第 4 天 Android 基础

第四章 网络编程（上）

- 使用 HttpURLConnection 访问网络
- Handler 机制
- 案例-新闻客户端
- 使用 SmartImageView 加载图片

1.1 网络编程预备知识

1.1.1 访问网络需要的权限

访问网络属于“侵犯用户利益”行为，因此必须在工程的 AndroidManifest.xml 中声明对应的权限。

【文件 1-1】 添加网络权限

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

1.1.2 ANR 异常

Application Not Response，应用程序无响应，简称 ANR 异常。在主线程中做一些耗时的操作，比如网络访问、文件拷贝等阻塞了主线程，导致主线程无法响应，这时就会报 ANR 异常。

因此对于耗时操作，必须在子线程中完成。

1.1.3 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，因此只有主线程才能修改界面，别的线程不允许修改，否则会报如下错误：

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

注意：

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？这个时候就应该使用 Handler 实现子线程与主线程之间的通信了。

关于 Handler 的使用会在接下来的案例中介绍，这里先点到为止。

1.1.4 主线程不能访问网络

Google 工程师考虑到网络访问是一个比较耗时的操作，从 Android 4.0 开始，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
```

因此，访问网络的操作必须在子线程中进行。

1.2 案例-网络源码查看器

通过该案例可以对 Android 的网络访问有个初步的认识。

1.2.1 需求

如图 1-1 所示。当用户在界面顶部的编辑框（EditText）中输入地址后，点击确定按钮，就开始访问网络，然后将获取到的网页源码显示在下部的 TextView 中，由于一个界面是显示不全如此多的网页源码的，因此需要用 ScrollView 将 TextView 给包裹起来。

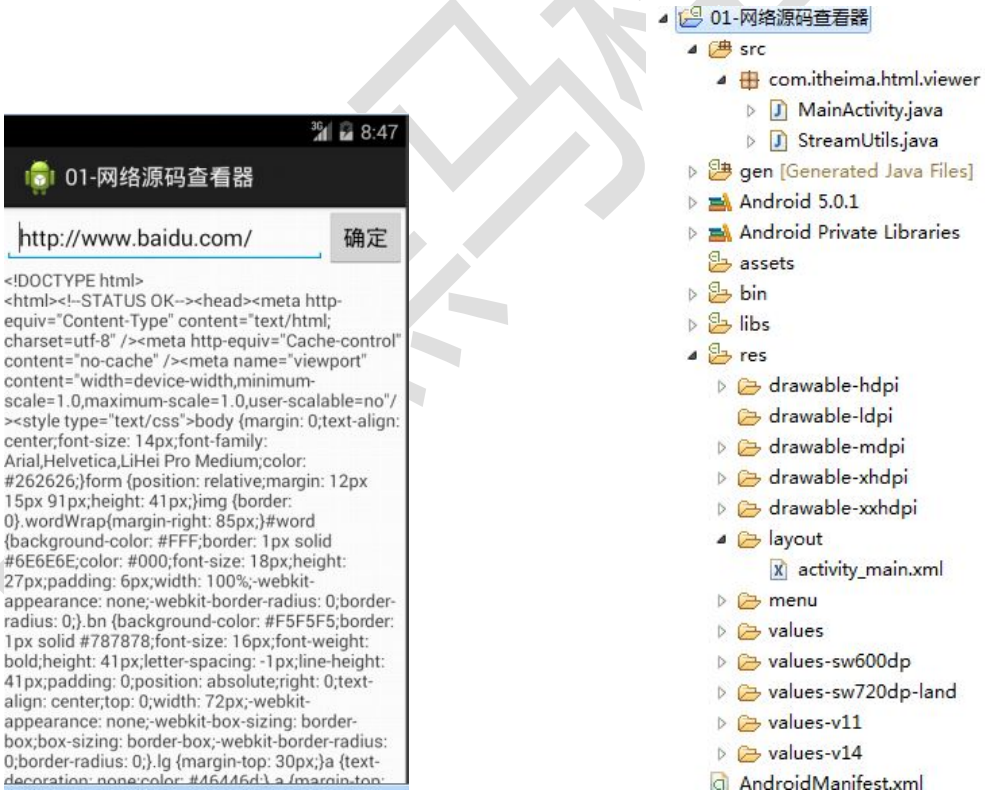


图 1-1　　网络源码查看器

1.2.2 编写布局

【文件 1-2】 activity_main.xml

```
1.    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <LinearLayout
8.         android:layout_width="match_parent"
9.         android:layout_height="wrap_content"
10.        android:orientation="horizontal" >
11.
12.        <EditText
13.            android:id="@+id/et_url"
14.            android:layout_width="0dp"
15.            android:layout_height="wrap_content"
16.            android:layout_weight="1"
17.            android:hint="请输入地址"
18.            android:text="http://www.baidu.com/" />
19.
20.        <Button
21.            android:layout_width="wrap_content"
22.            android:layout_height="wrap_content"
23.            android:onClick="load"
24.            android:text="确定" />
25.    </LinearLayout>
26.
27.    <ScrollView
28.        android:layout_width="match_parent"
29.        android:layout_height="0dp"
30.        android:layout_weight="1" >
31.
32.        <TextView
33.            android:id="@+id/tv_content"
34.            android:layout_width="match_parent"
35.            android:layout_height="wrap_content" />
36.    </ScrollView>
37.
38. </LinearLayout>
```

1.2.3 编写代码

【文件 1-3】 MainActivity.java

```
1.  package com.itheima.html.viewer;
```

```
2.
3. import java.io.InputStream;
4. import java.net.HttpURLConnection;
5. import java.net.URL;
6. import android.os.Bundle;
7. import android.os.Handler;
8. import android.os.Message;
9. import android.app.Activity;
10. import android.util.Log;
11. import android.view.View;
12. import android.widget.EditText;
13. import android.widget.TextView;
14. import android.widget.Toast;
15. /**
16.  * 网络源码查看器。
17.  *
18.  * @author wzy 2015-11-4
19.  *
20.  */
21. public class MainActivity extends Activity {
22.
23. private TextView tv_content;
24. private EditText et_url;
25. /*
26.  * 创建一个 Handler 对象，覆写类体 方法体
27.  */
28. private Handler handler = new Handler(){
29.     /*
30.      * 覆写 handleMessage 方法，在该方法中完成我们想做的工作，
31.      * 该方法是在主线程中被调用的，因此可以在这里修改 UI。
32.      */
33.     @Override
34.     public void handleMessage(Message msg) {
35.         //判断 Message 的类型，根据 msg 的 what 属性去获取其类型
36.         switch (msg.what) {
37.         //如果成功
38.         case RESULT_OK:
39.             //从 msg 的 obj 属性中获取数据，然后显示在 TextView 上。
40.             tv_content.setText(msg.obj.toString());
41.             break;
42.             //如果失败
43.         case RESULT_CANCELED:
44.             //弹吐司，给用户以提示
45.             Toast.makeText(MainActivity.this, "访问网页失败",
46.  Toast.LENGTH_SHORT).show();
```

```
47.             break;
48.         default:
49.             break;
50.         }
51.     };
52. };
53.
54. @Override
55. protected void onCreate(Bundle savedInstanceState) {
56.     super.onCreate(savedInstanceState);
57.     setContentView(R.layout.activity_main);
58.     //初始化控件
59.     et_url = (EditText) findViewById(R.id.et_url);
60.     tv_content = (TextView) findViewById(R.id.tv_content);
61. }
62.
63. /**
64.  * 加载网页源码
65.  *
66.  * @param view
67.  */
68. public void load(View view) {
69.     //获取用户输入的地址
70.     final String path = et_url.getText().toString().trim();
71.         /*
72.          * 网络访问必须在子线程中进行
73.          */
74.         new Thread(new Runnable() {
75.
76.             @Override
77.             public void run() {
78.                 try {
79.                     //1.创建一个 URL 对象，需要传入 url
80.                     URL url = new URL(path);
81.                     /*
82.                      * 2.使用 url 对象打开一个 HttpURLConnection，
83. 由于其返回的是 HttpURLConnection 的父类，
84.                      * 这里可以强制类型转换
85.                      */
86.                     HttpURLConnection connection =
87. (HttpURLConnection) url.openConnection();
88.                     /*
89.                      * 3. 配置 connection 连接参数
90.                      */
```

```
91.                     //设置联网超时时长，单位毫秒
92.                     connection.setConnectTimeout(5000);
93.                     /*
94.                      * 设置数据读取超时
95.                      * 注意：不是指读取数据总耗时超时，而是能够读取到数据流等待时长
96.                      */
97.                     connection.setReadTimeout(5000);
98.                     /*
99.                      * 设置请求方式，默认是 GET，但是为了增加代码易读性，建议显示指明为 GET
100.                     */
101.                    connection.setRequestMethod("GET");
102.                    //4.开始连接网络
103.                    connection.connect();
104.                    //5.以字节输入流的形式获取服务端发来的数据
105.                    InputStream inputStream = connection.getInputStream();
106.                    //6.将字节流转化为字符串（使用了自定义的 StreamUtils 工具类）
107.                    final String data =
108. StreamUtils.inputStream2String(inputStream);
109.                    /*
110.                     * 7.将获取到的数据封装到 Message 对象，然后发送给 Handler
111.                     */
112.                    //创建一个 Message 对象
113.                    Message msg = new Message();
114.                    /*
115.                     * 给 Message 对象的 what 属性设置一个 int 类型的值。
116.                     * 因为消息可能会有多个，因此为了区分这些不同的消息，
117.                     * 需要给消息设置 what 属性。
118.                     * RESULT_OK 是 Activity 的常量值为-1，
119.                     * 当然也可以自定义一个 int 类型的值。
120.                     */
121.                    msg.what = RESULT_OK;
122.                    /*
123.                     * 给 Message 对象的 obj 属性设置一个 Object 类型的值。
124.                     * 该值正式我们需要在 Message 对象上绑定的数据，这里绑定的
125.                     * 从网络上获取到的网页源码字符串。
126.                     */
127.                    msg.obj = data;
128.                    /*
129.                     * 给主线程发送消息。
130.                     * 发送后，系统会调用 handler 对象的 handleMessage（Message）方法，
131.                     * 该方法正是我们自己实现的。而且该方法是在主线程中执行的。
132.                     * 从而就实现了从子线程中访问网络数据，然后交给主线程，让主线程修改 UI。
133.                     *
134.                     */
135.                    handler.sendMessage(msg);
```

```
136.
137.                     } catch (Exception e) {
138.                         e.printStackTrace();
139.                         Log.d("tag", "遇到异常"+e,e);
140.                         /*
141.                          * 如果遇到异常，最好让主线程也知道子线程遇到异常了。
142.                          * 因此使用 handler 发送一个空消息，所谓的空消息是指
143.                          * 该消息没有 obj 值，只有一个 what 属性。
144.                          * 这里的 RESULT_CANCELED 就是一个 int 型的常量，
145.                          * 当然我们可以自定义。这里只不过是直接使用了 Activity 类的
146.                          * 一个常量值而已。
147.                          * 该消息发送后，系统依然会调用 handler 对象的
148. handleMessage（Message）
149.                          * 方法。
150.                          */
151.                         handler.sendEmptyMessage(RESULT_CANCELED);
152.                     }
153.                 }
154.             }).start();
155.
156.         }
157.     }
158.
159.
```

注意：

上面使用了一个将字节输入流转化为字符串的工具类，其源码如【文件 1-4】所示。

【文件 1-4】 StreamUtils.java

```
1. package com.itheima.html.viewer;
2.
3. import java.io.ByteArrayOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. /**
7.  * 将字节输入流转化为字符串的工具类。
8.  *
9.  * @author wzy 2015-10-25
10.  *
11.  */
12. public class StreamUtils {
13. /**
14.  * 将字节流转化为字符串,使用 Android 默认编码
15.  * @param inputStream
16.  * @return
17.  * @throws IOException
```

```
18.  */
19. public static String inputStream2String(InputStream inputStream) throws IOException{
20.     ByteArrayOutputStream baos = new ByteArrayOutputStream();
21.     int len = -1;
22.     byte[] buffer = new byte[1024];
23.     while((len = inputStream.read(buffer))!=-1){
24.         baos.write(buffer, 0, len);
25.     }
26.     inputStream.close();
27.     return new String(baos.toByteArray());
28. }
29. }
30.
```

1.2.4 添加权限

网络访问必须添加对应的权限。需要添加的权限见本文 1.1.1 节。

1.3 Handler 机制

1.3.1 Handler 机制

Android 的 Handler 机制（也有人叫消息机制）目的是为了跨线程通信，也就是多线程通信。之所以需要跨线程通信是因为在 Android 中主线程通常只负责 UI 的创建和修改，子线程负责网络访问和耗时操作，因此，主线程和子线程需要经常配合使用才能完成整个 Android 功能。

Handler 机制可以近似用图 1-2 展示。MainThread 代表主线程，newThread 代表子线程。

MainThread 是 Android 系统创建并维护的，创建的时候系统执行了 Looper.prepare();方法，该方法内部创建了 MessageQueue 消息队列（也叫消息池），该消息队列是 Message 消息的容器，用于存储通过 handler 发送过来的 Message。MessageQueue 是 Looper 对象的成员变量，Looper 对象通过 ThreadLocal 绑定在 MainThread 中。因此我们可以简单的这么认为：MainThread 拥有唯一的一个 Looper 对象，该 Looper 对象有用唯一的 MessageQueue 对象，MessageQueue 对象可以存储多个 Message。

MainThread 中需要程序员手动创建 Handler 对象，并覆写 Handler 中的 handleMessage(Message msg)方法，该方法将来会在主线程中被调用，在该方法里一般会写与 UI 修改相关的代码。

MainThread 创建好之后，系统自动执行了 Looper.loop();方法，该方法内部开启了一个“死循环”不断的去之前创建好的 MessageQueue 中取 Message。如果一有消息进入 MessageQueue，那么马上会被 Looper.loop();取出来，取出来之后就会调用之前创建好的 handler 对象的 handleMessage（Message）方法。

newThread 线程是我们程序员自定 new 出来的子线程。在该子线程中处理完我们的“耗时”或者网络访问任务后，调用主线程中的 handler 对象的 sendMessage（msg）方法，该方法一被执行，内部将就 msg 添加到了主线程中的 MessageQueue 队列中，这样就成为了 Looper.loop()的盘中餐了，等待着被消费。

上面的过程有点类似黑马 Java 基础班学习多线程时的生产者和消费者的过程。newThread 属于生产者，负责生产 Message，MainThread 属于消费者。这是一个很复杂的过程，但是 Android 显然已经将这种模式

给封装起来了，就叫 Handler 机制。我们使用时只需要在主线程中创建 Handler，并覆写 handler 中的 handleMessage 方法，然后在子线程中调用 handler 的 sendMessage（msg）方法即可。

图 1-2　Handler 原理图

1.3.2　获取 Message 的多种写法

在 Hanlder 机制中都需要用到 Message 对象，该对象的创建或者获取有多种方式。

1. Message msg = new Message(); 直接创建一个新的 Message 对象。

2. Message msg = handler.obtainMessage(); 通过 handler 对象获取一个 Message 对象，该对象从消息缓存池中获取的，可以提高 Message 的使用率，减少垃圾回收次数。

3. handler.post(Runnable r); post（）方法中传递一个 Runnable 对象，那么该对象会被主线程执行。该方法表面上看跟 Message 没任何关系，但是其内部帮我们封装了 Message。

1.4　案例-网络图片查看器

通过该案例可以学习如何将网络图片显示在到用户界面。

1.4.1　需求

如图 1-3，在编辑框输入图片的 url 地址，然后点击右侧的确定按钮。然后 app 就开始访问网络，将网络上的图片显示在界面。

图 1-3　网络图片查看器

1.4.2　编写布局

【文件 1-5】 activity_main.xml

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:orientation="vertical" >
6.
7.      <LinearLayout
8.          android:layout_width="match_parent"
9.          android:layout_height="wrap_content"
10.         android:orientation="horizontal" >
11.
12.         <EditText
13.             android:id="@+id/et_url"
14.             android:layout_width="0dp"
15.             android:layout_height="wrap_content"
16.             android:layout_weight="1"
17.             android:text="http://bbs.itheima.com/data/attachment/forum
18. /201511/04/121947sci9qd732yyyyyh4.jpg"
```

```
19.            android:hint="请输入 url 地址" />
20.
21.        <Button
22.            android:layout_gravity="center_vertical"
23.            android:layout_width="wrap_content"
24.            android:layout_height="wrap_content"
25.            android:onClick="load"
26.            android:text="确定" />
27.    </LinearLayout>
28.
29.    <ImageView
30.        android:layout_gravity="center_horizontal"
31.        android:id="@+id/iv"
32.        android:layout_width="wrap_content"
33.        android:layout_height="wrap_content" />
34.
35. </LinearLayout>
```

注意：

上面第 17 行布局代码中我将一个网络图片的地址“写死”了，因为图片地址的 url 太长了，因此仅仅为了方便演示才这么做的。大家在练习的时候可以将图片放到本地 tomcat 服务器中，然后通过本机 ip 地址进行访问。

1.4.3 编写代码

【文件 1-6】 MainActivity.java

```
1. package com.itheima.image.viewer;
2. import java.io.InputStream;
3. import java.net.HttpURLConnection;
4. import java.net.URL;
5. import android.os.Bundle;
6. import android.os.Handler;
7. import android.os.Message;
8. import android.app.Activity;
9. import android.graphics.Bitmap;
10. import android.graphics.BitmapFactory;
11. import android.view.View;
12. import android.widget.EditText;
13. import android.widget.ImageView;
14. import android.widget.Toast;
15. /**
16.  * 网络图片查看器
17.  *
18.  * @author wzy 2015-11-4
```

```
 *
 */
public class MainActivity extends Activity {

private EditText et_url;
private ImageView iv;
private Handler handler = new Handler(){
    @Override
    public void handleMessage(android.os.Message msg) {
        switch (msg.what) {
        case RESULT_OK://请求成功
            /*
             * 从 msg 的 obj 属性中拿到 bitmap 对象
             * 然后通过 iv.setImageBitmap(bitmap);
             * 方法将图片设置给 ImageView 对象。
             */
            iv.setImageBitmap((Bitmap)msg.obj);
            break;
        case RESULT_CANCELED:
            Toast.makeText(MainActivity.this,
"访问网络失败！", Toast.LENGTH_LONG).show();
            break;

        default:
            break;
        }
    };
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //初始化控件
    et_url = (EditText) findViewById(R.id.et_url);
    iv = (ImageView) findViewById(R.id.iv);
}
/*
 * 点击图片绑定的点击事件
 * 完成加载图片逻辑。
 */
public void load(View view){
    /*
     * 开启子线程，负责网络图片的请求
     */
```

```
64.     new Thread(new Runnable() {
65.
66.         @Override
67.         public void run() {
68.             try {
69.                 //获取地址
70.                 String path = et_url.getText().toString().trim();
71.                 //创建 URL 对象
72.                 URL url = new URL(path);
73.                 //获取 HttpURLConnection 对象
74.                 HttpURLConnection connection =
75. (HttpURLConnection) url.openConnection();
76.                 //设置配置信息
77.                 //请求方法
78.                 connection.setRequestMethod("GET");
79.                 //设置连接超时
80.                 connection.setConnectTimeout(5000);
81.                 //设置数据读取超时
82.                 connection.setReadTimeout(5000);
83.                 //开启连接
84.                 connection.connect();
85.                 //获取输入流
86.                 InputStream inputStream = connection.getInputStream();
87.                 /*
88.                  * 使用 sdk 提供的 BitmapFactory 工具类将字节输入流转化为 Bitmap 对象
89.                  */
90.                 Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
91.                 //创建一个新的 message 对象
92.                 Message msg = new Message();
93.                 msg.what = RESULT_OK;
94.                 //将 bitmap 对象赋值给 msg 的 obj 属性
95.                 msg.obj = bitmap;
96.                 /*
97.                  * 发送消息，因为在界面显示图片也属于修改 UI 操作，
98.                  * 因此必须让子线程干这活
99.                  */
100.                    handler.sendMessage(msg);
101.                } catch (Exception e) {
102.                    e.printStackTrace();
103.                    handler.sendEmptyMessage(RESULT_CANCELED);
104.                }
105.            }
106.        }).start();
107.    }
```

```
108.
109. }
110.
```

新技能：

上面代码第 35、90 行使用了两个新 api。

1、ImageView.setImageBitmap(Bitmap);

给 ImageView 对象设置 Bitmap 对象的图片。由于 ImageView 对象是布局中的一个控件，因此该方法属于修改 UI 操作，必须在主线程中调用。

2、Bitmap bitmap = BitmapFactory.decodeStream(InputStream is);

将字节输入流转换为 Bitmap 对象，该方法仅仅是将流转化为 Bitmap 对象，并没有牵扯到 UI 的东西，因此可以在子线程中使用。

1.4.4 添加权限

网络访问必须添加对应的权限。需要添加的权限见本文 1.1.1 节。

1.5 案例-新闻客户端

该案例属于一个综合案例，不仅用到了网络访问、图片加载、xml 的解析、handler 机制、线程池的使用还用到了 ListView 及其优化策略。

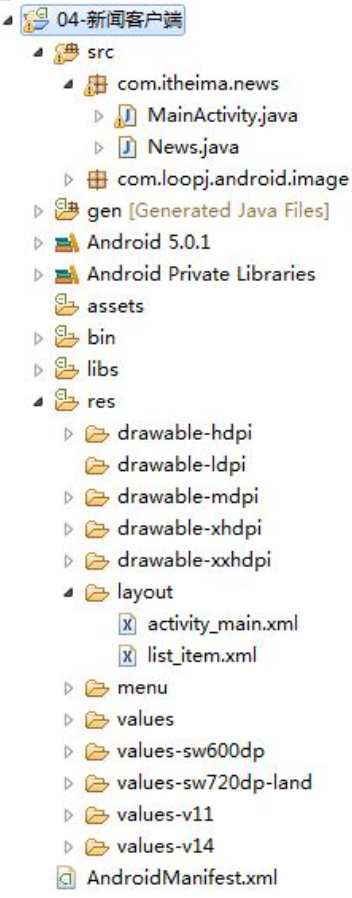
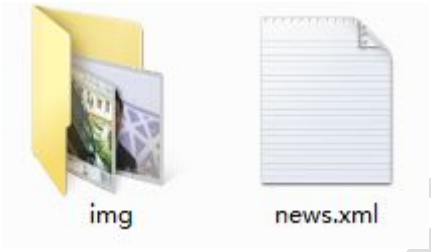


图 1-4　新闻客户端

1.5.1　前提准备

由于该案例需要用到网络数据，因此需要先搭建服务器。服务器的搭建也很简单，直接将如下图片和 news.xml 放到文件夹中（我起名叫 news），然后将该文件夹放到 tomcat 的 webapps 目录下，最后启动 tomcat 即可。



【文件 1-7】 news.xml 文件内容

```
1. <?xml version="1.0" encoding="UTF-8" ?>
2. <channel>
3. <item>
4.   <title>军报评徐才厚</title>
5.   <description>人死账不消 反腐步不停，支持，威武，顶，有希望了。
6.   </description>
7.   <image>http://10.0.2.2:8080/news/img/a.jpg</image>
8.   <type>1</type>
9.   <comment>163</comment>
10.   </item>
11.  <item>
12.   <title>女司机翻车后直奔麻将室</title>
13.   <description>女司机翻车后直奔麻将室，称大难不死手气必红
14.   </description>
15.   <image>http://10.0.2.2:8080/news/img/b.jpg</image>
16.   <type>2</type>
17.   </item>
18.   <item>
19.   <title>小伙当“男公关”以为陪美女</title>
20.   <description>来源：中国青年网，小伙当“男公关”以为陪美女，上工后被大妈吓怕  </description>
21.   <image>http://10.0.2.2:8080/news/img/c.jpg</image>
22.   <type>3</type>
23.   </item>
24.   <item>
25.   <title>男子看上女孩背影欲强奸</title>
26.   <description> 来源：新京报， 看到正脸后放弃仍被捕
27.   </description>
28.   <image>http://10.0.2.2:8080/news/img/d.jpg</image>
29.   <type>1</type>
```

```
30.   <comment>763</comment>
31.   </item>
32. </channel>
```

注意：

上面 xml 代码中<image>标签中的 url 是对应图片的地址。加载图片的实质就是先从 xml 中获取到图片的 url 地址，然后从网络中再获取其对应的图片。

1.5.2 需求分析

如图 1-4 所示。主界面是一个 ListView，ListView 的每一个条目都是一个复杂条目，ListView 中的所有数据全部来自网络。网络数据

步骤如下：

1、初始化网络数据

1. 访问网络获取 news.xml
2. 解析 news.xml，并将里面的每一个 item 封装成一个 JavaBean
3. 解析好数据后通过 Handler 发送消息给主线程，让主线程给 ListView 设置 Adapter
   因为网络访问必须在子线程中，而给 ListView 设置 Adapter 属于修改 UI 操作，因此当解析好网络数据后需要通过 Handler 发送消息。

2、给 ListView 设置适配器

1. 自定义一个 MyAdapter 继承 BaseAdapter
2. 覆写 getCount()和 getView()方法
3. getCount()方法返回新闻的条目个数
4. getView()中显然要将一个单独的布局填充为 View 对象
   通过 View.findViewById()将 list_item.xml 中的各个控件初始化，然后设置相应的值。其中 ImageView 需要处理。
5. 因为 getView()方法也是在主线程中被调用的，而此时需要在该方法中从网络上加载图片，因此需要在该方法中再开启一个子线程负责下载图片，然后通过 Handler 发送消息将图片显示在界面。

3、编写 ListView 的条目布局 list_item.xml

1.5.3 编写布局

【文件 1-8】 activity_main.xml

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.    xmlns:tools="http://schemas.android.com/tools"
3.    android:layout_width="match_parent"
4.    android:layout_height="match_parent"
5.    android:orientation="vertical">
6.
7.    <ListView
8.       android:id="@+id/lv"
9.       android:layout_width="match_parent"
```

```
10.         android:layout_height="0dp"
11.         android:layout_weight="1"
12.          />
13.
14. </LinearLayout>
```

【文件 1-9】 list_item.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content">
5.     <ImageView
6.        android:id="@+id/iv"
7.        android:layout_width="100dp"
8.        android:layout_height="80dp"
9.        android:layout_marginRight="5dp"
10.        android:background="@drawable/ic_launcher"
11.        />
12.     <TextView
13.        android:id="@+id/tv_title"
14.        android:layout_toRightOf="@id/iv"
15.        android:layout_width="wrap_content"
16.        android:layout_height="wrap_content"
17.        android:singleLine="true"
18.        android:ellipsize="middle"
19.        android:text="新闻的标题"
20.        android:textColor="#000000"
21.        android:textSize="20sp"
22.        />
23.     <TextView
24.        android:id="@+id/tv_desc"
25.        android:layout_width="wrap_content"
26.        android:layout_height="wrap_content"
27.        android:layout_toRightOf="@id/iv"
28.        android:layout_alignBottom="@id/iv"
29.        android:maxLines="3"
30.        android:text="这是新闻简介"
31.         android:layout_marginRight="15dp"
32.        />
33.     <TextView
34.      android:layout_alignBottom="@id/iv"
35.        android:layout_alignParentRight="true"
36.        android:layout_width="wrap_content"
37.        android:layout_height="wrap_content"
```

```
38.         android:id="@+id/tv_type"
39.         android:text="专题"
40.         android:textColor="#ff0000"
41.         />
42. </RelativeLayout>
43.
```

1.5.4 编写代码

【文件 1-10】 MainActivity.java

```
1.  package com.itheima.news;
2.
3. import java.io.IOException;
4. import java.io.InputStream;
5. import java.net.HttpURLConnection;
6. import java.net.URL;
7. import java.util.ArrayList;
8. import java.util.concurrent.ExecutorService;
9. import java.util.concurrent.Executors;
10. import org.xmlpull.v1.XmlPullParser;
11. import org.xmlpull.v1.XmlPullParserException;
12. import android.os.Bundle;
13. import android.os.Handler;
14. import android.app.Activity;
15. import android.graphics.Bitmap;
16. import android.graphics.BitmapFactory;
17. import android.util.Xml;
18. import android.view.View;
19. import android.view.ViewGroup;
20. import android.widget.BaseAdapter;
21. import android.widget.ImageView;
22. import android.widget.ListView;
23. import android.widget.TextView;
24. /**
25.  * 新闻客户端
26.  *
27.  * @author wzy 2015-11-4
28.  *
29.  */
30. public class MainActivity extends Activity {
31. /*
32.  * news.xml 文件的网络地址
33.  */
34. private static final String PATH = "http://10.0.2.2:8080/news/news.xml";
```

```
35. /*
36.  * 联网超时
37.  */
38. private static final int TIME_OUT = 5000;
39. private ListView lv;
40. /*
41.  * 用于存储新闻条目的集合
42.  */
43. private ArrayList<News> data = new ArrayList<News>();
44. /*
45.  * 定义一个缓存线程池
46.  */
47. private ExecutorService executorService = Executors.newCachedThreadPool();
48. //新建一个 Handler
49. private Handler handler = new Handler(){
50.     @Override
51.     public void handleMessage(android.os.Message msg) {
52.         /*
53.          * 当网络加载好之后发送消息
54.          * 然后给 ListView 设置适配器
55.          */
56.         if (msg.what==RESULT_OK) {
57.             /*
58.              * 给 ListView 设置适配器
59.              * 因为设置适配器也属于修改 UI 操作，因此该方法必须在主线程中
60.              */
61.             lv.setAdapter(new MyAdapter());
62.         }
63.     };
64. };
65.
66. @Override
67. protected void onCreate(Bundle savedInstanceState) {
68.     super.onCreate(savedInstanceState);
69.     setContentView(R.layout.activity_main);
70.     //获取 ListView 控件
71.     lv = (ListView) findViewById(R.id.lv);
72.     //初始化网络数据
73.     initData();
74. }
75.
76. /**
77.  * 从网络中加载数据
78.  */
```

```
79. private void initData() {
80.     //开启子线程，在线程中访问 news.xml
81.     new Thread(new Runnable() {
82.
83.         @Override
84.         public void run() {
85.             try {
86.                 URL url = new URL(PATH);
87.                 HttpURLConnection connection =
88. (HttpURLConnection) url.openConnection();
89.                 connection.setRequestMethod("GET");
90.                 connection.setReadTimeout(TIME_OUT);
91.                 connection.setConnectTimeout(TIME_OUT);
92.                 connection.connect();
93.                 InputStream inputStream = connection.getInputStream();
94.                 /*
95.                  *  解析 xml
96.                  *  自定义方法 parseNewsFromNet，将字节输入流中的 xml 数据进行解析
97. 然后封装数据到集合中
98.                  */
99.                 ArrayList<News> datas = parseNewsFromNet(inputStream);
100.                    if (datas!=null) {
101.                        //将解析好的数据赋值给 MainActivity 对象的成员变量
102.                        data = datas;
103.                    }
104.                    inputStream.close();
105.                    connection.disconnect();
106.                    /*
107.                     * 将数据解析好后消息告诉主线程
108.                     * 之所以发送空消息，是因为数据已经赋值给了 MainActivity 对象的成员变量
109.                     */
110.                    handler.sendEmptyMessage(RESULT_OK);
111.                } catch (Exception e) {
112.                    e.printStackTrace();
113.                }
114.            }
115.        }).start();
116.
117.
118.    }
119.
120.    /*
121.     * 从网络流中解析 xml
122.     * 将解析好的数据存储在集合中
123.     */
```

```
124.    private ArrayList<News> parseNewsFromNet(InputStream inputStream)
125.  throws XmlPullParserException, IOException {
126.        ArrayList<News> list = null;
127.        News news = null;
128.        XmlPullParser parser = Xml.newPullParser();
129.        parser.setInput(inputStream, "utf-8");
130.        int event = parser.next();
131.        while (event != XmlPullParser.END_DOCUMENT) {
132.            String name = parser.getName();
133.            if (event == XmlPullParser.START_TAG) {
134.                if ("channel".equals(name)) {
135.                    list = new ArrayList<News>();
136.                } else if ("item".equals(name)) {
137.                    news = new News();
138.                } else if ("title".equals(name)) {
139.                    news.setTitle(parser.nextText());
140.                } else if ("description".equals(name)) {
141.                    news.setDescription(parser.nextText());
142.                } else if ("image".equals(name)) {
143.                    news.setImage(parser.nextText());
144.                } else if ("type".equals(name)) {
145.                    news.setType(Integer.valueOf(parser.nextText()));
146.                } else if ("comment".equals(name)) {
147.                    news.setComment(Integer.valueOf(parser.nextText()));
148.                }
149.            } else if (event == XmlPullParser.END_TAG) {
150.                if ("item".equals(name)) {
151.                    list.add(news);
152.                }
153.            }
154.            event = parser.next();
155.        }
156.        return list;
157.    }
158.
159.    class MyAdapter extends BaseAdapter {
160.
161.        @Override
162.        public int getCount() {
163.            return data.size();
164.        }
165.
166.        @Override
167.        public Object getItem(int position) {
```

```
168.            return data.get(position);
169.        }
170.
171.        @Override
172.        public long getItemId(int position) {
173.            return position;
174.        }
175.
176.        @Override
177.        public View getView(int position, View convertView, ViewGroup parent) {
178.            View view = null;
179.            //使用 convertView
180.            if (convertView == null) {
181.                view = View.inflate(MainActivity.this, R.layout.list_item, null);
182.            }else {
183.                view = convertView;
184.            }
185.            /*
186.             * 从 view 中获取子控件
187.             * 注意：这里的 findViewById 是 View 类中的，在 Activity 中也有一个同名方法，
188.             * 因为我们是从 view 中查找子控件，因此必须使用 View.findViewById().
189.             * 如果直接使用 findViewById
190.             * 则不会报错。这里是初学者最容易犯的错误。
191.             */
192.            ImageView iv = (ImageView) view.findViewById(R.id.iv);
193.            TextView tv_title = (TextView) view.findViewById(R.id.tv_title);
194.            TextView tv_type = (TextView) view.findViewById(R.id.tv_type);
195.            TextView tv_desc = (TextView) view.findViewById(R.id.tv_desc);
196.            //获取图片的 url 地址
197.            String imageURL = data.get(position).getImage();
198.            /*
199.             * 自定义的方法，完成图片的加载
200.             * 参数 1：ImageView 对象
201.             * 参数 2：图片的 url 地址
202.             */
203.            loadImage(iv,imageURL);
204.            //设置新闻的标题
205.            String title = data.get(position).getTitle();
206.            tv_title.setText(title);
207.            //获取新闻的类型
208.            int type = data.get(position).getType();
209.            switch (type) {
210.            case 1:
211.                //如果是评论，则获取评论个数
212.                int comment = data.get(position).getComment();
```

```
213.                 tv_type.setText(comment + "跟帖");
214.                 break;
215.             case 2:
216.                 tv_type.setText("专题");
217.                 break;
218.             case 3:
219.                 tv_type.setText("活动");
220.                 break;
221.             default:
222.                 break;
223.             }
224.             //获取新闻的描述信息
225.             String description = data.get(position).getDescription();
226.             tv_desc.setText(description);
227.
228.             return view;
229.         }
230.
231.     }
232.     /**
233.      * 是异步方法,运行在子线程中
234.      * @param iv ImageView
235.      * @param imageURL 图片的 url 地址
236.      */
237.     public void loadImage(final ImageView iv,final String imageURL) {
238.         /*
239.          * 因为要执行访问网络操作，因此需要使用子线程
240.          * 这里使用了线程池来执行任务
241.          */
242.         executorService.execute(new Runnable() {
243.
244.             @Override
245.             public void run() {
246.                 try {
247.                     URL url = new URL(imageURL);
248.                     HttpURLConnection connection =
249. (HttpURLConnection) url.openConnection();
250.                     connection.setRequestMethod("GET");
251.                     connection.setConnectTimeout(TIME_OUT);
252.                     connection.setReadTimeout(TIME_OUT);
253.                     connection.connect();
254.                     InputStream inputStream = connection.getInputStream();
255.                     /*
256.                      * 使用 BitmapFactory 将字节输入流转换为 Bitmap 对象
```

```
                     */
                    final Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
                    /*
                     * 给 Handler 发送消息
                     * 让主线程执行 run 方法
                     * 在 run 方法中将 Bitmap 设置给 ImageView 对象
                     */
                    handler.post(new Runnable() {
                        @Override
                        public void run() {
                            iv.setImageBitmap(bitmap);
                        }
                    });
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });

    }

}

```

【文件 1-11】 News.java

```
package com.itheima.news;
/**
 * 封装新闻条目
 * @author wzy 2015-11-4
 *
 */
public class News {

 private String title;
private String description;
private String image;
private int type;
private int comment;
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public String getDescription() {
    return description;
```

```
22. }
23. public void setDescription(String description) {
24.     this.description = description;
25. }
26. public String getImage() {
27.     return image;
28. }
29. public void setImage(String image) {
30.     this.image = image;
31. }
32. public int getType() {
33.     return type;
34. }
35. public void setType(int type) {
36.     this.type = type;
37. }
38. public int getComment() {
39.     return comment;
40. }
41. public void setComment(int comment) {
42.     this.comment = comment;
43. }
44. @Override
45. public String toString() {
46.     return "News [title=" + title + ", description=" + description + ",
47. image=" + image + ", type=" + type + ", comment=" + comment + "]";
48. }
49.
50. }
51.
```

注意：

在文件【1-10】MainActivity.java 的 203 行，使用了 loadImage(iv,imageURL);方法，在 loadImage 方法中开启了线程池去加载图片，然后通过 handler 发送消息给 ImageView 对象设置 Bitmap 对象。这是因为 Adapter 中的 getView 方法是在主线程中被调用的,因此在该方法中无法访问网络,所有就调用了 loadImage()方法，在该方法中开启子线程下载图片，图片下载好之后再通过 handler 将消息发送给主线程，让主线程将 ImageView 设置 Bitmap 对象。

上面的过程是比较麻烦的，那么有没有现成的框架能直接在子线程中给 ImageView 设置网络图片呢？答案肯定是有的，而且目前有很多这样的框架，下面给出了两个典型的框架一个是 SmartImageView，该控件属于自定义控件，另一个是 ImageLoader。

1.6 SmartImageView

1.6.1 SmartImageView 简介

SmartImageView 是开源免费的图片加载框架，代码托管在 Github 上，用它可以替代 Android 自带的 ImgageView 控件。使用 SmartImageView 可以直接通过 url 加载图片，该过程是异步的，且支持图片的缓存，以便下次快速快速加载显示。

下载地址：https://github.com/loopj/android-smart-image-view

网站页面如图 1-5 所示。

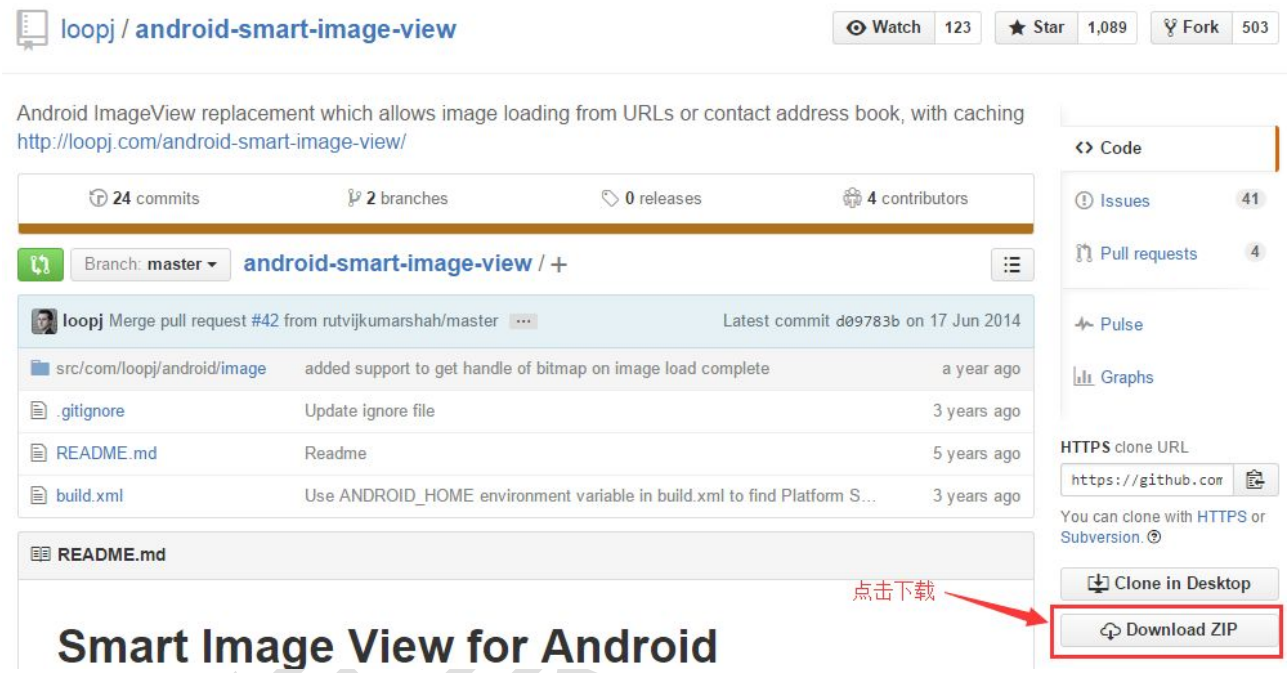


图 1-5　SmartImageView Github 网页截图

1.6.2 SmartImageView 的使用

在这里直接使用 1.5 节中的案例，直接修改。

1. 将下载好的 SmartImageView 的 src 下的整个源代码直接拷贝到自己本工程下的 src 目录下。

图 1-6　SmartImageView 位于的工程目录结构

2. 在 list_item.xml 布局文件中将 ImageView 替换为 com.loopj.android.image.SmartImageView ，如下图

所示。


```
<com.loopj.android.image.SmartImageView
    android:id="@+id/iv"
    android:layout_width="100dp"
    android:layout_height="80dp"
    android:layout_marginRight="5dp"
    android:background="@drawable/ic_launcher"
    />
```

3. 在代码中的 ImageView 也给替换为 SmartImageView

SmartImageView iv = (SmartImageView) view.findViewById(R.id.iv);

4. 给 SmartImageView 对象直接设置 url 即可

iv.setImageUrl(imageURL);