Android中的五中进程等级

1. 前台进程：

前台进程是用户当前正在使用的进程。只有一些前台进程可以在任何时候都存在。他们是最后一个被结束的。

2. 可见进程

可见进程是不包含前台的组件但是仍会影响用户在屏幕上所见内容的进程，除非前台进程需要获取它的资源，不然不会被中止。

3.服务进程

运行着一个通过startService() 方法启动的Service，它不会升级为上面两种级别。Service所在的进程虽然对用户不是直接可见的，但是他们执行了用户非常关注的任务（比如播放mp3，从网络下载数据）。只要前台进程和可见进程有足够的内存，系统不会回收他们。

4.后台进程

运行着一个对用户不可见的Activity（调用过 onStop() 方法)。这些进程对用户体验没有直接的影响，可以在服务进程、可见进程、前台进程需要内存的时候回收。

5.空进程

未运行任何程序组件。运行这些进程的唯一原因是作为一个缓存，缩短下次程序需要重新使用的启动时间。

从高优先级到低优先级:

1_前台进程(可看见，可操作)

2_可见进程(可看见，不可操作)

3_服务进程，通过startService()开启的服务进程(不可看见，不可操作)

4_后台进程(按HOME键，不可看见，不可操作)

5_空进程(按BACK键,不可看见，不可操作，好处：可缓存数据)

当Android系统内存不足时，按如下原则保存进程

前台进程->可见进程->【服务进程】->后台进程->空进程

【服务进程】有一个特点

1_内存不足，系统可以杀死这个服务进程

2_内存变得充足了，系统自己开启刚刚由系统杀死的服务进程

Andoid的单线程模式必须遵守两个规则：

●不要阻塞UI线程。用子线程来操作

●不要在UI线程之外访问Andoid的UI组件包。用主线程操作UI

服务Service

Service（服务）是一个没有用户界面的在后台运行执行耗时操作的应用组件。

一个组件能够绑定到一个Service与之交互（IPC机制），一个Service可能会处理网络操作，播

放音乐，操作文件 I/O或者与内容提供者（content provider）交互，所有这些活动都是在后台进行。

Service的两种状态

一、startService(开启)和stopService(停止)

onCreate->onStartCommand()->服务后台运行->onDestory()　当Activity销毁时，但Service没有销毁适合于自己书写的服务startService()后，有服务进程在后台长期运行Activity销毁，服务不会销毁

二、bindService(绑定)和unbindService(解绑):onCreate->onBind()->服务不在后台运行->onUnbind()->onDestory()

适合于借用第三书写的服务，我只是调用服务中的方法而以bindService()后，并没有服务进程在后台长期运行Activity销毁，一定要与服务解绑

onCreate()创建服务　　onBind(Intent intent)绑定服务返回数据（业务方法）给客户

onUnbind(Intent intent)解绑服务　　onDestroy()销毁服务

三、bindService(绑定)和unbindService(解绑)服务的程序逻辑：

● 主界面判断何进绑定或解绑服务

在判断之间要先创建　客户端与服务之间的监听器　用内部类实现implements ServiceConnection接口

重写onServiceConnected(ComponentName name, IBinder service)方法 该方法用于接收服务返回给客户的数据对象

创建监听器对象：　实现ServiceConnection接口的内部类对象

创建意图：　Intent intent = new Intent(this, 继承service的类.class);

参数一：意图，指明需要绑定的服务

参数二：监听器连接对象,该对象是客户端与服务的桥梁

参数三：标识位Context.BIND_AUTO_CREATE表示　如果第三方的服务暂时没有启动的话，就立刻让其启动

绑定服务：this.bindService(intent,myConn,Context.BIND_AUTO_CREATE);

解绑服务：this.unbindService(myConn);

以下音乐播放器：

```
/**
 * 客户端与服务
 * 之间的监听器
 */
private class MyConn implements ServiceConnection{
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {//new MyBinder()
        //为实例变量设置值
        myBinder = (MyBinder) service;
    }
    @Override
    public void onServiceDisconnected(ComponentName name) {
    }
}
```

```
    /**
     * 播放
     */
    public void click1(View view) {
        if(myConn!=null && myBinder!=null){
            myBinder.callPlayMusic("/mnt/sdcard/77.mp3");
        }
    }
    /**
     * 暂停
     */
    public void click2(View view) {
        if(myConn!=null && myBinder!=null){
            myBinder.callPauseMusic();
        }
    }
    /**
     * 停止
     */
    public void click3(View view) {
        if(myConn!=null && myBinder!=null){
            myBinder.callStopMusic();
        }
    }
    /**
```

```
 * 界面初始化
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //绑定服务
    if(myConn==null){
        myConn=new MyConn();
        Intent intent = new Intent(this,MusicService.class);
        //绑定服务，目的是调用服务中的方法
        this.bindService(intent,myConn,Context.BIND_AUTO_CREATE);
        //开启服务，目的是让服务进程长久在后台运行
        this.startService(intent);
    }
}
/**
 * 界面销毁
 */
@Override
protected void onDestroy() {
    //解绑服务
    if(myConn!=null){
        this.unbindService(myConn);
        myConn=null;
    }
    super.onDestroy();
}
```

●继承service的类中重写核心方法onBind(Intent intent)　　onDestroy()方法　　自定义业务方法　　其它方法更具情况而定

onBind(Intent intent)方法有一个返回值，如果要返回数据或业务方法给客户需要返回一个继承了extends Binder的内部类的对象

否则返回为null

继承了extends Binder的内部类,定义相关方法调用自定义业务方法即可

业务方法中定义相关的业务逻辑，如播放音乐，与第三方服务交互等

onDestroy()方法

释放资源，停止业务逻辑

以下音乐播放器：

```java
@Override
public IBinder onBind(Intent intent) {
    return new MyBinder();
}
public class MyBinder extends Binder{
    //注意：初学者不要递归调用
    public void callPlayMusic(String path){
        playMusic(path);
    }
    public void callPauseMusic(){
        pauseMusic();
    }
    public void callStopMusic(){
        stopMusic();
    }
}


private MediaPlayer mp;
/**
 * 播放
 * path表示需要播放音乐的路径
 */
public void playMusic(String path){
    if(mp == null){
        try {
            //创建MediaPlayer
            mp = new MediaPlayer();
            //设置音频类型,STREAM_MUSIC播放音乐
            mp.setAudioStreamType(AudioManager.STREAM_MUSIC);
            //设置音乐的来源，/mnt/sdcard/77.mp3
            mp.setDataSource(path);
            //加载音乐文件，同步加载，先加载文件完毕后，再播放音乐
            mp.prepare();
            //真正播放音乐
            //start[0....]
            mp.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }else{
        if(mp!=null){
            //继续播放[50...]
            mp.start();
        }
```

```
                mp.start();
            }
        }
    }
    /**
     * 暂停
     */
    public void pauseMusic(){
        if(mp!=null){
            //暂停
            mp.pause();
        }
    }
    /**
     * 停止
     */
    public void stopMusic(){
        if(mp!=null){
            //停止
            mp.stop();
            //释放资源
            mp.release();
            //强行mp设置为null
            mp=null;
        }
    }
```

```
@Override
public void onDestroy() {//服务销毁
    //停止音乐
    if(mp!=null){
        //停止
        mp.stop();
        //释放资源
        mp.release();
        //强行mp设置为null
        mp=null;
    }
    super.onDestroy();
}
```

Service的普通简单使用

1、创建一个普通类继承Service类重写方法　　android.app.Service包

一、onStartCommand()方法说明　一般为默认的重写内容不做处理

通过调用startService()请求Service启动时调用。一旦这个方法执行，这个服务就被开启并且在后台无限期地运行。如果你实现了这个方法，你就有责任在服务工作完毕后通过调用stopSelf()或者stopService()关闭服务（如果仅仅用来提供绑定，就不需要实现这个方法）

stopSelf()：用于服务自动销毁服务　　stopService()：手动调用停止服务

二、onBind()方法说明　　主要用于绑定服务

这个是bindService()绑定服务，当服务绑定的时候调用。在这个方法的实现中，需要通过返回一个IBinder对象提供客户端用来和Service通讯的接口，绑定服务需要重写一个自定义类继承Binder类，在自定义类中定义需要调用的业务方法，将自定义类做为onBind()方法的返回值。　当没有使用绑定服务时，可以不用关注这个类，这时候需要返回null。

三、onCreate()方法说明　服务创建　用于启动或创建一个管理或设备监听的创建启动

如：电话管理器，录音机等　如有用到这些可以重写后添加相应的代码

四、onDestory()方法说明　服务销毁

用来销毁停止服务　如调用stopSelf()或者stopService()关闭服务　　unbindService(解绑)

2、在清单文件中注册服务

<service android:name="cn.itheima.service.MyService"> </service>

某些操作需要添加相应的权限

如：电话，短信，录音机，开关机，网络，应用安装卸载，访问修改文件SD卡等

3、通过意图开启或停止服务

Intent intent = new Intent(this , 继承Service类的类名.class);

开启服务，【服务进程】会在后台，长期运行

this.startService(intent);　　或　　this.stopService(intent);

案例一：使用开启/停止服务 做电话窃听器

1、创建一个普通类继承Service类重写　onCreate()、onDestory()　方法

- 在onCreate()方法中初始化电话管理器，监听器，启动监听器

通过.getSystemService(Context.TELEPHONY_SERVICE);初始化电话管理器

继承extends PhoneStateListener类的内部类对象初始化电话监听器

参数一：创建的监听器，需要创建一个类 extends PhoneStateListener

参数二：监听手机状态

通过电话管理器对象.listen(监听器对象,PhoneStateListener.LISTEN_CALL_STATE); 启动监听服务

- 在 onDestory()方法中　服务停止　　取消电话监听

电话管理器对象.listen(监听器对象,PhoneStateListener.LISTEN_NONE);

- 继承 extends PhoneStateListener 类的内部类中重写 onCallStateChanged(int state, String

incomingNumber)方法，来判断电话不同的状态下要做的事。

如果TelephonyManager.CALL_STATE_RINGING电话响

就创建录音机，设置录音机的设备来源，录音格式，录音的编码，录音完存储位置，让录音机准备录音

```
new MediaRecorder();
.setAudioSource(MediaRecorder.AudioSource.MIC);
.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
.setOutputFile("/mnt/sdcard/luyin.3gp");
.prepare();
```

如果TelephonyManager.CALL_STATE_OFFHOOK: 接通，就开始录音"

```
recorder.start();
```

如果TelephonyManager.CALL_STATE_IDLE: 手机处于空闲 就向服务器上传录音并停止录音

```
//停止录音  recorder.stop();
//释放资源  recorder.release();
//GC便于回收  recorder=null;
```

● 主界面判断什么时候开启停止服务

意图开启服务

```
Intent intent = new Intent(MainActivity.this,PhoneService.class);
MainActivity.this.startService(intent);
```

意图停止服务

```
Intent intent = new Intent(MainActivity.this,PhoneService.class);
MainActivity.this.stopService(intent);
```

2、立即在清单文件中为其注册服务及样关权限

```
<!--  读 取 电 话 状 态 需 要 权 限  --><uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<!--  写 SD 卡 需 要 权 限  --><uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!--  录 音 需 要 权 限  --> <uses-permission android:name="android.permission.RECORD_AUDIO"/>
<service android:name="服务类"></service>
```

3、主线程开启服务

在onCreate()方法中初始化电话管理器，创建电话监听器，需要创建一个自定义类 extends PhoneStateListener类电话状态类

本地服务和远程服务

一、本地服务：服务(MusicService)与调用服务的组件(Activity)，是在同一进程中，叫本地服务

本地服务特点：服务类名你是知道

远程服务：服务与调用服务的组件，不在同一进程中，叫远程服务

远程服务特点：服务类名你不知道

二、进程间通信：二个应用之间的数据交互如果一个应用要用到另一个应用的功能，这样就必须跨进程访问，这候就需要进程间通信，即IPC 需要使用AIDL技术完成，

服务写一个AIDL文件，调用者拿到这个AIDL文件，在相同

的包下自动生成Java接口文件，所以调用者是基于这个Java接口

文件调用服务中的方法

AIDL

AIDL是 Android Interface definition language的缩写，一看就明白，它是一种android内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口

icp:interprocess communication :内部进程通信,远程服务通信

如：在线支付

支付服务系统：

1、创建类继承extends Service并在清单中注册服务同时必须要有意图的动作

2、定义好相关的业务方法

3、定义一个独立的接口类 interface 接口名 在接口中调用定好的业务方法

4、找到定义的接口文件将文件扩展名改为aidl，并将文件复制出一份到卓面以做为其他客户端要使用的接口上传

5、刷新工程目录之后被修改的aidl文件会报错，将aidl文件打开将文件中的public或其它权限修饰符删除保存即可

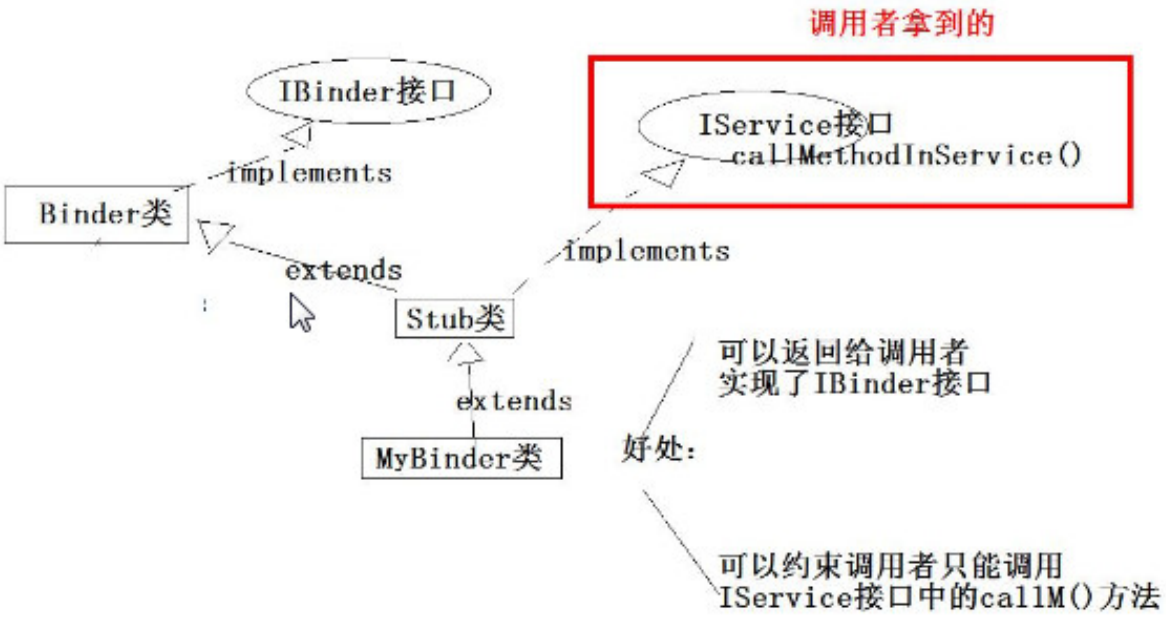
6、打开gen目录，找到自动生成的接口类文件打开，可以看到该文件的接口名及类名

7、回到服务类自定义一个内部类继承 extends 接口名.Stub 重写一个与aidl文件相同的方法在方法中调用业务方法

```java
public class ZhiFuFuWu extends Service {

    @Override
    public IBinder onBind(Intent intent) {

        return new MyBinder();
    }
    public class MyBinder extends Izhifu.Stub{
        public String SjZhifu(String name,String pwd){
            return zhifu(name,pwd);
        }
    }
    /*  服务的业务方法
     * 用户支付验证
     *  用户输入的用户名
     *   用户输入的密码
     *     返回一个验证结果     */
    public String zhifu(String name, String pwd) {
        if ("liu".equals(name) && "890912".equals(pwd)) {
            return "支付成功";
        } else {
            return "支付失败";
```



Izhifu.aidl

aidl接口文件

```
package liujinrong.zhifu;

interface Izhifu {
        String SjZhifu(String name,String pwd);
}
```

服务中内部类继承接口的方法

该方法在内部类中调用了服务的业务方法

调用支付服务系统的客户端：

1、下载到对应的aidl文件打开文件将文件中的包名复制出来，在项目中先创建一个包，包名就是aidl文件中复制的包名。必须一致

2、将aidl文件拷贝到创建的包下

3、在MainActivity类中先创建一个内部类实现implements ServiceConnection接口　客户端与服务之间的监听器

4、onServiceConnected(ComponentName name, IBinder service)方法中接口对象

aidl文件接口名　对象名 = 接口名.Stub.asInterface(service);

5、通过意图绑定服务

6、定义自己的业务方法，通过接口对象调用服务方法　返回结果 = 接口对象.服务业务方法

```
if (myconn == null) {
    myconn = new Myconn();
    Intent intent = new Intent();
    intent.setAction("www.liujinrong.cn");
    this.bindService(intent, myconn, Context.BIND_AUTO_CREATE);
}
```

绑定服务

```
public class Myconn implements ServiceConnection {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        Izhifu asIn = Izhifu.Stub.asInterface(service);
```

实现接口，创建客户端与服务之间的监听器

服务返回是接口对象，即aidl文件

```
public void ciclk(View v) {
    String name1 = name.getText().toString().trim();
    String pwd1 = pwd.getText().toString().trim();
    if (myconn != null && asIn != null) {
        try {
            String jieguo = asIn.SjZhifu(name1, pwd1);
                Toast.makeText(this, ""+name1 + jieguo + "本次付款:" + i + "元", 1)
                    .show();
```

客户端业务方法

通过远程服务提供的接口调用远程服务方法