<http://www.360doc.com/content/07/0625/10/25392_578395.shtml>

其他入门：<http://www.360doc.com/content/06/1228/22/11192_311194.shtml>

一般的情况下我们都是使用IE或者Navigator浏览器来访问一个WEB服务器，用来浏览页面查看信息或者提交一些数据等等。所访问的这些页面有的仅仅是一些普通的页面，有的需要用户登录后方可使用，或者需要认证以及是一些通过加密方式传输，例如HTTPS。目前我们使用的浏览器处理这些情况都不会构成问题。不过你可能在某些时候需要通过程序来访问这样的一些页面，比如从别人的网页中“偷”一些数据；利用某些站点提供的页面来完成某种功能，例如说我们想知道某个手机号码的归属地而我们自己又没有这样的数据，因此只好借助其他公司已有的网站来完成这个功能，这个时候我们需要向网页提交手机号码并从返回的页面中解析出我们想要的数据来。如果对方仅仅是一个很简单的页面，那我们的程序会很简单，本文也就没有必要大张旗鼓的在这里浪费口舌。但是考虑到一些服务授权的问题，很多公司提供的页面往往并不是可以通过一个简单的URL就可以访问的，而必须经过注册然后登录后方可使用提供服务的页面，这个时候就涉及到COOKIE问题的处理。我们知道目前流行的动态网页技术例如ASP、JSP无不是通过COOKIE来处理会话信息的。为了使我们的程序能使用别人所提供的服务页面，就要求程序首先登录后再访问服务页面，这过程就需要自行处理cookie，想想当你用java.net.HttpURLConnection来完成这些功能时是多么恐怖的事情啊！况且这仅仅是我们所说的顽固的WEB服务器中的一个很常见的“顽固”！再有如通过HTTP来上传文件呢？不需要头疼，这些问题有了“它”就很容易解决了！

我们不可能列举所有可能的顽固，我们会针对几种最常见的问题进行处理。当然了，正如前面说到的，如果我们自己使用java.net.HttpURLConnection来搞定这些问题是很恐怖的事情，因此在开始之前我们先要介绍一下一个开放源码的项目，这个项目就是Apache开源组织中的httpclient，它隶属于Jakarta的commons项目，目前的版本是2.0RC2。commons下本来已经有一个net的子项目，但是又把httpclient单独提出来，可见http服务器的访问绝非易事。

Commons-httpclient项目就是专门设计来简化HTTP客户端与服务器进行各种通讯编程。通过它可以让原来很头疼的事情现在轻松的解决，例如你不再管是HTTP或者HTTPS的通讯方式，告诉它你想使用HTTPS方式，剩下的事情交给httpclient替你完成。本文会针对我们在编写HTTP客户端程序时经常碰到的几个问题进行分别介绍如何使用httpclient来解决它们，为了让读者更快的熟悉这个项目我们最开始先给出一个简单的例子来读取一个网页的内容，然后循序渐进解决掉前进中的所有问题。

1． 读取网页(HTTP/HTTPS)内容

下面是我们给出的一个简单的例子用来访问某个页面

/\*

 \* Created on 2003-12-14 by Liudong

 \*/

**package** http.demo;

**import** java.io.IOException;

**import** org.apache.commons.httpclient.\*;

**import** org.apache.commons.httpclient.methods.\*;

/\*\*

 \* 最简单的HTTP客户端,用来演示通过GET或者POST方式访问某个页面

 \* **@author** Liudong

 \*/

**public** **class** SimpleClient {

**public** **static** **void** main(String[] args) **throws** IOException

    {

        HttpClient client = **new** HttpClient();

        //设置代理服务器地址和端口

        //client.getHostConfiguration().setProxy("proxy\_host\_addr",proxy\_port);

        //使用GET方法，如果服务器需要通过HTTPS连接，那只需要将下面URL中的http换成https

        HttpMethod method = **new** GetMethod("http://java.sun.com");

        //使用POST方法

        //HttpMethod method = new PostMethod("http://java.sun.com");

        client.executeMethod(method);

        //打印服务器返回的状态

        System.out.println(method.getStatusLine());

        //打印返回的信息

        System.out.println(method.getResponseBodyAsString());

        //释放连接

        method.releaseConnection();

    }  
}

在这个例子中首先创建一个HTTP客户端(HttpClient)的实例，然后选择提交的方法是GET或者POST，最后在HttpClient实例上执行提交的方法，最后从所选择的提交方法中读取服务器反馈回来的结果。这就是使用HttpClient的基本流程。其实用一行代码也就可以搞定整个请求的过程，非常的简单！

2． 以GET或者POST方式向网页提交参数  
  
其实前面一个最简单的示例中我们已经介绍了如何使用GET或者POST方式来请求一个页面，本小节与之不同的是多了提交时设定页面所需的参数，我们知道如果是GET的请求方式，那么所有参数都直接放到页面的URL后面用问号与页面地址隔开，每个参数用&隔开，例如：[http://java.sun.com?name=liudong&mobile=123456](http://java.sun.com/?name=liudong&mobile=123456)，但是当使用POST方法时就会稍微有一点点麻烦。本小节的例子演示向如何查询手机号码所在的城市，代码如下：

/\*

 \* Created on 2003-12-7 by Liudong

 \*/

**package** http.demo;

**import** java.io.IOException;

**import** org.apache.commons.httpclient.\*;

**import** org.apache.commons.httpclient.methods.\*;

/\*\*

 \* 提交参数演示

 \* 该程序连接到一个用于查询手机号码所属地的页面

 \* 以便查询号码段1330227所在的省份以及城市

 \* **@author** Liudong

 \*/

**public** **class** SimpleHttpClient {

**public** **static** **void** main(String[] args) **throws** IOException

    {

        HttpClient client = **new** HttpClient();

        client.getHostConfiguration().setHost("www.imobile.com.cn", 80, "http");

        HttpMethod method = getPostMethod();//使用POST方式提交数据

        client.executeMethod(method);

       //打印服务器返回的状态

        System.out.println(method.getStatusLine());

        //打印结果页面

        String response =

**new** String(method.getResponseBodyAsString().getBytes("8859\_1"));

       //打印返回的信息

        System.out.println(response);

        method.releaseConnection();

    }

    /\*\*

     \* 使用GET方式提交数据

     \* **@return**

     \*/

**private** **static** HttpMethod getGetMethod(){

**return** **new** GetMethod("/simcard.php?simcard=1330227");

    }

    /\*\*

     \* 使用POST方式提交数据

     \* **@return**

     \*/

**private** **static** HttpMethod getPostMethod(){

        PostMethod post = **new** PostMethod("/simcard.php");

        NameValuePair simcard = **new** NameValuePair("simcard","1330227");

        post.setRequestBody(**new** NameValuePair[] { simcard});

**return** post;

    }

}

在上面的例子中页面<http://www.imobile.com.cn/simcard.php>需要一个参数是simcard，这个参数值为手机号码段，即手机号码的前七位，服务器会返回提交的手机号码对应的省份、城市以及其他详细信息。GET的提交方法只需要在URL后加入参数信息，而POST则需要通过NameValuePair类来设置参数名称和它所对应的值

3． 处理页面重定向

在JSP/Servlet编程中response.sendRedirect方法就是使用HTTP协议中的重定向机制。它与JSP中的<jsp:forward …>的区别在于后者是在服务器中实现页面的跳转，也就是说应用容器加载了所要跳转的页面的内容并返回给客户端；而前者是返回一个状态码，这些状态码的可能值见下表，然后客户端读取需要跳转到的页面的URL并重新加载新的页面。就是这样一个过程，所以我们编程的时候就要通过HttpMethod.getStatusCode()方法判断返回值是否为下表中的某个值来判断是否需要跳转。如果已经确认需要进行页面跳转了，那么可以通过读取HTTP头中的location属性来获取新的地址。

|  |  |  |
| --- | --- | --- |
| 状态码 | 对应HttpServletResponse的常量 | 详细描述 |
| 301 | SC\_MOVED\_PERMANENTLY | 页面已经永久移到另外一个新地址 |
| 302 | SC\_MOVED\_TEMPORARILY | 页面暂时移动到另外一个新的地址 |
| 303 | SC\_SEE\_OTHER | 客户端请求的地址必须通过另外的URL来访问 |
| 307 | SC\_TEMPORARY\_REDIRECT | 同SC\_MOVED\_TEMPORARILY |

下面的代码片段演示如何处理页面的重定向

client.executeMethod(post);

        System.out.println(post.getStatusLine().toString());

        post.releaseConnection();

        //检查是否重定向

**int** statuscode = post.getStatusCode();

**if** ((statuscode == HttpStatus.SC\_MOVED\_TEMPORARILY) ||

            (statuscode == HttpStatus.SC\_MOVED\_PERMANENTLY) ||

            (statuscode == HttpStatus.SC\_SEE\_OTHER) ||

(statuscode == HttpStatus.SC\_TEMPORARY\_REDIRECT)) {

//读取新的URL地址

            Header header = post.getResponseHeader("location");

**if** (header != **null**) {

                String newuri = header.getValue();

**if** ((newuri == **null**) || (newuri.equals("")))

                    newuri = "/";

                GetMethod redirect = **new** GetMethod(newuri);

                client.executeMethod(redirect);

                System.out.println("Redirect:"+ redirect.getStatusLine().toString());

                redirect.releaseConnection();

            } **else**

                System.out.println("Invalid redirect");

        }

我们可以自行编写两个JSP页面，其中一个页面用response.sendRedirect方法重定向到另外一个页面用来测试上面的例子。

4． 模拟输入用户名和口令进行登录

本小节应该说是HTTP客户端编程中最常碰见的问题，很多网站的内容都只是对注册用户可见的，这种情况下就必须要求使用正确的用户名和口令登录成功后，方可浏览到想要的页面。因为HTTP协议是无状态的，也就是连接的有效期只限于当前请求，请求内容结束后连接就关闭了。在这种情况下为了保存用户的登录信息必须使用到Cookie机制。以JSP/Servlet为例，当浏览器请求一个JSP或者是Servlet的页面时，应用服务器会返回一个参数，名为jsessionid（因不同应用服务器而异），值是一个较长的唯一字符串的Cookie，这个字符串值也就是当前访问该站点的会话标识。浏览器在每访问该站点的其他页面时候都要带上jsessionid这样的Cookie信息，应用服务器根据读取这个会话标识来获取对应的会话信息。

对于需要用户登录的网站，一般在用户登录成功后会将用户资料保存在服务器的会话中，这样当访问到其他的页面时候，应用服务器根据浏览器送上的Cookie中读取当前请求对应的会话标识以获得对应的会话信息，然后就可以判断用户资料是否存在于会话信息中，如果存在则允许访问页面，否则跳转到登录页面中要求用户输入帐号和口令进行登录。这就是一般使用JSP开发网站在处理用户登录的比较通用的方法。

这样一来，对于HTTP的客户端来讲，如果要访问一个受保护的页面时就必须模拟浏览器所做的工作，首先就是请求登录页面，然后读取Cookie值；再次请求登录页面并加入登录页所需的每个参数；最后就是请求最终所需的页面。当然在除第一次请求外其他的请求都需要附带上Cookie信息以便服务器能判断当前请求是否已经通过验证。说了这么多，可是如果你使用httpclient的话，你甚至连一行代码都无需增加，你只需要先传递登录信息执行登录过程，然后直接访问想要的页面，跟访问一个普通的页面没有任何区别，因为类HttpClient已经帮你做了所有该做的事情了，太棒了！下面的例子实现了这样一个访问的过程。

/\*

 \* Created on 2003-12-7 by Liudong

 \*/

**package** http.demo;

**import** org.apache.commons.httpclient.\*;

**import** org.apache.commons.httpclient.cookie.\*;

**import** org.apache.commons.httpclient.methods.\*;

/\*\*

 \* 用来演示登录表单的示例

 \* **@author** Liudong

 \*/

**public** **class** FormLoginDemo {

**static** **final** String LOGON\_SITE = "localhost";

**static** **final** **int**    LOGON\_PORT = 8080;

**public** **static** **void** main(String[] args) **throws** Exception{

        HttpClient client = **new** HttpClient();

        client.getHostConfiguration().setHost(LOGON\_SITE, LOGON\_PORT);

       //模拟登录页面login.jsp->main.jsp

        PostMethod post = **new** PostMethod("/main.jsp");

        NameValuePair name = **new** NameValuePair("name", "ld");

        NameValuePair pass = **new** NameValuePair("password", "ld");

        post.setRequestBody(**new** NameValuePair[]{name,pass});

**int** status = client.executeMethod(post);

        System.out.println(post.getResponseBodyAsString());

        post.releaseConnection();

       //查看cookie信息

        CookieSpec cookiespec = CookiePolicy.getDefaultSpec();

        Cookie[] cookies = cookiespec.match(LOGON\_SITE, LOGON\_PORT, "/", **false**, client.getState().getCookies());

**if** (cookies.length == 0) {

           System.out.println("None");

       } **else** {

**for** (**int** i = 0; i < cookies.length; i++) {

               System.out.println(cookies[i].toString());

           }

       }

       //访问所需的页面main2.jsp

        GetMethod get = **new** GetMethod("/main2.jsp");

        client.executeMethod(get);

        System.out.println(get.getResponseBodyAsString());

        get.releaseConnection();

    }

}

5． 提交XML格式参数

提交XML格式的参数很简单，仅仅是一个提交时候的ContentType问题，下面的例子演示从文件文件中读取XML信息并提交给服务器的过程，该过程可以用来测试Web服务。

**import** java.io.File;

**import** java.io.FileInputStream;

**import** org.apache.commons.httpclient.HttpClient;

**import** org.apache.commons.httpclient.methods.EntityEnclosingMethod;

**import** org.apache.commons.httpclient.methods.PostMethod;

/\*\*

 \* 用来演示提交XML格式数据的例子

 \*/

**public** **class** PostXMLClient {

**public** **static** **void** main(String[] args) **throws** Exception {

        File input = **new** File(“test.xml”);

        PostMethod post = **new** PostMethod(“http://localhost:8080/httpclient/xml.jsp”);

        // 设置请求的内容直接从文件中读取

        post.setRequestBody(**new** FileInputStream(input));

**if** (input.length() < Integer.MAX\_VALUE)

            post.setRequestContentLength(input.length());

**else**            post.setRequestContentLength(EntityEnclosingMethod.CONTENT\_LENGTH\_CHUNKED);

        // 指定请求内容的类型

**post.setRequestHeader("Content-type", "text/xml; charset=GBK");**

        HttpClient httpclient = **new** HttpClient();

**int** result = httpclient.executeMethod(post);

        System.out.println("Response status code: " + result);

        System.out.println("Response body: ");

        System.out.println(post.getResponseBodyAsString());

        post.releaseConnection();

    }

}

6． 通过HTTP上传文件

httpclient使用了单独的一个HttpMethod子类来处理文件的上传，这个类就是MultipartPostMethod，该类已经封装了文件上传的细节，我们要做的仅仅是告诉它我们要上传文件的全路径即可，下面的代码片段演示如何使用这个类。

MultipartPostMethod filePost = **new** MultipartPostMethod(targetURL);

filePost.addParameter("fileName", targetFilePath);

HttpClient client = **new** HttpClient();

//由于要上传的文件可能比较大,因此在此设置最大的连接超时时间

client.getHttpConnectionManager().getParams().setConnectionTimeout(5000);

**int** status = client.executeMethod(filePost);

上面代码中，targetFilePath即为要上传的文件所在的路径。

7． 访问启用认证的页面

我们经常会碰到这样的页面，当访问它的时候会弹出一个浏览器的对话框要求输入用户名和密码后方可，这种用户认证的方式不同于我们在前面介绍的基于表单的用户身份验证。这是HTTP的认证策略，httpclient支持三种认证方式包括：基本、摘要以及NTLM认证。其中基本认证最简单、通用但也最不安全；摘要认证是在HTTP 1.1中加入的认证方式，而NTLM则是微软公司定义的而不是通用的规范，最新版本的NTLM是比摘要认证还要安全的一种方式。

下面例子是从httpclient的CVS服务器中下载的，它简单演示如何访问一个认证保护的页面：

**import** org.apache.commons.httpclient.HttpClient;

**import** org.apache.commons.httpclient.UsernamePasswordCredentials;

**import** org.apache.commons.httpclient.methods.GetMethod;

**public** **class** BasicAuthenticationExample {

**public** BasicAuthenticationExample() {

    }

**public** **static** **void** main(String[] args) **throws** Exception {

        HttpClient client = **new** HttpClient();

        client.getState().setCredentials(

            "www.verisign.com",

            "realm",

**new** UsernamePasswordCredentials("username", "password")

        );

        GetMethod get = **new** GetMethod("https://www.verisign.com/products/index.html");

        get.setDoAuthentication( **true** );

**int** status = client.executeMethod( get );

        System.out.println(status+"\n"+ get.getResponseBodyAsString());

        get.releaseConnection();

    }

}

8． 多线程模式下使用httpclient

多线程同时访问httpclient，例如同时从一个站点上下载多个文件。对于同一个HttpConnection同一个时间只能有一个线程访问，为了保证多线程工作环境下不产生冲突，httpclient使用了一个多线程连接管理器的类：MultiThreadedHttpConnectionManager，要使用这个类很简单，只需要在构造HttpClient实例的时候传入即可，代码如下：

MultiThreadedHttpConnectionManager connectionManager =

**new** MultiThreadedHttpConnectionManager();

HttpClient client = **new** HttpClient(connectionManager);

以后尽管访问client实例即可。

参考资料：

httpclient首页：    <http://jakarta.apache.org/commons/httpclient/>  
关于NTLM是如何工作：  <http://davenport.sourceforge.net/ntlm.html>