1进程的概念&进程优先级

① 大部分android应用 都跑在一个linux进程中(也可以跑在多个进程) 所有的组件都运行在一个线程里(主线程) 4大组件(activity service broadcastreceiver contentprovider) 都运行在主线程 四大组件做耗时操作都要开子线程
② android 试图保持所有的应用进程都存活在手机中 只有当手机内存不够用的时候才会杀死进程 android系统 通过进程中组件运行的情况 决定那个进程先被杀死 一共有5档优先级
1. Foreground process(前台进程)
1.1 有一个activity正在运行跟用户交互(activity的onResume方法被调用)
1.2 广播接收者正在执行onreceive 方法
1.3 service正在执行生命周期方法
2.Visible process(可视进程)
2.1 有一个activity 处于onPause状态(可见但不能被操作)
3.Service process(服务进程)
3.1 后台运行着一个用startservice开启的服务 一般这个服务虽说不能被用户看到 但是可能运行用户关心的操作(比如播放音乐)
系统会尽可能保留服务进程不被杀死
4.Background process(后台进程)
4.1后台进程是 只有activity处于onStop状态没有其他组件在运行, 后台进程可以被系统随时杀死, 后台会存在多个处于后台进程状态的应用,哪个先挂掉是按照LRU(最近使用的最后杀死,最少使用的最先杀死)的顺序来决定优先级的
5.Empty process(空进程)
5.1 空进程 没有任何组件活着的进程, 保持这个进程存活的目的是为了下次开启组件的时候速度更快一些,系统会随时杀死这些进程为了回收资源

2 startservice方式开启服务

①写一个类继承Service

```
1.  public class MyService extends Service {
```

②重写方法

```
1.  public class MyService extends Service {
2.
3.      @Override
4.      public IBinder onBind(Intent intent) {
5.          // TODO Auto-generated method stub
6.          return null;
7.      }
8.
```

```
    @Override
    public void onCreate() {
        super.onCreate();
        //当service第一次创建的时候系统会调用这个方法
        System.out.println("service onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //调用startservice方法的时候系统就会调用这个onStartCommand
        System.out.println("service onStartCommand");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        System.out.println("service onDestroy");
        //当service销毁的时候 会调用这个onDestroy
    }
}
```

③清单文件注册

```
<service android:name="cn.itcast.servicedemo.MyService"></service>
```

startservice开启服务的特点

①调用startservice之后 service走的生命周期方法 onCreate->onstartCommend

②多次调用startservice onCreate只会走一次 onStartCommend会调用多次(调用几次startservice onStartCommend方法就会调用几次)

③ 通过startservice开启service的activity退出之后 service不会销毁

④ 只有手动调用stopservice方法 才会销毁服务(或者在程序管理器页面停止服务)

⑤ startservice方式开启的服务可以提升应用的进程优先级

3电话录音机

新的API:

TelephonyManager 电话管理器

listen

PhoneStateListener

获取电话的状态需要权限

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

MediaRecorder

录音需要的权限

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

开机广播的权限

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

Service代码

```
public class RecordService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // 电话管理器
        TelephonyManager mananger = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
        MyPhoneStateListener listener = new MyPhoneStateListener();
        //通过电话管理器监听电话状态
        mananger.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
    }

    private class MyPhoneStateListener extends PhoneStateListener {
        private MediaRecorder recorder;

        @Override
        public void onCallStateChanged(int state, String incomingNumber) {
            switch (state) {
            case TelephonyManager.CALL_STATE_IDLE:
                //idle空闲 电话处于空闲状态
                System.out.println("空闲状态 如果有录音机 要停止录音");
                //录音机停止
                if(recorder !=null){
                     recorder.stop();
                     //重置录音机
                     recorder.reset();   // You can reuse the object by going back to setAudioSource() step
                     //释放录音机对象
                     recorder.release(); // Now the object cannot be reused
                }

                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                //hook钩子 offhook摘机 接通状态
                System.out.println("通话状态 如果有录音机 要开始录音");
```

```
                if(recorder!=null){
                    recorder.start();   // Recording is now started
                }
                break;
            case TelephonyManager.CALL_STATE_RINGING:
                //ring 铃声 ringing 响铃状态
                System.out.println("响铃状态 要准备一个录音机");
                recorder = new MediaRecorder();
                //设置音频输入源   MIC麦克风 只能记录自己的声音  VOICE_CALL语音通话 可以记录通话双方的声音
                 recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                 //设置输出文件格式 设置为3gp 早期手机媒体文件的格式
                 recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
                 //设置音频编码方式AMR_NB amr 手机上用来做手机铃声的音频格式 nb narrow band 窄带
                 recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
                 //设置保存的文件的路径
                 recorder.setOutputFile(getFilesDir().getAbsolutePath()+"/"+incomingNumber+".3gp");
                 //开始准备录音
                 try {
                    recorder.prepare();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                break;
            }
        }
    }
}
```

广播接收者代码

```
public class BootReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Intent service = new Intent(context,RecordService.class);
        //开启电话录音的服务
        context.startService(service);
    }
}
```

清单文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.itcast.recordcall"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
```

```
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="cn.itcast.recordcall.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="cn.itcast.recordcall.RecordService"></service>
        <receiver android:name="cn.itcast.recordcall.BootReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
                </intent-filter>
        </receiver>
    </application>

</manifest>
```

4使用服务注册特殊广播接收者

可以把必须动态注册的广播接收者放到service中注册 service可以长期在后台存活, 保证可以随时收到广播

Service代码

```
public class ScreenService extends Service {

    private ScreenReceivcer receiver;

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

    @Override
    public void onCreate() {
        super.onCreate();
        receiver = new ScreenReceivcer();
        //创建一个意图过滤器
        IntentFilter filter = new IntentFilter();
        //给广播添加需要监听的action
        filter.addAction("android.intent.action.SCREEN_OFF");
```

```
        filter.addAction("android.intent.action.SCREEN_ON");
        //动态注册一个广播接收者
        registerReceiver(receiver, filter);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        //注销receiver
        unregisterReceiver(receiver);
    }
}
```

广播接收者代码

```
public class ScreenReceivcer extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if("android.intent.action.SCREEN_OFF".equals(action)){
            System.out.println("屏幕熄灭 执行上传任务");
        }else{
            System.out.println("屏幕点亮,暂停上传任务");
        }
    }
}
```

清单文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.itcast.serviceregisterreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="cn.itcast.serviceregisterreceiver.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

```
        </activity>
        <service android:name="cn.itcast.serviceregisterreceiver.ScreenService"></service>

    </application>

</manifest>
```

5bindservice开启服务特点

① 通过bindservice开启服务要传递三个参数

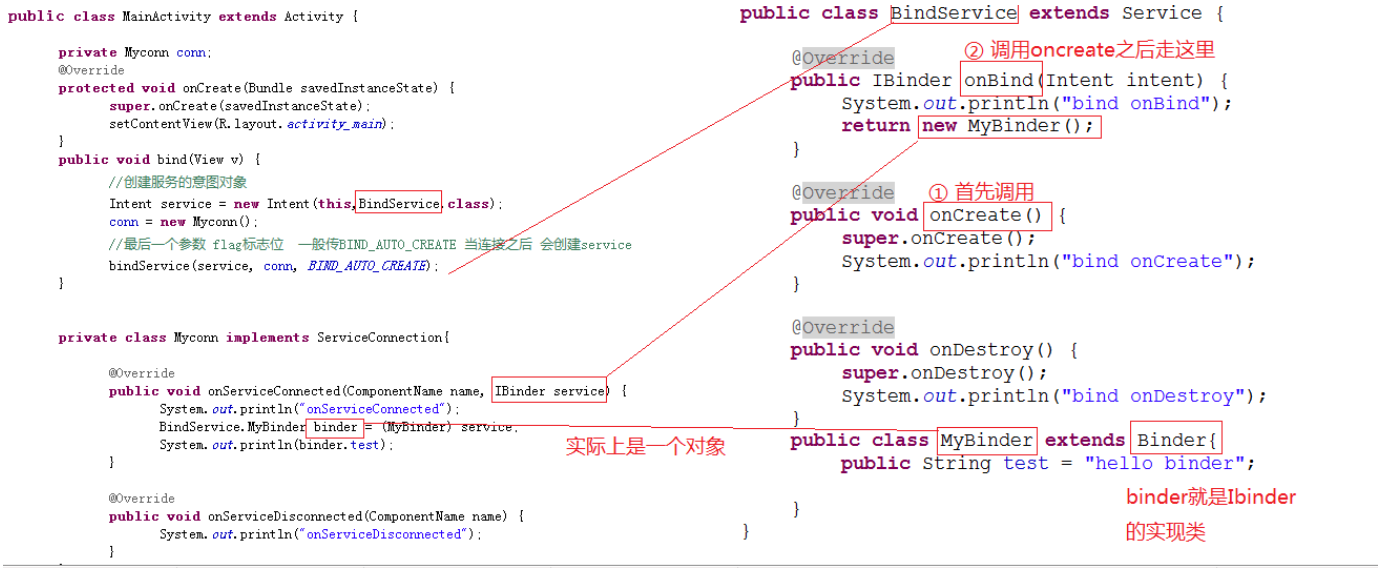
```
//创建服务的意图对象
        Intent service = new Intent(this,BindService.class);
        conn = new Myconn();
        //最后一个参数 flag标志位  一般传BIND_AUTO_CREATE 当连接之后 会创建service
        bindService(service, conn, BIND_AUTO_CREATE);
```

② 通过bindservice开启服务 走生命周期方法: onCreate()->onBind
多次调用bindservice onCreate()->onBind都只会执行一次
③ 如果service是在activity中通过bindservice方法打开的 service的生命周期就跟activity的生命周期关联起来(不求同生,但求同死)
④ 在activity退出的时候 如果不手动调用unbindservice方法会抛异常 提示serviceconnection泄露
⑤ unbindservice只能调用一次 多次调用会抛异常
⑥ 在service中的onBind方法 如果有返回值 那么 ServiceConnection中的onServiceConnected方法在连接创建之后就会被调用　其中第二个参数IBinder就是service中的onBind方法的返回值
通过这种方式

比较bindservice 和startservice之间区别

	走的生命周期方法	多次调用开启方法	service跟调用者的生命周期	关闭service的方法
bindservice	onCreate->onbind	oncreate onbind 只走一次	跟创建它的activity 不求同生,但求同死	unbindservice 只能执行一次(调用多次会有异常)
startservice	onCreate->onstartcommand	oncreate调用一次 onstartcommand 调用一次 执行一次	没关系	stopservice 可以调用多次

通过bindservice 可以获取到service中的一个内部类对象(Ibinder)
通过startservice开启的服务 不能够获取到service相关的对象



6 通过bindservice调用服务中的方法

①需要注意 四大组件 都不能直接使用new 构造 的方式创建, 四大组件要运行在框架中 需要通过系统的api创建对应的实例,自己new的四大组件 脱离了android的框架 只是普通的java类
② 通过bindservice调用步骤
2.1 在bindservice 方法传递的第二个参数 ServiceConnection 中 onServiceConnected方法 把传进来的 Ibinder对象保存起来
2.2 在 service里 创建内部类 继承BInder对象 在内部类中 暴露出方法 供activity调用
2.3 把内部类对象 作为onBind方法的返回值 返回去 在activity中可以获取到
2.4 通过获取到的service的内部类对象访问相关方法

7 音乐播放器框架

音乐播放器的应用 是通过activity作为控制的界面 实际播放的控制逻辑都要通过service实现
要求① activity 可以调用service中的方法 bindservice
② activity退出 service依然运行 startservice

7.1混合方式开启服务

startService bindservice 先后调用 顺序无所谓 但是都要调用
如果想销毁service 先后调用stopservice 和 unbindService

通过混合方式开启的服务 unbind之后 再次执行bindservice onbind 不会被调用 但是

ServiceConnection 中的 onServiceConnected每次都会被调用　也就是说 只要service没死 每次执行 bindservice之后 都会获取到service中的内部类对象 也就可以调用service中的方法

startservice 和bindservice虽然开了两次service 但是 只开启了一个servce对象

想销毁service 只要调用 stopservice 和unbindService 只要都调用了就可以关闭 跟顺序没关系

音乐播放器Activity代码

```
1.  public class MainActivity extends Activity {
2.      private MyConn conn;
3.      private IService binder;
4.
5.      @Override
6.      protected void onCreate(Bundle savedInstanceState) {
7.          super.onCreate(savedInstanceState);
8.          setContentView(R.layout.activity_main);
9.          Intent service = new Intent(this,MusicService.class);
10.         //开启一个服务
11.         startService(service);
12.         conn = new MyConn();
13.         //绑定一个服务
14.         bindService(service, conn, BIND_AUTO_CREATE);
15.     }
16.
17.     /**
18.      * 播放上一首歌曲的方法
19.      *
20.      * @param v
21.      */
22.     public void pre(View v) {
23.         binder.callPre();
24.     }
25.
26.     /**
27.      * 播放下一首歌曲的方法
28.      *
29.      * @param v
30.      */
31.     public void next(View v) {
32.         binder.callNext();
33.     }
34.
35.     /**
36.      * 开始播放歌曲的方法
37.      *
38.      * @param v
39.      */
40.     public void play(View v) {
41.         binder.callPlay();
42.     }
43.
```

```
    private class MyConn implements ServiceConnection{
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            binder = (IService) service;
            MyBinder binder2 = (MyBinder) service;
            binder2.playhigh();
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
        }
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //activity退出要解绑服务
        unbindService(conn);
    }
}
```

service服务代码

```
public class MusicService extends Service {

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

    @Override
    public void onCreate() {
        super.onCreate();
        //只执行一次的初始化操作 放到onCreate方法中
        System.out.println("音乐播放器初始化 MediaPlayer");
    }

    public void playMusic(){
        System.out.println("播放音乐...");
    }

    public void preSong(){
        System.out.println("播放上一首....");
    }

    public void nextSong(){
        System.out.println("播放下一首....");
    }

    public void playHighQulifyMusic(){
        System.out.println("播放无损音乐");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //如果需要通过activity多次传递数据给service 可以把相关逻辑放到onStartComma
```

```
nd当中
        return super.onStartCommand(intent, flags, startId);
    }

    public class MyBinder extends Binder implements IService{

        public void callPlay(){
            playMusic();
        }

        public void callPre(){
            preSong();
        }

        public void callNext(){
            nextSong();
        }

        public void playhigh(){
            playHighQulifyMusic();
        }
    }
}
```

可以通过接口实现只暴露部分方法

```
public interface IService {
    public void callPlay();
    public void callPre();
    public void callNext();

}
```

清单文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.itcast.musicplayer"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="cn.itcast.musicplayer.MainActivity"
            android:label="@string/app_name" >
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="cn.itcast.musicplayer.MusicService"></service>
    </application>

</manifest>
```

8 通过接口调用服务中的方法

9 aidl 安卓接口定义语言

aidl解决的是跨进程调用的问题

rpc remote procedure call 远程过程调用

ipc inter process communication 进程间通信(不同的应用之间进行数据传递和方法调用)

aidl流程

提供服务方法的应用

①创建service 在清单文件中注册 需要注意 要指定intent-filter 设置action

```
 <service android:name="cn.itcast.aidldemo.MyService">
            <intent-filter >
                <action android:name="cn.itcast.remotecall"/>
            </intent-filter>
        </service>
```

② 在service中 继承BInder对象 创建内部类 重写onBind方法 把创建的Binder对象做为onBind方法的返回值

```
public class MyBinder extends Binder{
        public void callRemoteMethod(){
            remotemethod();
        }
    }
```

③ 创建一个IService.java这个接口

```
1.  public interface IService {
2.      public void callRemoteMethod();
3.  }
```

④ MyBinder实现这个接口

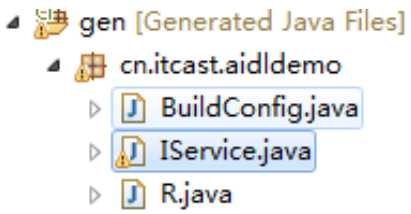
```
1.  public class MyBinder extends Binder implements IService{
2.          public void callRemoteMethod(){
3.              remotemethod();
4.          }
5.      }
```

⑤修改IService.java的扩展名为aidl 同时把aidl文件中的public都删除 如果成功 会发现在gen目录下有IService.java

aidl文件:

```
1 package cn.itcast.aidldemo;
2
3 interface IService {
4 void callRemoteMethod();
5 }
6
```

成功之后的gen目录:



⑥ 修改MyBinder类 让它继承Stub类 Stub就是在gen目录下生成的IService.java中的抽象类

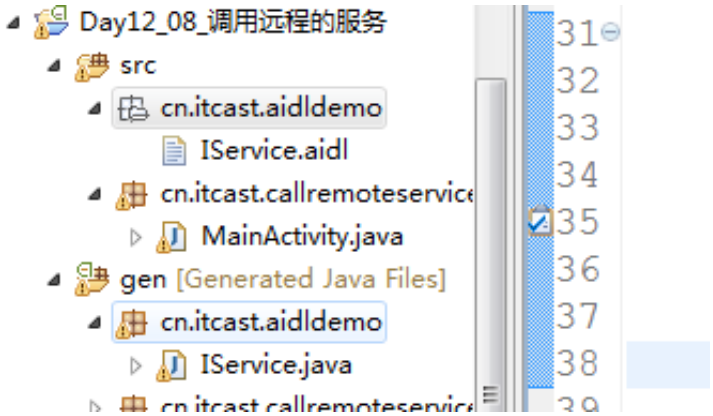
在需要进行远程访问的应用中

① 通过bindservice的方法访问service 注意 由于是跨进程访问 需要使用隐式意图

```
1.   Intent service = new Intent();
2.          service.setAction("cn.itcast.remotecall");
3.          conn = new Myconn();
4.          bindService(service, conn, BIND_AUTO_CREATE);
```

② 创建Myconn类 实现ServiceConnection接口

③ 在应用中创建一个跟提供服务的应用aidl文件所在包同报名的目录 并且把aidl文件复制过去 如果成功 会在gen目录下创建对应包名的 一个IService.java 文件



④ 在创建的Myconn中 onserviceConnected 方法里 调用 Stub.asInterface方法 把Ibinder对象转化为 IService对象

```
1.  private class Myconn implements ServiceConnection{
2.
3.          @Override
4.          public void onServiceConnected(ComponentName name, IBinder service) {
5.  // Stub就是生成的IService.java中的对象 asInterface就是这个对象的一个静态方法
6.              iService = Stub.asInterface(service);
7.          }
8.
9.          @Override
10.         public void onServiceDisconnected(ComponentName name) {
11.             // TODO Auto-generated method stub
12.
13.         }
14.
15.     }
```

⑤ 通过这个IService对象就可以访问远程的方法了

```
1.  public void call(View v){
2.          try {
3.              iService.callRemoteMethod();
4.          } catch (RemoteException e) {
5.              // TODO Auto-generated catch block
6.              e.printStackTrace();
7.          }
8.      }
```

aidl具体代码

提供服务的应用

service

```
1.  public class MyService extends Service {
2.
3.      @Override
4.      public IBinder onBind(Intent intent) {
```

```
        return new MyBinder();
    }

    @Override
    public void onCreate() {
        super.onCreate();
    }

    public void remotemethod(){
        System.out.println("remotemethod");
    }

    public class MyBinder extends Stub{
        public void callRemoteMethod(){
            remotemethod();
        }
    }
}
```

aidl文件代码

```
interface IService {
void callRemoteMethod();
}
```

清单文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.itcast.aidldemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="cn.itcast.aidldemo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="cn.itcast.aidldemo.MyService">
            <intent-filter >
```

```
                <action android:name="cn.itcast.remotecall"/>
            </intent-filter>
        </service>
    </application>

</manifest>
```

实现远程访问的应用代码

```
public class MainActivity extends Activity {

    private Myconn conn;
    private IService iService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent service = new Intent();
        service.setAction("cn.itcast.remotecall");
        conn = new Myconn();
        bindService(service, conn, BIND_AUTO_CREATE);
    }


    public void call(View v){
        try {
            iService.callRemoteMethod();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private class Myconn implements ServiceConnection{

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            iService = Stub.asInterface(service);
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            // TODO Auto-generated method stub

        }

    }
}
```

今天重点

概念 startservice 和 binderservice 区别
aidl的概念 ipc rpc

代码 电话拨号器(startservice) 音乐播放器框架(混合调用 bindservice方式调用服务中的方法)
aidl的流程(过一遍就可以了)