Day01 安卓基础知识

1.1G-4G 的介绍

2.Android 操作系统介绍

3.Android 历史介绍

4.Android 系统架构（重点）

android 是基于 linux 的操作系统.使用 java 开发的,Dalvik 虚拟机.

四个层次,从外向里:
1.应用程序层:自己开发的应用程序安装在手机上后,就属于这一层,电话拨号器,短信,浏览器都属于这一层;
2.应用程序框架层:开发应用程序时调用的 api 就属于这一层;
3.基础类库层:基础类库和 Dalvik 虚拟机就属于这一层;
4.linux 内核;

5.两种虚拟机的不同（重点）

JVM 和 Dalvik 虚拟机的差别:

（1）ＪＶＭ使用栈架构，占用大量的存储空间；Dalvik 使用寄存器，占用的空间小；
（2）加载数据的过程不一样，ＪＶＭ加载的是ｃｌａｓｓ文件，Ｄａｌｖｉｌ虚拟机加载的是ｄｅｘ文件；

6.SDKManger 的使用

用来管理ＳＤＫ的版本的；

7.模拟器的简介及创建

AVD：Android virtual Device 安卓虚拟设备；

```
VGA 480 * 640
QVGA 240*320
HVGA 320*480
WQVGA 240*400
FWVGA 480*854
WVGA 480*800

AVD: Android virtual Device
```

8. AVD 的创建

9. DDMS 介绍

10. SDK 目录

adb:android debug bridge 安卓调试桥;

11. 创建 HelloWorld

12. Android 目录结构（重点）

src/ java 原代码存放目录

gen/ 自动生成目录

gen 目录中存放所有由 Android 开发工具自动生成的文件。目录中最重要的就是 R.java 文件。 这个文件由 Android 开发工具自动产生的。Android 开发工具会自动根据你放入 res 目录的资源，同步更新修改 R.java 文件。正因为 R.java 文件是由开发工具自动生成的，所以我们应避免手工修改 R.java。R.java 在应用中起到了字典的作用，它包含了各种资源的 id，通过 R.java，应用可以很方便地找到对应资源。另外编绎器也会检查 R.java 列表中的资源是否被使用到，没有被使用到的资源不会编绎进软件中，这样可以减少应用在手机占用的空间。

res/ 资源(Resource)目录

在这个目录中我们可以存放应用使用到的各种资源，如 xml 界面文件，图片或数据。具体请看 ppt 下方备注栏。

libs/ 支持库目录

程序开发时需要的一些三方的 jar 包可以放在这个目录，系统会自动把里面的 jar 包，添加到环境变量。

assets 资源目录

Android 除了提供/res 目录存放资源文件外，在/assets 目录也可以存放资源文件，而且/assets 目录下的资源文件不会在 R.java 自动生成 ID，所以读取/assets 目录下的文件必须指定文件的路径，如：file:///android_asset/xxx.3gp

AndroidManifest.xml 项目清单文件

这个文件列出了应用程序所提供的功能，以后你开发好的各种组件需要在该文件中进行配置，如果应用使用到了系统内置的应用(如电话服务、互联网服务、短信服务、GPS 服务等等)，你还需在该文件中声明使用权限。

project.properties 项目环境信息，一般是不需要修改此文件

13. Android 的打包过程

IDE 在部署应用程序时,先检查 adb 工具是否正常运行,如果正常运行,使用 adb 工具把自己工程的 apk 软件上传到模拟器,上传结束后,模拟器会自动安装 apk 文件,安装完成后会自动打开软件的主界面.

14. ADB 命令

adb devices 列出所有已经启动的设备,连接已经掉线的模拟器.
adb shell 挂载到 linux 的命令行控制台;
adb install 安装 apk 软件;

15. 电话拨号器（重点）

做项目的步骤:

第一步:理解需求;
第二步:根据效果图设计 UI 界面,
第三步:使用代码实现功能;

代码:

MainActivity.java:

```
package com.example.helloworld;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
```

```
    private EditText et_number;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //加载布局文件,并转换成一个 view 对象显示在界面上
        setContentView(R.layout.activity_main);
        //初始化控件
        et_number = (EditText) findViewById(R.id.et_number);
    }

    /**
     * 按钮的单击事件的模版方法
     * @param view
     */
    public void call(View view){
        System.out.println("====call===========");
        //得到输入框中的内容
        String number = et_number.getText().toString().trim();
        if(TextUtils.isEmpty(number)){
            //提示用户电话号码不能为空
            Toast.makeText(this, "电话号码不能为空", Toast.LENGTH_SHORT).show();
            return;
        }else{
            //创建拨打电话号码的意图对象
            Intent intent  = new Intent(Intent.ACTION_CALL);
            //设置拨打的目标电话号码
            intent.setData(Uri.parse("tel:"+number));
            //调用手机自带的拨打电话号码的功能
            startActivity(intent);
        }
    }
}
```

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"
     >

    <EditText
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:hint="请输入电话号码"
        android:id="@+id/et_number" />

    <Button
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拨打"
        android:onClick="call"
        />

</LinearLayout>
```

注意在清单文件中添加权限:

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

16. 四种点击事件

第一种方式:在布局文件中给按钮添加 onClick,属性的值就是代码中方法的名称,方法的写法固定的:public void call(View view){};

示例:

(1)布局文件:

```
        <Button
             android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="拨打"
            android:onClick="call"
            />
```

(2)java 代码:

```
        public void call(View view){
            -------
        }
```

第二种方式:在布局文件中给按钮添加一个 ID,在代码中声明并初始化控件,再给这按钮添加一个单击事件的监听器:

示例:

(1)布局文件:

```
        <Button
             android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="拨打"
            android:id="@+id/btn_call"
            />
```

(2)java 代码:

```
            private Button btn_call;
            @Override
            protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                //加载布局文件,并转换成一个 view 对象显示在界面上
                setContentView(R.layout.activity_main);

                //初始化控件
                btn_call = (Button) findViewById(R.id.btn_call);
                //给按钮添加单击事件的监听器
                btn_call.setOnClickListener(new OnClickListener() {

                    /**
                     * 点击按钮时会调用这个方法
                     */
                    @Override
                    public void onClick(View v) {
                        //业务逻辑代码
                        Toast.makeText(MainActivity.this, "new OnClickListener()...",
0).show();

                    }
                });
            }
```

第三中方式:在布局文件中给按钮添加一个 ID,在代码中声明并初始化控件,再给这按钮添加一个单击事件的监听器,在代码中自定义单击事件的监听器:

示例:

(1)布局文件:

```
    <Button
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拨打"
        android:id="@+id/btn_call"
        />
```

(2)java 代码:

```
package com.example.call;


import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    private Button btn_call;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //加载布局文件,并转换成一个 view 对象显示在界面上
        setContentView(R.layout.activity_main);

        //初始化控件
        btn_call = (Button) findViewById(R.id.btn_call);
        //给按钮添加单击事件的监听器
        btn_call.setOnClickListener(new MyListener());
    }


    /**
     * 自定义单击事件的监听器
     * @author Administrator
     *
     */
    private class MyListener implements OnClickListener{

        @Override
        public void onClick(View v) {
            //业务逻辑代码
            Toast.makeText(MainActivity.this, " MyListener ...", 0).show();
```

```
            }
        }
    }
```

第四种方式:在布局文件中给按钮添加一个 ID,在代码中声明并初始化控件,再给这按钮添加一个单击事件的监听器,参数值为 this;需要让 activity 实现 OnClickListener 接口,并实现方法 onClick():

示例:

(1)布局文件:

```
    <Button
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拨打"
        android:id="@+id/btn_call"
        />
```

(2)java 代码:

```
package com.example.call;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity  implements OnClickListener{

    private Button btn_call;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //加载布局文件,并转换成一个 view 对象显示在界面上
        setContentView(R.layout.activity_main);

        //初始化控件
        btn_call = (Button) findViewById(R.id.btn_call);
        //给按钮添加单击事件的监听器
```

```
    //      btn_call.setOnClickListener(new MyListener());

            btn_call.setOnClickListener(this);
        }


        @Override
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.btn_call://判断当前点击的是哪个控件
                Toast.makeText(MainActivity.this, "实现接口的方式", 0).show();
                break;

            default:
                break;
            }

        }
    }
```

17. 框架布局

```
Android 中的五大布局:
    框架布局,线性布局,相对布局,表格布局,绝对布局;
    TableRow:表格中的一行;

框架布局的特点:
    在布局文件中,位于下面控件会显示在上面控件的上面;
```

代码:

```
<FrameLayout 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"
   >

    <TextView
        android:gravity="center"
        android:layout_gravity="center"
        android:layout_width="300dip"
        android:layout_height="300dip"
        android:background="#ff0000"
```

```
        android:text="红色 TextView" />

    <TextView
        android:gravity="center"
        android:layout_gravity="center"
        android:layout_width="200dip"
        android:layout_height="200dip"
        android:background="#000000"
        android:text="黑色 TextView" />

    <TextView
        android:gravity="center"
        android:layout_gravity="center"
        android:layout_width="100dip"
        android:layout_height="100dip"
        android:background="#0000ff"
        android:text="蓝色 TextView" />

</FrameLayout>
```

18. 线性布局（重点）

```
线性布局方向:
    1.垂直方向:每个控件都摆放在单独的一行;
    2.水平方向:把布局文件中所有的控件都摆放到同一行;
```

垂直方向布局的代码:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="第一个 textView"
        android:textSize="20sp"
        />

    <TextView
         android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="第二个 textView"
        android:textSize="20sp"
        android:background="#ff0000"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="按钮"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="002 按钮"
        />

</LinearLayout>
```

水平方向布局的代码:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="第一个 textView"
        android:textSize="20sp"
        />

    <TextView
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="第二个 textView"
        android:textSize="20sp"
        android:background="#ff0000"
        />

    <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="按钮"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="002 按钮"
        />

</LinearLayout>
```

19. 相对布局（重点）

```
特点:
    布局文件中的控件都是按照相对位置来摆放.
```

代码:

```
<RelativeLayout 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"
>

<TextView
    android:id="@+id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第一个 TextView"
    android:textSize="20sp" />

 <TextView
   android:layout_toRightOf="@id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第二个 TextView"
    android:textSize="20sp"
    android:id="@+id/tv_second" />

 <TextView
   android:layout_below="@id/tv_second"
```

```
   android:layout_toRightOf="@id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第三个 TextView"
    android:background="#00ff00"
    android:textSize="20sp" />

<RelativeLayout 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"
>

<TextView
    android:id="@+id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第一个 TextView"
    android:textSize="20sp" />

 <TextView
   android:layout_toRightOf="@id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第二个 TextView"
    android:textSize="20sp"
    android:id="@+id/tv_second" />

 <TextView
   android:layout_below="@id/tv_second"
   android:layout_toRightOf="@id/tv_first"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="第三个 TextView"
    android:background="#00ff00"
    android:textSize="20sp" />

</RelativeLayout>
```

Day02sd 卡保存、文件权限、存储数据、xml、帧动画

1. 单元测试的相关概念

根据测试时是否知道源代码分类:

(1)白盒测试:根据源代码的逻辑测试程序;
(2)黑盒测试:根据需求并且不知道源代码测试程序;

根据测试的粒度分类:

(1)方法测试:写完一个方法,并且能够独立运行;
(2)单元测试:写完一个能够独立运行的业务逻辑单元后测试;
(3)模块测试:写完整个模块后测试;
(4)集成测试:软件开发完成后做整体测试,客户端与服务器端的联调测试也属于集成测试;

根据测试的次数分类:

(1)压力测试:在单位时间内高频次的访问应用程序,工具有 LoadRunner,Jemeter;
(2)冒烟测试:monkey 测试;

2. 单元测试

JUnit 单元测试 does not specify a android.test.InstrumentationTestRunner instrumentation or does not declare uses-library android.test.runner in its AndroidManifest.xml

android.test.InstrumentationTestRunner:是一个指令集,可以告诉模拟器给指定的应用程序做单元测试.

android.test.runner:对应 android 应用程序做单元员测试用到的测试包.

单元测试的过程:IDE 工具把 android 工程打包成 apk 文件,然后使用 adb 工具把 apk 文件上传到模拟器上,模拟器根据应用程序(apk 软件)中的指令集对指定的应用程序做单元测试.

步骤:

1.写业务类方法;
2.给业务方法写单元测试的用例;
3.注意引用这两个包;

3. Logcat 日志工具的使用

Level 日志的级别:

E ERROR:错误信息,使用红色字体标识,程序抛出的异常信息也是 ERROR 信息;等级最高;
W WARN:警告信息,使用橙色字体标识,工程中没有指定需要的权限;等级较高;
I INFO:一般的调试信息,使用绿色字体标识,在代码中打印的一般的调试信息;等级高;
D DEBUG:调试信息,使用蓝色字体标识,在代码中送 log.d 打印的调试信息;等级一般;
V VERBOSE:所有的调试信息,使用黑色字体标识;等级最低;

4.把数据存储到文件并取数据显示在界面上

Androin 中存储数据的五种方式:
1.文件;
2.SharedPreferences 存储;
3.ContentProvider 内容提供者;
4.SQLite 数据库;
5.网络;

软件的内部存储目录:/data/data/包名/

回显数据的步骤:

1.在 onCreate 房中得到之前保存的数据;
2.显示在控件上;

5.存储到 SD 卡（重点）

在向 SD 卡中写数据之前,需要在清单文件中添加向 SD 写数据的权限;

代码:

```
package com.itheima.savetosd;

import java.io.File;
import java.io.FileOutputStream;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
```

```
import android.widget.Toast;

public class MainActivity extends Activity {

    private EditText et_data;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_data = (EditText) findViewById(R.id.et_data);
    }



    public void save(View view){
        //得到输入框中的数据
               String data = et_data.getText().toString().trim();

               if(TextUtils.isEmpty(data)){
                   Toast.makeText(this, "数据不能为空", 0).show();
                   return;
               }else{
                   try {
                       //判断 SD 卡是否已经插上

if(Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())){
                           //把数据保存到 SD 上
//                         File file = new File("/mnt/sdcard/a.txt");
                           //获得 SD 卡的根目
录:Environment.getExternalStorageDirectory()
                           File file = new
File(Environment.getExternalStorageDirectory()+"/a.txt");
                           FileOutputStream fos = new FileOutputStream(file);
                           fos.write(data.getBytes());

                           fos.close();
                       }

               } catch (Exception e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
```

```
}
}
```

manifest 清单文件中添加权限:

```
<!-- 添加向 SD 卡中写数据的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

6. 获取 SD 的大小及可用空间

代码:

```
package cn.itcast.sdspace;

import java.io.File;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.os.StatFs;
import android.text.format.Formatter;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

    private TextView tv_space;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tv_space = (TextView) findViewById(R.id.tv_space);
    }

    public void getSpace(View view) {
        // ------只兼容 2.3 及其以后的版本-------------------
        // 获得 SD 卡的跟目录 /mnt/sdcard/
        // File file = Environment.getExternalStorageDirectory();
        // // 获得总存储空间,单位是字节
        // long totalSpace = file.getTotalSpace();
```

```
        // // 获得可用存储空间,单位是字节
        // long usableSpace = file.getUsableSpace();
        // 转换数据单位
        // String totalSize = Formatter.formatFileSize(this, totalSpace);
        //
        // String usableSize = Formatter.formatFileSize(this, usableSpace);
        // 显示存储空间
        // tv_space.setText("SD 的存储空间:"+usableSize+"/"+totalSize);

        // ------兼容所有版本-------------------
        // 得到统计存储目录空间的对象
        StatFs fs = new StatFs(Environment.getExternalStorageDirectory() + "");
        // 得到总块数
        int blockCount = fs.getBlockCount();
        // 得到块的大小
        int blockSize = fs.getBlockSize();
        // 得到可用的块数
        int availCount = fs.getAvailableBlocks();

        // 计算可用存储空间
        long usableSpace = availCount * blockSize;
        // 计算总存储空间
        long totalSpace = blockCount * blockSize;

        // 转换数据单位
        String usableSize = Formatter.formatFileSize(this, usableSpace);
        // 转换数据单位
        String totalSize = Formatter.formatFileSize(this, totalSpace);
        // 显示存储空间
        tv_space.setText("SD 的存储空间:" + usableSize + "/" + totalSize);

    }
}
```

7. 文件的权限概念

```
- rwx rwx r--
第一位表示文件的类型:d 目录,l 连接,-文件;
第一组表示当前用户对这个文件拥有的权限,android 中表示当前应用程序;
第二组表示与当前用户在同一组中其他用户对这文件拥有的操作权限;
第三组表示出了第一组\第二组之外的其他用户对这个文件拥有的操作权限,android 中的其他软件;
```

8.权限的使用

9.SharedPreferences 存储数据（重点）

SharedPreferences:共享偏好;
SharedPreferences 的应用场景:保存给软件设置的参数;
SharedPreferences 内部使用类似 map 的数据结构来保存数据的;

代码:

```
package com.itheima.sp;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    private EditText et_qq;
    private EditText et_pwd;
    private SharedPreferences sp;
    /*
     *
     * 在 activity 被创建后调用这个方法,
     * 用来初始化这个实例对象的
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_qq = (EditText) findViewById(R.id.et_qq);
        et_pwd = (EditText) findViewById(R.id.et_pwd);

        sp = this.getSharedPreferences("config", Context.MODE_PRIVATE);
        //从 sp 中获取 qq
```

```
        String qq = sp.getString("qq", null);
        String pwd = sp.getString("pwd", null);
        //把 qq 显示在输入框上
        et_qq.setText(qq);
        et_pwd.setText(pwd);

    }

    public void login(View view){
        //取出 qq 和密码
        String qq = et_qq.getText().toString().trim();
        String pwd = et_pwd.getText().toString().trim();

        if(TextUtils.isEmpty(qq) || TextUtils.isEmpty(pwd)){
            Toast.makeText(this, "用户名或密码为空", 0).show();
            return;
        }else{
            //保存用户名和密码
//          //得到系统提供 sp 对象
//          SharedPreferences sp = this.getSharedPreferences("config",
Context.MODE_PRIVATE);
            //获得 sp 的编辑器
            Editor editor = sp.edit();
            //编辑需要保存的参数
            editor.putString("qq", qq);
            editor.putString("pwd", pwd);
            //提交保存数据
            editor.commit();
        }
    }
}
```

10. 序列化 xml 格式的数据到文件上

把数据写到 xml 格式的文件上;

应用场景:备份短信和联系人;
步骤:
(1)创建一个 xml 的序列器:

```
    XmlSerializer s = Xml.newSerializer();
```

(2)初始化序列化器

```
    FileOutputStream fos = new FileOutputStream(file);
    // os 输出流,
```

```
    // encoding 序列数据时使用的字符集编码
    s.setOutput(fos, "UTF-8");
(3)写文档的开头部分
(4)写根元素
(5)写数据对应的标签;
(6)写根元素
(7)结束文档
```

代码: try { // 保存用户名和密码到 xml 格式的文件上 // 把数据保存到 SD 卡的根目录下 File file = new File(Environment.getExternalStorageDirectory() + "/config.xml");

```
            // 创建 xml 的序列化器
            XmlSerializer s = Xml.newSerializer();

            // 初始化序列化器
            FileOutputStream fos = new FileOutputStream(file);
            // os 输出流,
            // encoding 序列数据时使用的字符集编码
            s.setOutput(fos, "UTF-8");

            // 开始写数据
            // 写 xml 文档的开头一行

            s.startDocument("UTF-8", true);
            // 写根元素开始标签
            s.startTag(null, "userInfo");

            // 写 qq 的开始标签
            s.startTag(null, "qq");
            // 写 qq 的值
            s.text(qq);
            // 写 qq 的结束标签
            s.endTag(null, "qq");

            // 写 pwd 的开始标签
            s.startTag(null, "pwd");
            // 写 pwd 的值
            s.text(pwd);
            // 写 pwd 的结束标签
            s.endTag(null, "pwd");

            //写根元素的结束标签
            s.endTag(null, "userInfo");
```

```
            //写文档的数据部分
            s.endDocument();


            fos.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
```

11.使用 pull 解析 xml 格式的数据（重要）

```
SAX,DOM,DOM4J,pull

pull 是一个轻量级的 xml 解析框架;是一个事件类型的解析器;

应用场景:解析服务器端返回的 xml 数据;还原短信或者联系人;

步骤:
    1.创建一个 xml 的解析器;
    2.初始化解析器;
    3.得到事件的解析类型;
    4.解析相关的开始标签,表签体,结束标签;
```

代码:

```
    //得到 xml 文件
            File file =new
File(Environment.getExternalStorageDirectory()+"/config.xml");

            //1.得到 xml 文件的解析器
            XmlPullParser parser = Xml.newPullParser();

            //2.初始化解析器
            FileInputStream fis =new FileInputStream(file);
            //inputStream 目标文件的输入流
            //inputEncoding 解析数据使用字符集编码
            parser.setInput(fis, "UTF-8");

            //得到解析的事件类型
//          XmlPullParser.START_DOCUMENT:解析到文件的开始部分的事件类型
//          XmlPullParser.END_DOCUMENT:解析到文件的结尾部分的事件类型
//          XmlPullParser.START_TAG:解析到标签开始位置的事件类型
```

```
//          XmlPullParser.END_TAG:解析到标签结束位置的事件类型
//          XmlPullParser.TEXT:解析到标签体的事件类型
            int type = parser.getEventType();

            while(type != XmlPullParser.END_DOCUMENT){
                switch (type) {
                case XmlPullParser.START_TAG://解析到标签开始位置的事件类型
                    //parser.getName()得到解析的当前标签的名称
                    if("userInfo".equals(parser.getName())){
                        userInfo = new UserInfo();
                    }else if("qq".equals(parser.getName())){
                        //得到标签体的内容
                        String qq = parser.nextText();
                        userInfo.setQq(qq);
                    }else if("pwd".equals(parser.getName())){
                        String pwd = parser.nextText();
                        userInfo.setPwd(pwd);
                    }
                    break;

                case XmlPullParser.END_TAG://解析到标签结束位置的事件类型

                    if("userInfo".equals(parser.getName())){
                        System.out.println("userInfo:"+userInfo);
                        userInfo = null;
                    }
                    break;
                }
                //让 pull 解析到下一个位置
                type = parser.next();
```

12.帧动画

帧动画:加载一组有序的图片,然后一帧一帧的播放;
参考源代码;

Day03 数据库与 listView

01_android 下数据库的创建（重点）

SQLite 是一个嵌入式的数据库,轻量级的数据库;

在 Android 平台上，集成了一个嵌入式关系型数据库–SQLite，SQLite3 支持 NULL、INTEGER、REAL（浮点数字）、TEXT(字符串文本)和 BLOB(二进制对象)数据类型，虽然它支持的类型只有五种，但实际上 sqlite3 也接受 varchar(n)、char(n)、decimal(p,s) 等数据类型，只不过在运算或保存时会转成对应的五种数据类型。 SQLite 最大的特点是你可以把各种类型的数据保存到任何字段中，而不用关心字段声明的数据类型是什么。例如：可以在 Integer 类型的字段中存放字符串，或者在布尔型字段中存放浮点数，或者在字符型字段中存放日期型值。

但有一种情况例外：定义为 INTEGER PRIMARY KEY 的字段只能存储 64 位整数， 当向这种字段保存除整数以外的数据时，将会产生错误。 另外，在编写 CREATE TABLE 语句时，你可以省略跟在字段名称后面的数据类型信息，如下面语句你可以省略 name 字段的类型信息：

主键必须是 integer 类型;

```
CREATE TABLE person (personid integer primary key autoincrement, name varchar(20));
```

SQLite 可以解析大部分标准 SQL 语句，如：

查询语句：select * from 表名 where 条件子句 group by 分组字句 having ... order by 排序子句
如：select * from person
　　select * from person order by id desc
　　select name from person group by name having count(*)>1
分页 SQL 与 mysql 类似，下面 SQL 语句获取 5 条记录，跳过前面 3 条记录
select * from Account limit 5 offset 3 或者 select * from Account limit 3,5
插入语句：insert into 表名(字段列表) values(值列表)。如： insert into person(name, age) values(‘传智’,3)
更新语句：update 表名 set 字段名=值 where 条件子句。如：update person set name=‘传智‘ where id=10
删除语句：delete from 表名 where 条件子句。如：delete from person where id=10

获取添加记录后自增长的 ID 值：SELECT last_insert_rowid()

SQLiteOpenHelper：管理数据库的版本；

在 android 应用程序中创建按数据库的步骤：
1、写一个 DBHelper，继承了 SQLiteOpenHelper,重新写了父类的构造方法、onCreate、onUpGrade:

```
//创建数据库
DBHelper helper = new DBHelper(this, "account.db", null, 1);
```

onCreate 是在数据库创建的时候调用的，主要用来初始化数据表结构和插入数据初始化的记录

onUpGrade 是在数据库版本升级的时候调用的，主要用来改变表结构

2、调用 db = helper.getWritableDatabase()，得到数据对象

数据库的存储位置：/data/data/包名/databases/数据库名

02_数据库 sql 语句的增删改查

创建表结构:

```
create table account (_id integer primary key autoincrement,name varchar(20),money varchar(20));

insert into account (name,money) values ('lf','100000000');

select name,money from account;

update account set money='2000000000' where name='lf';

delete from account where name ='lf';
```

03_android 下数据库的增删改查（重点）

使用 db.execSQL()方法的场景:多表关联操作;

rawQuery 从数据库表中查询出原始的数据;

代码:

04_数据库的另外一种增删改查方法（重点）

代码:

```
package com.itheima.sqlitedb;

import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Toast;

import com.itheima.sqlitedb.db.AccountDBHelper;

public class MainActivity extends Activity {

    private SQLiteDatabase db;
    private AccountDBHelper helper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        helper = new AccountDBHelper(this, "account.db", null, 1);

    }

    public void insert(View view) {

        db = helper.getReadableDatabase();
        // 封装列和值的对象,类似于 map 的数据结构
        ContentValues values = new ContentValues();
        values.put("name", "lf");
        values.put("money", "30000000000");
        // 插入数据
        // table 表名
        // nullColumnHack 是否为空的列
        // values 封装列和值的对象,类似于 map 的数据结构
        db.insert("account", null, values);
        // 数据库对象关闭
        db.close();
    }

    public void query(View view) {

        db = helper.getReadableDatabase();
        // 查询
        // table 表名
        // columns 查询出的列
        // selection 查询条件 where id=? and nam=?
        // selectionArgs 条件的参数值数组
        // groupBy 分组查询条件
        // having 分组的条件
```

```
        // orderBy 排序的条件
        Cursor cursor = db.query("account", new String[] { "name", "money" },
                "name = ?", new String[] { "lf" }, null, null, null);
        // 循环取出数据
        // cursor.moveToNext()让游标指向下一个数据,true表示指向了一条数据,false表示指向了结果集的结尾
        while (cursor.moveToNext()) {
            // 从cursor对象中取出列的值,根据列的索引,索引使用columns中的索引
            String name = cursor.getString(0);

            String money = cursor.getString(1);
            System.out.println("name=" + name + "; money=" + money);
            Toast.makeText(this, "name=" + name + "; money=" + money, 0).show();
        }

        cursor.close();

    }

    public void update(View view) {

        db = helper.getReadableDatabase();
        // 封装列和值的对象,类似于map的数据结构
        ContentValues values = new ContentValues();
        values.put("money", "500000000");
        // table 表名
        // values 更新的列及其值
        // whereClause 更新的条件
        // whereArgs 条件的值
        db.update("account", values, "name=?", new String[] { "lf" });
        // 数据库对象关闭
        db.close();
    }

    public void delete(View view) {

        db = helper.getReadableDatabase();
        // 封装列和值的对象,类似于map的数据结构
        // table 表名
        // values 更新的列及其值
        // whereClause 更新的条件
        // whereArgs 条件的值
        db.delete("account", "name=?", new String[] { "lf" });
        // 数据库对象关闭
```

```
        db.close();
    }
}
```

05_命令行查看数据库

使用 adb shell 命令进入 linux 的命令控制台;
使用 cd 命令切换/data/data/包名/databases/目录;
使用 sqlite3 数据库文件名,打开数据库文件;

06_数据库的事务（重点）

数据库的事务: 同一组操作要么同时成功要么同时失败;

转账:
lf 给强哥转出 1000 万;

lf-1000 万;
qg+1000 万;

代码:

```
//在一组业务操作之前开启事务
        db = helper.getReadableDatabase();

        try {
            //在一组业务操作之前开启事务
            db.beginTransaction();

            String sql = "update account set money=? where name=?";

            db.execSQL(sql, new String[]{"7000000000","lf"});

//          System.out.println(10/0);

            db.execSQL(sql, new String[]{"8000000000","qg"});
            Toast.makeText(this, "操作成功", 0).show();

            //告诉数据库使用已经成功
            db.setTransactionSuccessful();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }finally{
            //一组操作完成之后结束事务
            db.endTransaction();
        }
        db.close();
```

07_listview 的使用（重点）

listview 用来在界面上显示列表的;

listview 工作原理:
在显示一个条目时才调用 getView 这个方法,创建 item 的 view 对象.隐藏的条目占用的内存空间会被释放掉了.换句话来说,只有看见的条目占用内存空间.

使用的步骤:
1.在布局文件中使用 listview 控件,并添加一个 ID;

2.在 activity 中声明并初始化 listview 控件;

3.给 listview 填充数据;

布局文件:

```
<RelativeLayout 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"
tools:context=".MainActivity" >

<ListView
    android:id="@+id/lv"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true" >
</ListView>

</RelativeLayout>
```

代码:

```
package com.itheima.listview;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends Activity {

    private ListView lv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lv = (ListView) findViewById(R.id.lv);
        //给 listview 填充数据
        lv.setAdapter( new MyAdapter());
    }

    /**
     * 自定义数据适配器
     * @author Administrator
     *
     */
    private class MyAdapter extends BaseAdapter{

        /*
         * 得到 listview 中显示的 item 的个数
         *
         */
        @Override
        public int getCount() {
            return 100;
        }

        /**
         * 创建 item 对应的 view 对象的
         * position item 在 listview 中位置
         * convertView 缓存中的 view 对象
```

```
         * parent item 的父级控件
         */
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {

            TextView tv = new TextView(MainActivity.this);

            tv.setText("textView:"+position);
            tv.setTextSize(25);
            return tv;
        }




        /**
         * 根据位置得到 listview 中一个条目
         */
        @Override
        public Object getItem(int position) {

            return null;
        }

        /**
         * 根据位置得到 listview 中一个条目的 ID
         */
        @Override
        public long getItemId(int position) {
            return 0;
        }
    }
}
```

08_ArrayAdapter

示例代码:

```
    //需要显示的数据,是一个数组
    String[] objects = new String[]{"范冰冰","尔康","苍老师","高圆圆","黄渤","黄海波","
强哥"};
    //填充 listview
    //context 上下文对象
    //resource 条目对应的布局文件的资源 ID
```

```
    //textViewResourceId 布局文件中 textview 控件的 ID,在这个控件上显示数据
    //objects 需要显示的数据,是一个数组
    lv.setAdapter(new ArrayAdapter(this, R.layout.item, R.id.tv, objects));
```

布局文件:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >

<TextView
    android:id="@+id/tv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="25sp" />

</LinearLayout>
```

09_SimpleAdapter

代码:

```
package com.itheima.simpleadapter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;

public class MainActivity extends Activity {

    private ListView lv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        lv = (ListView) findViewById(R.id.lv);

        List<Map<String, String>> list = new ArrayList<Map<String, String>>();

        Map<String, String> map;
        // 准备需要显示的数据
        for (int i = 0; i < 100; i++) {
            map = new HashMap<String, String>();
            map.put("id", i + "");
            map.put("name", "name" + i);
            list.add(map);
        }
        // from map 中需要显示的列
        String[] from = new String[] { "id", "name" };
        // to item 布局文件中控件的 ID,用来显示数据的
        int[] to = new int[] { R.id.tv_id, R.id.tv_name };
        // 填充 listview
        // context 上下文对象
        // list 需要显示的数据 list 类型,list 中存放的数据是 map 类型的
        // resource 条目对应的布局文件的资源 ID
        // from map 中需要显示的列
        // to item 布局文件中控件的 ID,用来显示数据的
        lv.setAdapter(new SimpleAdapter(this, list, R.layout.item, from, to));
    }

}
```

布局文件:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <TextView
        android:id="@+id/tv_id"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="id" />
```

```
    <TextView
        android:id="@+id/tv_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="name"/>

</LinearLayout>
```

10_复杂 listview 界面的显示（重点）

步骤:

```
1.在布局文件上使用 listview:
2.在代码中声明并初始化 listview 空间;
3.给 listview 准备需要显示的数据;
4.填充数据:自定义了一个数据适配器,继承了 BaseAdapter;
    重写了 getCount():给 listView 设置显示的 item 的个数,就是显示数据的条数;
    重写 getView():创建 item 的界面并且给它填充数据;item 布局文件的作用就是为了显示条目的数据;
```

代码:

MainActivity.java:

```
package com.itheima.verboselistview;

import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;

import com.itheima.verboselistview.domain.NewsInfo;
import com.itheima.verboselistview.service.ParseNewsService;

public class MainActivity extends Activity {

    private ListView lv;
    private List<NewsInfo> list;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    lv = (ListView) findViewById(R.id.lv);
    //给 listview 准备数据
    list = ParseNewsService.parseNews();

    //填充 listview
    lv.setAdapter(new MyAdapter());
}


private class MyAdapter extends BaseAdapter{

    /**
     * 设置 listview 中显示的 item 的个数
     */
    @Override
    public int getCount() {
        return list.size();
    }

    /**
     * 创建一个 item 对应的 view 对像
     * position item 在 list 中的位置
     * convertView 缓存中的 view 对象
     * parent item 的父级控件
     *
     * view 视图 :能看到的图形,可以看见的界面或者控件(如 Button 等);
     */
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //把 item 对应的布局文件转换成一个 view 对象并返回
        //把布局文件加载为一个 view 类型的对象
        //root 父级控件 null 表示当前父级控件=listview
        View view = View.inflate(MainActivity.this, R.layout.item, null);
        // 从 view 中得到控件
        TextView tv_title = (TextView) view.findViewById(R.id.tv_title);
        TextView tv_desc = (TextView) view.findViewById(R.id.tv_desc);

        //从 list 集合中取出一个新闻条目显示 item 对应的界面上
        NewsInfo item = list.get(position);
```

```
            String title = item.getTitle();
            String desc = item.getDesc();

//          给 item 的界面填充数据
            tv_title.setText(title);
            tv_desc.setText(desc);

            return view;
        }

        @Override
        public Object getItem(int position) {
            // TODO Auto-generated method stub
            return null;
        }

        @Override
        public long getItemId(int position) {
            // TODO Auto-generated method stub
            return 0;
        }
    }
}
```

item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <ImageView
        android:id="@+id/iv_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        />

    <TextView
        android:layout_toRightOf="@id/iv_image"
        android:singleLine="true"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:text="我是新闻标题标题标题标题标题标题标题标题标题标题标题标题标题标题"
        android:textSize="20sp"
        android:id="@+id/tv_title"
        />

    <TextView
        android:layout_below="@id/tv_title"
        android:layout_toRightOf="@id/iv_image"
        android:maxLines="2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息我是描述信息"
        android:textSize="18sp"
        android:id="@+id/tv_desc"
        />

</RelativeLayout>
```

11_数据库 listview 界面的显示

```
在数据库中准备数据:

从数据库中查询出数据,放到一个 list 中:

把 list 中的数据显示在 listview 上:
```

代码:

```
package com.itheima.dbtolistview;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
```

```
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.itheima.dbtolistview.db.AccountDBHelper;
import com.itheima.dbtolistview.domain.AccountInfo;

public class MainActivity extends Activity {

    private SQLiteDatabase db;
    private AccountDBHelper helper;

    private ListView lv;
    // 存放从数据库中查询出来的数据
    private List<AccountInfo> list = new ArrayList<AccountInfo>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        helper = new AccountDBHelper(this, "account.db", null, 1);

        // 初始化 listview
        lv = (ListView) findViewById(R.id.lv);

    }

    public void insert(View view) {

        db = helper.getReadableDatabase();

        String sql = "insert into account (name,money) values (?,?)";
        for (int i = 0; i < 100; i++) {
            // sql 插入的语句
            // bindArgs 条件值
            db.execSQL(sql, new String[] { "qg" + i, "20000" + i });
        }

        db.close();
        Toast.makeText(this, "操作成功", 0).show();

    }
```

```
    /**
     * 查询数据显示在 listview 上
     * @param view
     */
    public void query(View view) {
        //準備數據
        queryData();
        // 填充数据
        lv.setAdapter(new MyAdapter());
    }

    public void queryData() {

        db = helper.getReadableDatabase();
        String sql = "select name,money from account";
        // sql 插入的语句
        // bindArgs 条件值
        Cursor cursor = db.rawQuery(sql, null);
        // // 循环取出数据
        AccountInfo info = null;

        // cursor.moveToNext()让游标指向下一个数据,true 表示指向了一条数据,false 表示指向了
结果集的结尾
        while (cursor.moveToNext()) {

            // 创建数据模型
            info = new AccountInfo();

            // 从 cursor 对象中取出列的值,根据列的索引,索引使用 columns 中的索引
            String name = cursor.getString(0);
            String money = cursor.getString(1);
            info.setName(name);
            info.setMoney(money);
            System.out.println("info======" + info);
            list.add(info);
        }

        cursor.close();

        db.close();
        Toast.makeText(this, "操作成功", 0).show();

    }
```

```
    private class MyAdapter extends BaseAdapter {

        /**
         * 设置 listview 显示 ite 的个数
         */
        @Override
        public int getCount() {
            return list.size();
        }

        /**
         * 创建 item 的界面
         */
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            // 把 item 的布局文件加载为 view 对象
            View view = View.inflate(MainActivity.this, R.layout.item, null);
            // 得到 item 布局文件中的空间,为了便于给它们填充数据
            TextView tv_name = (TextView) view.findViewById(R.id.tv_name);
            TextView tv_money = (TextView) view.findViewById(R.id.tv_money);

            // 给 item 的布局文件中的空间填充数据
            AccountInfo info = list.get(position);
            tv_name.setText(info.getName());
            tv_money.setText(info.getMoney());

            return view;
        }

        @Override
        public Object getItem(int position) {
            // TODO Auto-generated method stub
            return null;
        }

        @Override
        public long getItemId(int position) {
            // TODO Auto-generated method stub
            return 0;
        }

    }

}
```

12_对话框合集

Day04 网络编程

图片,文字,音频,视频;

> 客户端向服务器发送一个请求,服务器端返回数据,客户端把数据显示在界面;

01_网络图片查看器（重点）

> 需求:把服务器的一个图片加载客户端显示出来;
>
> 服务器端接口的地址:http://192.168.18.110:8080/meinv.jpg;

访问网络的步骤:

> 1.把网络地址转成一个 URL,打开一个 http 类型的连接;
> 2.设置连接的请求参数:请求方式 GET,POST,必须大写;连接的超时时间;
> 3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务器内部错误,结束服务器端返回的二进制数据流;
> 4.在清单文件中添加访问互联网的权限;

模版代码:

```
        //根据图片的网络地址,从服务器端加载这个图片,并显示出来
//          1.把网络地址转成一个 URL,打开一个 http 类型的连接;
                URL url = new URL(path);
                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
//          2.设置连接的请求参数:请求方式 GET,POST,必须大写;连接的超时时间;
                conn.setRequestMethod("GET");
                conn.setConnectTimeout(3000);
//          3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务器内部错误,接收服务器端返回的二进制数据流;
                int code = conn.getResponseCode();
                if(code == 200){
                    //接收服务器端返回的二进制数据流;
                    InputStream is = conn.getInputStream();
}
```

在清单文件中添加权限:

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

02_子线程不能修改 UI 界面

android.os.NetworkOnMainThreadException：网络在主线程上的异常：
原因:google 要求从 android4.0 开始,在主线程中不能有耗时的操作;
从 android4.0 开始,为让能够让 UI 界面运行的更加流畅,要求在主线程中不能有访问网络的操作,这样就避免因为在主线程访问网络时间太长导致主界面卡死等现象的发生.

在 activity 中的 oncreate 和单击事件的响应方法都是运行在主线程上的;

Only the original thread that created a view hierarchy can touch its views:
只有创建 UI 界面的线程才能修改 UI 界面;只有主线程才能修改 UI 界面,子线程不能直接修改 UI 界面;

03_消息处理机制的原理（重点）

使用 handler 的三个步骤:

```
1.在主线程中创建 handler 的成员变量;

    private Handler handler = new Handler(){
    };

2.在子线程中得到 handler 的引用,给 handler 发送一个消息;

    //2.在子线程中得到 handler 的引用,给 handler 发送消息
        Message msg = Message.obtain();
        msg.obj = result;
        handler.sendMessage(msg);

3.在 handler 中修改 UI 界面;

    private Handler handler = new Handler(){
    public void handleMessage(Message msg) {

        String result = (String) msg.obj;
        Toast.makeText(MainActivity.this, result, 0).show();
    };
    };
```

handler 的工作原理或者机制(handler,message,looper 三这之间的关系):

前提知识: 所有使用 UI 界面的操作系统,后台都运行着一个死循环,这个死循环叫做轮询器 Looper.这个死循环在不停的监听和接收用户发出的指令,一旦接收指令就立即执行.

在应用程序一启动的时候系统就给它提供了一 looper,子线程在修改 UI 界面时会把消息发送给 handler,handler 把消息放到 looper 内部维护的消息队列中,looper 还有一个死循环,它在不停的从消息队列中取消息,取到消息后在发送给 handler,handler 再去修改 UI 界面;

04_网络 HTML 查看器

获得服务器端一个页面的源代码显示在客户端的界面上;

代码:

MainActivity.java:

```
package com.itheima.htmlview;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.itheima.htmlview.utils.StreamTools;

public class MainActivity extends Activity {

    private EditText et_path;
    private TextView tv_result;
    //1.在主线程创建 handler 的成员变量
    private Handler handler = new Handler(){
        //3.修改 UI 界面
        public void handleMessage(Message msg) {
            String result = (String) msg.obj;
            tv_result.setText(result);
```

```
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et_path = (EditText) findViewById(R.id.et_path);
        tv_result = (TextView) findViewById(R.id.tv_result);
    }

    public void viewHtml(View view){
        final String path = et_path.getText().toString().trim();
        if(TextUtils.isEmpty(path)){
            Toast.makeText(this, "请输入 html 页面的网络地址", 0).show();
            return;
        }else{

            new Thread(){
                public void run() {
                    try{
//                      1.把网络地址转成一个 URL,打开一个 http 类型的连接;
                        URL url = new URL(path);
                        HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
//                      2.设置连接的请求参数:请求方式 GET,POST,必须大写;连接的超时时间;
                        conn.setRequestMethod("GET");
                        conn.setConnectTimeout(3000);
//                      3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务
器内部错误,接收服务器端返回的二进制数据流;
                        int code = conn.getResponseCode();
                        if(code == 200){
                            //接收服务器端返回的二进制数据流;
                            InputStream is = conn.getInputStream();

                            //把二进制 刘转换成字符串
                            String result = StreamTools.parseString(is);

                            //2.在子线程中得到 handler 的引用,给 handler 发送消息
                            Message msg = Message.obtain();
                            msg.obj = result;
                            handler.sendMessage(msg);
                        }
                    }catch (Exception e) {
```

```
                    e.printStackTrace();
                }
            };
        }.start();
//

    }

}
}
```

StreamTools.java:

```
package com.itheima.htmlview.utils;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class StreamTools {

    private static String result;

    /**
     * 把二进制输入里转成字符串
     * @param is
     * @return
     */
    public static String parseString(InputStream is) {

        try {
            //创建一个输出流
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            int len = -1;
            byte[] buffer = new byte[1024];

            while( (len = is.read(buffer)) != -1){
                //把数据从 buffer 中写到输出流中
                baos.write(buffer, 0, len);
            }
//          把输出流转成字符串
            result = new String(baos.toByteArray());

        } catch (Exception e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return result;

    }

}
```

在清单文件中添加权限:

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

05_消息处理常用另一个 API

```
runOnUiThread(new Runnable() {

    @Override
    public void run() {

            iv_pic.setImageBitmap(bm);
        }
    });
```

06_新闻客户端

新闻客户端代码实现的步骤:

```
1.设计 UI 界面包含 item 的 UI 界面;
2.在代码中声明并初始化 listview;
3.访问网络得到服务器端返回的二进制数据流(里面是 xml 的数据);
4.解析 xml 数据,放到 list 中;
5.把 list 中的新闻数据填充到 listview 上;
```

代码:

MainActivity.java:

```
package com.itheima.newsclient;

import java.io.InputStream;
import java.net.HttpURLConnection;
```

```
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;

import com.itheima.newsclient.domain.NewsInfo;
import com.loopj.android.image.SmartImageView;

public class MainActivity extends Activity {

    private ListView lv;
    private String path = "http://192.168.18.110:8080/news.xml";

    private List<NewsInfo> list = new ArrayList<NewsInfo>();

    //1.在主线程中创建 handler 的成员变量
    private Handler handler = new Handler(){
        //3.修改 UI 界面
        public void handleMessage(Message msg) {
//          5.把 list 中的新闻数据填充到 listview 上;
            lv.setAdapter(new MyAdapter());
        };
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //填充数据
        lv = (ListView) findViewById(R.id.lv);

        //准备数据:获得网络上 xml 的数据并解析出来放到 list 中
        new Thread(){
```

```
            public void run() {
                try {
                    //访问网络上 xml 文件,解析 xml 数据
//                  1.把网络地址转成一个 URL,打开一个 http 类型的连接;
                    URL url = new URL(path);
                    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
//                  2.设置连接的请求参数:请求方式 GET,POST,必须大写;连接的超时时间;
                    conn.setRequestMethod("GET");
                    conn.setConnectTimeout(3000);
//                  3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务器内部错误,接收服务器端返回的二进制数据流;
                    int code = conn.getResponseCode();
                    if(code == 200){
                        //接收服务器端返回的二进制数据流;
                        InputStream is = conn.getInputStream();
                        //解析服务器端返回的 xml 数据
                        list = ParseNewsService.parseNews(is);

                        //2.在子线程中得到 handler 的引用,给 handler 发送消息
                        Message msg = Message.obtain();
                        handler.sendMessage(msg);
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            };
        }.start();

    }

    private class MyAdapter extends BaseAdapter{

        /**
         * 设置 listview 显示的条目的个数
         */
        @Override
        public int getCount() {
            return list.size();
        }

        /**
         * 创建 item 的界面
         */
```

```
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            //把 item 的布局文件加载为 view 对象
            View view = View.inflate(MainActivity.this, R.layout.item, null);
            //得到 item 布局文件中控件
            SmartImageView iv_image = (SmartImageView)
view.findViewById(R.id.iv_image);
            TextView tv_title = (TextView) view.findViewById(R.id.tv_title);
            TextView tv_desc = (TextView) view.findViewById(R.id.tv_desc);
            TextView tv_type = (TextView) view.findViewById(R.id.tv_type);
//          TextView tv_commentNum = (TextView) view.findViewById(R.id.tv_commentNum);

            //给 item 中的控件填充数据
            NewsInfo info = list.get(position);

            tv_title.setText(info.getTv_title());
            tv_desc.setText(info.getTv_desc());
//          tv_commentNum.setText(info.getTv_commentNum());

            //使用 smartImageView 加载网络上一个图片
            iv_image.setImageUrl(info.getImage_url());

            String type = info.getTv_type();
            if("1".equals(type)){
                tv_type.setText(info.getTv_commentNum()+ " 评论");
                tv_type.setTextColor(Color.BLACK);
            }else if("2".equals(type)){
                tv_type.setText("专题");
                tv_type.setTextColor(Color.RED);
            }else if("3".equals(type)){
                tv_type.setText("视频");
                tv_type.setTextColor(Color.BLUE);
            }
            return view;
        }
        @Override
        public Object getItem(int position) {
            // TODO Auto-generated method stub
            return null;
        }

        @Override
        public long getItemId(int position) {
            // TODO Auto-generated method stub
```

```
            return 0;
        }


    }

}
```

ParseNewsService.java:

```
package com.itheima.newsclient;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import org.xmlpull.v1.XmlPullParser;

import com.itheima.newsclient.domain.NewsInfo;

import android.util.Xml;

public class ParseNewsService {
    /**
     * 解析 xml 文件中的新闻条目,返回一个 list
     * @return
     */
    public static  List<NewsInfo> parseNews(InputStream is){
        List<NewsInfo> list = new ArrayList<NewsInfo>();
        try {
            //解析 xml 文件中的新闻数据
            //1.创建 xml 的解析器
            XmlPullParser parser = Xml.newPullParser();
            //2.初始化解析器
            parser.setInput(is, "UTF-8");
            //3.得到解析的事件类型
            int type = parser.getEventType();
            NewsInfo item = null;
            while(type != XmlPullParser.END_DOCUMENT){
                switch (type) {
                case XmlPullParser.START_TAG:
                    if("item".equals(parser.getName())){//判断是否解析到了 item 标签的开
始位置

                        item = new NewsInfo();
                    }else if("title".equals(parser.getName())){
```

```
                    String title = parser.nextText();
                    item.setTv_title(title);
                }else if("description".equals(parser.getName())){
                    String description = parser.nextText();
                    item.setTv_desc(description);
                }else if("image".equals(parser.getName())){
                    String image = parser.nextText();
                    item.setImage_url(image);
                }
                else if("type".equals(parser.getName())){
                    String newsType = parser.nextText();
                    item.setTv_type(newsType);
                }
                else if("comment".equals(parser.getName())){
                    String comment = parser.nextText();
                    item.setTv_commentNum(comment);
                }
                break;

            case XmlPullParser.END_TAG:
                if("item".equals(parser.getName())){//判断是否解析到了 item 标签的开
始位置
                    list.add(item);
                    System.out.println("item==="+item);
                    item = null;
                }
                break;
            }
            //解析到下一个位置
            type = parser.next();
        }

        is.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return list;
  }
}
```

NewsInfo.java:

```
package com.itheima.newsclient.domain;
```

```
public class NewsInfo {

    private String tv_title;
    private String tv_desc;
    private String tv_type;
    private String tv_commentNum;
    private String image_url;

    public String getTv_title() {
        return tv_title;
    }

    public void setTv_title(String tv_title) {
        this.tv_title = tv_title;
    }

    public String getTv_desc() {
        return tv_desc;
    }

    public void setTv_desc(String tv_desc) {
        this.tv_desc = tv_desc;
    }

    public String getTv_type() {
        return tv_type;
    }

    public void setTv_type(String tv_type) {
        this.tv_type = tv_type;
    }

    public String getTv_commentNum() {
        return tv_commentNum;
    }

    public void setTv_commentNum(String tv_commentNum) {
        this.tv_commentNum = tv_commentNum;
    }

    public String getImage_url() {
        return image_url;
    }
```

```
    public void setImage_url(String image_url) {
        this.image_url = image_url;
    }

    @Override
    public String toString() {
        return "NewsInfo [tv_title=" + tv_title + ", tv_desc=" + tv_desc
                + ", tv_type=" + tv_type + ", tv_commentNum=" + tv_commentNum
                + ", image_url=" + image_url + "]";
    }

}
```

activity_main.xml:

```
<RelativeLayout 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" >

    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <com.loopj.android.image.SmartImageView
        android:id="@+id/iv_image"
        android:layout_width="80dip"
        android:layout_height="80dip"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:layout_toRightOf="@id/iv_image"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="我是新闻标题"
        android:id="@+id/tv_title"
        android:singleLine="true"
        android:textSize="16sp"
        android:textColor="#000000"
        />

    <TextView
        android:layout_below="@id/tv_title"
         android:layout_toRightOf="@id/iv_image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="我是新闻描述我是新闻描述"
        android:id="@+id/tv_desc"
        android:maxLines="3"
         android:textSize="14sp"
        />

    <TextView

       android:layout_below="@id/tv_desc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="专题"
        android:id="@+id/tv_type"
        android:textSize="10sp"
          android:layout_alignParentRight="true"

        />
</RelativeLayout>
```

在清单文件中添加权限:

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

07_使用 smartImageView 显示新闻图片（重点）

步骤:

```
1.把框架的源代码拷贝到自己的工程中;
2.在布局文件中使用自定义的控件;
    <com.loopj.android.image.SmartImageView
    android:id="@+id/iv_image"
```

```
    android:layout_width="80dip"
    android:layout_height="80dip"
    android:src="@drawable/ic_launcher" />

3.在代码中使用自定义的控件;
    SmartImageView iv_image = (SmartImageView) view.findViewById(R.id.iv_image);
    //使用 smartImageView 加载网络上一个图片
    iv_image.setImageUrl(info.getImage_url());
```

在清单文件中添加权限:

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

08_smartImageView 的工作原理

```
过程与网络图片查看器一样;
```

09_使用 GET 方式向服务器端提交数据（重点）

http://192.168.18.110:8080/web/servlet/LoginServlet 使用 GET 方式提交数据时,是把提交的参数组拼到 url 地址的后面; 如:
http://192.168.18.110:8080/web/servlet/LoginServlet?username=123&password=abc ?username=123&password=abc

10_使用 POST 方式提交数据（重点）

url:

```
http://192.168.18.110:8080/web/servlet/LoginServlet
```

与 get 方式的不同点:

```
1.请求方式不同使用 POST 方式;
2.URL 地址不一样,没有把参数组拼到后面;
3.多了 Content-Type,Content-Length;
4.把提交的参数组拼了一个字符串;
5.以二进制流的形式写给服务器端的;
```

示例代码:

```
    //组拼参数到 URL 地址的后面
                        String data = "username="+qq+"&password="+pwd;
```

```
                        //访问网络上 xml 文件,解析 xml 数据
    //                  1.把网络地址转成一个 URL,打开一个 http 类型的连接;
                        URL url = new URL(path);
                        HttpURLConnection conn = (HttpURLConnection)
url.openConnection();

//              2.设置连接的请求参数:请求方式 GET,POST,必须大写;连接的超时时间;
                        conn.setRequestMethod("POST");
                        conn.setConnectTimeout(3000);

                        conn.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
                        conn.setRequestProperty("Content-Length",data.length()+"");

                        //以二进制流的形式写给服务器端的;
                        //设置是否允许把数据写到服务器端,true 表示允许,false 表示不允许
                        conn.setDoOutput(true);
                        //把数据写到服务器端
                        OutputStream os = conn.getOutputStream();
                        os.write(data.getBytes());


//              3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务器内
部错误,接收服务器端返回的二进制数据流;
                        int code = conn.getResponseCode();
                        if(code == 200){
                            //接收服务器端返回的二进制数据流;
                            InputStream is = conn.getInputStream();
                            //把输入流转成字符串
                            String result = StreamTools.parseString(is);

                            //2.在子线程中得到 handler 的引用,给 handler 发送消息
                            Message msg = Message.obtain();
                            msg.obj = result;
                            handler.sendMessage(msg);

}
```

在清单文件中添加权限:

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

11_补间动画（tweens）

补间动画:让图片发生透明度,旋转,缩放,平移;
alpha,rotate,scale,translate;

代码:

```
package com.itheima.tweens;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.AlphaAnimation;
import android.view.animation.AnimationSet;
import android.view.animation.RotateAnimation;
import android.view.animation.ScaleAnimation;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);
    }

    /**
     * 透明度动画
     *
     * @param view
     */
    public void alpha(View view) {
        // fromAlpha 开始是透明度的值 0 表示完全透明,1 表示完全不透明
        // toAlpha 结束时透明度的值
        AlphaAnimation aa = new AlphaAnimation(0, 1.0f);
        // 设置动画播放的时长
        aa.setDuration(3000);
        // 设置重复播放的次数
        aa.setRepeatCount(2);
        // 设置重复播放的模式
        aa.setRepeatMode(AlphaAnimation.REVERSE);
```

```
        // 在 imageview 上播放动画
        iv.startAnimation(aa);
    }


    /**
     * 旋转动画
     *
     * @param view
     */
    public void rotate(View view) {

        RotateAnimation ra = new RotateAnimation(0, 360,
RotateAnimation.RELATIVE_TO_SELF, 0.5f,  RotateAnimation.RELATIVE_TO_SELF,  0.5f);
        // 设置动画播放的时长
        ra.setDuration(3000);
        // 设置重复播放的次数
        ra.setRepeatCount(2);
        // 设置重复播放的模式
        ra.setRepeatMode(RotateAnimation.REVERSE);
        // 在 imageview 上播放动画
        iv.startAnimation(ra);
    }

    /**
     * 缩放动画
     *
     * @param view
     */
    public void scale(View view) {

        ScaleAnimation sa = new ScaleAnimation(0, 1.0f, 0, 1.0f,
ScaleAnimation.RELATIVE_TO_SELF, 0.5f, ScaleAnimation.RELATIVE_TO_SELF, 0.5f);
        // 设置动画播放的时长
        sa.setDuration(3000);
        // 设置重复播放的次数
        sa.setRepeatCount(2);
        // 设置重复播放的模式
        sa.setRepeatMode(ScaleAnimation.REVERSE);
        // 在 imageview 上播放动画
        iv.startAnimation(sa);
    }

    /**
```

```
     * 平移动画
     *
     * @param view
     */
    public void trans(View view) {

        TranslateAnimation ta = new
TranslateAnimation(TranslateAnimation.RELATIVE_TO_PARENT, 0,
TranslateAnimation.RELATIVE_TO_PARENT,
                1.0f, TranslateAnimation.RELATIVE_TO_PARENT, 0,
TranslateAnimation.RELATIVE_TO_PARENT, 1.0f);
        // 设置动画播放的时长
        ta.setDuration(3000);
        // 设置重复播放的次数
        ta.setRepeatCount(2);
        // 设置重复播放的模式
        ta.setRepeatMode(TranslateAnimation.REVERSE);
        // 在 imageview 上播放动画
        iv.startAnimation(ta);
    }


    /**
     * 动画集合
     * @param view
     */
public void set(View view) {

        //shareInterpolator 动画运行的速度与轨迹的综合体
        AnimationSet ta = new AnimationSet(true);
//      // 设置动画播放的时长
//      ta.setDuration(3000);
//      // 设置重复播放的次数
//      ta.setRepeatCount(2);
//      // 设置重复播放的模式
//      ta.setRepeatMode(TranslateAnimation.REVERSE);

        AlphaAnimation aa = new AlphaAnimation(0, 1.0f);
        // 设置动画播放的时长
        aa.setDuration(3000);
        // 设置重复播放的次数
        aa.setRepeatCount(2);
        // 设置重复播放的模式
        aa.setRepeatMode(AlphaAnimation.REVERSE);
```

```
        ta.addAnimation(aa);

        RotateAnimation ra = new RotateAnimation(0, 360,
RotateAnimation.RELATIVE_TO_SELF, 0.5f,  RotateAnimation.RELATIVE_TO_SELF,  0.5f);
        // 设置动画播放的时长
        ra.setDuration(3000);
        // 设置重复播放的次数
        ra.setRepeatCount(2);
        // 设置重复播放的模式
        ra.setRepeatMode(RotateAnimation.REVERSE);

        ta.addAnimation(ra);

        ScaleAnimation sa = new ScaleAnimation(0, 1.0f, 0, 1.0f,
ScaleAnimation.RELATIVE_TO_SELF, 0.5f, ScaleAnimation.RELATIVE_TO_SELF, 0.5f);
        // 设置动画播放的时长
        sa.setDuration(3000);
        // 设置重复播放的次数
        sa.setRepeatCount(2);
        // 设置重复播放的模式
        sa.setRepeatMode(ScaleAnimation.REVERSE);

        ta.addAnimation(sa);

        TranslateAnimation taa = new
TranslateAnimation(TranslateAnimation.RELATIVE_TO_PARENT, 0,
TranslateAnimation.RELATIVE_TO_PARENT,
                1.0f, TranslateAnimation.RELATIVE_TO_PARENT, 0,
TranslateAnimation.RELATIVE_TO_PARENT, 1.0f);
        // 设置动画播放的时长
        taa.setDuration(3000);
        // 设置重复播放的次数
        taa.setRepeatCount(2);
        // 设置重复播放的模式
        taa.setRepeatMode(TranslateAnimation.REVERSE);
        //把动画对象放到动画集合中
        ta.addAnimation(taa);

        // 在 imageview 上播放动画
        iv.startAnimation(ta);
    }
}
```

Day05 网络编程第二天课程：

01_post 方式提交数据的中文乱码解决(重点)

出现中文乱码的情况:

```
1.服务器端返回的数据是中文:

2.客户端给服务器端提交的数据是中文:
```

解决办法:

```
让客户端与服务器使用的字符集编码一样;
客户端使用 URLEncoder.encode(qq)对中文进行转码
```

02_get 提交数据乱码的解决(重点)

```
解决办法:
让客户端与服务器使用的字符集编码一样;
客户端使用 URLEncoder.encode(qq)对中文进行转码;
```

03_使用 HttpClient 向服务器端提交数据(重点)

```
把 HttpClient 作为浏览器来使用:

1.打开一个浏览器;
2.输入网址;
3.按回车;
```

使用 GET 发送请求:

```
package com.itheima.login;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
```

```
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import com.itheima.login.utils.StreamTools;

public class MainActivity extends Activity {

    private EditText et_qq;
    private EditText et_pwd;



    private Handler handler = new Handler(){
        public void handleMessage(Message msg) {

            String result = (String) msg.obj;
            Toast.makeText(MainActivity.this, result, 0).show();
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_pwd = (EditText) findViewById(R.id.et_pwd);
        et_qq = (EditText) findViewById(R.id.et_qq);
    }

    public void login(View view){
        final String qq = et_qq.getText().toString().trim();
        final String pwd = et_pwd.getText().toString().trim();
        if(TextUtils.isEmpty(qq)|| TextUtils.isEmpty(pwd)){
            Toast.makeText(this, "用户名和密码都不能为空", 0).show();
            return;
        }else{
            //访问网络把数据传递给服务器端
```

```
            new Thread(){
                public void run() {
                    try {
                        String path =
"http://192.168.18.110:8080/web/servlet/LoginServlet";
                        //组拼参数到 URL 地址的后面
                        path = path + "?username="+URLEncoder.encode(qq,
"UTF-8")+"&password="+URLEncoder.encode(pwd, "UTF-8");
//                      1.打开一个浏览器;
                        HttpClient client = new DefaultHttpClient();
//                      2.输入网址;
                        HttpGet http = new HttpGet(path);
//                      3.按回车;
                        HttpResponse response = client.execute(http);
                        //得到响应码
                        int code = response.getStatusLine().getStatusCode();
                        if(code == 200){
                            //先得到相应对象中的数据实体,在得到里面的数据内容
                            InputStream is = response.getEntity().getContent();
                            //把输入流转成字符串
                            String result = StreamTools.parseString(is);
                            //2.在子线程中得到 handler 的引用,给 handler 发送消息
                            Message msg = Message.obtain();
                            msg.obj = result;
                            handler.sendMessage(msg);
                        }

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                };
            }.start();

        }
    }

}
```

使用 POST 发送请求:

```
package com.itheima.login;

import java.io.InputStream;
import java.io.OutputStream;
```

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import com.itheima.login.utils.StreamTools;

public class MainActivity extends Activity {

    private EditText et_qq;
    private EditText et_pwd;

    private Handler handler = new Handler(){
        public void handleMessage(Message msg) {
            String result = (String) msg.obj;
            Toast.makeText(MainActivity.this, result, 0).show();
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_pwd = (EditText) findViewById(R.id.et_pwd);
        et_qq = (EditText) findViewById(R.id.et_qq);
    }
```

```
    public void login(View view){
        final String qq = et_qq.getText().toString().trim();
        final String pwd = et_pwd.getText().toString().trim();
        if(TextUtils.isEmpty(qq)|| TextUtils.isEmpty(pwd)){
            Toast.makeText(this, "用户名和密码都不能为空", 0).show();
            return;
        }else{
            //访问网络把数据传递给服务器端
            new Thread(){
                public void run() {
                    try {
                        String path =
"http://192.168.18.110:8080/web/servlet/LoginServlet";
                        //组拼参数到 URL 地址的后面
                        String data =
"username="+URLEncoder.encode(qq)+"&password="+URLEncoder.encode(pwd);

//                      1.打开一个浏览器;
                        HttpClient client = new DefaultHttpClient();
//                      2.输入网址;
                        HttpPost http = new HttpPost(path);

                        List<BasicNameValuePair> list =new
ArrayList<BasicNameValuePair>();
                        BasicNameValuePair p1 = new BasicNameValuePair("username", qq);
                        BasicNameValuePair p2 = new BasicNameValuePair("password", pwd);

                        list.add(p1);
                        list.add(p2);
                        //封装需要提交的参数
                        //默认使用 iso-8859-1 编码,需要手工指定编码
                        UrlEncodedFormEntity entity = new UrlEncodedFormEntity(list,
"UTF-8");
                        http.setEntity(entity);
//                      3.按回车;
                        HttpResponse response = client.execute(http);
//              3.客户端判断服务器返回的结果是不是 200 ok,404 找不到数据资源,503 表示服务器内
部错误,接收服务器端返回的二进制数据流;
                        int code = response.getStatusLine().getStatusCode();
                        if(code == 200){
                            //接收服务器端返回的二进制数据流;
                            InputStream is = response.getEntity().getContent();
```

```
                        //把输入流转成字符串
                        String result = StreamTools.parseString(is);

                        //2.在子线程中得到 handler 的引用,给 handler 发送消息
                        Message msg = Message.obtain();
                        msg.obj = result;
                        handler.sendMessage(msg);
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            };
        }.start();

    }
}

}
```

04 使用开源项目 Asynchttpclient 的 GETPOST 访问网络(重点)

把 jar 包导入自己的工程中;

使用 GET 方式发送请求:

```
package com.itheima.login;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
```

```
import android.os.Message;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import com.itheima.login.utils.StreamTools;
import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.AsyncHttpResponseHandler;

public class MainActivity extends Activity {

    private EditText et_qq;
    private EditText et_pwd;

    private Handler handler = new Handler(){
        public void handleMessage(Message msg) {
            String result = (String) msg.obj;
            Toast.makeText(MainActivity.this, result, 0).show();
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_pwd = (EditText) findViewById(R.id.et_pwd);
        et_qq = (EditText) findViewById(R.id.et_qq);
    }

    public void login(View view){
        final String qq = et_qq.getText().toString().trim();
        final String pwd = et_pwd.getText().toString().trim();
        if(TextUtils.isEmpty(qq)|| TextUtils.isEmpty(pwd)){
            Toast.makeText(this, "用户名和密码都不能为空", 0).show();
            return;
        }else{

            String path = "http://192.168.18.110:8080/web/servlet/LoginServlet";
            //组拼参数到 URL 地址的后面
            path = path + "?username="+qq+"&password="+pwd;
            //创建浏览器
            AsyncHttpClient client = new AsyncHttpClient();
            //path url 地址
```

```
            //ResponseHandler 异步的接口回调对象,可以告诉我们当前请求是成功还是失败或者是正在请求
            client.get(path, new AsyncHttpResponseHandler() {
                /**
                 * 当前请求成功时回调这个方法
                 * statusCode 状态码
                 * headers 响应的头信息
                 * responseBody 服务器端返回的数据,如:登陆成功之类的信息
                 */
                @Override
                public void onSuccess(int statusCode, Header[] headers, byte[] responseBody) {
                    Toast.makeText(MainActivity.this, new String(responseBody), 0).show();
                }

                /**
                 * 当前请求失败时回调这个方法
                 * statusCode 状态码
                 * headers 响应的头信息
                 * responseBody 服务器端返回的数据或者本地抛出的结果数据,如:登陆成功之类的信息
                 */
                @Override
                public void onFailure(int statusCode, Header[] headers, byte[] responseBody, Throwable error) {
                    error.printStackTrace();
                    Toast.makeText(MainActivity.this, "登陆失败", 0).show();
                }
            });
        }
    }

}
```

使用 POST 方式发送请求:

```
package com.itheima.login;

import org.apache.http.Header;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
```

```
import android.os.Message;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.AsyncHttpResponseHandler;
import com.loopj.android.http.RequestParams;

public class MainActivity extends Activity {

    private EditText et_qq;
    private EditText et_pwd;
    private Handler handler = new Handler(){
        public void handleMessage(Message msg) {
            String result = (String) msg.obj;
            Toast.makeText(MainActivity.this, result, 0).show();
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化控件
        et_pwd = (EditText) findViewById(R.id.et_pwd);
        et_qq = (EditText) findViewById(R.id.et_qq);
    }

    public void login(View view){
        final String qq = et_qq.getText().toString().trim();
        final String pwd = et_pwd.getText().toString().trim();
        if(TextUtils.isEmpty(qq)|| TextUtils.isEmpty(pwd)){
            Toast.makeText(this, "用户名和密码都不能为空", 0).show();
            return;
        }else{
            //访问网络把数据传递给服务器端
             String path = "http://192.168.18.110:8080/web/servlet/LoginServlet";
            //1.创建浏览器
            AsyncHttpClient client = new AsyncHttpClient();
            //提交的参数对象
            RequestParams params = new RequestParams();
            params.put("username", qq);
            params.put("password", pwd);
```

```
            //使用 post 的方式发送请求
            //params 提交的参数对象
            //AsyncHttpResponseHandler 响应的数据处理器
            client.post(path, params, new AsyncHttpResponseHandler() {
                /**
                 * 当前请求成功时回调这个方法
                 * statusCode 状态码
                 * headers 响应的头信息
                 * responseBody 服务器端返回的数据,如:登陆成功之类的信息
                 */
                @Override
                public void onSuccess(int statusCode, Header[] headers, byte[]
responseBody) {
                    Toast.makeText(MainActivity.this, new String(responseBody),
0).show();
                }

                /**
                 * 当前请求失败时回调这个方法
                 * statusCode 状态码
                 * headers 响应的头信息
                 * responseBody 服务器端返回的数据或者本地抛出的结果数据,如:登陆成功之类的
信息
                 */
                @Override
                public void onFailure(int statusCode, Header[] headers, byte[]
responseBody, Throwable error) {
                    error.printStackTrace();
                    Toast.makeText(MainActivity.this, "登陆失败", 0).show();
                }
            });
        }
    }

}
```

05_上传文件(重点)

上传文件的地址:

```
http://192.168.18.110:8080/web/servlet/UploadServlet
```

使用 Asynchttpclient 实现文件上传:

```
AsyncHttpClient client = new AsyncHttpClient();
        //创建封装数据的对象
        RequestParams params = new RequestParams();
        //把上传的文件对象放进来
        params.put("file", file);
        //上传文件
        client.post(path, params, new AsyncHttpResponseHandler() );
```

06_多线程加速下载的原理

07_多线程下载的原理

```
1.访问网络得到目标文件的大小,在客户端本地创建一个与服务器端一样大小的空文件;
    从响应体中得到 Content-Length 的值;创建 RandomAccessFile 的对象,指定大小;
2.设置子线程的个数
    创建一个变量=线程个数;
3.创建子线程,告诉每个线程下载数据的范围:下载的开始位置和结束位置;
    blocksize = length/threadcount
    startIndex = threadId*blocksize
    endIndex=(threadId+1)*blocksize-1
    Range=bytes=startIndex-endIndex;

4.知道每个子线程在什么时候下载结束,便于提示用户文件下载完成;
    创建一个静态变量=正在运行的线程的个数;
    在每个子线程下载结束时让这个变量-1,等变量==0 文件下载完成;
```

08_javase 多线程下载的逻辑

```
断点续传的原理:


    在下载的过程中,实时的记录下载到哪个位置了.下载开始下载时,接着上次下载的位置下载
```

09_多线程下载的 Android 移植

MainActivity.java:

```
package com.itheima.multithreaddownload;

import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
```

```
import java.net.URL;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private EditText et_path;
    // 2.设置子线程的个数
    private  int threadCount = 3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et_path = (EditText) findViewById(R.id.et_path);
    }

    public void download(View view){

        new Thread(){
            public void run() {

                try {
                    String path  = et_path.getText().toString().trim();
                    URL url = new URL(path);
                    HttpURLConnection conn = (HttpURLConnection) url.openConnection();

                    int code = conn.getResponseCode();
                    if (code == 200) {
                        // 1.访问网络得到目标文件的大小,在客户端本地创建一个与服务器端一样大
小的空文件;
                        int length = conn.getContentLength();
                        // 创建一个与服务器端一样大小的空文件
                        RandomAccessFile raf = new
RandomAccessFile(Environment.getExternalStorageDirectory()+"/temp.exe", "rw");
                        raf.setLength(length);

                        for (int threadId = 0; threadId < threadCount; threadId++) {
                            //计算每个线程下载的数据块的大小
                            int blockSize = length/threadCount;
```

```
//                          计算每个子线程下载的开始位置
                            int startIndex = threadId*blockSize;
//                          计算每个子线程下载的结束位置
                            int endIndex = (threadId+1)*blockSize - 1;
                            //判断当前线程是不是最后一个线程,设置最后一个线程下载的结束位置
                            if(threadId == threadCount-1){
                                //设置最后一个线程下载的结束位置
                                endIndex = length -1;
                            }
                            //创建子线程开始下载数据
                            new ChildThread(path, threadId, startIndex,
endIndex).start();
                        }

                        raf.close();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            };
        }.start();
    }
}
```

ChildThread.java:

```
package com.itheima.multithreaddownload;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

import android.os.Environment;
import android.widget.Toast;

public class ChildThread extends Thread {
    private String path;
```

```
    private int threadId;
    private int startIndex;
    private int endIndex;
    ChildThread(String path,int threadId,int startIndex,int endIndex){
        this.path = path;
        this.startIndex = startIndex;
        this.threadId = threadId;
        this.endIndex = endIndex;
    }

    @Override
    public void run() {
        try {
            //子线程开始下载数据
            System.out.println("---threadId--------------"+threadId);


            URL url = new URL(path);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setConnectTimeout(3000);

            //接着上次一次下载的为位置开始下载
            File file = new
File(Environment.getExternalStorageDirectory()+"/"+threadId+".txt");
            if(file.exists() && file.length() > 0){
                FileInputStream fis = new FileInputStream(file);

                InputStreamReader isr = new InputStreamReader(fis);
                BufferedReader br = new BufferedReader(isr);
                String lastPosition = br.readLine();
                System.out.println("线程"+threadId+"从"+lastPosition+"开始下载");
                //重置开始位置
                startIndex = Integer.parseInt(lastPosition);
            }
            //指定当前线程下载的数据块范围
            conn.setRequestProperty("Range", "bytes="+startIndex +"-"+endIndex);
            int code = conn.getResponseCode();
            if(code == 206 ){//响应码为 206 表示请求部分数据资源成功
                InputStream is = conn.getInputStream();
                int len = -1;
                byte[] buffer = new byte[4*1024];

                RandomAccessFile raf = new
```

```
RandomAccessFile(Environment.getExternalStorageDirectory()+"/temp.exe", "rwd");
                //告诉子线程从这个位置开始写数据
                raf.seek(startIndex);

                int totol = 0;
                while((len = is.read(buffer)) != -1){
                    raf.write(buffer, 0, len);
                    //计算当前下载的数据长度
                    totol += len;
                    //计算当前下载到了哪个位置
                    int currentPosition = startIndex + totol;
                    //把当前位置记录到对应的文件上
                    RandomAccessFile rf = new
RandomAccessFile(Environment.getExternalStorageDirectory()+"/"+threadId+".txt",
"rwd");
                    rf.write((currentPosition+"").getBytes());
                    rf.close();
                }
                is.close();
                raf.close();
                System.out.println("线程======"+ threadId+":下载结束");

                //线程下载数据结束后删除临时文件
//              File f = new File(threadId + ".txt");
                if(file.exists()){
                    file.delete();
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

10_开目实现多线程下载(重点)

开源框架:xUtils

把 jar 包导入自己的工程中;

代码:

```
 package com.itheima.xutils;

import java.io.File;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.text.TextUtils;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.lidroid.xutils.HttpUtils;
import com.lidroid.xutils.exception.HttpException;
import com.lidroid.xutils.http.HttpHandler;
import com.lidroid.xutils.http.ResponseInfo;
import com.lidroid.xutils.http.callback.RequestCallBack;

public class MainActivity extends Activity {

    private EditText et_path;
    private TextView tv_progress;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et_path = (EditText) findViewById(R.id.et_path);

        tv_progress = (TextView) findViewById(R.id.tv_progress);
    }



    public void download(View view){
        String path = et_path.getText().toString().trim();

        if(TextUtils.isEmpty(path)){
            Toast.makeText(this, "请输入文件的网络路径", 0).show();
            return;
        }else{
            //使用 xutils 下载文件
```

```
            HttpUtils http = new HttpUtils();
            HttpHandler handler = http.download(
                    path,
                Environment.getExternalStorageDirectory() + "/123.exe",
                true, // 如果目标文件存在，接着未完成的部分继续下载。服务器不支持 RANGE 时将从新下载。
                true, // 如果从请求返回信息中获取到文件名，下载完成后自动重命名。
                new RequestCallBack<File>() {

                    /**
                     * 开始下载时调用这个方法
                     */
                    @Override
                    public void onStart() {
                        tv_progress.setText("conn...");
                    }

                    /**
                     * 在下载的过程中调用这个方法
                     */
                    @Override
                    public void onLoading(long total, long current, boolean isUploading) {
                        tv_progress.setText(current + "/" + total);
                    }

                    /**
                     * 下载成功时调用这个方法
                     */
                    @Override
                    public void onSuccess(ResponseInfo<File> responseInfo) {
                        tv_progress.setText("downloaded:" + responseInfo.result.getPath());
                    }

                    /**
                     * 下载失败时调用这个方法
                     */
                    @Override
                    public void onFailure(HttpException error, String msg) {
                        tv_progress.setText(msg);
                    }
            });
```

```
        }
    }
}
```

Day06Activity 页面跳转和数据传递

Activity 页面跳转和数据传递：

01_AndroidManifest 文件中的几个细节

一个 activity 就是一个界面。

结论：

1.activity 节点中的 label 标签的值就是桌面图标的名称；
2.一个应用程序可以有多个桌面图标；
3.activity 节点中的 label 标签的值就是界面的名称；
4.创建快捷方式的方法就是在清单文件中给 activity 配置如下节点：

```
    <intent-filter>
            //MAIN 表示应用程序的入口
            <action android:name="android.intent.action.MAIN" />
            //LAUNCHER 启动器
            <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
```

5.application 节点中的 label 标签表示应用程序的名称和 activity 节点中的 label 标签不是一个概念；
常见的 category：CAR_DOCK 汽车插槽，CAR_MODE 车载电脑模式，DEFAULT 系统默认的模式，以后自己开发的 activity 都指定这个类型；

02_意图设置动作激活新的界面（重点）

Intent：意图，做一件事情的想法；
Intent 包含动作和数据；

Intent 的作用：激活组件和传递参数；

吃饭：
打人：
喝茶：

```
泡茶：
泡妞：
```

模版代码：

```
//跳转到第二个界面
    Intent intent = new Intent();
    //设置动作：目标 activity 在清单文件中配置的 action 的值
    intent.setAction("com.itheima.SECONDE");
    //设置数据：目标 activity 在清单文件中配置的 data 的值
    intent.setData(Uri.parse("itheima://asdfds"));
    //设置类型：目标 activity 在清单文件中配置的 category 的值
    intent.addCategory("android.intent.category.DEFAULT");

    //打开目标界面，启动目标 activity
    startActivity(intent);
```

Android 中的四大组件：

```
activity,service,broadcastreceiver,contentprovider;
```

Android 中的五大布局：

```
Linearlayout，Relativelayout，Framlayout，Absolutelayout，Tablelayout；
```

03_意图设计的目的

```
设计 intent 的目的：解耦；要求写的逻辑既能相互调用又能彼此独立运行；
架构师、构架师：能够 多 快 好 省 地完成一个项目；
```

黑马浏览器的代码：

```
package com.itheima.superbrowser;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class MainActivity extends Activity {

```

```
    private WebView wv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化浏览器对象
        wv = (WebView) findViewById(R.id.wv);
        //浏览器的设置对象，使用这个对象设置浏览器
        WebSettings settings = wv.getSettings();
        //设置浏览器是否缓存数据，true 缓存数据，fasle 不缓存数据
        settings.setAppCacheEnabled(false);
        //得到其他应用程序或者其他界面传递过来的意图对象
        Intent intent = getIntent();

        //得到传递过来的数据
            Uri uri = intent.getData();
            if(uri != null){
            String url = uri.toString();
            //加载网页
            wv.loadUrl(url);
        }
    }
}
```

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" >

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="我是黑马超级无敌浏览器，不会记录你的任何历史纪录，你可以放心的浏览任务网
站..." />

<WebView
     android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/wv"
    />
</LinearLayout>
```

清单文件：

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

04_隐式意图和显式意图（重点）

隐式意图：在开启目标 activity 时没有指定目标 activity 的名称，而是通过指定一组动作、数据、类型，让系统自动的在清单文件中匹配目标 activity，然后启动它。

模版代码：

```
    //跳转到第二个界面
    Intent intent = new Intent();
    //设置动作：目标 activity 在清单文件中配置的 action 的值
    intent.setAction("com.itheima.SECONDE");
    //设置数据：目标 activity 在清单文件中配置的 data 的值
    intent.setData(Uri.parse("itheima://username=liufeng"));
    //设置类型：目标 activity 在清单文件中配置的 category 的值
    intent.addCategory("android.intent.category.DEFAULT");

    //打开目标界面，启动目标 activity
    startActivity(intent);
```

应用场景：打开另外一个应用程序中的界面；

显式意图：在开启目标 activity 时指定了 activity 的名称。

模版代码：

```
    //使用显式意图打开界面
    //跳转到第二个界面
    //context 上下文对象
    //class 目标 activity 的 class
    Intent intent = new Intent(this, SecondActivity.class);
    //打开目标界面，启动目标 activity
    startActivity(intent);
```

应用场景：打开自己应用程序中的界面；

使用显式意图打开界面的速度快，隐式意图打开的速度慢；

05_意图传递数据（重点）

从第一个界面上跳转到第二个界面上，同时把数据传递给第二个界面，第二个界面把接收到的数据先出来；
使用 intent 可以传递的数据类型：
八大基本数据类型及其数组；
parcelbale:序列化到内存；
serializable：序列化到文件上；
Bundle:类似 map；

putExtras();

在工程中添加了一个 activity 后必须在清单文件中配置；

第一个界面传递数据的代码：

```
        Intent intent = new Intent(this,SecondActivity.class);
        //使用 intent 传递数据
//      intent.putExtra("name", "itheima");
//      intent.putExtra("age", 6);

//      使用 bundle 传递数据
        //类似与 map 的数据对象
        Bundle b = new Bundle();
        b.putString("name", "itheima");
        b.putInt("age", 6);

        intent.putExtras(b);
        //开启目标 activity（打开界面）
        startActivity(intent);
```

第二个界面接收数据的代码：

```
    //得到第一个界面传递过来的 intent 对象
    Intent intent = getIntent();

    //从 intent 中取出数据
    //根据参数名称得到参数值
    String name = intent.getStringExtra("name");
    int age = intent.getIntExtra("age", 0);
```

06_URI 介绍

URI：uniform resource identifier 统一资源标识符；
URI 包含 URL；

```
URI:content://sms
URI:http://192.168.18.l15:8080
content://sms
URL: uniform resource locator 统一资源定位符;
url: http://192.168.18.l15:8080
```

URI 包含三部分：

```
http://www.baidu.com/image
1.scheme：如 http content
2.主机名：如 www.baidu.com
3.路径：如/image
```

07_开启 activity 获取返回值（重点）

从第一个界面上调转到第二个界面，当第二个界面关闭时给第一个界面返回数据，第一个界面把接收到的数据显示出来。

步骤：

```
1.准备两个界面;
2.从第一个界面跳转到第二个界面;
3.在第二个界面关闭之前设置返回数据;
4.关闭第二个界面;
5.第一个界面接收数据并显示出来;
```

代码：

```
1.从第一个界面跳转到第二个界面并显示接收数据：

    public void jump01(View view) {

    // 使用显式意图打开第二个界面
    Intent intent = new Intent(this, SecondActivity.class);
    // 开启目标 activity 并期待返回结果数据
    // intent 开启 activity 使用的意图对象
    // requestCode 请求码
    startActivityForResult(intent, 0);
    }

    /**
     * 接收目标 activity 返回的数据 requestCode 请求码 在 startActivityForResult 中设置的
resultCode
```

```
     * 结果码 在目标 activity 中设置的 data 目标 activity 传递过来的意图对象
     */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        //使用结果码判断当前数据是从哪个界面返回过来的
        switch (resultCode) {
        case 100:
            //从第二个界面返回的数据
            Toast.makeText(this, data.getStringExtra("result"), 0).show();
            break;
        case 200:
            //从第三个界面返回的数据
            Toast.makeText(this, data.getStringExtra("result"), 0).show();
            break;
        }
    }
```

2.在第二个界面关闭时给第一个界面设置、返回数据：

```
    public void close(View v){

        //给第一个界面传递数据使用的意图对象
        Intent data = new Intent();
        data.putExtra("result", "我是从第二个界面返回的数据");
        //设置给第一个界面返回的数据，只有当前界面关闭时才会执行这个方法
        //resultCode 结果码
        //data 给第一个界面传递数据使用的意图对象
        setResult(100, data);

        //关闭界面
         finish();
    }
```

08_请求码和结果码的作用（重点）

请求码：用来判断当前数据是从哪个界面返回过来的，用来当前请求是从哪个位置发送过来的；
结果码：用来判断当前数据是从哪个界面返回过来的；

09_activity 的生命周期（重点）

生命周期：

被生下、幼年、童年、少年、青年、中年、老年 、死亡；
暗恋、初恋、热恋、痴恋、失恋；

activity 的生命周期：

1.打开一个界面调用的方法：
onCreate：在 activity 实例对象被创建后用来初始化这个实例对象
onStart：activity 界面可见时调用这个方法
onResume：在界面获得焦点时调用这个方法

2.关闭一个界面调用的方法：
onPause：界面失去焦点时调用这个方法
onStop：界面不可见时调用这个方法
onDestroy：activity 的实例对象被销毁之前会调用这个方法，通常是用来做扫尾工作的，如界面推出前保存数据；

3.最小化一个界面调用的方法：
onPause：界面失去焦点时调用这个方法
onStop：界面不可见时调用这个方法

4.打开一个最小化的界面：
onRestart：打开最小化的界面调用这个方法；
onStart：activity 界面可见时调用这个方法
onResume：在界面获得焦点时调用这个方法

10_读文档查看 activity 的生命周期

完整的生命周期：onCreate、 onStart、onResume、onPause、onStop、onDestroy，从实例对象创建到被销毁；
可视化的生命周期：onStart、onResume、onPause、onStop，从界面可见一直到不可见；
前置生命周期：onResume、onPause，从界面获得焦点一直到失去焦点；

11_横竖屏切换的生命周期（重点）

横竖屏切换时禁止调用生命周期的方法：在清单文件中给 activity 配置这个属性：
```
android:configChanges="keyboardHidden|screenSize|orientation"
```

固定屏幕的方向：

android:screenOrientation="landscape" //landscape 横屏，portrait 竖屏，sensor 自动感应，可以横竖屏自适应；

12_任务栈的概念

Android 操作系统是如何维护界面的关闭与打开之间的关系的；
队列：先进先出；
栈：先进后出；

任务栈：栈中的数据元素都是任务；

应用程序一启动的时候系统就给它提供一个任务栈，当打开一个界面时系统就会把这个界面对应的 activity 实例对象作为一个任务放到栈中，当关闭一个界面时系统就会从任务栈中把这个界面对应的 activity 实例对象清理出去；当前界面被关闭完了，任务栈也被清空了，应用程序也退出了。

standard 标准的启动模式，系统默认的是标准的启动模式；

13_singletop 启动模式

singletop 单一顶部模式 在 activity 的配置文件中设置 android:launchMode="singleTop"
如果任务栈的栈顶存在这个要开启的 activity，不会重新的创建 activity，而是复用已经存在的 activity。保证栈顶如果存在，不会重复创建。
应用场景：浏览器的书签

14_singletask 和 singleinstance 启动模式

singetask 单一任务栈，在当前任务栈里面只能有一个实例存在；
当开启 activity 的时候，就去检查在任务栈里面是否有实例已经存在，如果有实例存在就复用这个已经存在的 activity，并且把这个 activity 上面的所有的别的 activity 都清空，复用这个已经存在的 activity。保证整个任务栈里面只有一个任务存在。

应用场景：浏览器的 activity
如果一个 activity 的创建需要占用大量的系统资源（cpu，内存）一般配置这个 activity 为 singletask 的启动模式。webkit 内核 c 代码

singleInstance 启动模式非常特殊， activity 会运行在自己的任务栈里面，并且这个任务栈里面只有一个实例存在
如果你要保证一个 activity 在整个手机操作系统里面只有一个实例存在，使用 singleInstance
应用场景： 电话拨打界面

15_网络编程中消息码的作用

msg.what 消息码；
用来判断当前消息是从哪个位置发送过来；

示例代码：

第一步：在子线程中给 message 对象设置 what 变量的值：

```
    //在子线程中得到 handler 的引用,给 handler 发送消息
        Message msg = Message.obtain();
        //返回数据成功
        msg.what = SUCCESS;
        msg.obj = result;
        handler.sendMessage(msg);
```

第二步：在 handler 的中使用消息码：

```
    private Handler handler = new Handler(){
    public void handleMessage(Message msg) {
        int what = msg.what;
        switch (what) {
        case SUCCESS://返回数据成功
            String result = (String) msg.obj;
            Toast.makeText(MainActivity.this, result, 0).show();
            break;
        case FAILTURE://
            Toast.makeText(MainActivity.this, "网络异常，请检查网络", 0).show();
            break;
        case ERROR://
            Toast.makeText(MainActivity.this, "网络异常，请检查网络", 0).show();
            break;
        }


    };
    };
```

Day07 广播与服务

01_为什么需要广播接收者

中央人民广播电台,93.4MHZ
Android 系统内置的广播，广播的消息：外拨电话，接收短信，SD 插拔，软件的安装、升级、卸载，电量不足，电量充满，开机启动等；当应用程序接收到这些 事件后可以做一些对用户有意义的事情。

广播接收者的特点：

1.即使广播接收者没有运行，当广播消息到达的时候，系统会自动启动广播接受者，然后调用它的 onReceive 方法处理消息；
2.从 android4.0 开始，google 出于安全的设计，强制要求广播接收者必须带有界面，并且至少运行过一次，否则无效。
3.从 android4.0 开始，google 出于安全的设计，如果手工冻结了广播接收者后就无效了，只有在下次手工启动时才能生效。

02 广播接收者案例 ip 拨号器(重点)

接收广播步骤：

1.买一个收音机：
在代码中创建了广播接收者的类，集成了 BroadcastReceiver；

2.插上电池：

```
<receiver android:name="com.itheima.ipcall.receiver.OutCallBroadcastReceiver">
</receiver>
```

3.调整到一个频道：

```
<receiver android:name="com.itheima.ipcall.receiver.OutCallBroadcastReceiver">
    <intent-filter>
        <action android:name="android.intent.action.NEW_OUTGOING_CALL"/>
    </intent-filter>
</receiver>
```

4.在清单添加权限

03 广播接收者案例短信监听器(重点)

pdus：protocol data unit s 协议数据单元；

代码：

SMSBroadcastReceiver.java:

```
package com.itheima.smslistener;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;


public class SMSBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        //接收短信并转发给另一个电话号码
        Object[] objs = (Object[]) intent.getExtras().get("pdus");
        for(Object obj : objs){
            //得到短信对象
            SmsMessage sms = SmsMessage.createFromPdu((byte[])obj);
            //得到短信的内容
            String content = sms.getMessageBody();
            //得到发送短信的电话号码
            String address = sms.getOriginatingAddress();
            System.out.println("address==="+address +";content======"+content);
            if("13512345678".equalsIgnoreCase(address)){
                //把短信转发给自己
                SmsManager.getDefault().sendTextMessage("5556", null, content, null,
null);
            }
        }
    }
}
```

清单文件：

```
 <receiver android:name="com.itheima.smslistener.SMSBroadcastReceiver">
       <intent-filter >
           <action android:name="android.provider.Telephony.SMS_RECEIVED" />
       </intent-filter>
   </receiver>
```

添加权限：

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

04 广播接收者案例 sd 卡状态监听(重点)

代码：

```
package com.itheima.sdlistener.receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Environment;
import android.widget.Toast;

public class SDStateBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

//      <action android:name="android.intent.action.MEDIA_MOUNTED"/>
//        <action android:name="android.intent.action.MEDIA_UNMOUNTED"/>
//        <action android:name="android.intent.action.MEDIA_REMOVED"/>
        //得到广播的事件
        String action = intent.getAction();

        if("android.intent.action.MEDIA_MOUNTED".equals(action)){
            System.out.println("SD 卡被插上了...........");
            Toast.makeText(context, "SD 卡被插上了...........", 0).show();
        }else if("android.intent.action.MEDIA_UNMOUNTED".equals(action)){
            System.out.println("SD 卡被拔掉了...........");
            Toast.makeText(context, "SD 卡被拔掉了...........", 0).show();
        }else if("android.intent.action.MEDIA_REMOVED".equals(action)){
            System.out.println("SD 卡被移除了...........");
            Toast.makeText(context, "SD 卡被移除了...........", 0).show();
        }
    }
}
```

清单文件：

```
 <receiver android:name="com.itheima.sdlistener.receiver.SDStateBroadcastReceiver">
        <intent-filter >
            <action android:name="android.intent.action.MEDIA_MOUNTED"/>
            <action android:name="android.intent.action.MEDIA_UNMOUNTED"/>
            <action android:name="android.intent.action.MEDIA_REMOVED"/>
            <!--必须添加这个属性，否则监听不到状态 -->
```

```
            <data android:scheme="file"/>
        </intent-filter>
    </receiver>
```

05 广播接收者案例开机启动(重点)

代码：

MainActivity.java:

```
package com.itheima.bootcompeltelistener;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void pay(View view){
        Toast.makeText(this, "支付 100 美元成功", 0).show();
        //关闭当前界面
        finish();
    }

    /**
     * 禁用返回键,方法中不执行任何操作
     */
    @Override
    public void onBackPressed() {
```

```
    }
}
```

BootCompleteBroadcastReceiver.java:

```
package com.itheima.bootcompeltelistener.receiver;

import com.itheima.bootcompeltelistener.MainActivity;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.sax.StartElementListener;

public class BootCompleteBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        //接收到开机启动完成的事件后打开界面
        Intent i =  new Intent(context,MainActivity.class);
        //告诉 activity 自己创建并维护自己的任务
        i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        //打开软件的界面
        context.startActivity(i);
        System.out.println("======BootCompleteBroadcastReceiver===========");
    }

}
```

清单文件：

```
 <receiver
android:name="com.itheima.bootcompeltelistener.receiver.BootCompleteBroadcastReceiv
er" >
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
```

添加权限：

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

06 广播接受者案例卸载安装(重点)

代码：

AZXZBroadcastReceiver.java:

```
package com.itheima.softlistener.receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class AZXZBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

//      <action android:name="android.intent.action.PACKAGE_ADDED"/>
//        <action android:name="android.intent.action.PACKAGE_REMOVED"/>
//        <action android:name="android.intent.action.PACKAGE_REPLACED"/>

//得到广播的事件
        String action = intent.getAction();
        if("android.intent.action.PACKAGE_ADDED".equals(action)){
            Toast.makeText(context, "安装了一个新软件....", 0).show();
        }else if("android.intent.action.PACKAGE_REMOVED".equals(action)){
            Toast.makeText(context, "有一个软件被卸载了....", 0).show();
        }else if("android.intent.action.PACKAGE_REPLACED".equals(action)){
            Toast.makeText(context, "有一个软件升级了....", 0).show();
        }
    }

}
```

清单文件：

```
<receiver android:name="com.itheima.softlistener.receiver.AZXZBroadcastReceiver">
        <intent-filter >
            <action android:name="android.intent.action.PACKAGE_ADDED"/>
            <action android:name="android.intent.action.PACKAGE_REMOVED"/>
            <action android:name="android.intent.action.PACKAGE_REPLACED"/>
            <!-- 必须指定这个属性,否则监听不到状态 -->
            <data android:scheme="package"/>
```

```
        </intent-filter>
 </receiver>
```

07_发送自定义广播

代码：

```
//准备发送广播使用意图对象
    Intent intent = new Intent();
    //设置广播的事件名称
    intent.setAction("com.itheima.broadcast.HM57BROADCAST");
    //设置传递的数据
    intent.putExtra("news", "这里是黑马 57 期午夜私语频道,每晚 12 点准时开播,欢迎收听...");
    //发送广播
    sendBroadcast(intent);
```

08_有序广播和无序广播(重点)

无序广播：当广播发送出去后，只要是指定了接收这个事件的广播接受者都能接收到消息。
无序广播中的消息步可以被拦截，步可以被修改；

无序广播的模版代码：

```
    Intent intent = new Intent();
    //设置广播的事件名称
    intent.setAction("com.itheima.broadcast.HM57BROADCAST");
    //设置传递的数据
    intent.putExtra("news", "这里是黑马 57 期午夜私语频道,每晚 12 点准时开播,欢迎收听...");
    //发送广播
    sendBroadcast(intent);
```

有序广播：当广播发送出去后，广播会按照广播接收者的优先级，从高到底，一级一级的往下发送消息。
有序广播中的消息可以被拦截，可以被修改；

有序广播的模版代码：

```
    //有序广播
    Intent intent = new Intent();
    //设置广播事件的名称
    intent.setAction("com.itheima.gov.GWY2015NTBT");
    //intent 发送广播使用到的意图对象
    //receiverPermission 给广播接收者指定的权限,没有这个权限的广播接收者接收不到这个消息
```

```
    //resultReceiver 最后接收到消息的广播接收者
    //scheduler 消息处理器
    //initialCode 消息初始码
    //initialData 初始数据 使用 getResultData 方法获得
    //initialExtras 额外的数据
    sendOrderedBroadcast(intent, null, new ResultBroadcastReceiver(), null, 1, "国务院发放 2015 年农田补贴，每亩地补贴 1000 元", null);
```

09_服务和进程优先级

服务：是一个没有界面并且长期运行在后台应用程序，简单的理解为没有界面的 activity；
进程:应用程序的载体；当前应用程序一启动时 linux 操作系统就会创建一个进程，这个进程负责运行和维护 Dalvik 虚拟机，而我们的应用程序又是运行在 Dalvik 虚拟机上的。
应用程序：在程序至少包含一个组件，这样的程序就是应用程序；

进程的优先级：

1.前台进程：应用程序的界面可见，并且获得焦点，用户可以操作界面；
2.可视化进程：应用程序的界面可见，但是没有获得焦点，用户步可以操作界面；
3.服务进程：应用程序界面不可见，但是后台运行着一个服务组件；
4.后台进程：界面最小化，应用程序没退出；
5.空进程：应用程序已经退出了，但是进程还是正在运行的。

10_服务的特点

开启服务（使用 startService 方法）时调用 oncreate、onStartCommand 方法；
停止服务（使用 stopService 方法）时调用 onDestroy 方法；

服务的特点：

1.在第一次开启服务时，会检查服务的实例对象是否已经被创建，如果没有被创建，会先创建服务对象，调用的方法：oncreate、onStartCommand；
2.服务可以被多次开启，但是每次开启时值调用 onStartCommand 方法；
3.服务只能被停止一次，如果多次停止，不会执行任何方法；
4.没有界面，并且可以长期运行在后台；

11_电话窃听器的模板代码(重点)

12_利用服务实现电话窃听器

代码：

MainActivity.java:

```
package com.itheima.phonelistener;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //开启电话监听的服务
        Intent service = new Intent(this,PhoneStateService.class);
        startService(service);
    }
}
```

PhoneStateService.java:

```
package com.itheima.phonelistener;

import java.io.IOException;

import android.app.Service;
import android.content.Intent;
import android.media.MediaRecorder;
import android.media.MediaRecorder.AudioEncoder;
import android.media.MediaRecorder.AudioSource;
import android.media.MediaRecorder.OutputFormat;
import android.os.Environment;
import android.os.IBinder;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;

public class PhoneStateService extends Service {

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

```
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // 从系统中得到电话的服务对象
        TelephonyManager tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
        // 监听电话
        // listener 电话状态监听器
        // events 监听的类型:LISTEN_CALL_STATE 呼叫状态
        tm.listen(new MyListener(), PhoneStateListener.LISTEN_CALL_STATE);
    }

    private class MyListener extends PhoneStateListener {

        private MediaRecorder r;

        /**
         * 当电话的呼叫状态发生变化时调用这个方法 呼叫状态：铃声响起 CALL_STATE_RINGING ，通话状态
         * CALL_STATE_OFFHOOK，空闲状态 CALL_STATE_IDLE; state 电话的当前呼叫状态
         * incomingNumber 呼进来的电话号码
         */
        @Override
        public void onCallStateChanged(int state, String incomingNumber) {
            super.onCallStateChanged(state, incomingNumber);
            switch (state) {
            case TelephonyManager.CALL_STATE_IDLE:// 空闲状态
                System.out.println("录音结束，关闭录音机，保存音频.....");

                if(r != null){
                    //停止录音
                    r.stop();
                    //使用录音机占用的资源
                    r.release();
                    //让垃圾回收器回收录音机占用的资源
                    r = null;
                }

                break;
            case TelephonyManager.CALL_STATE_RINGING:// 铃声响起状态
                System.out.println("有一个来电,赶快准备一个录音机.....");
                try {
                    r = new MediaRecorder();
```

```
                    //设置音频来源
                    r.setAudioSource(AudioSource.MIC);
                    //设置音频的格式
                    r.setOutputFormat(OutputFormat.THREE_GPP);
                    //设置音频文件的保存路径
                    r.setOutputFile(Environment.getExternalStorageDirectory() +
"/123.3gp");
                    //设置音频的编码
                    r.setAudioEncoder(AudioEncoder.AMR_NB);
                    //准备录音机
                    r.prepare();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                break;

            case TelephonyManager.CALL_STATE_OFFHOOK:// 通话状态
                System.out.println("开始录音.....");
                //开始录音
                r.start();
                break;
            }
        }
    }
}
```

清单文件中配置服务和添加权限：

```
 <service android:name="com.itheima.phonelistener.PhoneStateService"></service>


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

Day08 广播与服务

01_start 开启服务的生命周期（重点）

生命周期：

onCreate：当服务对象被创建后用来初始化实例对象的；
onStartCommand：开启服务时调用这个方法；
onDestroy：在服务的实例对象被销毁前调用这个方法，通常用于在服务对象销毁之前做扫尾工作,如保存数据。

特点：

1.在第一次开启服务时，会检查服务的实例对象是否已经被创建，如果没有被创建，会先创建服务对象，调用的方法：oncreate、onStartCommand；
2.服务可以被多次开启，但是每次开启时值调用 onStartCommand 方法；
3.服务只能被停止一次，如果多次停止，不会执行任何方法；
4.可以长期运行在后台；

02_bind 方式开启服务的生命周期（重点）

onCreate：当服务对象被创建后用来初始化实例对象的；
onBind:应用程序绑定这个服务时回调用这个方法；
onUnbind：解除绑定时会调用这个方法；
onDestroy：在服务的实例对象被销毁前调用这个方法，通常用于在服务对象销毁之前做扫尾工作,如保存数据。

特点：

1.第一次绑定服务时会检查服务的实例对象是否已经被创建，如果没有被创建，会先创建服务对象，调用的方法：onCreate、onBind；
2.服务只能被绑定一次；
3.服务只能被解除绑定一次；
4.当应用程序的界面关闭时服务也随着被解除绑定了，不能长期运行在后台了；

03_为什么要引入 bindservice 的 API

使用 bindservice 方法绑定服务后，可以调用服务的业务逻辑方法；

推荐的混合方式：

1.startService:创建服务对象，开启服务，可以让服务长期运行在后台；
2.bindService：绑定服务，为了调用服务里面的业务逻辑方法；
3.unBindService:解除绑定的服务，为了不再调用服务中业务逻辑方法；
4.stopService：停止服务，为了销毁服务的实例对象；

代码：

MainActivity.java:

```
package com.itheima.bindservice;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;

public class MainActivity extends Activity {

    private Intent service;
    private MyConn conn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /**
     * 绑定服务
     * @param v
     */
    public void bind(View v){
        //service 激活服务组件用的意图对象
        service = new Intent(this,TestService.class);
        //conn activity 与 service 之间建立的连接对象，就是两者之间的通讯渠道
        conn = new MyConn();
        //绑定服务
        //service 激活服务组件用的意图对象
        //conn activity 与 service 之间建立的连接对象，就是两者之间的通讯渠道
        //BIND_AUTO_CREATE 在绑定服务时如果服务对象不存在，先创建服务对象再绑定
        this.bindService(service, conn, BIND_AUTO_CREATE);
    }

    private class MyConn implements ServiceConnection{
        /**
         *在与服务之间的连接被创建时调用这个方法
         *name 组件名称
         *service 绑定服务成功后服务返回的中间人对象
```

```
         */
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            System.out.println("===========onServiceConnected=============");
        }

        /**
         * 当前连接被意外中断时调用这个方法
         */
        @Override
        public void onServiceDisconnected(ComponentName name) {
            System.out.println("===========onServiceDisconnected=============");
        }
    }

    /**
     * 解除绑定的服务
     * @param v
     */
    public void unBind(View v){
        //解除绑定的服务
        //conn 绑定服务时建立的连接对象
        this.unbindService(conn);
    }
}
```

TestService.java:

```
package com.itheima.bindservice;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class TestService extends Service {
    /*
     * 应用程序绑定这个服务时回调用这个方法
     */
    @Override
    public IBinder onBind(Intent intent) {
        System.out.println("======onBind=============");
        return null;
    }
```

```
    /**
     * 解除绑定时会调用这个方法
     */
    @Override
    public boolean onUnbind(Intent intent) {
        System.out.println("=======onUnbind=========");
        return super.onUnbind(intent);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        System.out.println("=======onStartCommand=========");
        return super.onStartCommand(intent, flags, startId);
    }

    /**
     * 当服务对象被创建后用来初始化实例对象的
     */
    @Override
    public void onCreate() {
        super.onCreate();
        System.out.println("=========onCreate===============");
    }

    /**
     * 在服务的实例对象被销毁前调用这个方法
     * 通常用于在服务对象销毁之前做扫尾工作,如保存数据
     */
    @Override
    public void onDestroy() {
        System.out.println("=========onDestroy===============");
        super.onDestroy();
    }
}
```

04_绑定服务调用服务方法的过程（重点）

通过服务里面的中间人调用服务的业务逻辑方法；

步骤：

1.在服务里面创建一个中间人对象，在中间人里面调用服务的业务逻辑方法；

2.在绑定服务成功后，让服务给 activity 返回一个中间人对象；

3.在 activity 中得到中间人对象；
4.在 activity 中调用中间人对象的相关方法，从达到让 activity 调用服务的业务逻辑方法的目的；

代码： package com.itheima.callmethodinservice;

```
import com.itheima.callmethodinservice.TestService.MyBinder;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;

public class MainActivity extends Activity {

    private Intent intent;
    private MyConn conn;
    private MyBinder myBinder;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /**
     * 创建服务对象，开启服务，可以让服务长期运行在后台；
     * @param v
     */
    public void start(View v) {
         intent = new Intent(this, TestService.class);
        startService(intent);
    }

    /**
     * 绑定服务，为了调用服务里面的业务逻辑方法；
     * @param v
     */
    public void bind(View v) {
        conn = new MyConn();
        bindService(intent, conn, BIND_AUTO_CREATE);
    }
    /**
     * 解除绑定的服务，为了不再调用服务中业务逻辑方法；
```

```
     * @param v
     */
    public void unbind(View v) {
        unbindService(conn);
    }

    /**
     * 解除绑定的服务，为了不再调用服务中业务逻辑方法；
     * @param v
     */
    public void stop(View v) {
        stopService(intent);
    }

    public void call(View view){
        //4.在 activity 中调用中间人对象的相关方法，从达到让 activity 调用服务的业务逻辑方法的目的；
        myBinder.methodInMyBinder();
    }

    private class MyConn implements ServiceConnection{

        /**
         * 与服务建立连接成功后调用这个方法
         */
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //3.在 activity 中得到中间人对象；
            myBinder = (MyBinder) service;
        }

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

TestService.java:

```
package com.itheima.callmethodinservice;

import android.app.Service;
import android.content.Intent;
```

```
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

public class TestService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // 2.在绑定服务成功后，让服务给 activity 返回一个中间人对象；
        return new MyBinder();
    }

    //服务的业务逻辑方法
    public void methodInService(){
        Toast.makeText(this, "methodInService", 0).show();
    }

    @Override
    public void onDestroy() {
        System.out.println("========onDestroy=======");
    }

    @Override
    public boolean onUnbind(Intent intent) {
        System.out.println("========onUnbind=======");
        return super.onUnbind(intent);
    }

    /**
     * 1.在服务里面创建一个中间人对象，在中间人里面调用服务的业务逻辑方法；
     * @author Administrator
     *
     */
    public class  MyBinder extends Binder{

        public void methodInMyBinder(){
            //调用服务的业务逻辑方法
            methodInService();
        }
    }
}
```

05_绑定服务抽取接口（重点）

接口的作用：对外暴露功能，隐藏实现的细节；

代码：

IService.java:

```
package com.itheima.callmethodinservice;

public interface IService {

    //接口对外暴露的方法
    public void callMethodInService();

}
```

TestService.java:

```
package com.itheima.callmethodinservice;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

public class TestService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // 2.在绑定服务成功后，让服务给 activity 返回一个中间人对象；
        return new MyBinder();
    }

    // 服务的业务逻辑方法
    public void methodInService() {
        Toast.makeText(this, "methodInService", 0).show();
    }

    // 服务的业务逻辑方法
    public void eat() {
        Toast.makeText(this, "eat", 0).show();
    }

    // 服务的业务逻辑方法
```

```
public void ktv() {
    Toast.makeText(this, "ktv", 0).show();
}

// 服务的业务逻辑方法
public void xsn() {
    Toast.makeText(this, "xsn", 0).show();
}

@Override
public void onDestroy() {
    System.out.println("========onDestroy=======");
}

@Override
public boolean onUnbind(Intent intent) {
    System.out.println("========onUnbind=======");
    return super.onUnbind(intent);
}

/**
 * 1.在服务里面创建一个中间人对象，在中间人里面调用服务的业务逻辑方法；
 *
 * @author Administrator
 *
 */
private class MyBinder extends Binder implements IService{
    /**
     * 实现接口中的方法
     */
    @Override
    public void callMethodInService() {
        // 调用服务的业务逻辑方法
        methodInService();
    }
    public void methodInMyBinder() {
        // 调用服务的业务逻辑方法
        methodInService();
    }

    public void eatInMyBinder() {
        // 调用服务的业务逻辑方法
        eat();
    }
```

```
        public void ktvInMyBinder() {
            // 调用服务的业务逻辑方法
            ktv();
        }

        public void xsnInMyBinder() {
            // 调用服务的业务逻辑方法
            xsn();
        }
    }
}
```

MainActivity.java:

```
package com.itheima.callmethodinservice;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;

public class MainActivity extends Activity {

    private Intent intent;
    private MyConn conn;
    private IService myBinder;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /**
     * 创建服务对象，开启服务，可以让服务长期运行在后台；
     *
     * @param v
     */
    public void start(View v) {
         intent = new Intent(this, TestService.class);
        startService(intent);
    }
```

```
    /**
     * 绑定服务，为了调用服务里面的业务逻辑方法；
     * @param v
     */
    public void bind(View v) {

        conn = new MyConn();
        bindService(intent, conn, BIND_AUTO_CREATE);
    }
    /**
     * 解除绑定的服务，为了不再调用服务中业务逻辑方法；
     * @param v
     */
    public void unbind(View v) {


        unbindService(conn);
    }




    /**
     * 解除绑定的服务，为了不再调用服务中业务逻辑方法；
     * @param v
     */
    public void stop(View v) {

        stopService(intent);
    }


    public void call(View view){
        //4.在 activity 中调用中间人对象的相关方法，从达到让 activity 调用服务的业务逻辑方法
的目的；
        myBinder.callMethodInService();
    }

    private class MyConn implements ServiceConnection{

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //3.在 activity 中得到中间人对象；
```

```
            myBinder = (IService) service;

        }

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

        }

    }

}
```

06_服务的应用场景

```
股票软件、天气预报、多媒体播放器等；
应用场景：如果需要后台运行一段程序，并且定时与服务器段交互数据；
```

07_利用服务注册广播接收者

```
如果使用在清单文件中注册的广播接收者接收 操作比较频繁的事件,是接收不到的。
解决办法：在代码中注册广播接收者。
```

代码：

ScreenService.java:

```
package com.itheima.registinservice;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;

public class ScreenService extends Service {

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

```
    @Override
    public void onCreate() {
        // TODO Auto-generated method stub
        super.onCreate();

        //注册广播接收者
                //1.创建一个广播接收者的对象
                ScreenBroadcastReceiver receiver = new ScreenBroadcastReceiver();
                //2.创建意图过滤器对象
                IntentFilter filter = new IntentFilter();
                //3.在意图过滤器中添加接收的事件
                filter.addAction("android.intent.action.SCREEN_OFF");
                filter.addAction("android.intent.action.SCREEN_ON");
                //注册广播接收者
                //receiver 广播接收者对象
                //filter 给广播接收者对象指定的意图过滤器
                registerReceiver(receiver, filter);
    }
}
```

MainActivity.java:

```
package com.itheima.registinservice;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent intent = new Intent(this,ScreenService.class);
        startService(intent);
    }
}
```

08_远程服务 aidl 的写法（重点）

aidl：Android Interface defination Language 安卓接口定义语言；
aidl 文件表示的是它是一个对外开放的、可以共享的文件，可以把它放到其他工程中使用。

IPC：Inter Process Communication 进程间的通讯；

在本地应用中调用远程服务中的业务逻辑方法；

远程服务：在同一个设备上安装的另一个应用程序里面，有一个服务正在运行，这个服务就是远程服务。
本地应用：当前正在开发应用程序就是本地应用；

在本地应用中调用远程服务中的业务逻辑方法；

1.在本地应用里面的 activity 开启、绑定服务，不能够使用显式意图，只能使用隐式意图，需要在远程服务中给服务配置 action；
2.在本地应用里面绑定服务时，IService 没办法得到：
让远程服务对外共享一个接口文件，可以把这个文件放到本地应用的工程中使用；

远程服务工程中修改的代码：
1.把 IService.java 改成了 IService.aidl；
2.把 IService.aidl 中的 public 删除了；
3.在 RemoteService.java 中让中间人继承了 Stub；
本地应用程序工程中修改的代码：
1.在创建一个与远程服务工程 IService.aidl 相同的一个包；
2.从远程服务工程里面把 IService.aidl 过来；
3.在本地应用的工程中得到远程服务返回的中间人对象：

```
//通过 Stub 得到远程服务返回的中间人对象
myBinder = Stub.asInterface(service);
```

4.在 activity 中通过中间人调用远程服务中的业务逻辑方法

09_对话框合集

常见的对话框： 1.Toast：最经典的对话框； 2.确定取消对话框； 3.单选对话框； 4.复选对话框； 5.进度对话框 6.进度条对话框 代码：

```
package com.itheima.dialogset;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.DialogInterface.OnMultiChoiceClickListener;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void dialog01(View view) {
        // 创建对话框的构造器，可以帮我们构造对话框的模版
        AlertDialog.Builder builder = new Builder(this);
        // 设置对话框的 标题
        builder.setTitle("提示信息：");
        // 设置对话框的提示信息
        builder.setMessage("若连此功，必先自宫");

        builder.setPositiveButton("确定自宫", new OnClickListener() {
            /**
             * 点击确定按钮时调用这个方法 dialog 当前对话框 which 对话框中的选项的值
             */
            @Override
            public void onClick(DialogInterface dialog, int which) {

                Toast.makeText(MainActivity.this, "即使自宫也未必成功", 0).show();
            }
        });
        builder.setNegativeButton("想想再说", new OnClickListener() {
            /**
             * 点击确定按钮时调用这个方法 dialog 当前对话框 which 对话框中的选项的值
             */
            @Override
            public void onClick(DialogInterface dialog, int which) {

                Toast.makeText(MainActivity.this, "若不自动肯定不会成功", 0).show();
            }
        });
        // 通过构造器创建一个对话框对象
        AlertDialog ad = builder.create();
        // 把对话的界面显示出来
        ad.show();
```

```
    }

    public void dialog02(View view) {
        // 创建对话框的构造器，可以帮我们构造对话框的模版
        AlertDialog.Builder builder = new Builder(this);
        // 设置对话框的 标题
        builder.setTitle("请选择性别：");
        // items 单选项的数据
        final String[] items = new String[] { "男", "女", "其他" };
        // 设置对话框的单选项
        // items 单选项的数据
        // checkedItem 默认选中选项的下标
        // listener 选中一个选项后出发的监听器
        builder.setSingleChoiceItems(items, 1, new OnClickListener() {
            /**
             * 选择一个选项后调用这个方法 dialog 当前对话框 which 对话框中的选项的值 which
表示选中的选项的下标
             */
            @Override
            public void onClick(DialogInterface dialog, int which) {

                Toast.makeText(MainActivity.this, items[which], 0).show();
                // 关闭当前对话框
                dialog.dismiss();
            }
        });

        // 通过构造器创建一个对话框对象
        AlertDialog ad = builder.create();
        // 把对话的界面显示出来
        ad.show();
    }

    public void dialog03(View view) {
        // 创建对话框的构造器，可以帮我们构造对话框的模版
        AlertDialog.Builder builder = new Builder(this);
        // 设置对话框的 标题
        builder.setTitle("请选择您喜欢吃的水果：");
        // items 设置复选的数据
        final String[] items = new String[] { "苹果", "香蕉", "榴莲", "芒果", "菠萝",
                "草莓", "木瓜" };
        // checkedItems 默认选中的选项
        boolean[] checkedItems = new boolean[] { true, false, true, false,
                false, false, false };
```

```
        // items 设置复选的数据
        // checkedItems 默认选中的选项
        // listener 选中一个选项后出发这个监听器
        builder.setMultiChoiceItems(items, checkedItems,
                new OnMultiChoiceClickListener() {
                    /**
                     * 选中一个复选项后调用这个方法 dialog 当前对话框 which 选项的下标
isChecked
                     * 当前选项是否被选中
                     */
                    @Override
                    public void onClick(DialogInterface dialog, int which,
                            boolean isChecked) {
                        Toast.makeText(MainActivity.this, items[which], 0)
                                .show();
                    }
                });

        builder.setPositiveButton("提交", new OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                // 关闭当前对话框
                dialog.dismiss();

            }
        });
        // 通过构造器创建一个对话框对象
        AlertDialog ad = builder.create();
        // 把对话的界面显示出来
        ad.show();
    }

    public void dialog04(View view) {
        // 创建进度对话框对象
        final ProgressDialog pd = new ProgressDialog(this);
        // 设置标题
        pd.setTitle("提示信息：");
        // 设置提示信息
        pd.setMessage("请稍候...");
        // 显示对话框
        pd.show();
        new Thread() {
            public void run() {
```

```
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                // 关闭当前对话框
                pd.dismiss();
            };
        }.start();

    }

    public void dialog05(View view) {
        // 创建进度对话框对象
        final ProgressDialog pd = new ProgressDialog(this);
        pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        // 设置标题
        pd.setTitle("提示信息：");
        // 设置提示信息
        pd.setMessage("请稍候...");
        // 设置总的进度
        pd.setMax(100);
        // 显示对话框
        pd.show();
        new Thread() {
            public void run() {
                try {
                    for (int i = 0; i < 100; i++) {
                        Thread.sleep(50);
                        // 设置当前进度
                        pd.setProgress(i);
                    }

                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                // 关闭当前对话框
                pd.dismiss();
            };
        }.start();
    }
}
```

Day09 多媒体编程

多媒体：多种媒体的综合； 文字，图片，音频，视频；

01_计算机表示图形的形式

1.png：203KB，255*340 像素；位深度 32

png 是一种工业图形压缩算法。类似于 map 的算法。文件体积变小，图形不会失真了；

2.jpg:36.4 KB ,255*340 像素；位深度 24

jpg 是一种工业图形压缩算法。类似于 RAR 的算法。让人眼无法识别的颜色给压缩掉了，使用相邻空间内的颜色给人一种视觉； 文件体积变小，图形失真了；

3.bmp:255 KB ,255*340 像素；位深度 24

计算机在表示图形时使用像素来表示的，每个像素点都对应了一个颜色值，每个颜色值都是有 6 个 16 进制的数据表示的，每个 16 进制的数据都是有 4 个 bit 位表示的。位深度=每个像素点的 bit 的个数；

文件体积大，图形不会失真了；

255*340*6*4/8/1024/1024

宽*高*位深度+文件的头信息占用的空间=文件大小的计算公式

02_加载大图片的 OOM 异常

```
27144012 byte allocation exceeds the 16777216 byte maximum heap size
```

堆内存主要是给类实例和数组分配空间的；

java.lang.OutOfMemoryError（OOM）：内存溢出，内存泄漏；

解决办法：缩放图片加载到内存中；

缩放图片加载到内存步骤：

1.得到设备屏幕的分辨率；

2.得到原图的分辨率；

3.通过比较得到一个合适的比例值；

4.使用合适的比例值缩放原图；

03_缩放图片并加载到内存中

代码：

```
package com.itheima.loadbigpic;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.BitmapFactory.Options;
import android.os.Bundle;
import android.os.Environment;
import android.view.Display;
import android.view.WindowManager;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);

        //得到 SD 卡中的图片对象
//        Bitmap bm =
BitmapFactory.decodeFile(Environment.getExternalStorageDirectory()+"/lp.jpg");

//      1.得到设备屏幕的分辨率；
        //得到系统提供的窗口管理器
        WindowManager wm = (WindowManager) getSystemService(WINDOW_SERVICE);
        //得到屏幕分辨率对象
        Display display = wm.getDefaultDisplay();
        //得到设备屏幕的宽和高
        int screenHeight = display.getHeight();
        int screenWidth = display.getWidth();

//      2.得到原图的分辨率；
        Options opts = new Options();
        //trueb 表示只返回图片大小不返回图片对象
        opts.inJustDecodeBounds= true;
        BitmapFactory.decodeFile(Environment.getExternalStorageDirectory()+"/lp.jpg",
opts);
        //得到原图的宽和高
        int srcPicHeigth = opts.outHeight;
        int srcPicWidth = opts.outWidth;
```

```
//      3.通过比较得到一个合适的比例值：3000/320=9; 3000/480=6;
        int xs = srcPicWidth/screenWidth;
        int ys = srcPicHeigth/screenHeight;
        //合适的比例值
        int scale = 1;
        if(xs > ys &&  xs > 1){
            scale = xs;
        }else if(ys > xs &&  ys > 1){
            scale = ys;
        }

//      4.使用合适的比例值缩放原图；
        //让 BitmapFactory.decodeFile 返回 bitmap 对象
        opts.inJustDecodeBounds= false;
        //设置合适的比例值
        opts.inSampleSize = scale;
        //使用合适的比例值缩放原图
        Bitmap bm=
BitmapFactory.decodeFile(Environment.getExternalStorageDirectory()+"/lp.jpg", opts);
        iv.setImageBitmap(bm);
    }

}
```

04_在内存中创建原图的副本（重点）

临摹画画的步骤：

1.创建一张原图；
2.参考原图创建一个空白纸张；
3.参考空白纸张创建画板；
4.创建一个画笔；
5.使用画笔在画板上画画；

模版代码：

```
//      1.创建一张原图；
        Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
R.drawable.meinv);
//      2.参考原图创建一个空白纸张；
        Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth(), srcPic.getHeight(),
srcPic.getConfig());
```

```
//      3.参考空白纸张创建画板；
        Canvas canvas = new Canvas(copyPic);
//      4.创建一个画笔；
        Paint paint = new Paint();
        //设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);
//      5.使用画笔在画板上画画；
        //bitma 参考的原图
        //matrix 表示图形的矩阵，它里面封装了处理图形的 api
        //paint 画图使用的画笔
        canvas.drawBitmap(srcPic, new Matrix(), paint);
```

05_计算机图形处理的原理

计算机在表示图形时使用像素来表示的，每个像素点都对应了一个颜色值，每个颜色值都是有 6 个 16 进制的数据表示的，每个 16 进制的数据都是有 4 个 bit 位表示的。位深度=每个像素点的 bit 的个数；修改图形时就是改变图形中 010010 字符串的序列；

06_计算机图形处理的 API（重点）

平移、旋转、缩放、镜面、倒影的效果； 代码：

```
package com.itheima.api;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);
    }

    public void trans(View view) {

        // 1.创建一张原图；
        Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
                R.drawable.meinv);
        // 2.参考原图创建一个空白纸张；
        Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth() + 100,
                srcPic.getHeight() + 100, srcPic.getConfig());
        // 3.参考空白纸张创建画板；
        Canvas canvas = new Canvas(copyPic);
        // 4.创建一个画笔；
        Paint paint = new Paint();
        // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);

        Matrix matrix = new Matrix();
        // 在水平方向和垂直方向分别平移 100 个像素点
        matrix.setTranslate(100, 100);
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, matrix, paint);
        iv.setImageBitmap(copyPic);

    }

    public void rotate(View view) {

        // 1.创建一张原图；
        Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
                R.drawable.meinv);
        // 2.参考原图创建一个空白纸张；
        Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth(),
                srcPic.getHeight(), srcPic.getConfig());
        // 3.参考空白纸张创建画板；
        Canvas canvas = new Canvas(copyPic);
        // 4.创建一个画笔；
        Paint paint = new Paint();
```

```
        // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);

        Matrix matrix = new Matrix();
        // 在围绕着图片中心点旋转 90 度
        matrix.setRotate(90, srcPic.getWidth()/2, srcPic.getHeight()/2);
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, matrix, paint);
        iv.setImageBitmap(copyPic);

    }

    public void scale(View view) {

        // 1.创建一张原图；
        Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
                R.drawable.meinv);
        // 2.参考原图创建一个空白纸张；
        Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth(),
                srcPic.getHeight(), srcPic.getConfig());
        // 3.参考空白纸张创建画板；
        Canvas canvas = new Canvas(copyPic);
        // 4.创建一个画笔；
        Paint paint = new Paint();
        // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);

        Matrix matrix = new Matrix();
        // 在围绕着图片中心点把宽和高都是缩小 1/2
        matrix.setScale(0.5f, 0.5f, srcPic.getWidth()/2, srcPic.getHeight()/2);
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, matrix, paint);
        iv.setImageBitmap(copyPic);

    }


    public void jm(View view) {
```

```
    // 1.创建一张原图；
    Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
            R.drawable.meinv);
    // 2.参考原图创建一个空白纸张；
    Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth(),
            srcPic.getHeight(), srcPic.getConfig());
    // 3.参考空白纸张创建画板；
    Canvas canvas = new Canvas(copyPic);
    // 4.创建一个画笔；
    Paint paint = new Paint();
    // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
    paint.setColor(Color.BLACK);

    Matrix matrix = new Matrix();
    // 在水平方向和垂直方向分别平移 100 个像素点
    matrix.setScale(-1.0f, 1.0f);
    //向右平移图片的宽度
    matrix.postTranslate(srcPic.getWidth(), 0);
    // 5.使用画笔在画板上画画；
    // bitma 参考的原图
    // matrix 表示图形的矩阵，它里面封装了处理图形的 api
    // paint 画图使用的画笔
    canvas.drawBitmap(srcPic, matrix, paint);
    iv.setImageBitmap(copyPic);
}

public void dy(View view) {

    // 1.创建一张原图；
    Bitmap srcPic = BitmapFactory.decodeResource(getResources(),
            R.drawable.meinv);
    // 2.参考原图创建一个空白纸张；
    Bitmap copyPic = Bitmap.createBitmap(srcPic.getWidth(),
            srcPic.getHeight(), srcPic.getConfig());
    // 3.参考空白纸张创建画板；
    Canvas canvas = new Canvas(copyPic);
    // 4.创建一个画笔；
    Paint paint = new Paint();
    // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
    paint.setColor(Color.BLACK);

    Matrix matrix = new Matrix();
    // 在水平方向和垂直方向分别平移 100 个像素点
```

```
        matrix.setScale(1.0f, -1.0f);
        //向下平移图片的高度
        matrix.postTranslate(0, srcPic.getHeight());
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, matrix, paint);
        iv.setImageBitmap(copyPic);
    }


}
```

07_撕衣服

让两张图片重叠显示，在上面这张图片上移动的时候，可以把手所到之处的像素点改变成透明色，可以看到下面一张图片；

代码：

```
package com.itheima.syf;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;
    private Bitmap srcPic;
    private Bitmap copyPic;
    private Canvas canvas;
    private Paint paint;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);

        srcPic = BitmapFactory.decodeResource(getResources(),
                R.drawable.pre4);
        copyPic = Bitmap.createBitmap(srcPic.getWidth(),
                srcPic.getHeight(), srcPic.getConfig());
        canvas = new Canvas(copyPic);
        paint = new Paint();
        // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, new Matrix(), paint);

        iv.setImageBitmap(copyPic);
        // 给 imageview 控件添加触摸监听器
        iv.setOnTouchListener(new OnTouchListener() {
            /**
             * 当初触摸到这个 imageview 控件上时就回调这个方法 v 当前 imageview 对象 event
             * 触摸的事件对象，里面包含了与事件相关的信息 返回 true 表示当前事件已经被消费掉，
false 表示当前事件正在进行还没有结束
             */
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                // 得到触摸事件类型
                int action = event.getAction();
                switch (action) {
                case MotionEvent.ACTION_DOWN:// 按下

                    break;

                case MotionEvent.ACTION_MOVE:// 移动
                    //得到当前像素点的坐标
                    int startX = (int) event.getX();
                    int startY = (int) event.getY();
```

```
                    for(int i=-10; i<10; i++){
                        for(int j=-10;j<10;j++){
                            int r = (int) Math.sqrt(i*i+j*j);
                            if(r<=10){
                                // 把图片上的当前像素点改成透明色
                                copyPic.setPixel(startX+i, startY+j,
Color.TRANSPARENT);
                                //把修改后的图片显示在 imageview 上
                                iv.setImageBitmap(copyPic);
                            }
                        }
                    }

                    //得到下一个像素点
                     startX = (int) event.getX();
                     startY = (int) event.getY();
                    break;
                case MotionEvent.ACTION_UP:// 抬起
                    break;
                }
                // 返回 true 表示当前事件已经被消费掉，false 表示当前事件正在进行还没有结束
                return true;
            }
        });
    }
}
```

08_画画板

代码：

```
package com.itheima.painter;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Bitmap.CompressFormat;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
```

```
import android.graphics.Paint;
import android.os.Bundle;
import android.os.Environment;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;
    private Canvas canvas;
    private Bitmap copyPic;
    private Bitmap srcPic;
    private Paint paint;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);

        srcPic = BitmapFactory.decodeResource(getResources(),
                R.drawable.bg);
        copyPic = Bitmap.createBitmap(srcPic.getWidth(),
                srcPic.getHeight(), srcPic.getConfig());
        canvas = new Canvas(copyPic);
        paint = new Paint();
        // 设置画笔默认的颜色，在画画的过程中会使用原图相对应的颜色来画画
        paint.setColor(Color.BLACK);
        // 5.使用画笔在画板上画画；
        // bitma 参考的原图
        // matrix 表示图形的矩阵，它里面封装了处理图形的 api
        // paint 画图使用的画笔
        canvas.drawBitmap(srcPic, new Matrix(), paint);

        iv.setImageBitmap(copyPic);
        // 给 imageview 添加触摸的监听器
        iv.setOnTouchListener(new OnTouchListener() {
            private int startX;
            private int startY;

            /**
```

```
         * 触摸时回调这个方法
         */
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            // 得到当前的触摸事件类型
            int action = event.getAction();
            switch (action) {
            case MotionEvent.ACTION_DOWN:// 按下

                startX = (int) event.getX();
                startY = (int) event.getY();
                break;

            case MotionEvent.ACTION_MOVE:// 移动

                //得到移动后的坐标点
                int endX = (int) event.getX();
                int endY = (int) event.getY();

                //在画板上画一条线
                canvas.drawLine(startX, startY, endX, endY, paint);
                //把修改后的图片显示出来
                iv.setImageBitmap(copyPic);

                //得到新的开始坐标点
                startX = (int) event.getX();
                startY = (int) event.getY();
                break;

            case MotionEvent.ACTION_UP:// 抬起

                break;


            }

            return true;
        }
    });
}

public void red(View view){
    //把画笔的颜色改为红色
    paint.setColor(Color.RED);
```

```
    }

    public void green(View view){
        //把画笔的颜色改为绿色
        paint.setColor(Color.GREEN);
    }

    public void blue(View view){
        //把画笔的颜色改为绿色
        paint.setColor(Color.BLUE);
    }
    public void brush(View view){
        //把画笔的颜色改为绿色
        paint.setStrokeWidth(10);
    }

    public void save(View view){

        try {
            File file = new File(Environment.getExternalStorageDirectory()+"/123.jpg");
            FileOutputStream fos = new FileOutputStream(file);
            //压缩保存一张图片
            //format 图片 的格式
            //quality 图片的压缩比率 0-100 100 表示无损压缩
            //stream 文件的输出流
            copyPic.compress(CompressFormat.JPEG, 100, fos);
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

09_视频播放器 videoview（重点）

代码：

```
    //初始化播放器
    vv = (VideoView) findViewById(R.id.vv);
    //指定播放的数据资源，既可以是本地的资源文件也可以是网络上的
    vv.setVideoPath("http://192.168.18.115:8080/11.mp4");
```

```
    //开始播放
    vv.start();
```

10_音乐播放器 API

11_影音播放器的常用 API

代码：

```
package com.itheima.mediaplayer;

import android.app.Activity;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnPreparedListener;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    private MediaPlayer mediaPlayer;
    private int currentPosition;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void play(View view) {
        try {
            mediaPlayer = new MediaPlayer();
            // 设置音频流类型
            mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
            // 设置播放的音视频资源
            mediaPlayer.setDataSource("http://192.168.18.115:8080/xpg.mp3");
            // 准备多媒体播放器 prepare 是一个同步的方法，只有等文件加载完成后才能播放。
            // mediaPlayer.prepare(); // might take long! (for buffering, etc)
            // 准备多媒体播放器 prepare 是一个异步的方法，可以边加载边播放播放。
            mediaPlayer.prepareAsync();
            // 添加准备完成的监听器
            mediaPlayer.setOnPreparedListener(new OnPreparedListener() {
                /**
```

```
             * 准备完成后调用这个方法
             */
            @Override
            public void onPrepared(MediaPlayer mp) {
                // 开始播放
                mediaPlayer.start();
            }
        });

    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void pause(View view) {
    if (mediaPlayer != null && mediaPlayer.isPlaying()) {
        // 暂停
        mediaPlayer.pause();
        // 得到当前播放的位置
        currentPosition = mediaPlayer.getCurrentPosition();
    }

}

public void resume(View view) {

    if (mediaPlayer != null) {
        // 设置开始播放的位置
        mediaPlayer.seekTo(currentPosition);
        // 开始播放
        mediaPlayer.start();
    }

}

public void replay(View view) {
    //先停止原来那个播放器
    stop(view);
    // 从头开始播放
    play(view);
}

public void stop(View view) {
    if(mediaPlayer != null){
```

```
            //停止播放
            mediaPlayer.stop();
            //释放多媒体对象占用的资源
            mediaPlayer.release();
            mediaPlayer = null;
        }
    }
}
```

12_影音播放器的生命周期函数

idle:空闲状态，调用 reset 方法，刚创建这个对象时；
initialized:初始化状态：setDataSource 方法后；
prepared：调用 prepare 方法后；
started：开始状态，调用 start 方法；
playbackcompleted:播放完成状态；
preparing:正在准备状态，prepareAsync 执行完成之前；
stoped：停止状态，调用了 stop 方法；
paused：暂停状态，调用了 pause 方法；
released：释放状态，调用 release 方法；
Error：错误状态，播放出现异常；

13_照相机拍照（重点）

调用手机自带的拍照功能：

```
 //创建拍照的意图对象
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    Uri fileUri = Uri.fromFile(new
File(Environment.getExternalStorageDirectory()+"/a.jpg"));
    //告诉相机照片的存储位置
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the image file name
    //调用手机自带的拍照功能，打开拍照的界面
    startActivityForResult(intent, 0);
```

需要在清单文件中添加拍照的权限：

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

14_录像机应用

调用相机的录像功能：

```
 //创建录像的意图对象
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    Uri fileUri = Uri.fromFile(new
File(Environment.getExternalStorageDirectory()+"/a.mp4"));
    //告诉相机照片的存储位置
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the image file name
    //调用手机自带的拍照功能，打开拍照的界面
    startActivityForResult(intent, 0);
```

需要在清单文件中添加拍照的权限：

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

Day10 内容提供者

01_为什么需要内容提供者

自己应用程序中的数据表不允许其他应用程序访问,自己可以写一些访问数据库的接口,让其他应用程序可以通过这些接口访问自己应用程序中的数据库表.

02_编写内容提供者

步骤:

```
    1.在代码中创建一个内容提供者的类,继承 contentprovider;
    2.在清单文件中配置一个 provider 界面,name 指向内容提供者这个类,从 android4.1 之后需要添加一个 android:exported="true"属性,如果没添加这个属性或者值为 false 表示其他应用程序没有权限访问这个内容提供者;必须添加一个 android:authorities="com.itheima.sqlitedb.AccountContentProvider"用来唯一标识这个内容提供者,就相当于网站的域名;
```

新创建一个 Android 工程，工程名《MyContentProvider》，包名：com.itheima.provider。

2 在 com.itheima.provider.dao 包下新建一个 PersonOpenHelper 类继承 SQLiteOpenHelper 类，该类用于创建数据库。

```
public class PersonOpenHelper extends SQLiteOpenHelper {

    public PersonOpenHelper(Context context, String name,
CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    /**
     * 对外提供一个简单的构造函数，使用默认的数据库和默认的版本号
     * @param context
     */
    public PersonOpenHelper(Context context){
        super(context, "person.db", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "create table person (id integer primary key
autoincrement,name varchar(20),phone varchar(20),age integer,address
varchar(50));";
        db.execSQL(sql);

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {

    }

}
```

3 在 com.itheima.contenProvider.provider 包中创建， PersonContentProvider 类继承 ContentProvider 类。同时将该 Provider 在 AndroidManifest.xml 中注册。

```
<provider
android:exported="true"
android:name="com.itheima.contenProvider.provider.PersonContentPro
vider"android:authorities="com.itheima.person" />
```

PersonContentProvider 类是核心业务代码，也是本文档的重要内容，代码清单如下：

```
public class PersonContentProvider extends ContentProvider {
    //用于存放并匹配个 Uri 标识信息，一般在静态代码块中对其信息进行初始化操
作
    private static UriMatcher matcher;
    //声明一个用于操作数据库对象
    private PersonOpenHelper openHelper;
    //主机名信息：对应清单文件的 authorities 属性
    private static final String AUTHORITY = "com.itheima.person";
    //数据库 表名
    private static final String TABLE_PERSON_NAME = "person";
    //Uri 匹配成功的返回码
    private static final int PERSON_INSERT_CODE = 1000;
    private static final int PERSON_DELETE_CODE = 10001;
    private static final int PERSON_UPDATE_CODE = 10002;
    private static final int PERSON_QUERYALL_CODE = 10003;
    private static final int PERSON_QUERYONE_CODE = 10004;
    //静态代码块，用于初始化 UriMatcher
    static{
        //NO_MATCH:没有 Uri 匹配的时候返回的状态码（-1）
        matcher = new UriMatcher(UriMatcher.NO_MATCH);
        //添加一个分机号：
        //对 person 表进行添加操作，如果
Uri=content://com.itheima.person/person/insert,则返回
PERSON_INSERT_CODE
        matcher.addURI(AUTHORITY, "person/insert",
PERSON_INSERT_CODE);
        //对 person 表进行删除操作,如果 Uri=
content://com.itheima.person/person/delete,则返回 PERSON_DELETE_CODE
        matcher.addURI(AUTHORITY, "person/delete",
PERSON_DELETE_CODE);
        //对 person 表进行修改操作,如果 Uri=
content://com.itheima.person/person/update,则返回 PERSON_UPDATE_CODE
        matcher.addURI(AUTHORITY, "person/update",
PERSON_UPDATE_CODE);
```

```
//对 person 表进行查询所有操作,如果 Uri=
content://com.itheima.person/person,则返回 PERSON_QUERYALL_CODE
        matcher.addURI(AUTHORITY, "person", PERSON_QUERYALL_CODE);
        //对 person 表进行查询单个操作,如果 Uri=
content://com.itheima.person/person/#,(#: 代表数字)则返回
PERSON_QUERYONE_CODE
        matcher.addURI(AUTHORITY, "person/#", PERSON_QUERYONE_CODE);
    }
    @Override
    public boolean onCreate() {
        openHelper = new PersonOpenHelper(getContext());
        return false;
    }
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
        //用匹配器去匹配 uri，如果匹配成功则返回匹配器中对应的状态码
        int matchCode = matcher.match(uri);
        SQLiteDatabase db = openHelper.getReadableDatabase();
        switch (matchCode) {
        case PERSON_QUERYALL_CODE:
            return db.query(TABLE_PERSON_NAME, projection, selection,
selectionArgs, null, null, sortOrder);
        case PERSON_QUERYONE_CODE:
        //使用 ContentUris 工具类解析出 uri 中的 id
            long parseId = ContentUris.parseId(uri);
            return db.query(TABLE_PERSON_NAME, projection,"id=?", new
String[]{parseId+""}, null, null, sortOrder);
        default:
            throw new IllegalArgumentException("Uri 匹配失败: "+uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //新插入对象的 id
        long id = db.insert(TABLE_PERSON_NAME, null, values);
        db.close();
        //使用 ContentUris 工具类将 id 追加到 uri 中，返回给客户
        return ContentUris.withAppendedId(uri, id);
```

```
    }

    @Override
    public int delete(Uri uri, String selection, String[]
selectionArgs) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //返回删除的个数
        int count = db.delete(TABLE_PERSON_NAME, selection,
selectionArgs);
        //关闭数据库
        db.close();
        return count;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
String[] selectionArgs) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //返回更新的个数
        int count = db.update(TABLE_PERSON_NAME, values, selection,
selectionArgs);
        //更新数据库
        db.close();
        return count;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

}
```

03_内容提供者工作的原理

内容提供者解析器使用通过内容提供者的主机名来匹配的.如果内容提供者里面指定了路径,在内容提供者解析器中也需要加上路径.内容提供者解析器中的增删改查方法与内容提供者中的增删改查方法一一对应;

04_内容提供者的增删改查的实现

需要在内容提供者里面添加匹配器,用来匹配其他应用程序传过来的 uri 是否合法;

05_内容提供者的使用场景

应用场景:当前需要其他应用程序访问自己应用程序中的数据库表;

06_插入短信

sms 表中关心的列:address,date,type,body;
使用短信应用提供的内容提供者向它数据库表中插入一条数据;

uri: content://sms/

权限:
android:readPermission="android.permission.READ_SMS"
 android:writePermission="android.permission.WRITE_SMS"

07_内容提供者 uri 的写法

形式:

content://主机名
content://主机名/路径
content://主机名/路径/数据记录 ID

08_短信的备份

从短信应用的数据库中查询出短信数据,再序列化到 xml 格式的文加上;

09_短信的还原操作

读取 SD 卡上的 xml 文件,解析其中的数据,插入到短信应用列表中;

10_联系人数据库的表结构

关心的表:
data :联系人信息项的值;
raw_contacts:raw_contact_id 联系人 ID;

```
mimetypes:联系人信息项的名称;

data 表中关心的列:raw_contact_id,mimetype_id,data1;
```

联系人内容提供者的主机名:contacts;com.android.contacts

需要添加的权限:

```
 android:readPermission="android.permission.READ_CONTACTS"
 android:writePermission="android.permission.WRITE_CONTACTS"


raw_contacts 表的 uri:content://com.android.contacts/raw_contacts/

data 表的 uri:content://com.android.contacts/data/
```

11_联系人的还原

步骤：

1. 操作 raw_contacts 表，获取全部的 id
2. 根据获取到的每一条 id 去查询 data 表中的数据
3. 将查询到的展示是用户界面
4. 通过代码给系统联系人插入一条联系人信息

1 新建一个 Android 工程《操作系统联系人》。包名：com.itheima.contacts。

2 使用默认生成的布局文件和 MainActivity 类。在布局文件里将用户联系人展示在 ListView 中，因此我们需要为 ListView 的 Item 创建一个布局文件，由于该布局文件比较简单，因此我们只给出效果图，布局文件清单就不再给出。activity_main.xml 文件清单：

```
<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" >

    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >
    </ListView>

    <LinearLayout
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:onClick="queryContacts"
            android:text="获取联系人" />

        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:onClick="insertContacts"
            android:text="插入联系人" />
    </LinearLayout>

</LinearLayout>
```

想想还是将 list_item.xml 的布局给出吧，哎，任性了。

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:baselineAligned="false"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:orientation="horizontal"
    >
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_weight="1"
        >
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="联系人"
            android:textColor="#0000ff"
            />
        <TextView
            android:id="@+id/tv_name"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="张三"
            android:textColor="#0000ff"
            />
    </LinearLayout>
    <LinearLayout
        android:layout_height="match_parent"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:orientation="vertical"
        >
         <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="电话"
            android:textColor="#ff0000"
            />
        <TextView
            android:id="@+id/tv_phone"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="12333333"
            android:textColor="#ff0000"
            />
```

```
</LinearLayout>
    <LinearLayout
        android:layout_height="match_parent"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:orientation="vertical"
        >
         <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="邮件"
            android:textColor="#00ff00"
            />
        <TextView
            android:id="@+id/tv_email"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="12333333@qq.com"
            android:textColor="#00ff00"
            />

    </LinearLayout>
</LinearLayout>
```

3 编写 MainActivity 类代码，在这里我们用到了自定义的 Contact 对象，用于封装属性，比较简单因此就不再给出 Contact 类代码清单。

```
public class MainActivity extends Activity {

    private ListView lv;
    private List<Contact> list;
    private MyAdapter adapter;
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            Toast.makeText(MainActivity.this, "数据获取成功。",
1).show();
            //更新数据
            adapter.notifyDataSetChanged();
        };
```

```
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lv = (ListView) findViewById(R.id.lv);
        list = new ArrayList<Contact>();
        adapter = new MyAdapter();
        lv.setAdapter(adapter);
    }
    /**
     * 查询所有的系统联系人，这里的业务放在子线程中操作
     */
    public void queryContacts(View view) {
        new Thread(new Runnable() {

            @Override
            public void run() {
                ContentResolver contentResolver = getContentResolver();
                Uri uri =
Uri.parse("content://com.android.contacts/raw_contacts");
                Cursor cursor = contentResolver.query(uri, new String[]
{ "contact_id" }, null, null, null);
                list.clear();
                int i = 0;
                while (cursor.moveToNext()) {
                    i++;
                    //先从 raw_contacts 表中获取 contact_id
                    String contact_id = cursor.getString(0);
                    if (TextUtils.isEmpty(contact_id)) {
                        continue;
                    }
                    //根据 contact_id 从 data 表中查询具体的数据
                    Cursor cursor2 =
contentResolver.query(Uri.parse("content://com.android.contacts/da
ta"), new String[] { "data1", "mimetype" }, "contact_id=?", new
String[] { contact_id }, null);
                    Contact contact = new Contact();
                    while (cursor2.moveToNext()) {
                        String data = cursor2.getString(0);
```

```
                    String mimetype = cursor2.getString(1);
                    if
("vnd.android.cursor.item/name".equals(mimetype)) {
                        contact.setName(data);
                    } else if
("vnd.android.cursor.item/phone_v2".equals(mimetype)) {
                        contact.setPhone(data);
                    } else if
("vnd.android.cursor.item/email_v2".equals(mimetype)) {
                        contact.setEmail(data);
                    } else {
                        contact.setOther(data);
                    }
                }
                list.add(contact);
                cursor2.close();
            }
            cursor.close();
            //发送一个空消息，更新 ListView
            handler.sendEmptyMessage(RESULT_OK);
        }
    }).start();

}

class MyAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return list.size();
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
```

```
        /**
         * 这里面通过 ViewHolder 类将其他子属性值绑定在 View 上面，这里对
ListView 作了进一步的优化处理
         */
        @Override
        public View getView(int position, View convertView, ViewGroup
parent) {
            View view;
            ViewHolder holder;
            if (convertView != null) {
                view = convertView;
                holder = (ViewHolder) view.getTag();
            } else {
                view = View.inflate(MainActivity.this,
R.layout.list_item, null);
                holder = new ViewHolder();
                holder.tv_name = (TextView)
view.findViewById(R.id.tv_name);
                holder.tv_phone = (TextView)
view.findViewById(R.id.tv_phone);
                holder.tv_email = (TextView)
view.findViewById(R.id.tv_email);
                view.setTag(holder);
            }
            Contact contact = list.get(position);
            holder.tv_name.setText(contact.getName());
            holder.tv_email.setText(contact.getEmail());
            holder.tv_phone.setText(contact.getPhone());
            return view;
        }
    }

    class ViewHolder {
        TextView tv_name;
        TextView tv_email;
        TextView tv_phone;
    }
    /**
     * 往系统联系人表中插入一条数据，这里为了方便演示，我们直接插入一条固定
的数据
     */
```

```
    public void insertContacts(View view) {
        //创建一个自定义的 Contact 类，将要网系统联系人表中插入的字段封装起
来
        Contact contact = new Contact();
        contact.setEmail("itheima@itcast.cn");
        contact.setName("王二麻子"+new Random().nextInt(1000));
        contact.setPhone("9999999"+new Random().nextInt(100));
        contact.setOther("北京市中关村软件园");
        //获取 ContentResolver 对象
        ContentResolver resolver = getContentResolver();
        //操作 raw_contacts 表的 uri
        Uri raw_uri =
Uri.parse("content://com.android.contacts/raw_contacts");
        //操作 data 表的 uri
        Uri data_uri =
Uri.parse("content://com.android.contacts/data");
        //在插入数据之前先查询出当前最大的 id
        Cursor cursor = resolver.query(raw_uri, new String[]
{ "contact_id" }, null, null, "contact_id desc limit 1");
        int id = 1;
        if (cursor!=null) {
            boolean moveToFirst = cursor.moveToFirst();
            if (moveToFirst) {
                id = cursor.getInt(0);
            }
        }
        cursor.close();
        //要插入数据的 contact_id 值
        int newId = id + 1;
        // 给 raw_contact 表中添加一天记录
        ContentValues values = new ContentValues();
        values.put("contact_id", newId);
        resolver.insert(raw_uri, values);
        // 在 data 表中添加数据
        // 添加 name
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/name");
        values.put("data1", contact.getName());
        resolver.insert(data_uri, values);
        // 添加 phone
        values = new ContentValues();
```

```
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/phone_v2");
        values.put("data1", contact.getPhone());
        resolver.insert(data_uri, values);
        // 添加地址
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype",
"vnd.android.cursor.item/postal-address_v2");
        values.put("data1", contact.getOther());
        resolver.insert(data_uri, values);
        // 添加 email
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/email_v2");
        values.put("data1", contact.getEmail());
        resolver.insert(data_uri, values);

        Toast.makeText(this, "成功插入联系人" + contact, 0).show();

    }

}
```

4 在 AndroidManifest.xml 中添加添加如下权限：

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

5 部署该工程于模拟器。点击获取联系人，然后在点击插入联系人，然后在点击获取联系人。发现我们不仅成功的将所有的联系人读取出来了同时也将新联系人插入到了数据库中。

运行效果图如下：

12_内容观察者

内容观察者:可以观察到内容提供者数据的变化,可以给你提供一些方法做相关的业务逻辑操作;

我们通过 ContentObserver 实现了对系统发送短信的监听。这是因为系统短信数据数据库中插入新的短信的时候通过 ContentResolver 调用了 notifyChange 方法，正是该方法对外发出了消息，这样我们的 ContentObserver 才监听到了短信的发送。

那么我们也可以自定义我们的 ContentProvider，在 ContentProvider 中发送消息，被我们的 ContentObserver 监听到。

为了方便演示，我们直接使用该文档 1.1 章节中创建的工程《MyContentProvider》。我们修改 PersonContentProvider 类中的 insert 方法，当有新用户插入数据库的时候发出消息以被 ContentObserver 监听。

```
public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //新插入对象的 id
        long id = db.insert(TABLE_PERSON_NAME, null, values);
        db.close();
        //声明一个 uri，通过这个 uri ContentObserver 可以进行监听
        Uri notifyUri = Uri.parse("com.itheima.person");
        /*
         * notifyChange 方法中第一个参数指定一个被监听的 uri
         * 第二个参数是指定 ContentObserver，如果不为 null 则代表只有指定的
ContentObserver 可以接收到消息
         * 为 null 则所有的 ContentObserver 都有机会接收到该消息
         */
        getContext().getContentResolver().notifyChange(notifyUri,
null);
        //使用 ContentUris 工具类将 id 追加到 uri 中，返回给客户
        return ContentUris.withAppendedId(uri, id);
    }
```

同样的，接下来同监听短信一样，我们已经可以通过自定义 ContentObserver 对我们的 person 数据库的添加动作进行监听了。由于监听的原理跟第 5 章完成一样，因此这里就不再给出详细的代码。

13_短信窃听器

用户使用系统自带的短信程序发送短信，程序会通过 ContentProvider 把短信保存进数据库，并且发出一个数据变化通知，使用 ContentObserver 对数据变化进行监听，在用户发送短信时，就会被 ContentObserver 窃听到短信。

发出的短信有这样几个过程：草稿箱->发件箱->已发送。所以只需查询发件箱中的信息即可，处于正在发送状态的短信放在发送箱中。

需求：创建一个应用程序，监听用户发送的短信。

1. 新创建一个 Android 工程《短信窃听》，包名：com.itheima.contentObserver。
2. 新创建一个 SmsContentObserver 类继承 ContentObserver 类，覆写 onChange 方法。在该方法中实现核心业务逻辑。

```
public class SmsContentObserver extends ContentObserver {
    //声明一个 Content 对象，在构造函数中对其进行实例化
    private Context context;
    private Handler handler;
    //声明构造函数
```

```
    public SmsContentObserver(Context context,Handler handler){
        super(handler);
        this.context = context;
        this.handler = handler;
    }
    //覆写父类 onChange 方法
    @Override
    public void onChange(boolean selfChange) {
        //通过查看发件箱中的短信即可观察到新短信的发送
        Uri uri = Uri.parse("content://sms/outbox");
        //获取 Context 对象的 ContentResolver 对象
        ContentResolver resolver = context.getContentResolver();
        String[] projection = {"address","body","date"};
        //获取发件箱中的短信
        Cursor cursor = resolver.query(uri, projection, null, null,
null);
        if(cursor!=null&&cursor.moveToNext()){
            String address = cursor.getString(0);
            String body = cursor.getString(1);
            Long date = cursor.getLong(2);
            //调用 DateFormat 的方法将 long 类型的时间转化为系统相关时间
            String dateStr =
DateFormat.getTimeFormat(context).format(date);
            //打印观察到正在发送的短信
            System.out.println("address="+address+" body="+body+"
date="+dateStr);
        }
        cursor.close();
    }
}
```

3 在 MainActivity 类中注册 SmsContentObserver。

```
public class MainActivity extends Activity {
    private SmsContentObserver observer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //实例化一个自定义的 SMSContentObserver 对象
```

```
        observer = new SmsContentObserver(this, null);
        //操作系统短信的 uri
        Uri uri = Uri.parse("content://sms");
        /**
         * 获取 ContentResolver,然后注册 ContentObserver
         */
        getContentResolver().registerContentObserver(uri, true,
observer);
    }

}
```

4 因为我们读取系统短信了，因此需要在 AndroidManifest.xml 中添加权限。

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

5 将该工程部署在模拟器上并运行，然后打开短息功能，给 5557 模拟器（可以随意指定个号码，不需要启动 5557 模拟器）发送一条短信。观察 LogCat 发现，我们的 ContentObserver 成功监听到了短信内容。

```
address=555-7 body=hello 5557 i am 5554! date=下午 4:47
```

Day11 新特性、样式与主题

---

fragment 新特性（重点）

01_fragment 入门

Android3.0 开始引入这个新特性。主要用于实现一个多任务的界面。
fragement 是 activity 的子界面，是运行在 activity 里面的。

步骤：

1、添加一个 fragment 一个类型，继承 fragment；
2、在 fragment 中加载对应的布局文件；
3、得到 fragment 的管理器
4、得到 fragment 的事务管理器

5、在指定的容器中填充 fragment
6、提交 fragment 的事务

代码：

```
    //加载功能一对应的界面
    //得到 fragment 的管理器
    FragmentManager  fm = this.getFragmentManager();
    //得到 fragment 的事务管理器
    FragmentTransaction ft = fm.beginTransaction();

    Fragment01 f01 = new Fragment01();
    //在指定的容器中填充 fragment
    ft.replace(R.id.fl_container, f01);
    //提交 fragment 的事务
    ft.commit();
```

02_fragment 的向下兼容

把所有与 fragment 相关 API 都是从 surpport.vx 里面导入。

代码：

MainActivity.java:

```
package com.itheima.fragmentsy;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends FragmentActivity {

    private FragmentManager fm;
    private Fragment01 f01;

    private EditText data;
    private TextView tv_info;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    data = (EditText) findViewById(R.id.data);
    tv_info = (TextView) findViewById(R.id.tv_info);
  fm = this.getSupportFragmentManager();
    //得到 fragment 的事务管理器
    FragmentTransaction ft = fm.beginTransaction();

    f01 = new Fragment01();
    //在指定的容器中填充 fragment
    ft.replace(R.id.fl_container, f01);
    //提交 fragment 的事务
    ft.commit();
}


public void submit(View view){
    String dataStr = data.getText().toString();
    f01.getData(dataStr);
}


public void open01(View view){
    //加载功能一对应的界面

    //得到 fragment 的事务管理器
    FragmentTransaction ft = fm.beginTransaction();

    //在指定的容器中填充 fragment
    ft.replace(R.id.fl_container, f01);
    //提交 fragment 的事务
    ft.commit();
}

public void open02(View view){
    //加载功能一对应的界面

    //得到 fragment 的事务管理器
    FragmentTransaction ft = fm.beginTransaction();

    Fragment02 f02 = new Fragment02();
```

```
        //在指定的容器中填充 fragment
        ft.replace(R.id.fl_container, f02);
        //提交 fragment 的事务
        ft.commit();
    }

    public void open03(View view){
        //加载功能一对应的界面

        //得到 fragment 的事务管理器
        FragmentTransaction ft = fm.beginTransaction();

        Fragment03 f03 = new Fragment03();
        //在指定的容器中填充 fragment
        ft.replace(R.id.fl_container, f03);
        //提交 fragment 的事务
        ft.commit();
    }


    public void getInfo(String info){
        tv_info.setText(info);
    }

}
```

Fragment01.java:

```
package com.itheima.fragmentsy;

import android.app.Activity;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Fragment01 extends Fragment {
```

```
private TextView tv_data;

private EditText et_info;

private Button bt_send;
/**
 *  初始化 fragment 对象
 *  加载布局文件
 */
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {

    View view = View.inflate(getActivity(), R.layout.fragment01, null);

    et_info = (EditText) view.findViewById(R.id.et_info);
    //在对应的布局文件中的控件
    tv_data = (TextView) view.findViewById(R.id.tv_data);

    bt_send = (Button) view.findViewById(R.id.bt_send);
    //给按钮添加单击事件的响应方法
    bt_send.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            //把数据传递 activity

            send();
        }
    });
    return view;
}

public void getData(String data){
    tv_data.setText(data);
}

public void send(){
    String info = et_info.getText().toString();
    //得到运行 fragment 的 activity
    Activity a = this.getActivity();
    //判断 activity 是不是 mainactivity
    if(a instanceof MainActivity){
        MainActivity ma = (MainActivity)a;
        //调用 activity 的方法，把数据传递过去
```

```
            ma.getInfo(info);

        }
    }
}
```

03_fragment 的数据传递

步骤：
1、在 fragment 中提供了一个接收数据的方法；
2、在 activity 中调用 fragment 的方法，把数据传递过去；

代码：

```
package com.itheima.fragmentsy;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class Fragment01 extends Fragment {

    private TextView tv_data;
    /**
     *  初始化 fragment 对象
     *  加载布局文件
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {

        View view = View.inflate(getActivity(), R.layout.fragment01, null);
        //在对应的布局文件中的控件
        tv_data = (TextView) view.findViewById(R.id.tv_data);
        return view;
    }

    public void getData(String data){
        tv_data.setText(data);
```

```
    }
}
```

mainactivity 中的方法：

```
public void submit(View view){
    String dataStr = data.getText().toString();
    f01.getData(dataStr);
}
```

在 fragment 中把数据传递给 activity：

```
步骤：
1、在 fragment 得到目标 activity；
2、调用 activity 的方法把数据传递过去；
```

代码：

```
package com.itheima.fragmentsy;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Fragment01 extends Fragment {

    private TextView tv_data;

    private EditText et_info;

    private Button bt_send;
    /**
     *  初始化 fragment 对象
     *  加载布局文件
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

```
            Bundle savedInstanceState) {

        View view = View.inflate(getActivity(), R.layout.fragment01, null);

        et_info = (EditText) view.findViewById(R.id.et_info);
        //在对应的布局文件中的控件
        tv_data = (TextView) view.findViewById(R.id.tv_data);

        bt_send = (Button) view.findViewById(R.id.bt_send);
        //给按钮添加单击事件的响应方法
        bt_send.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //把数据传递 activity
                send();
            }
        });
        return view;
    }

    public void send(){
        String info = et_info.getText().toString();
        //得到运行 fragment 的 activity
        Activity a = this.getActivity();
        //判断 activity 是不是 mainactivity
        if(a instanceof MainActivity){
            MainActivity ma = (MainActivity)a;
            //调用 activity 的方法，把数据传递过去
            ma.getInfo(info);
        }
    }
}
```

04_fragment 的生命周期

与 activity 生命周期的方法相比，多了：
onCreateView：加载布局文件
onDestroyView：销毁 fragment 的视图对象

属性动画

01_为什么使用属性动画

02_属性动画入门

03_3 种常见属性动画

代码：

```
package com.itheima.propertiesanimi;

import android.animation.Animator;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private ImageView iv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv = (ImageView) findViewById(R.id.iv);
    }

    public void trans(View view) {
        // 把动画对象作用到按钮上
        ObjectAnimator oa = ObjectAnimator.ofFloat(iv, "translationX", 0, 10,
                20, 40, 60, 100, 120, 140);
        oa.setDuration(3000);
        oa.setRepeatCount(2);
        oa.setRepeatMode(ObjectAnimator.REVERSE);

        oa.start();
    }

    public void rotate(View view) {
```

```
    // 把动画对象作用到按钮上
    ObjectAnimator ra = ObjectAnimator.ofFloat(iv, "rotation", 0, 30,
            60, 90, 120, 150, 180);
    ra.setDuration(3000);
    ra.setRepeatCount(2);
    ra.setRepeatMode(ObjectAnimator.REVERSE);

    ra.start();
}

public void alpha(View view) {


    // 把动画对象作用到按钮上
    ObjectAnimator aa = ObjectAnimator.ofFloat(iv, "alpha", 0, 0.2f,
            0.4f, 0.6f, 0.8f, 1.0f);
    aa.setDuration(3000);
    aa.setRepeatCount(2);
    aa.setRepeatMode(ObjectAnimator.REVERSE);

    aa.start();
}

public void scale(View view) {

    // 把动画对象作用到按钮上
    ObjectAnimator sa = ObjectAnimator.ofFloat(iv, "scaleY", 0, 0.2f,
            0.4f, 0.6f, 0.8f, 1.0f);
    sa.setDuration(3000);
    sa.setRepeatCount(2);
    sa.setRepeatMode(ObjectAnimator.REVERSE);

    sa.start();
}

public void set(View view) {

    AnimatorSet set = new AnimatorSet();

    ObjectAnimator oa = ObjectAnimator.ofFloat(iv, "translationX", 0, 10,
            20, 40, 60, 100, 120, 140);
    oa.setDuration(3000);
    oa.setRepeatCount(2);
```

```
        oa.setRepeatMode(ObjectAnimator.REVERSE);

        ObjectAnimator ra = ObjectAnimator.ofFloat(iv, "rotation", 0, 30,
                60, 90, 120, 150, 180);
        ra.setDuration(3000);
        ra.setRepeatCount(2);
        ra.setRepeatMode(ObjectAnimator.REVERSE);

        ObjectAnimator aa = ObjectAnimator.ofFloat(iv, "alpha", 0, 0.2f,
                0.4f, 0.6f, 0.8f, 1.0f);
        aa.setDuration(3000);
        aa.setRepeatCount(2);
        aa.setRepeatMode(ObjectAnimator.REVERSE);

        // 把动画对象作用到按钮上
                ObjectAnimator sa = ObjectAnimator.ofFloat(iv, "scaleY", 0, 0.2f,
                        0.4f, 0.6f, 0.8f, 1.0f);
                sa.setDuration(3000);
                sa.setRepeatCount(2);
                sa.setRepeatMode(ObjectAnimator.REVERSE);

        //组合播放动画
        set.playTogether(oa,ra,aa,sa);

        set.start();
    }
}
```

样式与主题（重点）

01_样式

样式：主要作用于控件上的，修饰控件的一些属性；

自定义样式：

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <style name="button_color_stype">
        <item name="android:background" >#ff0000</item>
```

```
        <item name="android:textColor">#00ff00</item>
    </style>

    <style name="textview_color_stype">
        <item name="android:background" >#ff0000</item>
        <item name="android:textColor">#0000ff</item>
    </style>

    <style name="textview_color_stype_largesize" parent="textview_color_stype">
        <item name="android:textSize" >20sp</item>
        <item name="android:background" >#0000ff</item>
    </style>

</resources>
```

02_主题

主题:界面或者整个应用程序的风格；
theme
定义主题的方法与定义样式完全一样；

代码： <?xml version="1.0" encoding="utf-8"?> <resources xmlns:android="http://schemas.android.com/apk/res/android">

```
</resources>



<activity
        android:name="com.itheima.style.MainActivity"
        android:label="@string/app_name"
        android:theme="@style/my_theme_activity_background"
         >
```