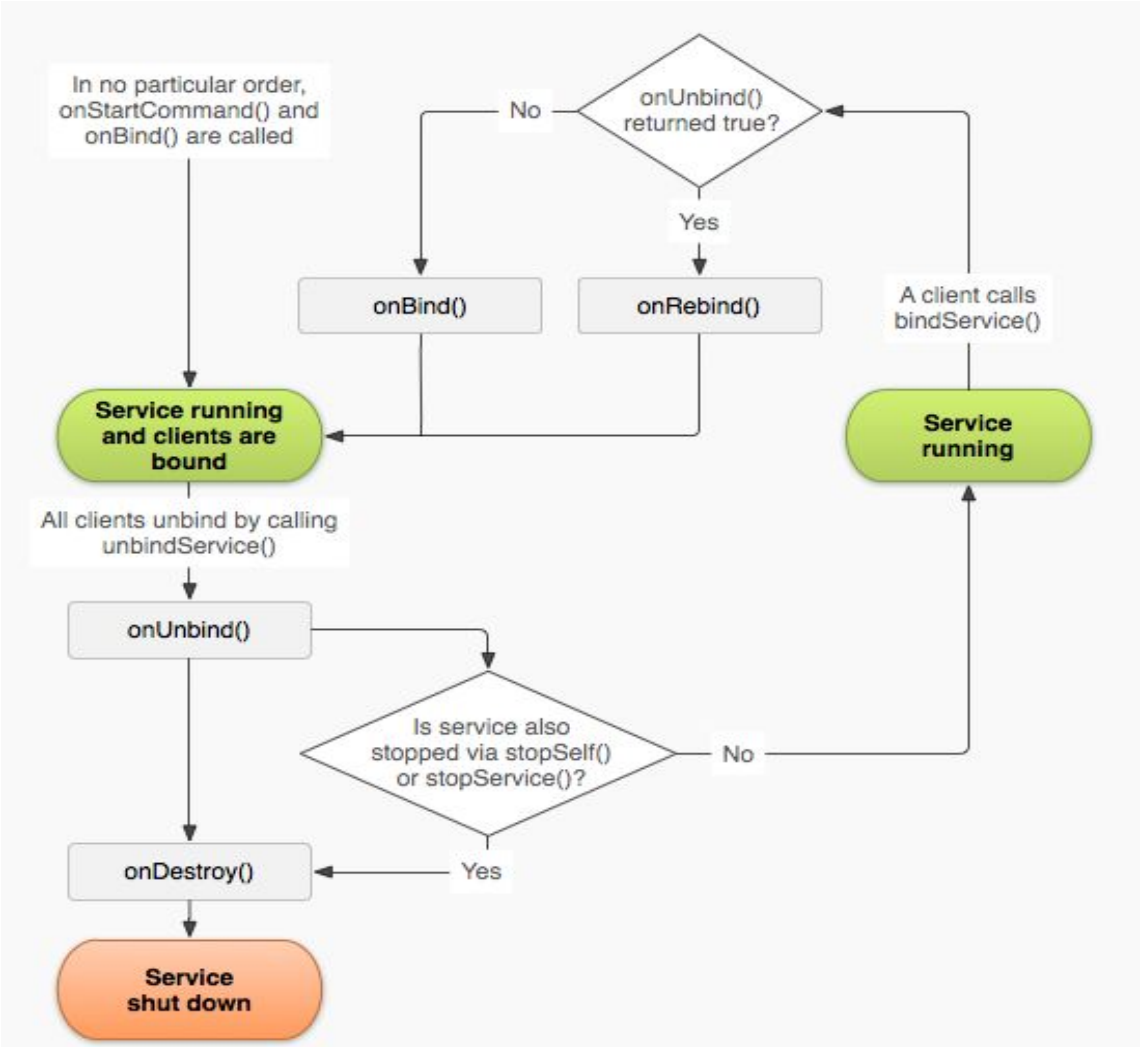


Figure 1. The lifecycle for a service that is started and also allows binding.

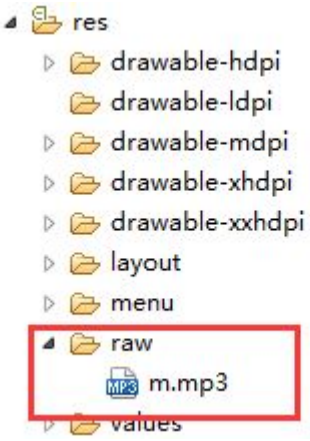
5.Android 中服务的调用（★★★）

5.1 案例-本地服务调用音乐播放器

1 新创建一个 Android 工程《音乐播放器》，包名：com.itheima.musicPlayer。

在 res 目录下新建一个文件夹 raw（名字必须为 raw，约定大于配置的原则），然后在 raw 目录中拷贝进一个音乐文件，注意文件名必须遵循 Android 资源文件的命名规则。

目录结构如下图：



2 在 src 目录下，新建一个 MediaService 继承 Service 类，在该类中实现核心服务的方法。

```
public class MediaService extends Service {
    //声明一个 MediaPlayer 对象
    private MediaPlayer player;

    @Override
    public IBinder onBind(Intent intent) {
        System.out.println("服务返回 MediaController 对象了......");
        return new MediaController();
    }

    @Override
    public void onCreate() {
        System.out.println("音乐服务已经被创建......");
        //初始化音乐播放器
        player = MediaPlayer.create(this, R.raw.m);
    }
    //自定义一个 Binder 对象，Binder 是 IBinder 接口的子类
    class MediaController extends Binder{
        public void play(){
            player.start();
        }
        public void pause(){
            player.pause();
        }
```

```
        public void stop(){
            player.stop();
        }
        //获取音乐的总时长
        public int getDuration(){
            return player.getDuration();
        }
        //获取当前播放位置
        public int getCurrentPostion(){
            return player.getCurrentPosition();
        }
        //判断是否在播放
        public boolean isPlaying(){
            return player.isPlaying();
        }

    }
}
```

3 这是使用系统默认的布局文件，activity_main.xml 清单如下：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="音乐播放器"
        android:textColor="#ff0000"
        android:textSize="28sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:orientation="horizontal" >

        <Button
            android:id="@+id/bt_play"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="play"
            android:text="播放" />

        <Button
            android:id="@+id/bt_pause"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="pause"
            android:text="暂停" />

        <Button
```

4 使用默认的 MainActivity 类，在该类中完成业务的控制，代码清单如下：

```
public class MainActivity extends Activity {
    //声明进度条
    private ProgressBar pb;
    //声明自定义的 MediaController 对象
    private MediaController mediaController;
    private boolean isRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //实例化进度条
        pb = (ProgressBar) findViewById(R.id.pb);
        //创建一个用于启动服务的显示意图，指向我们自定义的 MediaService 类
        Intent intent = new Intent(this, MediaService.class);
        //绑定服务，同时服务开启，如果成功则返回 true 否则返回 false
        isRunning = bindService(intent, new MediaConnection(),
BIND_AUTO_CREATE);
        if (isRunning) {
```

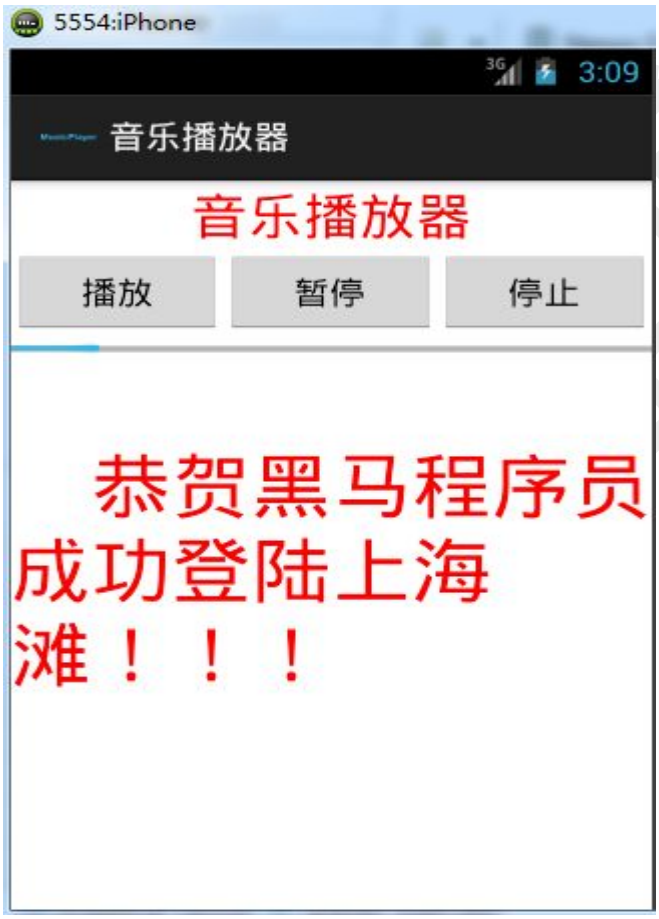
```
            System.out.println("音乐播放器服务绑定成功！");
        } else {
            System.out.println("音乐播放器服务绑定失败！");
        }
    }
    //用于循环更新当前播放进度
    private void updateProgressBar() {
        new Thread(new Runnable() {

            @Override
            public void run() {
                while (true) {
                    SystemClock.sleep(400);

    pb.setProgress(mediaController.getCurrentPostion());
                    if (mediaController.getDuration() ==
mediaController.getCurrentPostion()) {
                        break;
                    }
                }
            }
        }).start();
    }

    public void play(View view) {
        if (mediaController!=null) {
            //如果音乐正在播放则不能再次播放
            if(mediaController.isPlaying()){
                Toast.makeText(this, "音乐播放中", 0).show();
                return;
            }else {
                mediaController.play();
                Toast.makeText(this, "音乐开是播放", 0).show();
            }
        }
    }
    //暂停
    public void pause(View view) {
        if (mediaController != null) {
            mediaController.pause();
        }
```

```
    }
    //停止
    public void stop(View view) {
        if (mediaController!=null) {
            //停止的时候将进度条设置为初始位置
            pb.setProgress(0);
            mediaController.stop();
            Toast.makeText(this, "音乐已经关闭！", 0).show();
        }
    }
    //新建一个 ServiceConnection 类
    class MediaConnection implements ServiceConnection {
        /**
         * 当 service 被绑定的时候回调该函数
         */
        @Override
        public void onServiceConnected(ComponentName name, IBinder
service) {
            //返回的 IBinder 对象其实就是我们自定义的 MediaController 类对
象
            mediaController = (MediaController) service;
            //给进度条设置最大值
            pb.setMax(mediaController.getDuration());
            //更新进度条
            updateProgressBar();
            System.out.println("服务已经连接......");
        }
        /**
         * 服务被关闭或者断开的时候调用该方法
         */
        @Override
        public void onServiceDisconnected(ComponentName name) {
            System.out.println("服务已经断开......");
        }

    }

}
```

5 在 AndroidManifest.xml 中注册 Service。

```
<service android:name="com.itheima.musicPlayer.MediaService"/>
```

6 将工程部署到模拟器上，点击播放，发现成功播放了音乐。点击暂停，发现音乐暂停了，然后点击播放，音乐再次响起。点击停止，问题来了，我们发现点击停止后再次点击播放音乐没能再次播放，因为这里面直接调用 MediaPlayer 的 stop 方法是有 bug 的。因此为了解决这样的问题，我们应该将停止调用层 pause 方法，同时只需调用 MediaPlayer 的 seekTo（int）方法将音乐设置到开始位置。



控制台输出信息如下：

Tag	Text
System.out	音乐播放器服务绑定成功!
System.out	音乐服务已经被创建.....
System.out	服务返回MediaController对象了.....
System.out	服务已经连接.....

5.2 案例-远程服务调用商城支付

在 Android 平台中，各个组件运行在自己的进程中，他们之间是不能相互访问的，但是在程序之间是不可避免的要传递一些对象，在进程之间相互通信。为了实现进程之间的相互通信，Android 采用了一种轻量级的实现方式 RPC(Remote Procedure Call 远程进程调用)来完成进程之间的通信，并且 Android 通过接口定义语言（Android Interface Definition Language，AIDL）来生成两个进程之间相互访问的代码，例如，你在 Activity 里的代码需要访问 Service 中的一个方法，那么就可以通过这种方式来实现了。

AIDL 是 Android 的一种接口描述语言; 编译器可以通过 aidl 文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程的目的. 如果需要在一个 Activity 中, 访问另一个 Service 中的某个对象, 需要先将对象转化成 AIDL 可识别的参数(可能是多个参数), 然后使用 AIDL 来传递这些参数, 在消息的接收端, 使用这些参数组装成自己需要的对象。

AIDL RPC 机制是通过接口来实现的，类似 Windows 中的 COM 或者 Corba，但他是轻量级的，客户端和被调用实现之间是通过代理模式实现的，代理类和被代理类实现同一个接口 IBinder 接口。

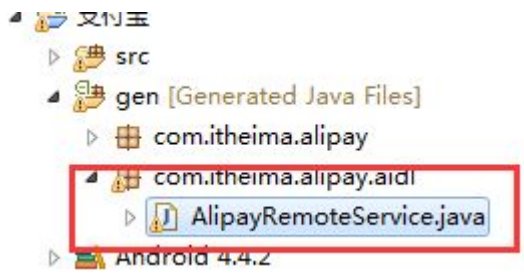
下面是案例-商城支付的步骤：

需求：分别创建两个工程，模拟一个支付平台，暂且叫支付宝，模拟一个商户端，叫商户。商户可以调用支付宝发布的远程服务进行收款操作。

1 新创建一个 Android 工程《支付宝》，包名：com.itheima.alipay。在 src 目录下创建 com.itheima.alipay.aidl 包，然后在该包下创建 AlipayRemoteService.aidl 文件。在该文件中只声明一个接口，在接口里声明一个方法。文件清单如下：

```
package com.itheima.alipay.aidl;

interface AlipayRemoteService{
    boolean forwardPayMoney(float money);
}
```

Tips：当该 aidl 文件创建好以后 ADT 会自动在 gen 目录下创建对应的类。



2 在《支付宝》src 目录下创建 com.itheima.alipay.service 包，在该包中新建一个 Service，叫 AlipayService，该类实现付款功能。代码清单如下：

```
public class AlipayService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return new PayController();
    }
    public boolean pay(float money){
        System.out.println("成功付款"+money);
        return true;
    }
    /**
     * 因为 Stub 已经继承了 IBinder 接口，因此 PayController 类也间接继承了该接口
     * @author wzy Dec 13, 2014
     *
     */
    public class PayController extends Stub{

        @Override
        public boolean forwardPayMoney(float money) throws RemoteException {
```

```
            return pay(money);
        }

    }
}
```

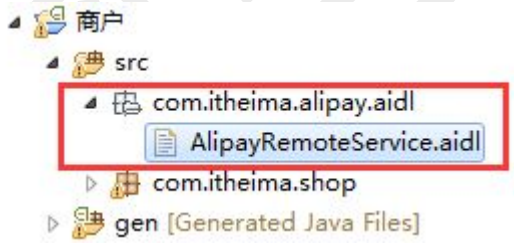
3 在《支付宝》工程的 AndroidManifest.xml 中注册该 AlipayService。

```
  <service
android:name="com.itheima.alipay.service.AlipayService">
            <intent-filter >
                <action android:name="com.itheima.alipay"></action>
            </intent-filter>
 </service>
```

4 创建一个新 Android 工程，名字叫《商户》，包名：com.itheima.shop。

使用默认的布局文件和默认的 MainActivity 类。

将《支付宝》工程中的 AlipayRemoteService.aidl 文件拷贝到《商户》工程的 src 目录下，同时注意添加对应的包名，**要求包名必须跟该文件在原工程中的包名严格一致**。《商户》src 目录结构如下图：



5 编辑 activity_main.xm 布局文件

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
```

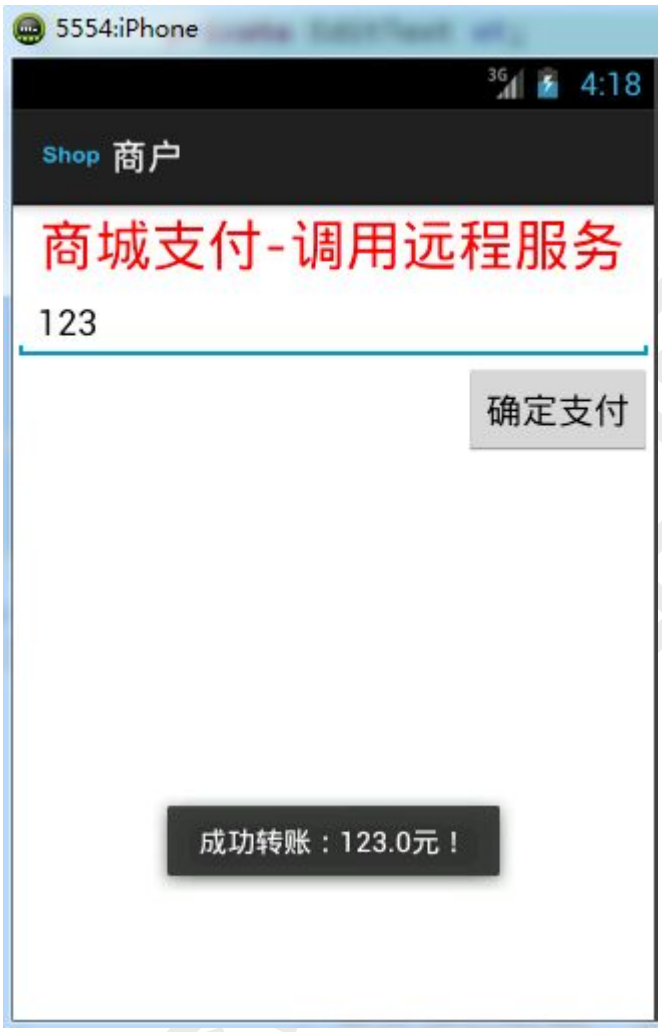
```
    <TextView
        android:layout_gravity="center_horizontal"
        android:textColor="#ff0000"
        android:textSize="28sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="商城支付-调用远程服务" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="请输入要转正的金额"
        android:inputType="number"
        android:id="@+id/et"
        />
    <Button
        android:layout_gravity="right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="确定支付"
        android:onClick="pay"
        />

</LinearLayout>
```

6 编写 MainActivity 类，在该类中实现核心方法

```
public class MainActivity extends Activity {
    //声明一个 AlipayRemoteService 对象，该类是根据 aidl 文件自动生成
    private AlipayRemoteService alipayRemoteService;
    private EditText et;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText) findViewById(R.id.et);
        //创建一个隐式意图，用于启动《支付宝》中的 Service
        Intent intent = new Intent();
        intent.setAction("com.itheima.alipay");
        //绑定远程服务
```

```
        boolean bindService = bindService(intent, new MyConnection(),
Context.BIND_AUTO_CREATE);
        if (bindService) {
            Toast.makeText(this, "服务绑定成功", 1).show();
            System.out.println("服务绑定成功");
        }else {
            Toast.makeText(this, "服务绑定失败", 1).show();
            System.out.println("服务绑定失败");
        }
    }
    public void pay(View view){
        float money = Float.valueOf(et.getText().toString());
        try {
            alipayRemoteService.forwardPayMoney(money);
        } catch (RemoteException e) {
            e.printStackTrace();
            Toast.makeText(this, "付款失败", 1).show();
        }
        Toast.makeText(this, "成功转账："+money+"元！", 0).show();
    }
    class MyConnection implements ServiceConnection{
        /**
         * 通过 Stub 的静态方法 asInterface 将 IBinder 对象转化为本地
AlipayRemoteService 对象
         */
        @Override
        public void onServiceConnected(ComponentName name, IBinder
service) {
            System.out.println("服务已经连接。。。");
            alipayRemoteService = Stub.asInterface(service);
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            System.out.println("服务已经关闭。");
        }

    }
}
```

7 先将《支付宝》部署到模拟器，然后将《商户》部署到模拟器，然后在《商户》界面输入一个金额，然后点击确定支付，发现《商户》工程已经成功通过远程服务调用了《支付宝》中的服务。运行图如下：



至此，本文档完！