宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Alpha 0.01 版

Android-网络编程&消息处理&开源框架

1.Android 网络编程（★★★）

在当今互联网时代，我们在公司写的 Android 程序基本不可能不联网的，网络编程是任何一个 Android 程序员必备的技能。

1.1 网络编程的预备知识

- 访问网络的 Android 应用都必须加上访问互联网的权限：

```
android.permission.INTERNET
```

- 开启子线程执行网络或者耗时的操作

Tips:在 Android 中凡是对 UI 的更新、“耗时”操作等都需要在子线程中进行。

★ 1.Android4.0 以上版本，Google 更加在意 UI 界面运行的流畅性，强制要求访问网络的操作不允许在主线程中执行，只能在子线程中进行，在主线程请求网络时，会报如下错误：

```
android.os.NetworkOnMainThreadException
```

★ 2.ANR 异常：Application Not Response,应用程序无响应。在主线程中做一些耗时的操作，阻塞了主线程，当用户点击其时，主线程无法响应，这是就会出 ANR 异常。

- 子线程不能修改 UI

主线程也叫 UI 线程,Activity 中的 onCreate 方法和点击事件的方法都是运行在主线程中的。主线程创建的界面,只有主线程才能修改，别的线程不允许修改 UI，否则会报如下错

误：

```
CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
```

如果子线程修改了 UI，系统会验证当前线程是不是主线程，如果不是主线程，就会终止运行。

Tips:既然子线程不能修改主线程的 UI 那么，我们的子线程如果需要修改 UI 该怎么办呢？

解决方式：使用 Handler 实现子线程与主线程之间的通信。

消息处理机制原理：所有使用 UI 界面的操作系统，后台都运行着一个死循环（Looper），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。关于 Handler 的使用会在接下来的案例中介绍，这里先点到为止。

- 模拟器如何访问本地 Tomcat

模拟器把它自己作为了 localhost，也就是说，代码中使用 localhost 或者 127.0.0.1 来访问，都是访问模拟器自己！若想在模拟器上面访问我们的电脑，那么就使用 android 内置的 IP: 10.0.0.2，10.0.0.2 是模拟器设定的特定 ip,在模拟器上用 **10.0.0.2** 就能成功访问我们的电脑本机。

1.2 案例-网络图片查看器

需求：实现一个网络图片查看器，在页面输入框填入网络图片的地址，点击访问按钮，可以访问网络并获取图片，并显示在界面上。

1. 新创建一个工程，工程名字《网络图片查看器》。
2. 在工程清单文件（AndroidManifest.xml）中添加访问网络权限

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

3 使用并修改默认的布局文件（activity_main.xml）。

```
<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"
    tools:context=".MainActivity"
    android:orientation="vertical"
    >

    <ImageView
        android:layout_height="0dp"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:id="@+id/iv"
        />
    <EditText
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:hint="请输入地址"

android:text="http://g.hiphotos.baidu.com/image/pic/item/6159252dd4
2a2834d277681659b5c9ea14cebfde.jpg"
        android:id="@+id/et"
        />
    <Button
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="获取图片"
        android:onClick="getImage"
        />
```

4 使用并编辑默认的 Activity（MainActivity.java）

```
public class MainActivity extends Activity {

    private ImageView imageView;
    //创建一个 Handler 对象，用户接收子线程发送的消息，然后更新 UI
    private Handler handler = new Handler(){

        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            //将数据强转转化为 Bitmap,然后显示在 ImageView 控件中
            imageView.setImageBitmap((Bitmap) msg.obj);
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imageView = (ImageView)findViewById(R.id.iv);
    }

    public void getImage(View view) {
        //开启一个子线程 处理网络请求
            new Thread(new Runnable() {

                @Override
                public void run() {
                    try {
                        //从页面获取 URL 地址
                        EditText editText =
(EditText)findViewById(R.id.et);
                        String path = editText.getText().toString();
                        //调用 Android API 中的 TextUtils 工具类判断路径是否为
空
                        if (TextUtils.isEmpty(path)) {
                            Looper.prepare();
                            Toast.makeText(MainActivity.this, "请输入 URL",
0).show();
                            Looper.loop();
                            return;
                        }
                        URL url = new URL(path);
```

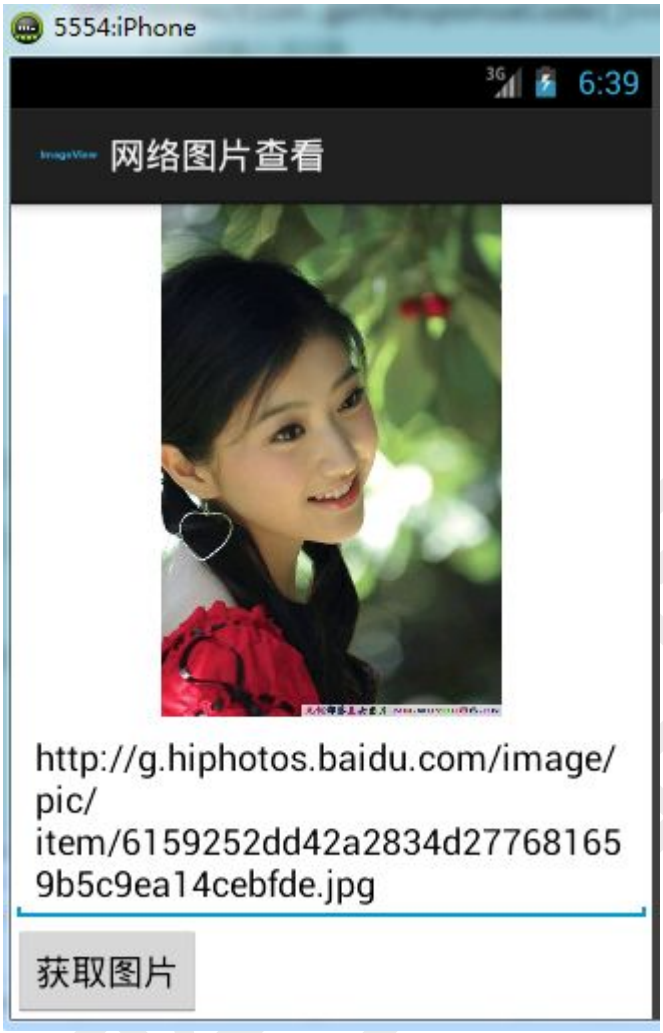
```
                    //获取 HttpURLConnection 链接对象
                    HttpURLConnection connection =
(HttpURLConnection)url.openConnection();
                    //设置请求方法为 GET 方式
                    connection.setRequestMethod("GET");
                    //设置链接超时时间
                    connection.setConnectTimeout(50000);
                    //设置输入流读取超时时间
                    connection.setReadTimeout(50000);
                    //打开链接，发送请求
                    connection.connect();
                    //判断返回的状态码
                    if(connection.getResponseCode()==200){
                        //获取输入流对象
                        InputStream inputStream =
connection.getInputStream();
                        //调用 Android API 提供的 BitmapFactory 工具类将字
节流转化为位图
                        Bitmap bitmap =
BitmapFactory.decodeStream(inputStream);
                        //创建一个新的消息
                        Message message = new Message();
                        //将数据绑定消息
                        message.obj = bitmap;
                        //调用 handler 发送消息给主线程
                        handler.sendMessage(message);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }

            }
        }).start();
    }

}
```

5 运行上面的工程，效果图如下：输入百度图片的一个网址

http://g.hiphotos.baidu.com/image/pic/item/6159252dd42a2834d277681659b5c9ea14cebfde.jpg



1.3 案例-网页源码查看器

需求：实现一个网页源码查看器，在页面输入框填入网页的地址，点击访问按钮，可以读取网页并显示在界面上。

Tips：该案例跟 1.2 章节案例类似，因此可以直接在 1.2 章节中创建的工程中修改，当然新建工程也可以。使用 Button 按钮、EditText 控件、TextView 控件设置布局，在网络访问时，将接收到的二进制流转换成字符串，将字符串交由 handler 修改 TextView 的显示

即可。

因为布局比较简单，处于节约篇幅的考虑就不再给出布局清单，这里只给出 Activity 类代码清单。MainActivity.java 代码清单如下：

```
public class MainActivity extends Activity {

    //定义两个常量用于代表消息的类型
    private static final int ERROR = 0;
    private static final int OK = 1;
    Handler handler = new Handler(){

        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            //获取界面的 TextView 对象实例
            TextView tv = (TextView)findViewById(R.id.tv);
            //如果接收到的信息为 OK，则将结果显示出来
            if (msg.what==OK) {
                tv.setText((String)msg.obj);
            //如果 ERROR 则提示错误信息
            }else if (msg.what==ERROR) {
                tv.setText("对不起，页面加载失败！");
            }
        }

    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    //界面 Button 绑定的事件
    public void load(View view){
        EditText editText = (EditText)findViewById(R.id.et);
        if (TextUtils.isEmpty(editText.getText().toString())) {
            Toast.makeText(this, "请输入网页的 URL",
Toast.LENGTH_SHORT).show();
            return;
        }
```

```
        //调用本类自定义方法，实现网页抓取
        getContent(editText.getText().toString());
    }
    private void getContent(final String path){
        final StringBuilder sb = new StringBuilder();
        new Thread(new Runnable() {
            @Override
            public void run() {
                URL url;
                try {
                    //根据 url 地址创建一个 URL 对象
                    url = new URL(path);
                    //创建一个 HttpURLConnection 对象
                    HttpURLConnection connection =
(HttpURLConnection)url.openConnection();
                    //设置连接超时为 5 秒
                    connection.setConnectTimeout(5000);
                    //设置读取数据流超时为 5 秒
                    connection.setReadTimeout(5000);
                    //设置请求方式为 GET
                    connection.setRequestMethod("GET");
                    //开始连接
                    connection.connect();
                    //获取返回状态码
                    int responseCode = connection.getResponseCode();
                    if(responseCode==200){//如果成功
                        //获取字节流
                        InputStream inputStream =
connection.getInputStream();
                        //将字节流转化为 BufferedReader
                        BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(inputStream));
                        String tmp = null;
                        while((tmp=bufferedReader.readLine())!=null){
                            sb.append(tmp);
                        }
                        //新创建个消息对象
                        Message msg = new Message();
                        //设置消息的类型为 OK（自定义，用于自己区分不同的消息）
                        msg.what = OK;
```

```
                    //给消息绑定数据
                    msg.obj = sb.toString();
                    //将消息发给主线程
                    handler.sendMessage(msg);
                }else {//如果请求失败
                    //发送一个空消息
                    handler.sendEmptyMessage(ERROR);
                }
            } catch (Exception e) {
                e.printStackTrace();
                handler.sendEmptyMessage(ERROR);
            }
        }
    }).start();
  }
}
```

运行以上程序，结果图如下：

1.4 案例-新闻客户端

需求：使用 ListView 控件实现一个新闻客户端，新闻信息以 XML 文件的格式存储 通过网络访问存放新闻信息的 XML 文件，解析此文件，逐条信息生成 ListView 的 item，添加到 ListView 中，在界面上呈现出来。

1 创建一个新工程，工程名为《新闻客户端》，包名为 com.itheima.news，在清单文件中添加访问网络权限。

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

2 使用并编辑默认布局文件 activity_main.xml，布局文件清单如下：

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

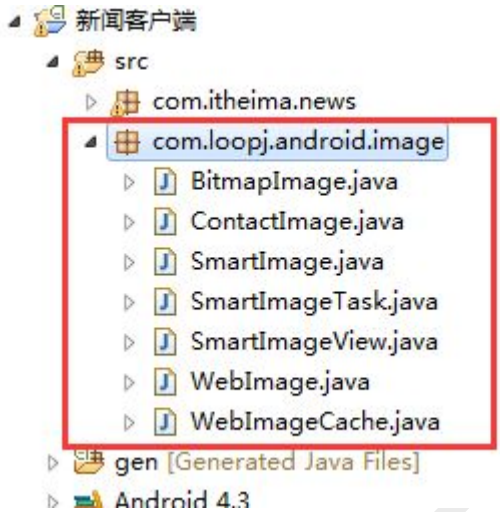
3 在 res/layout 目录下创建 item.xml 文件，作为第二步骤中 ListView 的 item 布局文件。该布局文件采用相对布局，布局清单如下：

```
<?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="wrap_content"
    android:id="@+id/rl">
```

```
    <!-- 引入第三方控件用于显示图片 -->
    <com.loopj.android.image.SmartImageView
        android:id="@+id/iv"
        android:layout_height="match_parent"
       android:layout_width="60dp"
       android:contentDescription="news"
        />
    <!-- 创建一个 LinearLayout，采用垂直布局方式 -->
    <LinearLayout
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:orientation="vertical"
        android:layout_toRightOf="@id/iv"
        >
        <!-- 用于显示新闻的标题 -->
        <TextView
            android:layout_height="0dp"
            android:layout_width="match_parent"
            android:singleLine="true"
            android:id="@+id/tv_title"
            android:layout_weight="1"
            />
        <!-- 用于显示新闻正文内容 -->
        <TextView
            android:layout_height="0dp"
            android:layout_width="match_parent"
            android:singleLine="true"
            android:id="@+id/tv_content"
            android:layout_weight="3"
            />
        <!-- 用于显示评论次数 -->
        <TextView
            android:layout_height="0dp"
            android:layout_width="match_parent"
            android:singleLine="true"
            android:id="@+id/tv_commen"
            android:layout_weight="1"
            />
    </LinearLayout>

</RelativeLayout>
```

Tips:在上面的布局文件中我们采用了开源框架 SmartImageView 控件来显示图片。在布局文件声明此控件时需写全包路径，否则将无法找到。关于 SmartImageView 的用法将会在下面的撞见中详细介绍。该框架很小巧，只需将如下文件添加到工程中。



4 编写一个 JavaBean 用于封装新闻对象。类名：News

```
public class News {
    private Integer id;//新闻 id
    private String title;//新闻标题
    private String detail;//新闻内容详情
    private String image;//新闻图片链接地址
    private String comment;//评论
    //省略 setter&getter 方法
```

5 在本地tomcat的webapps目录下创建news文件夹 然后将news.xml和image文件添加到改文件夹中。查看本机的 IP 地址，并修改 news.xml 文件。然后启动 tomcat。部署后的 tomcat 目录结构如下截图。

其中 news.xml 的文件清单如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<newslist>
    <news>
        <title>黑马程序员 37 期就业快报</title>
        <detail>热烈祝贺黑马程序员 37 期平均薪水突破 12k</detail>
        <comment>15687</comment>
        <image>http://172.16.0.61:8080/a/images/a6.jpg</image>
    </news>
    <news>
        <title>程序员因写代码太乱被杀害</title>
        <detail>凶手是死者同事，维护死者代码时完全看不懂而痛下杀手
</detail>
        <comment>16359</comment>
        <image>http://172.16.0.61:8080/a/images/a7.jpg</image>
    </news>
    <news>
        <title>产品经理因频繁改需求被杀害</title>
        <detail>凶手是一名程序员，因死者对项目需求频繁改动而痛下杀手
</detail>
        <comment>14112</comment>
        <image>http://172.16.0.61:8080/a/images/a7.jpg</image>
    </news>
    <news>
        <title>3Q 大战宣判：腾讯获赔 500 万</title>
        <detail>最高法驳回 360 上诉，维持一审宣判.</detail>
        <comment>6427</comment>
        <image>http://172.16.0.61:8080/a/images/a1.jpg</image>
    </news>
    <news>
        <title>今日之声:北大雕塑被戴口罩</title>
        <detail>市民：因雾霾起诉环保局；公务员谈"紧日子"：坚决不出
去.</detail>
        <comment>681</comment>
        <image>http://172.16.0.61:8080/a/images/a2.jpg</image>
    </news>
</newslist>
```

6 使用并编辑默认 MainActivity.java，在该类中实现所有方法。

```
public class MainActivity extends Activity {
    //设置新闻访问地址
    private String path = "http://172.16.0.67:8080/news/news.xml";
    Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            final List<News> list = (List<News>)msg.obj;
            //实例化 ListView 对象
            ListView listView = (ListView)findViewById(R.id.lv);
            //给 ListView 设置适配器
            listView.setAdapter(new BaseAdapter() {

                @Override
                public View getView(int position, View convertView,
ViewGroup parent) {
                    if(convertView==null){
                        convertView =
(RelativeLayout)View.inflate(MainActivity.this, R.layout.item,
null);
                    }else{
                        return convertView;
                    }
                        News news = list.get(position);
                        //实例化 SmartImageView 对象
                        SmartImageView imageView =
(SmartImageView)convertView.findViewById(R.id.iv);
                        //给 SmartImageView 设置一个图像的 URL 则
SmartImageView 会自动加载图片
                        imageView.setImageUrl(news.getImage());
                        TextView connenTextView =
(TextView)convertView.findViewById(R.id.tv_commen);
                        connenTextView.setText("评论:
"+news.getComment());
                        TextView contentTextView =
(TextView)convertView.findViewById(R.id.tv_content);
                        contentTextView.setText(news.getDetail());
                        TextView titleTextView = (TextView)
convertView.findViewById(R.id.tv_title);
                        titleTextView.setText(news.getTitle());
```

```
            return convertView;
        }
        @Override
        public long getItemId(int position) {
            return position;
        }
        @Override
        public Object getItem(int position) {
            return null;
        }
        @Override
        public int getCount() {
            return list.size();
        }
    });
  }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //调用本类私有方法完成网络访问
    getString();
}
//开启子线程下载数据
private String getString() {
      new Thread(new Runnable() {
          @Override
          public void run() {
              XmlPullParserFactory parserFactory;
              //创建一个集合，用于存储新闻
              List<News> list = new ArrayList<News>();
              try {
                  //获取 XML 解析器工厂
parserFactory = XmlPullParserFactory.newInstance();
                  //获取解析器
XmlPullParser parser = parserFactory.newPullParser();
```

```
                    URL url = new URL(path);
                    //打开一个链接
       HttpURLConnection connection = (HttpURLConnection)
   url.openConnection();
                    //设置连接超时时间
                    connection.setConnectTimeout(5000);
                    connection.setReadTimeout(5000);
                    connection.setRequestMethod("GET");
                    //开始链接
                    connection.connect();
                    //获取数据流
                    InputStream inputStream =
connection.getInputStream();
                    //给解析器设置流对象
                    parser.setInput(new
InputStreamReader(inputStream));
                    News news = new News();
                    //解析 XML 数据

   while(parser.next()!=XmlPullParser.END_DOCUMENT){
                        int eventType = parser.getEventType();
                        switch (eventType) {
                        case XmlPullParser.START_TAG:
                            String tagName = parser.getName();
                            if (tagName.equals("newslist")) {
                                continue;
                            }else if (tagName.equals("news")) {
                                news = new News();
                                System.out.println("创建新的新闻对象");
                            }else if (tagName.equals("title")) {
                                String title = parser.nextText();
                                news.setTitle(title);
                            }else if (tagName.equals("detail")) {
                                String detail = parser.nextText();
                                news.setDetail(detail);
                            }else if (tagName.equals("comment")) {
                                String comment = parser.nextText();
                                news.setComment(comment);
                            }else if (tagName.equals("image")) {
                                String image = parser.nextText();
```

```
                            news.setImage(image);
                        }else {
                            continue;
                        }
                        break;
                    case XmlPullParser.END_TAG:
                        String endTag = parser.getName();
                         if (endTag.equals("news")) {
                            //将解析出来的数据添加到集合中
                            list.add(news);
                        }
                        break;
                    default:
                        continue;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
                Looper.prepare();
                Toast.makeText(MainActivity.this, "请求网络失败
"+e, 0).show();
                Looper.loop();
            }
            //如果当前消息池中有则返回一个，没有则创建
            Message msg = Message.obtain();
            //给当前消息绑定数据
            msg.obj=list;
            handler.sendMessage(msg);
        }
    }).start();
    return null;
  }
}
```

运行上面的程序，效果图如下：

1.5 案例-向服务器提交数据

该案例模拟 Android 客户端给服务器端发送登录申请，客户端将用户名和密码发给服务器，由服务器来验证登录是否成功。在该案例中我们分别用 GET 和 POST 两种方式提交数据。

在该案例前我们需要做一些前期工作，首先在本地 tomcat 服务器上发布一个用户登录的服务。服务端用 JavaWEB 阶段学过的 Servlet 实现。这里直接给出 Servlet 代码，关于 JavaWEB 的其他知识这里不再涉及。

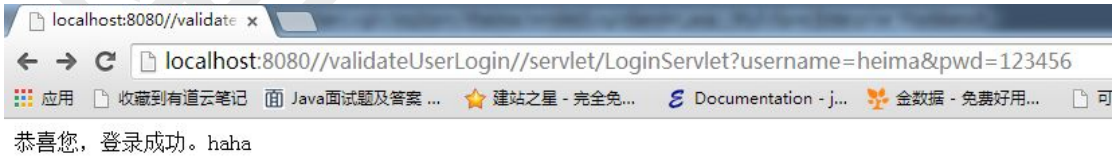
- 以 GET 方式向客户端提交数据

1 LoginServlet 核心业务代码清单如下：

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      String username = request.getParameter("username");
      String pwd = request.getParameter("pwd");
      response.setCharacterEncoding("UTF-8");
      response.setHeader("Content-Type", "text/html;
charset=UTF-8");
      PrintWriter writer = response.getWriter();
      if ("heima".equals(username)&&"123456".equals(pwd)) {//登录成
功
         writer.write("恭喜您，登录成功。haha");
      }else {//登录失败
         writer.write("对不起，密码或者用户名不正确，登录失败！");
      }
      writer.close();
   }
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

      doGet(request, response);
   }
```

2 将该 Servlet 发布到本地 tomcat，然后运行 tomcat，首先通过浏览器测试服务，发现发布成功。



3 新创建一个 Android 工程，工程名《用户登录》。使用默认的布局文件和 Activity。

布局文件清单如下：

```
 <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="向服务器提交数据"
        android:textColor="#0000ff" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="用户名" />

    <EditText
        android:id="@+id/et_username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="请输入用户名" />
     <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="密码" />
    <EditText
        android:id="@+id/et_pwd"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:hint="请输入密码" />
   <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="登陆"
        android:layout_gravity="right"
android:onClick="login"
        />
</LinearLayout>
```

MainActivity 代码清单文件如下：

```
public class MainActivity extends Activity {
    private EditText et_username;
    private EditText et_pwd;
    private final String PATH =
"http://172.16.0.61:8080/validateUserLogin/servlet/LoginServlet";
    private static final String METHOD_GET = "GET";
    private static final String METHOD_POST = "POST";
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
              Toast.makeText(MainActivity.this, msg.obj.toString(),
1).show();
        };
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et_username = (EditText) findViewById(R.id.et_username);
        et_pwd = (EditText) findViewById(R.id.et_pwd);
    }

    public void login(View view) {
        final String username = et_username.getText().toString();
        final String pwd = et_pwd.getText().toString();
        if (TextUtils.isEmpty(username)) {
            Toast.makeText(this, "对不起，用户名不能为空！", 0).show();
            return;
        } else if (TextUtils.isEmpty(pwd)) {
            Toast.makeText(this, "对不起，密码不能为空！", 0).show();
            return;
        }
        // 开启一个子线程访问网络，提交数据，对用户名和密码进行验证
        new Thread(new Runnable() {
            @Override
            public void run() {
                String result = null;
                try {
                    String path = PATH+"?username="+username+"&pwd="+pwd;
                    URL url = new URL(path);
```

```
    HttpsURLConnection connection = (HttpsURLConnection)
url.openConnection();
                connection.setConnectTimeout(5000);
                connection.setReadTimeout(5000);
                connection.setRequestMethod(METHOD_GET);
                connection.connect();
                int responseCode = connection.getResponseCode();
                if (responseCode == 200) {
                    InputStream in = connection.getInputStream();
                    ByteArrayOutputStream out = new
ByteArrayOutputStream();
                    byte[] bytes = new byte[1024];
                    int len = -1;
                    while ((len = in.read(bytes)) != -1) {
                        out.write(bytes, 0, len);
                    }
                    in.close();
                    result = out.toString();
                }
            } catch (Exception e) {
                e.printStackTrace();
                result = e.getMessage();
            } finally {
                Message msg = Message.obtain();
                msg.obj = result;
                handler.sendMessage(msg);
            }
        }
    }).start();
  }
}
```

4 运行上面的工程，输入账号 heima，密码 123456，发现登录成功。

运行截图如下：

以 POST 方式向客户端提交数据

该方式跟以 POST 方式提交数据只有稍微的区别，因此为了方便演示，我们直接在上面的工程中修改代码。

```
//String path = PATH+"?username="+username+"&pwd="+pwd;
                String path = PATH;
                URL url = new URL(path);
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                connection.setConnectTimeout(5000);
                connection.setReadTimeout(5000);
                //connection.setRequestMethod(METHOD_GET);
                connection.setRequestMethod(METHOD_POST);
                connection.setDoOutput(true);
```

```
                String data = "username="+username+"&pwd="+pwd;
                OutputStream outStream =
connection.getOutputStream();
                outStream.write(data.getBytes());
                outStream.close();
```

在上面的代码清单中，蓝色部分为老的内容，需要替换，黄色的部分是新添加的部分。

运行上面的代码会得到跟以 GET 方式提交同样的结果。

Tips：在上面的 2 个案例中，如果我们如果提交的用户名是中文，我们可能会遇到乱码问题，因此下面将结合上面的案例介绍一下常见的中文乱码问题。

1.6 中文乱码问题

Android 操作系统默认使用的编码是 UTF-8，解决中文乱码问题的关键是确保服务器和客户端使用的编码一致。

1.6.1 GET 方式中文乱码的解决

客户端：在拼接字符串的时候，对传递的数据通过 URLEncoder 类进行一下编码，使用的编码方式与服务器约定的编码方式一致。这里假设服务端的编码方式为 UTF-8，发送数据到服务器的处理代码如下：

```
String usernameEncoder=URLEncoder.encode(username,"UTF-8");

String passwordEncoder=URLEncoder.encode(password,"UTF-8");

String urlPath=path+"?username="+usernameEncoder+"&password="+passwordEncoder;
```

接收来自服务器的数据时，处理代码如下：

```
//获取响应的内容
InputStream is=conn.getInputStream();
//读取流，转换为字符串
ByteArrayOutputStream baos=new ByteArrayOutputStream();
int len=-1;
byte[] bys=new byte[1024];
while((len=is.read(bys))!=-1){
          baos.write(bys, 0, len);
   }
String result=baos.toString("UTF-8");
baos.close();
```

服务端：Tomcat 默认采用 iso-8859-1 的编码接收客户端传递来的数据，因此在获取 request 请求中传递来的参数时，需先用 iso-8859-1 对参数进行解码，然后再以客户端传递的数据的编码方式进行编码。这里假设客户端发送的数据其编码方式为 UTF-8，获取请求参数时处理代码如下：

```
String username=request.getParameter("username");

byte[] bys=username.getBytes("iso-8859-1");

username=new String(bys,"UTF-8");
```

发送数据给客户端时，处理代码如下：

```
response.getOutputStream().write("登录成功".getBytes("UTF-8"));
```

1.6.2 POST 方式中文乱码的解决

客户端：发送数据给服务器的时候，对发送的数据进行编码。

```
//要发送到服务端的数据
String data="username="+username+"&password="+password;
//写数据
conn.getOutputStream().write(data.getBytes("UTF-8"));
```

接收数据时的处理逻辑与 GET 方式相同。

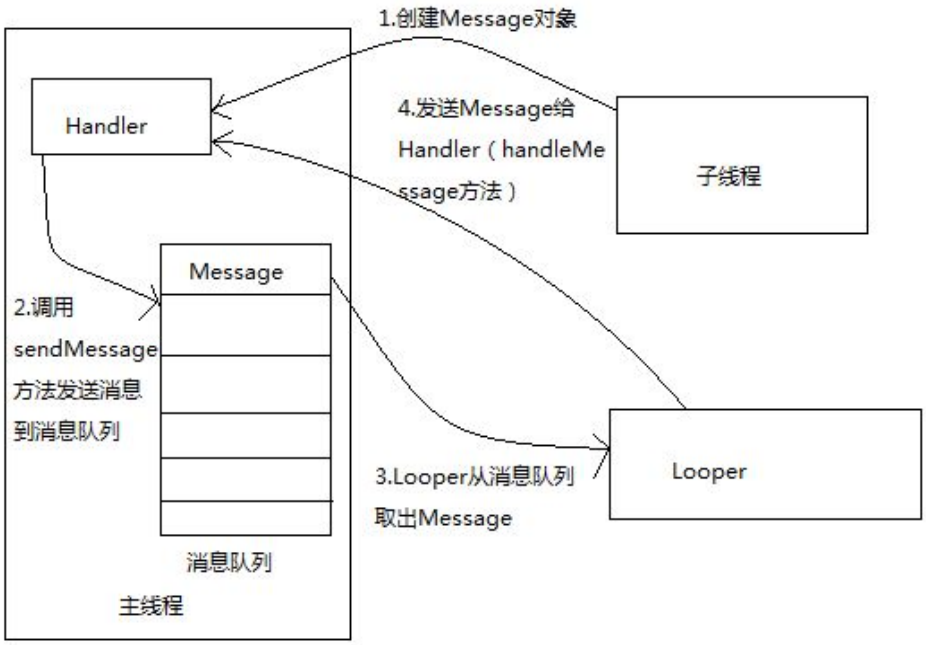
服务端：发送数据给客户端时的处理逻辑与 GET 方式相同，接收 request 请求发送来的参数时，可以采用先使用 iso-8859-1 解码再使用客户端的编码方式进行编码，也可以通过如下处理代码实现：

```
request.setCharacterEncoding("UTF-8");
```

2.Android 消息处理机制（★★★★）

2.1 Looper、Message、Handler 的关系

当我们的 Android 应用程序的进程一创建的时候，系统就给这个进程提供了一个 Looper，Looper 是一个死循环，它内部维护这个一个消息队列。Looper 不停地从消息队列中取消息（Message），取到消息就发送给了 Handler，最后 Handler 根据接收到的消息去修改 UI。Handler 的 sendMessage 方法就是将消息添加到消息队列中。



2.2 runOnUiThread

Activity 中提供了一个 runOnUiThread 方法，用于进行消息处理。此方法是通过线程合并——join 来实现消息处理的。

线程合并：主线程将子线程的任务拿到自己这里来执行并终止子线程。

实例代码如下：

```
/**
        * Runs the specified action on the UI thread. If the current
thread is
        * the UI thread, then the action is executed immediately. If the
        * current thread is not the UI thread, the action is posted to
the
        * event queue of the UI thread.
        * 上面的意思为：在 UI 线程中运行我们的任务，如果当前线程是 UI 线程，
则立即执行，如果不是则该任务发送到 UI 线程的事件队列。
        */
       runOnUiThread(new Runnable() {

           @Override
           public void run() {
               //自定义我们的业务代码
           }
       });
```

2.3 postDelayed

该方法是 Handler 对象提供的，Handler 给消息队列发送一个消息，发送成功则返回 true，否则返回 false，如果返回 false 一般是由于 looper 进程不存在导致的。该方法主要用于定时任务。如果返回 true 也不一定代表着我们的定时任务就执行了，因为很可能在定时任务的时间未到之前我们的 looper 进程退出了，那么消息也就丢失了。

执行该任务的线程用的就是 Handler 对象所在的线程。

```
/**
* Causes the Runnable r to be added to the message queue, to be run
* after the specified amount of time elapses. The runnable will be run
* on the thread to which this handler is attached. Parameters: r The
* Runnable that will be executed. delayMillis The delay (in
* milliseconds) until the Runnable will be executed. Returns: Returns
* true if the Runnable was successfully placed in to the message queue.
* Returns false on failure, usually because the looper processing the
* message queue is exiting. Note that a result of true does not mean
* the Runnable will be processed -- if the looper is quit before the
* delivery time of the message occurs then the message will be dropped.
* 上面代码翻译如下:
* 该方法将一个 Runnable 对象 r 添加到消息队列，在指定的时间后会被执行。
* 这个 Runnable 对象会运行在当前 handler 所在的线程中。
* 第一个参数：Runnable 要执行的任务
* 第二个参数：delayMillis(单位：毫秒) runnable 任务被执行前的延迟时间
* 返回值：boolean ，如果该 Runnable 被成功添加到消息队列则返回 true，否则
返回 false
* 不过，通常返回 false 是因为 looper 进程处理消息队列退出。
* 注意：返回 true 不代表着 Runnable 被执行，如果 looper 在延时任务还没被执行
前退出了，那么消息就丢失掉了。
        */
       boolean flag = handler.postDelayed(new Runnable() {
           @Override
           public void run() {
           }
       }, 2000);
```

2.4 postAtTime

该方法也属于 Handler 对象，唯一不同的是该方法设置的定时任务是一个绝对时间，指的是 Android 系统的开机时间，如果想设置从当前时间算起 2 秒后执行该任务则可以将时间这样写：`SystemClock.uptimeMillis()+2000`，其中 SystemClock.uptimeMillis()是系统运行时间。

```
/**
* Causes the Runnable r to be added to the message queue, to be run at
* a specific time given by uptimeMillis. The time-base is
* android.os.SystemClock.uptimeMillis. The runnable will be run on the
* thread to which this handler is attached. Parameters: r The Runnable
* that will be executed. uptimeMillis The absolute time at which the
* callback should run, using the android.os.SystemClock.uptimeMillis
* time-base. Returns: Returns true if the Runnable was successfully
* placed in to the message queue. Returns false on failure, usually
* because the looper processing the message queue is exiting. Note that
* a result of true does not mean the Runnable will be processed -- if
* the looper is quit before the delivery time of the message occurs
* then the message will be dropped.
* 意译：给消息队列发出一个消息，让指定的任务在指定的时间执行。这里的时间是
绝对时间，是相对于 android.os.SystemClock.uptimeMillis 的时间
* 如果我们想在当前时间的 2 秒后执行该任务则将时间设置为：
SystemClock.uptimeMillis()+2000 即可。
        */
       boolean postAtTime = handler.postAtTime(new Runnable() {

           @Override
           public void run() {
               // TODO Auto-generated method stub
           }
       }, SystemClock.uptimeMillis()+2000);
   }
```

3.网络编程中常用的框架（★★★）

3.1 开源框架 SmartImageView

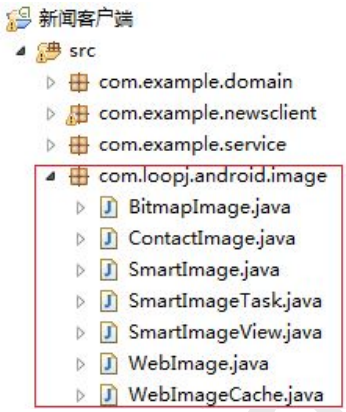
此框架提供很多实用的功能，其中可使用 setImageUrl 方法将一个网络图片的地址赋值给此控件，此控件内部的处理逻辑将访问网络获取指定 URL 的图片资源，并显示到客户

端。该框架的使用很简单。

★使用步骤：

1. 登录 github 开源大仓库，下载此开源控件。或者使用我在百度云盘上提供的附件。*http://pan.baidu.com/s/1o6MDhJs*。

2. 添加此开源框架的包到工程下



3. 在布局页面使用此控件：必须使用包名.类名这样的全路径，否则无法找到

```
<!-- 引入第三方控件用于显示图片 -->
  <com.loopj.android.image.SmartImageView
      android:id="@+id/iv"
      android:layout_height="70dp"
     android:layout_width="70dp"
     android:contentDescription="news"
      />
```

4. 调用 setImageUrl,将图片的网络地址赋值给控件即可

```
//实例化 SmartImageView 对象
SmartImageView imageView =
(SmartImageView)convertView.findViewById(R.id.iv);
//给 SmartImageView 设置一个图像的 URL 则 SmartImageView 会自动加载图片
imageView.setImageUrl(news.getImage());
```

3.2 HttpClient

HttpClient 是 apache 开源组织研发的一个 API，被 Android 引入使用。（URL 以及 HttpUrlConnection 是 java 自带的 API）。 HttpClient 设计的思想是模拟浏览器的操作来实现网络访问。

使用步骤：

1. 定义一个客户端对象：即获得一个 HttpClient 对象（打开浏览器）
2. 定义请求方法（输入网址）：Get——HttpGet/POST——HttpPost
3. 设置请求的参数/请求头信息/连接超时时间/读取数据超时时间等
4. 执行请求(敲回车)：execute 方法——此方法会返回一个 HttpResponse 对象
5. 获取状态码：response.getStatusLine().getStatusCode()
6. 若状态码是 200，获取服务器返回的数据:

   InputStream is=response.getEntity().getContent()
7. 操作结束后断开连接：

   client.getConnectionManager().shutdown()

3.2.1 使用 HttpClient 发送 GET 请求

为了方便演示，我们直接使用本文档 1.5 章节的工程，只需修改发送用户数据的核心方法即可。

代码清单如下：

```
String path = PATH+"?username="+username+"&pwd="+pwd;
                //获取一个客户端对象
                HttpClient client = new DefaultHttpClient();
                //定义请求方式，并传递请求地址
                HttpGet httpGet = new HttpGet(path);
                //获取参数对象
                HttpParams params = httpGet.getParams();
                //设置链接超时时间
                HttpConnectionParams.setConnectionTimeout(params,
5000);
                //设置读取数据超时时间
                HttpConnectionParams.setSoTimeout(params, 5000);
                //开始发送请求，同时返回 HttpResponse 对象,该句代码属于耗
时操作
                HttpResponse httpResponse = client.execute(httpGet);
                //获取状态码
                int code =
httpResponse.getStatusLine().getStatusCode();
                if (code==200) {
                    //获取返回数据流对象
                    InputStream in =
httpResponse.getEntity().getContent();
                    ByteArrayOutputStream bos = new
ByteArrayOutputStream();
                    byte[] buff = new byte[1024];
                    int len = -1;
                    while((len=in.read(buff))!=-1){
                        bos.write(buff, 0, len);
                    }
                    in.close();
                    result = bos.toString();
                    bos.close();

                }else {
                    result = "请求失败！";
                }
//关闭链接
client.getConnectionManager().shutdown();
```

3.2.2 使用 HttpClient 发送 POST

使用上节中的工程，只需修改核心功能代码即可。代码清单如下：

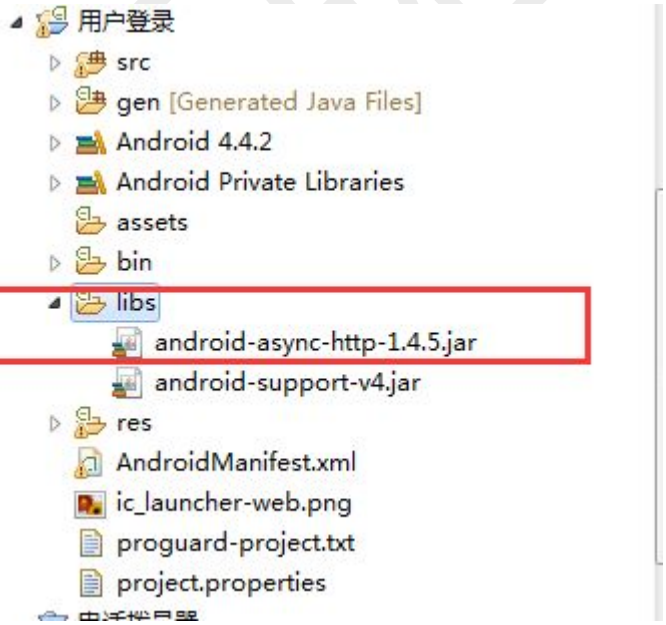
```
//创建客户端对象
HttpClient client = new DefaultHttpClient();
//创建一个 POST 请求对象
HttpPost httpPost = new HttpPost(PATH);
//获取请求对象的参数
HttpParams httpParams = httpPost.getParams();
//设置链接超时时间
HttpConnectionParams.setConnectionTimeout(httpParams, 5000);
//设置读取数据流超时时间
HttpConnectionParams.setSoTimeout(httpParams, 5000);
//设置请求参数集合，将请求参数都添加到该集合
List<BasicNameValuePair> parameters = new
ArrayList<BasicNameValuePair>();
//将用户信息添加到集合中
parameters.add(new BasicNameValuePair("username", username));
parameters.add(new BasicNameValuePair("pwd", pwd));
//创建一个请求体对象
UrlEncodedFormEntity entity = new UrlEncodedFormEntity(parameters );
//给 httpPost 对象设置请求体
httpPost.setEntity(entity);
HttpResponse httpResponse = client.execute(httpPost);
//获取状态码
int code = httpResponse.getStatusLine().getStatusCode();
if (code==200) {
//获取返回数据流对象
InputStream in = httpResponse.getEntity().getContent();
ByteArrayOutputStream bos = new ByteArrayOutputStream();
byte[] buff = new byte[1024];
int len = -1;
while((len=in.read(buff))!=-1){
   bos.write(buff, 0, len);
            }
in.close();
result = bos.toString();
bos.close();
```

```
            }else {
                result = "请求失败！";
            }
            //关闭链接
            client.getConnectionManager().shutdown();
```

3.3 开源框架 AsyncHttpClient

AsyncHttpClient 是一个 Android 的异步 HTTP 函数库，使用此开源项目访问网络时，**无需开启子线程**，因为调用其方法时传递的接口参数中的方法就是执行在主线程中的，使用起来非常方便。登录 github，搜索 async-http,下载此开源项目，添加 jar 包到工程下的 lib 目录，直接使用即可。百度云盘下载地址：http://pan.baidu.com/s/1qWz80Ew

为了演示 AsyncHttpClient 的使用，我们依然使用上一个章节的《用户登陆》工程，只需将下载的 jar 包添加到工程中，如图所示。



3.3.1 使用 GET 方式提交数据给服务器

这里只在原有工程的基础上修改核心代码，代码清单如下：

```
//创建一个 AsyncHttpClient 对象
      AsyncHttpClient client = new AsyncHttpClient();
      //调用 get 方法，发送 get 方式的请求
      client.get(PATH, new AsyncHttpResponseHandler() {
          /**
           * 处理成功的结果
           * statusCode 状态码
           * headers 响应头消息
           * responseBody 相应的主体内容
           */
          @Override
          public void onSuccess(int statusCode, Header[] headers,
byte[] responseBody) {
              String result = new String(responseBody);
              Toast.makeText(MainActivity.this, "请求结果: "+result,
1).show();
          }
          /**
           * 处理失败的结果
           * statusCode 状态码
           * headers 响应头消息
           * responseBody 相应的主体内容
           */
          @Override
          public void onFailure(int statusCode, Header[] headers,
byte[] responseBody, Throwable throwable) {
              Toast.makeText(MainActivity.this, "请求网络失败！",
0).show();
          }
      });
```

3.3.2 使用 POST 方式提交数据给服务器

基本逻辑与 GET 方式相同，不同的是 GET 方式调用的是 GET 方法，POST 方式调用的

是 POST 方法；GET 方式将请求参数拼接在 URL 后面传输给服务器，POST 方式需要使用 RequestParams 对象封装请求参数给 post。

```
AsyncHttpClient client = new AsyncHttpClient();

        RequestParams params = new RequestParams();
        params.put("username", "heima");
        params.put("pwd", "12346");
        client.post(PATH, params,new AsyncHttpResponseHandler() {
        //省略其他一样的代码
```

3.3.3 上传文件到服务器

上传文件其实用的就是 POST 方法，只需要给 RequestParams 对象设置一个 File 类型的参数即可。实例代码如下：

```
AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        params.put("username", "heima");
        params.put("pwd", "12346");
        File file = new
File(Environment.getExternalStorageDirectory().getAbsoluteFile()+"a
.txt");
        params.put("file", file);
                client.post(PATH, params,new AsyncHttpResponseHandler()
    {
```

至此，本文档完！