第 10 章　看看精彩的世界，使用网络技术

如果你在玩手机的时候不能上网，那你一定会感到特别的枯燥乏味。没错，现在早已不是玩单机的时代了，无论是 PC、手机、平板、还是电视几乎都会具备上网的功能，到未来甚至是手表、眼镜、拖鞋等等设备也可能会逐个加入到这个行列，21 世纪的确是互联网的时代。

那么不用多说，Android 手机肯定也是可以上网的，所以作为开发者的我们就需要考虑如何利用网络来编写出更加出色的应用程序，像 QQ、微博、新闻等常见的应用都会大量地使用到网络技术。本章主要会讲述如何在手机端使用 HTTP 协议和服务器端进行网络交互，并对服务器返回的数据进行解析，这也是 Android 中最常使用到的网络技术了，下面就让我们一起来学习一下吧。

10.1　WebView 的用法

有时候我们可能会碰到一些比较特殊的需求，比如说要求在应用程序里展示一些网页。相信每个人都知道，加载和显示网页通常都是浏览器的任务，但是需求里又明确指出，不允许打开系统浏览器，而我们当然也不可能自己去编写一个浏览器出来，这时应该怎么办呢？

不用担心，Android 早就已经考虑到了这种需求，并提供了一个 WebView 控件，借助它我们就可以在自己的应用程序里嵌入一个浏览器，从而非常轻松地展示各种各样的网页。

WebView 的用法也是相当简单，下面我们就通过一个例子来学习一下吧。新建一个 WebViewTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

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

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

```
</LinearLayout>
```

可以看到，我们在布局文件中使用到了一个新的控件，WebView。这个控件当然也就是用来显示网页的了，这里的写法很简单，给它设置了一个 id，并让它充满整个屏幕。

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private WebView webView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        webView = (WebView) findViewById(R.id.web_view);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.setWebViewClient(new WebViewClient() {
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String
url) {
                view.loadUrl(url); // 根据传入的参数再去加载新的网页
                return true; // 表示当前WebView可以处理打开新网页的请求，不用借助
系统浏览器
            }
        });
        webView.loadUrl("http://www.baidu.com");
    }

}
```

MainActivity 中的代码也很短，首先使用 findViewById()方法获取到了 WebView 的实例，然后调用 WebView 的 getSettings()方法可以去设置一些浏览器的属性，这里我们并不去设置过多的属性，只是调用了 setJavaScriptEnabled()方法来让 WebView 支持 JavaScript 脚本。

接下来是非常重要的一个部分，我们调用了 WebView 的 setWebViewClient()方法，并传入了 WebViewClient 的匿名类作为参数，然后重写了 shouldOverrideUrlLoading()方法。这就表明当需要从一个网页跳转到另一个网页时，我们希望目标网页仍然在当前 WebView 中显示，而不是打开系统浏览器。

最后一步就非常简单了，调用 WebView 的 loadUrl()方法，并将网址传入，即可展示相应网页的内容，这里就让我们看一看百度的首页是长什么样的吧。

另外还需要注意，由于本程序使用到了网络功能，而访问网络是需要声明权限的，因此

我们还得修改 AndroidManifest.xml 文件，并加入权限声明，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.webviewtest"
    android:versionCode="1"
    android:versionName="1.0" >
    ……
    <uses-permission android:name="android.permission.INTERNET" />
    ……
</manifest>
```

在开始运行之前，首先需要保证你的手机或模拟器是联网的，如果你使用的是模拟器，只需保证电脑能正常上网即可。然后就可以运行一下程序了，效果如图 10.1 所示。

图　10.1

可以看到，WebViewTest 这个程序现在已经具备了一个简易浏览器的功能，不仅成功将百度的首页展示了出来，还可以通过点击链接浏览更多的网页。

当然，WebView 还有很多更加高级的使用技巧，我们就不再继续进行探讨了，因为那不是本章的重点。这里先介绍了一下 WebView 的用法，只是希望你能对 HTTP 协议的使用有一个最基本的认识，接下来我们就要利用这个协议来做一些真正的网络开发工作了。

10.2 使用 HTTP 协议访问网络

如果真的说要去深入分析 HTTP 协议，可能需要花费整整一本书的篇幅。这里我当然不会这么干，因为毕竟你是跟着我学习 Android 开发的，而不是网站开发。对于 HTTP 协议，你只需要稍微了解一些就足够了，它的工作原理特别的简单，就是客户端向服务器发出一条 HTTP 请求，服务器收到请求之后会返回一些数据给客户端，然后客户端再对这些数据进行解析和处理就可以了。是不是非常简单？一个浏览器的基本工作原理也就是如此了。比如说上一节中使用到的 WebView 控件，其实也就是我们向百度的服务器发起了一条 HTTP 请求，接着服务器分析出我们想要访问的是百度的首页，于是会把该网页的 HTML 代码进行返回，然后 WebView 再调用手机浏览器的内核对返回的 HTML 代码进行解析，最终将页面展示出来。

简单来说，WebView 已经在后台帮我们处理好了发送 HTTP 请求、接收服务响应、解析返回数据，以及最终的页面展示这几步工作，不过由于它封装得实在是太好了，反而使得我们不能那么直观地看出 HTTP 协议到底是如何工作的。因此，接下来就让我们通过手动发送 HTTP 请求的方式，来更加深入地理解一下这个过程。

10.2.1 使用 HttpURLConnection

在 Android 上发送 HTTP 请求的方式一般有两种，HttpURLConnection 和 HttpClient，本小节我们先来学习一下 HttpURLConnection 的用法。

首先需要获取到 HttpURLConnection 的实例，一般只需 new 出一个 URL 对象，并传入目标的网络地址，然后调用一下 openConnection()方法即可，如下所示：

```
URL url = new URL("http://www.baidu.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

得到了 HttpURLConnection 的实例之后，我们可以设置一下 HTTP 请求所使用的方法。常用的方法主要有两个，GET 和 POST。GET 表示希望从服务器那里获取数据，而 POST 则表示希望提交数据给服务器。写法如下：

```
connection.setRequestMethod("GET");
```

接下来就可以进行一些自由的定制了，比如设置连接超时、读取超时的毫秒数，以及服务器希望得到的一些消息头等。这部分内容根据自己的实际情况进行编写，示例写法如下：

```
connection.setConnectTimeout(8000);
connection.setReadTimeout(8000);
```

之后再调用 getInputStream()方法就可以获取到服务器返回的输入流了，剩下的任务就是对输入流进行读取，如下所示：

```
InputStream in = connection.getInputStream();
```

最后可以调用 disconnect()方法将这个 HTTP 连接关闭掉，如下所示：

```
connection.disconnect();
```

下面就让我们通过一个具体的例子来真正体验一下 HttpURLConnection 的用法。新建一个 NetworkTest 项目，首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/send_request"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send Request" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <TextView
            android:id="@+id/response"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </ScrollView>

</LinearLayout>
```

注意这里我们使用了一个新的控件，ScrollView，它是用来做什么的呢？由于手机屏幕的空间一般都比较小，有些时候过多的内容一屏是显示不下的，借助 ScrollView 控件的话就可以允许我们以滚动的形式查看屏幕外的那部分内容。另外，布局中还放置了一个 Button 和一个 TextView，Button 用于发送 HTTP 请求，TextView 用于将服务器返回的数据显示出来。

接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    public static final int SHOW_RESPONSE = 0;

    private Button sendRequest;
```

```
private TextView responseText;

private Handler handler = new Handler() {

    public void handleMessage(Message msg) {
        switch (msg.what) {
        case SHOW_RESPONSE:
            String response = (String) msg.obj;
            // 在这里进行UI操作，将结果显示到界面上
            responseText.setText(response);
        }
    }

};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    sendRequest = (Button) findViewById(R.id.send_request);
    responseText = (TextView) findViewById(R.id.response_text);
    sendRequest.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    if (v.getId() == R.id.send_request) {
        sendRequestWithHttpURLConnection();
    }
}

private void sendRequestWithHttpURLConnection() {
    // 开启线程来发起网络请求
    new Thread(new Runnable() {
        @Override
        public void run() {
            HttpURLConnection connection = null;
            try {
                URL url = new URL("http://www.baidu.com");
                connection = (HttpURLConnection) url.openConnection();
```

```
                    connection.setRequestMethod("GET");
                    connection.setConnectTimeout(8000);
                    connection.setReadTimeout(8000);
                    InputStream in = connection.getInputStream();
                    // 下面对获取到的输入流进行读取
                    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
                    StringBuilder response = new StringBuilder();
                    String line;
                    while ((line = reader.readLine()) != null) {
                        response.append(line);
                    }
                    Message message = new Message();
                    message.what = SHOW_RESPONSE;
                    // 将服务器返回的结果存放到Message中
                    message.obj = response.toString();
                    handler.sendMessage(message);
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    if (connection != null) {
                        connection.disconnect();
                    }
                }
            }
        }).start();
    }

}
```

可以看到，我们在 Send Request 按钮的点击事件里调用了 sendRequestWithHttpURL-Connection()方法，在这个方法中先是开启了一个子线程，然后在子线程里使用 HttpURLConnection 发出一条 HTTP 请求，请求的目标地址就是百度的首页。接着利用 BufferedReader 对服务器返回的流进行读取，并将结果存放到了一个 Message 对象中。这里为什么要使用 Message 对象呢？当然是因为子线程中无法对 UI 进行操作了。我们希望可以将服务器返回的内容显示到界面上，所以就创建了一个 Message 对象，并使用 Handler 将它发送出去。之后又在 Handler 的 handleMessage()方法中对这条 Message 进行处理，最终取出结果并设置到 TextView 上。

完整的一套流程就是这样，不过在开始运行之前，仍然别忘了要声明一下网络权限。修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.networktest"
    android:versionCode="1"
    android:versionName="1.0" >
    ……
    <uses-permission android:name="android.permission.INTERNET" />
    ……
</manifest>
```

好了，现在运行一下程序，并点击 Send Request 按钮，结果如图 10.2 所示。

图 10.2

是不是看得头晕眼花？没错，服务器返回给我们的就是这种 HTML 代码，只是通常情况下浏览器都会将这些代码解析成漂亮的网页后再展示出来。

那么如果是想要提交数据给服务器应该怎么办呢？其实也不复杂，只需要将 HTTP 请求的方法改成 POST，并在获取输入流之前把要提交的数据写出即可。注意每条数据都要以键值对的形式存在，数据与数据之间用&符号隔开，比如说我们想要向服务器提交用户名和密码，就可以这样写：

```
connection.setRequestMethod("POST");
DataOutputStream out = new DataOutputStream(connection.getOutputStream());
out.writeBytes("username=admin&password=123456");
```

好了，相信你已经将 HttpURLConnection 的用法很好地掌握了，下面我们来学习一下 HttpClient 的用法吧。

10.2.2　使用 HttpClient

HttpClient 是 Apache 提供的 HTTP 网络访问接口，从一开始的时候就被引入到了 Android API 中。它可以完成和 HttpURLConnection 几乎一模一样的效果，但两者之间的用法却有较大的差别，那么我们自然要看一下 HttpClient 是如何使用的了。

首先你需要知道，HttpClient 是一个接口，因此无法创建它的实例，通常情况下都会创建一个 DefaultHttpClient 的实例，如下所示：

```
HttpClient httpClient = new DefaultHttpClient();
```

接下来如果想要发起一条 GET 请求，就可以创建一个 HttpGet 对象，并传入目标的网络地址，然后调用 HttpClient 的 execute()方法即可：

```
HttpGet httpGet = new HttpGet("http://www.baidu.com");
httpClient.execute(httpGet);
```

如果是发起一条 POST 请求会比 GET 稍微复杂一点，我们需要创建一个 HttpPost 对象，并传入目标的网络地址，如下所示：

```
HttpPost httpPost = new HttpPost("http://www.baidu.com");
```

然后通过一个 NameValuePair 集合来存放待提交的参数，并将这个参数集合传入到一个 UrlEncodedFormEntity 中，然后调用 HttpPost 的 setEntity()方法将构建好的 UrlEncodedFormEntity 传入，如下所示：

```
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("username", "admin"));
params.add(new BasicNameValuePair("password", "123456"));
UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "utf-8");
httpPost.setEntity(entity);
```

接下来的操作就和 HttpGet 一样了，调用 HttpClient 的 execute()方法，并将 HttpPost 对象传入即可：

```
httpClient.execute(httpPost);
```

执行 execute()方法之后会返回一个 HttpResponse 对象，服务器所返回的所有信息就会包含在这里面。通常情况下我们都会先取出服务器返回的状态码，如果等于 200 就说明请求和

响应都成功了，如下所示：

```
if (httpResponse.getStatusLine().getStatusCode() == 200) {
    // 请求和响应都成功了
}
```

接下来在这个 if 判断的内部取出服务返回的具体内容，可以调用 getEntity()方法获取到一个 HttpEntity 实例，然后再用 EntityUtils.toString()这个静态方法将 HttpEntity 转换成字符串即可，如下所示：

```
HttpEntity entity = httpResponse.getEntity();
String response = EntityUtils.toString(entity);
```

注意如果服务器返回的数据是带有中文的，直接调用 EntityUtils.toString()方法进行转换会有乱码的情况出现，这个时候只需要在转换的时候将字符集指定成 utf-8 就可以了，如下所示：

```
String response = EntityUtils.toString(entity, "utf-8");
```

好了，基本的用法就是如此，接下来就让我们把 NetworkTest 这个项目改用 HttpClient 的方式再实现一遍吧。

由于布局部分完全不用改动，所以现在直接修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.send_request) {
            sendRequestWithHttpClient();
        }
    }

    private void sendRequestWithHttpClient() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    HttpClient httpClient = new DefaultHttpClient();
                    HttpGet httpGet = new HttpGet("http://www.baidu.com");
                    HttpResponse httpResponse = httpClient.execute(httpGet);
                    if (httpResponse.getStatusLine().getStatusCode() == 200) {
                        // 请求和响应都成功了
                        HttpEntity entity = httpResponse.getEntity();
```

```
                        String response = EntityUtils.toString(entity,
"utf-8");
                        Message message = new Message();
                        message.what = SHOW_RESPONSE;
                        // 将服务器返回的结果存放到Message中
                        message.obj = response.toString();
                        handler.sendMessage(message);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    ……
}
```

这里我们并没有做太多的改动，只是添加了一个 sendRequestWithHttpClient()方法，并在 Send Request 按钮的点击事件里去调用这个方法。在这个方法中同样还是先开启了一个子线程，然后在子线程里使用 HttpClient 发出一条 HTTP 请求，请求的目标地址还是百度的首页，HttpClient 的用法也正如前面所介绍的一样。然后为了能让结果在界面上显示出来，这里仍然是将服务器返回的数据存放到了 Message 对象中，并用 Handler 将 Message 发送出去。

仅仅只是改了这么多代码，现在我们可以重新运行一下程序了。点击 Send Request 按钮后，你会看到和上一小节中同样的运行结果，由此证明，使用 HttpClient 来发送 HTTP 请求的功能也已经成功实现了。

这样的话，相信你就已经把 HttpURLConnection 和 HttpClient 的基本用法都掌握得差不多了。

经验值：+25000　　目前经验值：154905

级别：头领鸟

捡到宝物：以前我就不怕走夜路，自从身体能够发光后我就更不怕了，现在我身体发出的光芒已经可以照亮周围 1 米见方，走夜路时给人的感觉很温暖。唯一的问题是似乎有些影响睡眠，因为睡觉时我也总是亮着，俗话说有一利必有一弊还真不是盖的。我曾试图用意念去关闭光芒，发现不管用，也曾在周围没人的时候试过“芝麻，关灯”，尽管我知道这很蠢，但不试过总是不死心，结果意料之中的不管用。但我隐约记得那次参加 TNND 编程大赛时看到那些或气宇轩昂或牛逼轰轰的高阶选手也并不发光啊，不去想了。人就是个习惯，开着灯不也一样睡么。这天我又一次露宿野外，是一个山洞，事实上自从上次我钉帐篷时发现了上

古典籍《算法本源》后，我就有意识地尽量不住旅店，尽量露宿在野外。因为我知道，身处我这种境地，要想捡到什么宝物的话，那只能在野外、深谷、山洞等地方，你指望别人在宾馆里落下一个那么好的大宝物是很不现实的，有谁会带着祖传宝贝到处瞎几吧逛呢（当然那些热衷参加鉴宝栏目的除外）。功夫不负有心人，在这个山洞中我捡到了宝物，也是一本书，书名叫《编译子》，应该是讲述编译原理的，作者名叫有编氏，看书老旧的程度，应该是上古的，但内容很艰深，目前还读不懂。同那本《算法本源》一样，书的前言中也提到，当阅读者的编程级别提升至某个层次时，将更容易看懂这本书，但具体是什么级别，书中也没有说，只说“造化弄人，因人而异”。我现在知道了，想必这句话在上古时期大概就相当于我们人界的书上都喜欢来上这么一句“由于笔者水平所限，书中错误在所难免，敬请读者批评指正。”只不过上古时的书因为作者特别牛逼，所以书中没有错误，因此看不懂都是读者的问题，而今天人界的书，因为受作者水平所限，错误不可避免，因此看不懂都是作者的问题。还是上古的人牛逼啊。想到这里，我心中不禁一阵感慨，大师，我无功受禄，请受后辈一拜。我在山洞里插草为香，对上古大师有编氏祭拜了一番，然后睡下，一夜无梦。清晨。继续前进。

10.3 解析 XML 格式数据

通常情况下，每个需要访问网络的应用程序都会有一个自己的服务器，我们可以向服务器提交数据，也可以从服务器上获取数据。不过这个时候就出现了一个问题，这些数据到底要以什么样的格式在网络上传输呢？随便传递一段文本肯定是不行的，因为另一方根本就不会知道这段文本的用途是什么。因此，一般我们都会在网络上传输一些格式化后的数据，这种数据会有一定的结构规格和语义，当另一方收到数据消息之后就可以按照相同的结构规格进行解析，从而取出他想要的那部分内容。

在网络上传输数据时最常用的格式有两种，XML 和 JSON，下面我们就来一个个地进行学习，本节首先学一下如何解析 XML 格式的数据。

在开始之前我们还需要先解决一个问题，就是从哪儿才能获取一段 XML 格式的数据呢？这里我准备教你搭建一个最简单的 Web 服务器，在这个服务器上提供一段 XML 文本，然后我们在程序里去访问这个服务器，再对得到的 XML 文本进行解析。

搭建 Web 服务器其实非常简单，有很多的服务器类型可供选择，这里我准备使用 Apache 服务器。首先你需要去下载一个 Apache 服务器的安装包，下载地址是：http://httpd.apache.org/download.cgi。

下载完成后双击就可以进行安装了，如图 10.3 所示。

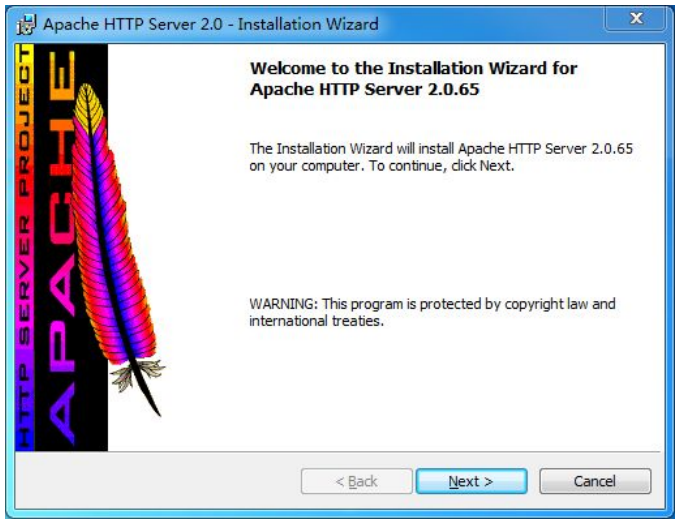

图　10.3

然后一直点击 Next，会提示让你输入自己的域名，我们随便填一个域名就可以了，如图 10.4 所示。

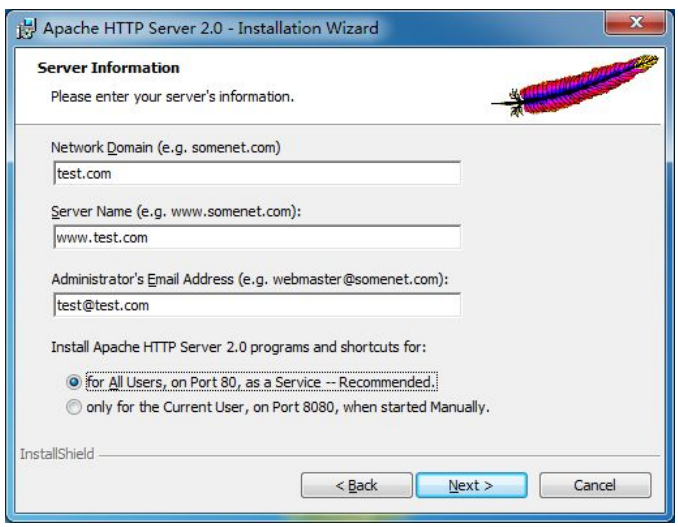

图　10.4

接着继续一直点击 Next 会提示让你选择程序安装的路径，这里我选择安装到 C:\Apache 目录下，之后再继续点击 Next 就可以完成安装了。安装成功后服务器会自动启动起来，你可以打开电脑的浏览器来验证一下。在地址栏输入 127.0.0.1，如果出现了如图 10.5 所示的

界面，就说明服务器已经启动成功了。

图　10.5

接下来进入到 C:\Apache\Apache2\htdocs 目录下，在这里新建一个名为 get_data.xml 的文件，然后编辑这个文件，并加入如下 XML 格式的内容。

```
<apps>
    <app>
        <id>1</id>
        <name>Google Maps</name>
        <version>1.0</version>
    </app>
    <app>
        <id>2</id>
        <name>Chrome</name>
        <version>2.1</version>
    </app>
    <app>
        <id>3</id>
        <name>Google Play</name>
        <version>2.3</version>
    </app>
</apps>
```

这时在浏览器中访问 http://127.0.0.1/get_data.xml 这个网址，就应该出现如图 10.6 所示的内容。

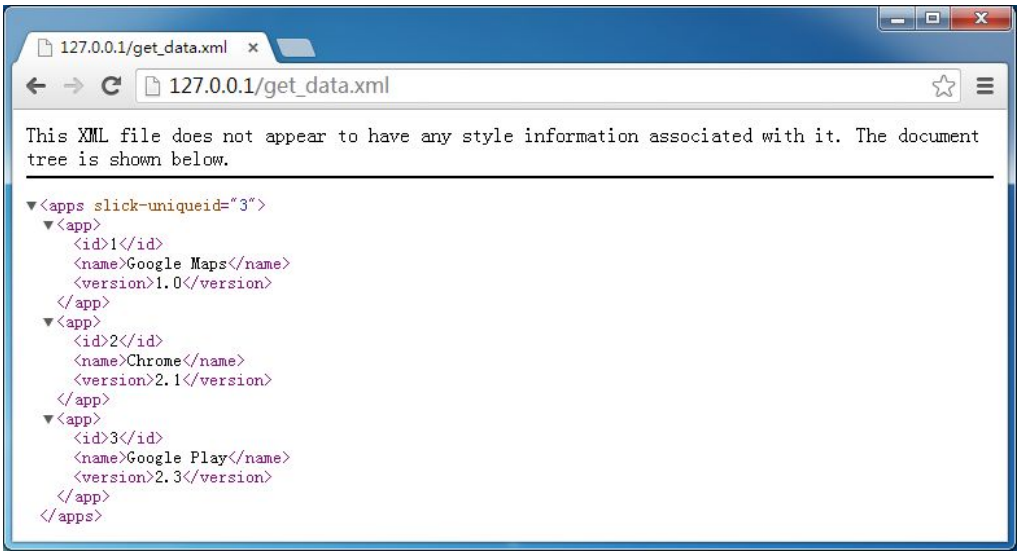

图　10.6

好了，准备工作到此结束，接下来就让我们在 Android 程序里去获取并解析这段 XML 数据吧。

10.3.1　Pull 解析方式

解析 XML 格式的数据其实也有挺多种方式的，本节中我们学习比较常用的两种，Pull 解析和 SAX 解析。那么简单起见，这里仍然是在 NetworkTest 项目的基础上继续开发，这样我们就可以重用之前网络通信部分的代码，从而把工作的重心放在 XML 数据解析上。

既然 XML 格式的数据已经提供好了，现在要做的就是从中解析出我们想要得到的那部分内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    private void sendRequestWithHttpClient() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    HttpClient httpClient = new DefaultHttpClient();
```

```
                    // 指定访问的服务器地址是电脑本机
                    HttpGet httpGet = new HttpGet("http://10.0.2.2/get_data.xml");
                    HttpResponse httpResponse = httpClient.execute(httpGet);
                    if (httpResponse.getStatusLine().getStatusCode() == 200) {
                        // 请求和响应都成功了
                        HttpEntity entity = httpResponse.getEntity();
                        String response = EntityUtils.toString(entity,
"utf-8");
                        parseXMLWithPull(response);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    private void parseXMLWithPull(String xmlData) {
        try {
            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
            XmlPullParser xmlPullParser = factory.newPullParser();
            xmlPullParser.setInput(new StringReader(xmlData));
            int eventType = xmlPullParser.getEventType();
            String id = "";
            String name = "";
            String version = "";
            while (eventType != XmlPullParser.END_DOCUMENT) {
                String nodeName = xmlPullParser.getName();
                switch (eventType) {
                // 开始解析某个结点
                case XmlPullParser.START_TAG: {
                    if ("id".equals(nodeName)) {
                        id = xmlPullParser.nextText();
                    } else if ("name".equals(nodeName)) {
                        name = xmlPullParser.nextText();
                    } else if ("version".equals(nodeName)) {
                        version = xmlPullParser.nextText();
                    }
                    break;
                }
```

```
                    // 完成解析某个结点
                    case XmlPullParser.END_TAG: {
                        if ("app".equals(nodeName)) {
                            Log.d("MainActivity", "id is " + id);
                            Log.d("MainActivity", "name is " + name);
                            Log.d("MainActivity", "version is " + version);
                        }
                        break;
                    }
                    default:
                        break;
                    }
                    eventType = xmlPullParser.next();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

可以看到，这里首先是将 HTTP 请求的地址改成了 http://10.0.2.2/get_data.xml，10.0.2.2 对于模拟器来说就是电脑本机的 IP 地址。在得到了服务器返回的数据后，我们并不再去发送一条消息，而是调用了 parseXMLWithPull()方法来解析服务器返回的数据。

下面就来仔细看下 parseXMLWithPull()方法中的代码吧。这里首先要获取到一个 XmlPullParserFactory 的实例，并借助这个实例得到 XmlPullParser 对象，然后调用 XmlPullParser 的 setInput()方法将服务器返回的 XML 数据设置进去就可以开始解析了。解析的过程也是非常简单，通过 getEventType()可以得到当前的解析事件，然后在一个 while 循环中不断地进行解析，如果当前的解析事件不等于 XmlPullParser.END_DOCUMENT，说明解析工作还没完成，调用 next()方法后可以获取下一个解析事件。

在 while 循环中，我们通过 getName()方法得到当前结点的名字，如果发现结点名等于 id、name 或 version，就调用 nextText()方法来获取结点内具体的内容，每当解析完一个 app 结点后就将获取到的内容打印出来。

好了，整体的过程就是这么简单，下面就让我们来测试一下吧。运行 NetworkTest 项目，然后点击 Send Request 按钮，观察 LogCat 中的打印日志，如图 10.7 所示。

Tag	Text
MainActivity	id is 1
MainActivity	name is Google Maps
MainActivity	version is 1.0
MainActivity	id is 2
MainActivity	name is Chrome
MainActivity	version is 2.1
MainActivity	id is 3
MainActivity	name is Google Play
MainActivity	version is 2.3

图 10.7

可以看到，我们已经将 XML 数据中的指定内容成功解析出来了。

10.3.2 SAX 解析方式

Pull 解析方式虽然非常的好用，但它并不是我们唯一的选择。SAX 解析也是一种特别常用的 XML 解析方式，虽然它的用法比 Pull 解析要复杂一些，但在语义方面会更加的清楚。

通常情况下我们都会新建一个类继承自 DefaultHandler，并重写父类的五个方法，如下所示：

```
public class MyHandler extends DefaultHandler {

    @Override
    public void startDocument() throws SAXException {
    }

    @Override
    public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
    }

    @Override
    public void characters(char[] ch, int start, int length) throws
SAXException {
    }

    @Override
```

```
    public void endElement(String uri, String localName, String qName) throws
SAXException {
    }

    @Override
    public void endDocument() throws SAXException {
    }

}
```

这五个方法一看就很清楚吧？startDocument()方法会在开始 XML 解析的时候调用，startElement()方法会在开始解析某个结点的时候调用，characters()方法会在获取结点中内容的时候调用，endElement()方法会在完成解析某个结点的时候调用，endDocument()方法会在完成整个 XML 解析的时候调用。其中，startElement()、characters()和 endElement()这三个方法是有参数的，从 XML 中解析出的数据就会以参数的形式传入到这些方法中。需要注意的是，在获取结点中的内容时，characters()方法可能会被调用多次，一些换行符也被当作内容解析出来，我们需要针对这种情况在代码中做好控制。

那么下面就让我们尝试用 SAX 解析的方式来实现和上一小节中同样的功能吧。新建一个 ContentHandler 类继承自 DefaultHandler，并重写父类的五个方法，如下所示：

```
public class ContentHandler extends DefaultHandler {

    private String nodeName;

    private StringBuilder id;

    private StringBuilder name;

    private StringBuilder version;

    @Override
    public void startDocument() throws SAXException {
        id = new StringBuilder();
        name = new StringBuilder();
        version = new StringBuilder();
    }

    @Override
    public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
```

```
            // 记录当前结点名
            nodeName = localName;
        }

        @Override
        public void characters(char[] ch, int start, int length) throws
    SAXException {
            // 根据当前的结点名判断将内容添加到哪一个StringBuilder对象中
            if ("id".equals(nodeName)) {
                id.append(ch, start, length);
            } else if ("name".equals(nodeName)) {
                name.append(ch, start, length);
            } else if ("version".equals(nodeName)) {
                version.append(ch, start, length);
            }
        }

        @Override
        public void endElement(String uri, String localName, String qName) throws
    SAXException {
            if ("app".equals(localName)) {
                Log.d("ContentHandler", "id is " + id.toString().trim());
                Log.d("ContentHandler", "name is " + name.toString().trim());
                Log.d("ContentHandler", "version is " + version.toString().trim());
                // 最后要将StringBuilder清空掉
                id.setLength(0);
                name.setLength(0);
                version.setLength(0);
            }
        }

        @Override
        public void endDocument() throws SAXException {
        }

    }
```

可以看到，我们首先给 id、name 和 version 结点分别定义了一个 StringBuilder 对象，并在 startDocument()方法里对它们进行了初始化。每当开始解析某个结点的时候，startElement()

方法就会得到调用，其中 localName 参数记录着当前结点的名字，这里我们把它记录下来。接着在解析结点中具体内容的时候就会调用 characters()方法，我们会根据当前的结点名进行判断，将解析出的内容添加到哪一个 StringBuilder 对象中。最后在 endElement()方法中进行判断，如果 app 结点已经解析完成，就打印出 id、name 和 version 的内容。需要注意的是，目前 id、name 和 version 中都可能是包括回车或换行符的，因此在打印之前我们还需要调用一下 trim()方法，并且打印完成后还要将 StringBuilder 的内容清空掉，不然的话会影响下一次内容的读取。

接下来的工作就非常简单了，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    private void sendRequestWithHttpClient() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    HttpClient httpClient = new DefaultHttpClient();
                    // 指定访问的服务器地址是电脑本机
                    HttpGet httpGet = new HttpGet("http://10.0.2.2:8080/
get_data.xml");
                    HttpResponse httpResponse = httpClient.execute(httpGet);
                    if (httpResponse.getStatusLine().getStatusCode() == 200) {
                        // 请求和响应都成功了
                        HttpEntity entity = httpResponse.getEntity();
                        String response = EntityUtils.toString(entity,
"utf-8");
                        parseXMLWithSAX(response);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    ……
    private void parseXMLWithSAX(String xmlData) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            XMLReader xmlReader = factory.newSAXParser().getXMLReader();
```

```
                ContentHandler handler = new ContentHandler();
                // 将ContentHandler的实例设置到XMLReader中
                xmlReader.setContentHandler(handler);
                // 开始执行解析
                xmlReader.parse(new InputSource(new StringReader(xmlData)));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

在得到了服务器返回的数据后，我们这次去调用 parseXMLWithSAX()方法来解析 XML 数据。parseXMLWithSAX()方法中先是创建了一个 SAXParserFactory 的对象，然后再获取到 XMLReader 对象，接着将我们编写的 ContentHandler 的实例设置到 XMLReader 中，最后调用 parse()方法开始执行解析就好了。

现在重新运行一下程序，点击 Send Request 按钮后观察 LogCat 中的打印日志，你会看到和图 10.7 中一样的结果。

除了 Pull 解析和 SAX 解析之外，其实还有一种 DOM 解析方式也算挺常用的，不过这里我们就不再展开进行讲解了，感兴趣的话你可以自己去查阅一下相关资料。

经验值：+26000　　目前经验值：180905

级别：头领鸟

赢得宝物：战胜魔界使者 XML。拾取 XML 掉落的宝物，XML 智能解析器一套。XML 是那种标准的外强中干类型，看似强大，但你真正出手后，会发现这货其实功力平平，多少有辱使者的称谓。不过做为神界与魔界的官方沟通渠道之一，XML 还是能够完成本职工作的。

10.4　解析 JSON 格式数据

现在你已经掌握了 XML 格式数据的解析方式，那么接下来我们要去学习一下如何解析 JSON 格式的数据了。比起 XML，JSON 的主要优势在于它的体积更小，在网络上传输的时候可以更省流量。但缺点在于，它的语义性较差，看起来不如 XML 直观。

在开始之前，我们还需要在 C:\Apache\Apache2\htdocs 目录中新建一个 get_data.json 的文件，然后编辑这个文件，并加入如下 JSON 格式的内容：

```
[{"id":"5","version":"5.5","name":"Angry Birds"},
{"id":"6","version":"7.0","name":"Clash of Clans"},
{"id":"7","version":"3.5","name":"Hey Day"}]
```

这时在浏览器中访问 http://127.0.0.1/get_data.json 这个网址，就应该出现如图 10.8 所示的内容。

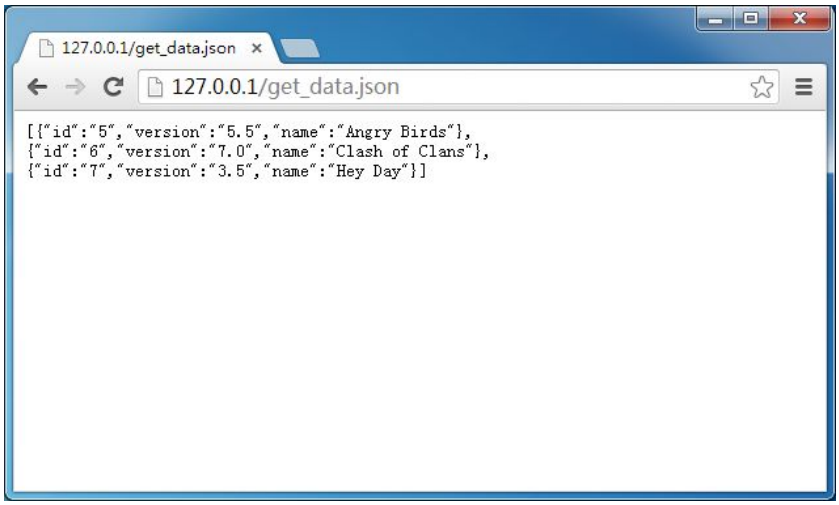

图　10.8

好了，这样我们把 JSON 格式的数据也准备好了，下面就开始学习如何在 Android 程序中解析这些数据吧。

10.4.1　使用 JSONObject

类似地，解析 JSON 数据也有很多种方法，可以使用官方提供的 JSONObject，也可以使用谷歌的开源库 GSON。另外，一些第三方的开源库如 Jackson、FastJSON 等也非常不错。本节中我们就来学习一下前两种解析方式的用法。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    private void sendRequestWithHttpClient() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    HttpClient httpClient = new DefaultHttpClient();
                    // 指定访问的服务器地址是电脑本机
                    HttpGet httpGet = new HttpGet("http://10.0.2.2/
get_data.json");
```

```
                    HttpResponse httpResponse = httpClient.execute(httpGet);
                    if (httpResponse.getStatusLine().getStatusCode() == 200) {
                        // 请求和响应都成功了
                        HttpEntity entity = httpResponse.getEntity();
                        String response = EntityUtils.toString(entity,
"utf-8");
                        parseJSONWithJSONObject(response);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    ……
    private void parseJSONWithJSONObject(String jsonData) {
        try {
            JSONArray jsonArray = new JSONArray(jsonData);
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                String id = jsonObject.getString("id");
                String name = jsonObject.getString("name");
                String version = jsonObject.getString("version");
                Log.d("MainActivity", "id is " + id);
                Log.d("MainActivity", "name is " + name);
                Log.d("MainActivity", "version is " + version);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

首先记得要将 HTTP 请求的地址改成 http://10.0.2.2/get_data.json，然后在得到了服务器返回的数据后调用 parseJSONWithJSONObject()方法来解析数据。可以看到，解析 JSON 的代码真的是非常简单，由于我们在服务器中定义的是一个 JSON 数组，因此这里首先是将服务器返回的数据传入到了一个 JSONArray 对象中。然后循环遍历这个 JSONArray，从中取出的每一个元素都是一个 JSONObject 对象，每个 JSONObject 对象中又会包含 id、name 和 version 这些数据。接下来只需要调用 getString()方法将这些数据取出，并打印出来即可。

好了，就是这么简单！现在重新运行一下程序，并点击 Send Request 按钮，结果如图 10.9 所示。

Tag	Text
MainActivity	id is 5
MainActivity	name is Angry Birds
MainActivity	version is 5.5
MainActivity	id is 6
MainActivity	name is Clash of Clans
MainActivity	version is 7.0
MainActivity	id is 7
MainActivity	name is Hey Day
MainActivity	version is 3.5

图　10.9

10.4.2　使用 GSON

如何你认为使用 JSONObject 来解析 JSON 数据已经非常简单了，那你就太容易满足了。谷歌提供的 GSON 开源库可以让解析 JSON 数据的工作简单到让你不敢想象的地步，那我们肯定是不能错过这个学习机会的。

不过 GSON 并没有被添加到 Android 官方的 API 中，因此如果想要使用这个功能的话，则必须要在项目中添加一个 GSON 的 Jar 包。首先我们需要将 GSON 的资源压缩包下载下来，下载地址是：http://code.google.com/p/google-gson/downloads/list。

然后将资源包进行解压，会看到如图 10.10 所示的几个文件。

名称	类型	大小
gson-2.2.4.jar	Executable Jar File	186 KB
gson-2.2.4-javadoc.jar	Executable Jar File	244 KB
gson-2.2.4-sources.jar	Executable Jar File	125 KB
LICENSE	文件	12 KB
README	文件	1 KB

图　10.10

其中 gson-2.2.4.jar 这个文件就是我们所需要的了，现在将它拷贝到 NetworkTest 项目的 libs 目录下，GSON 库就会自动添加到 NetworkTest 项目中了，如图 10.11 所示。

图 10.11

那么GSON库究竟是神奇在哪里呢？其实它主要就是可以将一段JSON格式的字符串自动映射成一个对象，从而不需要我们再手动去编写代码进行解析了。

比如说一段 JSON 格式的数据如下所示：

```
{"name":"Tom","age":20}
```

那我们就可以定义一个 Person 类，并加入 name 和 age 这两个字段，然后只需简单地调用如下代码就可以将 JSON 数据自动解析成一个 Person 对象了：

```
Gson gson = new Gson();
Person person = gson.fromJson(jsonData, Person.class);
```

如果需要解析的是一段 JSON 数组会稍微麻烦一点，我们需要借助 TypeToken 将期望解析成的数据类型传入到 fromJson()方法中，如下所示：

```
List<Person> people = gson.fromJson(jsonData, new TypeToken<List<Person>>()
{}.getType());
```

好了，基本的用法就是这样，下面就让我们来真正地尝试一下吧。首先新增一个 App 类，并加入 id、name 和 version 这三个字段，如下所示：

```
public class App {

    private String id;

    private String name;

    private String version;

    public String getId() {
        return id;
    }

    public void setId(String id) {
```

```
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }

}
```

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ......
    private void sendRequestWithHttpClient() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    HttpClient httpClient = new DefaultHttpClient();
                    // 指定访问的服务器地址是电脑本机
                    HttpGet httpGet = new HttpGet("http://10.0.2.2/
get_data.json");
                    HttpResponse httpResponse = httpClient.execute(httpGet);
                    if (httpResponse.getStatusLine().getStatusCode() == 200) {
                        // 请求和响应都成功了
                        HttpEntity entity = httpResponse.getEntity();
                        String response = EntityUtils.toString(entity,
"utf-8");
                        parseJSONWithGSON(response);
```

```
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    ……
    private void parseJSONWithGSON(String jsonData) {
        Gson gson = new Gson();
        List<App> appList = gson.fromJson(jsonData, new
TypeToken<List<App>>() {}.getType());
        for (App app : appList) {
            Log.d("MainActivity", "id is " + app.getId());
            Log.d("MainActivity", "name is " + app.getName());
            Log.d("MainActivity", "version is " + app.getVersion());
        }
    }
}
```

现在重新运行程序，点击 Send Request 按钮后观察 LogCat 中的打印日志，你会看到和图 10.9 中一样的结果。

好了，这样我们就算是把 XML 和 JSON 这两种数据格式最常用的几种解析方法都学习完了，在网络数据的解析方面，你已经成功毕业了。

10.5 网络编程的最佳实践

目前你已经掌握了 HttpURLConnection 和 HttpClient 的用法，知道了如何发起 HTTP 请求，以及解析服务器返回的数据，但也许你还没有发现，之前我们的写法其实是很有问题的。因为一个应用程序很可能会在许多地方都使用到网络功能，而发送 HTTP 请求的代码基本都是相同的，如果我们每次都去编写一遍发送 HTTP 请求的代码，这显然是非常差劲的做法。

没错，通常情况下我们都应该将这些通用的网络操作提取到一个公共的类里，并提供一个静态方法，当想要发起网络请求的时候只需简单地调用一下这个方法即可。比如使用如下的写法：

```
public class HttpUtil {

    public static String sendHttpRequest(String address) {
        HttpURLConnection connection = null;
```

```
        try {
            URL url = new URL(address);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(8000);
            connection.setReadTimeout(8000);
            connection.setDoInput(true);
            connection.setDoOutput(true);
            InputStream in = connection.getInputStream();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line);
            }
            return response.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return e.getMessage();
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }

}
```

以后每当需要发起一条 HTTP 请求的时候就可以这样写：

```
String address = "http://www.baidu.com";
String response = HttpUtil.sendHttpRequest(address);
```

在获取到服务器响应的数据后我们就可以对它进行解析和处理了。但是需要注意，网络请求通常都是属于耗时操作，而 sendHttpRequest()方法的内部并没有开启线程，这样就有可能导致在调用 sendHttpRequest()方法的时候使得主线程被阻塞住。

你可能会说，很简单嘛，在 sendHttpRequest()方法内部开启一个线程不就解决这个问题了吗？其实不是像你想象中的那么容易，因为如果我们在 sendHttpRequest()方法中开启了一个线程来发起 HTTP 请求，那么服务器响应的数据是无法进行返回的，所有的耗时逻辑都是

在子线程里进行的，sendHttpRequest()方法会在服务器还来得及响应的时候就执行结束了，当然也就无法返回响应的数据了。

那么遇到这种情况应该怎么办呢？其实解决方法并不难，只需要使用 Java 的回调机制就可以了，下面就让我们来学习一下回调机制到底是如何使用的。

首先需要定义一个接口，比如将它命名成 HttpCallbackListener，代码如下所示：

```
public interface HttpCallbackListener {

    void onFinish(String response);

    void onError(Exception e);

}
```

可以看到，我们在接口中定义了两个方法，onFinish()方法表示当服务器成功响应我们请求的时候调用，onError()表示当进行网络操作出现错误的时候调用。这两个方法都带有参数，onFinish()方法中的参数代表着服务器返回的数据，而 onError()方法中的参数记录着错误的详细信息。

接着修改 HttpUtil 中的代码，如下所示：

```
public class HttpUtil {

    public static void sendHttpRequest(final String address, final
HttpCallbackListener listener) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                try {
                    URL url = new URL(address);
                    connection = (HttpURLConnection) url.openConnection();
                    connection.setRequestMethod("GET");
                    connection.setConnectTimeout(8000);
                    connection.setReadTimeout(8000);
                    connection.setDoInput(true);
                    connection.setDoOutput(true);
                    InputStream in = connection.getInputStream();
                    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
                    StringBuilder response = new StringBuilder();
```

```
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
                if (listener != null) {
                    // 回调onFinish()方法
                    listener.onFinish(response.toString());
                }
            } catch (Exception e) {
                if (listener != null) {
                    // 回调onError()方法
                    listener.onError(e);
                }
            } finally {
                if (connection != null) {
                    connection.disconnect();
                }
            }
        }
    }).start();
}

}
```

我们首先给 sendHttpRequest()方法添加了一个 HttpCallbackListener 参数，并在方法的内部开启了一个子线程，然后在子线程里去执行具体的网络操作。注意子线程中是无法通过 return 语句来返回数据的，因此这里我们将服务器响应的数据传入了 HttpCallbackListener 的 onFinish()方法中，如果出现了异常就将异常原因传入到 onError()方法中。

现在 sendHttpRequest()方法接收两个参数了，因此我们在调用它的时候还需要将 HttpCallbackListener 的实例传入，如下所示：

```
HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
    @Override
    public void onFinish(String response) {
        // 在这里根据返回内容执行具体的逻辑
    }

    @Override
    public void onError(Exception e) {
        // 在这里对异常情况进行处理
```

```
        }
    });
```

这样的话，当服务器成功响应的时候我们就可以在 onFinish()方法里对响应数据进行处理了，类似地，如果出现了异常，就可以在 onError()方法里对异常情况进行处理。如此一来，我们就巧妙地利用回调机制将响应数据成功返回给调用方了。

另外需要注意的是，onFinish()方法和 onError()方法最终还是在子线程中运行的，因此我们不可以在这里执行任何的 UI 操作，如果需要根据返回的结果来更新 UI，则仍然要使用上一章中我们学习的异步消息处理机制。

10.6 小结与点评

本章中我们主要学习了在 Android 中使用 HTTP 协议来进行网络交互的知识，虽然 Android 中支持的网络通信协议有很多种，但 HTTP 协议无疑是最常用的一种。通常我们有两种方式来发送 HTTP 请求，分别是 HttpURLConnection 和 HttpClient，相信这两种方式你都已经很好地掌握了吧？

接着我们又学习了 XML 和 JSON 格式数据的解析方式，因为服务器响应给我们的数据一般都是属于这两种格式的。无论是 XML 还是 JSON，它们各自又拥有多种的解析方式，这里我们只是学习了最常用的几种，如果以后你的工作中还需要用到其他的解析方式，可以自行去学习。

本章的最后同样是最佳实践环节，在这次的最佳实践中，我们主要学习了如何利用 Java 的回调机制来将服务器响应的数据进行返回。其实除此之外，还有很多地方都可以使用到 Java 的回调机制，希望你能举一反三，以后在其他地方需要用到回调机制时都能够灵活地使用。

好了，关于 Android 网络编程部分的内容就学习这么多，下一章中我们该去学习一下 Android 特色开发的相关内容了。

经验值：+22000　　目前经验值：202905

级别：头领鸟

赢得宝物：战胜魔界使者 JSON。拾取 JSON 掉落的宝物，JSON 智能解析器一套。与魔界左使 XML 相比，作为魔界右使的 JSON 功力显然更高杆一些。不过在如今的我面前，也已很难走上 5 个回合。