www.leadfar.org

李腾飞学习笔记
HttpClient 入门

版本 1.0

修订历史记录

日期	版本	说明	作者
8/12/2010	1.0	创建	李腾飞

目录

HttpClient 入门

1. HttpClient 概述

HttpClient 是 apache 组织下面的一个用于处理 HTTP 请求和响应的开源工具。它不是一个浏览器，也不处理客户端缓存等浏览器的功能。它只是一个类库！它在 JDK 的基本类库基础上做了更好的封装！
HttpClient 目前（写作本文日期：2010 年 8 月）最新版本是 4.0.1，官方网址：
http://hc.apache.org/httpcomponents-client-4.0.1/index.html

HttpClient 项目依赖于 HttpCore（处理核心的 HTTP 协议）、commons-codec（处理与编码有关的问题的项目）和 commons-logging（处理与日志记录有关问题的项目）。

如果你希望能够通过 HttpClient 向服务器上传文件等与 multipart 编码类型有关的请求，以及其它复杂的 MIME 类型，那么，你需要另外一个依赖包：HttpMime（它是专门处理与 MIME 类型有关问题的项目），在下载的 HttpClient 包中（下载地址为：http://hc.apache.org/downloads.cgi）已经包含了 HttpMime。

在本文中，我们使用的 HttpClient 版本为：4.0.1GA

开始使用 HttpClient，我们加入了下列依赖包：
httpclient-4.0.1.jar
httpcore-4.0.1.jar
httpmime-4.0.1.jar
　　- 又依赖于 mime4j（apache-mime4j-0.6.jar）
commons-codec-1.4.jar
commons-logging-1.1.1.jar
commons-io-1.4.jar – 为了更方便处理与 IO 有关的需求

【如果你下载的是包含依赖包的压缩包，那么上述依赖包（除 commons-io-1.4.jar 外）都已经在其中了！】

2. 了解 JDK 中有关 HTTP URL 处理的 API

因为 HttpClient 是对 JDK 中 java.net.*包下面的有关基础类库的封装，所以，我们有必要了解一下，这些基础类库的简单用法和概念。

2.1 最简单的获取网页内容的示例

```
try {
	String urlString = "http://localhost:8080/cms/";
	URL url = new URL(urlString); //代表了一个网址
	InputStream is =  url.openStream(); //获得网页的内容

	//将InputStream转换为Reader，并使用缓冲读取，提高效率，同时可以按行读取内容
	BufferedReader br = new BufferedReader(new InputStreamReader(is,"UTF-
8"));
	String line = null;
	while((line = br.readLine()) != null){
		System.out.println(line);
	}
	is.close();
```

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

上述例子，直接通过 URL 获得输入流，然后对其进行处理！

2.2 URLConnection 的基本用法

上述例子太过简单，假如你需要通过代理来访问网络，那么，你需要的是 URLConnection！即，在获取内容之前，先设置代理！

```
      public void testFetch02(){
            try {
                  String urlString =
"http://www.ibm.com/developerworks/cn/java/j-javaroundtable/index.html";
                  URL url = new URL(urlString); //代表了一个网址

                  //首先创建HTTP代理，指定代理的地址和端口
                  Proxy proxy = new Proxy(Proxy.Type.HTTP,new
InetSocketAddress("79.120.193.53",80));

                  /**
                   * 首先打开一个连接对象
                   * 可以通过这个对象，在真正发起请求之前，设置一些其它的信息
                   * 比如：代理服务器等
                   */
                  URLConnection conn = url.openConnection(proxy);
                  InputStream is =  conn.getInputStream(); //获得网页的内容

                  //将InputStream转换为Reader，并使用缓冲读取，提高效率，同时可以按行
读取内容
                  BufferedReader br = new BufferedReader(new
InputStreamReader(is,"UTF-8"));
                  String line = null;
                  while((line = br.readLine()) != null){
                        System.out.println(line);
                  }
                  is.close();
            } catch (Exception e) {
                  e.printStackTrace();
            }
```

可以设置 HTTP 或 SOCKS 代理，只要指定不同的代理类型即可！

2.3 HttpURLConnection 的用法

HttpURLConnection 是 URLConnection 的子类，它提供了更多与 HTTP 有关的处理方法，
* 比如：如果你希望获得服务器响应的HTTP代码，比如：2XX，3XX等
* 比如：你希望设置是否自动进行客户端重定向（缺省是自动重定向）
* 比如：你希望指定向服务器提交的 HTTP METHOD（GET 或 POST 等）

等等等等

```
            try {
                  String urlString =
"http://localhost:8080/cms/backend/main.jsp";
                  URL url = new URL(urlString); //代表了一个网址

                  //设置是否自动进行重定向，缺省这个值为true
                  HttpURLConnection.setFollowRedirects(false);
                  HttpURLConnection conn =
(HttpURLConnection)url.openConnection();
                  //设置HTTP METHOD
                  conn.setRequestMethod("GET");
                  int code = conn.getResponseCode();
                  System.out.println("服务器响应代码为: "+code);
                  InputStream is =  conn.getInputStream();

                  //将InputStream转换为Reader，并使用缓冲读取，提高效率，同时可以按行
读取内容
                  BufferedReader br = new BufferedReader(new
InputStreamReader(is,"UTF-8"));
                  String line = null;
                  while((line = br.readLine()) != null){
                        System.out.println(line);
                  }
                  is.close();
            } catch (Exception e) {
                  e.printStackTrace();
            }
```

上述例子中，由于设置了不自动重定向，所以，加入 main.jsp 返回的是一个 3XX 代码（客户端重定向），那么将获取不到任何内容！假如设置了自动重定向，将获得重定向之后的网页的内容！

3. 使用 HttpClient 获取网页内容

在进行下述例子之前，请添加上面所用到的依赖包！

3.1 使用 GET 方式向后台递交请求

```
      public void testFetch01(){
            try {
                  //HttpClient主要负责执行请求
                  HttpClient httpclient = new DefaultHttpClient();

                  //利用HTTP GET向服务器发起请求
                  HttpGet get = new HttpGet("http://localhost:8080/cms");

                  //获得服务器响应的的所有信息
                  HttpResponse response = httpclient.execute(get);

                  //获得服务器响应回来的消息体（不包括HTTP HEAD）
```

```
                    HttpEntity entity = response.getEntity();
                    if(entity != null){
                          InputStream is =  entity.getContent();
                          //将InputStream转换为Reader，并使用缓冲读取，提高效率，同时
可以按行读取内容
                          BufferedReader br = new BufferedReader(new
InputStreamReader(is,"UTF-8"));
                          String line = null;
                          while((line = br.readLine()) != null){
                                System.out.println(line);
                          }

                          is.close();
                    }

                    //释放所有的链接资源，一般在所有的请求处理完成之后，才需要释放
                    httpclient.getConnectionManager().shutdown();
              } catch (ClientProtocolException e) {
                    e.printStackTrace();
              } catch (IOException e) {
                    e.printStackTrace();
              }
        }
```

3.2 自动获得响应的编码信息

我们希望程序能够自动分辨响应的内容的编码

```
        public void testFetch02(){
              try {
                    //HttpClient主要负责执行请求
                    HttpClient httpclient = new DefaultHttpClient();

                    //利用HTTP GET向服务器发起请求
                    HttpGet get = new HttpGet("http://www.baidu.com/");//new
HttpGet("http://localhost:8080/cms");

                    //获得服务器响应的的所有信息
                    HttpResponse response = httpclient.execute(get);

                    //获得服务器响应回来的消息体（不包括HTTP HEAD）
                    HttpEntity entity = response.getEntity();
                    if(entity != null){
                          //获得响应的字符集编码信息
                          //即获取HTTP HEAD的：Content-
Type:text/html;charset=UTF-8中的字符集信息
                          String charset =
EntityUtils.getContentCharSet(entity);
                          System.out.println("响应的字符集是："+charset);
                          InputStream is =  entity.getContent();
                          //使用响应中的编码来解释响应的内容
                          BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                          String line = null;
                          while((line = br.readLine()) != null){
```

```
                        System.out.println(line);
                    }

                    is.close();
                }

                //释放所有的链接资源，一般在所有的请求处理完成之后，才需要释放
                httpclient.getConnectionManager().shutdown();
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

上述例子，主要是通过 **EntityUtils.*getContentCharSet*** 方法来获得编码信息

3.3 设置代理服务器，访问网站

```
        public void testFetch03(){
            try {
                //HttpClient主要负责执行请求
                HttpClient httpclient = new DefaultHttpClient();

                //设置代理服务器

        httpclient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY, new
HttpHost("121.12.249.207",3128));

                //利用HTTP GET向服务器发起请求
                HttpGet get = new HttpGet("http://www.baidu.com/");//new
HttpGet("http://localhost:8080/cms");

                //获得服务器响应的的所有信息
                HttpResponse response = httpclient.execute(get);

                //获得服务器响应回来的消息体（不包括HTTP HEAD）
                HttpEntity entity = response.getEntity();
                if(entity != null){
                    //获得响应的字符集编码信息
                    //即获取HTTP HEAD的：Content-
Type:text/html;charset=UTF-8中的字符集信息
                    String charset =
EntityUtils.getContentCharSet(entity);
                    System.out.println("响应的字符集是："+charset);
                    InputStream is =  entity.getContent();
                    //使用响应中的编码来解释响应的内容
                    BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                    String line = null;
                    while((line = br.readLine()) != null){
                        System.out.println(line);
                    }
```

```
                        is.close();
                    }

                    //释放所有的链接资源，一般在所有的请求处理完成之后，才需要释放
                    httpclient.getConnectionManager().shutdown();

                } catch (ClientProtocolException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
```

主要通过 `httpclient.getParams().setParameter` 方法设置代理服务器。
`HttpHost` 是 `HttpClient` 中用来表示一台主机的类。

3.4 获得重定向之后的网址信息

HttpClient 缺省情况下自动处理客户端重定向，即当你访问网页（比如 A 网页）之后，假设被重定向到了 B 网页，那么，HttpClient 将自动返回 B 网页的内容，无需再编程处理它！有时候我们可能想要知道 A 网页被重定向到了哪里，也就是取得 B 网页的网址，那么可以通过下述例子获得：

```
        public void testFetch04(){
                try {
                    //HttpClient主要负责执行请求
                    HttpClient httpclient = new DefaultHttpClient();

                    HttpContext context = new BasicHttpContext();

                    //利用HTTP GET向服务器发起请求
                    HttpGet get = new
HttpGet("http://localhost:8080/cms/backend/main.jsp");

                    //获得服务器响应的的所有信息
                    HttpResponse response = httpclient.execute(get,context);

                    //获得重定向之后的主机地址信息
                    HttpHost targetHost =
(HttpHost)context.getAttribute(ExecutionContext.HTTP_TARGET_HOST);
                    System.out.println(targetHost); // http://localhost:8080

                    //获得实际的请求对象的URI（即重定向之后的
"/cms/backend/login.jsp"）
                    HttpUriRequest actualRequest = (HttpUriRequest)
context.getAttribute(ExecutionContext.HTTP_REQUEST);
                    System.out.println(actualRequest.getURI());

                    //获得服务器响应回来的消息体（不包括HTTP HEAD）
                    HttpEntity entity = response.getEntity();

                    if(entity != null){
                        //获得响应的字符集编码信息
```

```
                        //即获取HTTP HEAD的：Content-
Type:text/html;charset=UTF-8中的字符集信息
                        String charset =
EntityUtils.getContentCharSet(entity);
                        System.out.println("响应的字符集是："+charset);
                        InputStream is =  entity.getContent();
                        //使用响应中的编码来解释响应的内容
                        BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                        String line = null;
                        while((line = br.readLine()) != null){
                              System.out.println(line);
                        }

                        is.close();
                  }

                  //释放所有的链接资源，一般在所有的请求处理完成之后，才需要释放
                  httpclient.getConnectionManager().shutdown();
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
```

HttpContext，实际上是客户端用来在多次请求-响应的交互中，保持状态信息用的。假如我们在调用 httpclient.execute 方法的时候，将 HttpContext 对象作为参数传给这个方法（请看上述例子），那么 HttpClient 将把请求-响应交互过程中的状态信息存储在 HttpContext 中。

比如上面的例子中，HttpClient 把主机信息和真正的请求对象（所谓真正的请求对象，因为我们发出的是 main.jsp 的请求，但这个请求实际上被重定向到了 login.jsp，所以真正的请求对象实际上是 login.jsp）等信息（请参考文档说明）放到了 HttpContext 中！

我们自己也可以利用 HttpContext 来存放一些我们想要存放的其它信息，以便下次请求的时候，能够把这些信息拿出来使用！

3.5 自动 Cookie 处理

HttpClient 能够支持自动 Cookie 处理。设想一个典型的场景：首先打开登录页面，然后输入用户名和密码登录，然后访问那些只有登录之后才能访问的网页……
如果我们用浏览器，因为浏览器可以将登录之后的会话信息用 Cookie 存储在本地，所以，登录之后的每次请求，都会自动向服务器发送 Cookie 的信息，我们利用 HttpClient，这些过程都全部可以自动化处理了。

```
      public void testFetch05(){
            try {
                  //HttpClient主要负责执行请求
                  HttpClient httpclient = new DefaultHttpClient();

                  HttpContext context = new BasicHttpContext();
```

```
                    //利用HTTP GET向服务器发起请求，
                    HttpGet get = new
HttpGet("http://localhost:8080/cms/backend/login.jsp");

                    //获得服务器响应的的所有信息
                    HttpResponse response = httpclient.execute(get,context);

                    //获得服务器响应回来的消息体（不包括HTTP HEAD）
                    HttpEntity entity = response.getEntity();
                    String charset = null;
                    if(entity != null){
                          //获得响应的字符集编码信息
                          //即获取HTTP HEAD的：Content-
Type:text/html;charset=UTF-8中的字符集信息
                          charset = EntityUtils.getContentCharSet(entity);
                          System.out.println("响应的字符集是："+charset);
                          InputStream is =  entity.getContent();
                          //使用响应中的编码来解释响应的内容
                          BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                          String line = null;
                          while((line = br.readLine()) != null){
                                System.out.println(line);
                          }

                          is.close();
                    }

                    //************* 执行登录请求 ********************//
                    HttpPost post = new
HttpPost("http://localhost:8080/cms/backend/LoginServlet");

                    //添加POST参数
                    List<NameValuePair> nvps = new ArrayList<NameValuePair>();
                    nvps.add(new BasicNameValuePair("username","admin"));
                    nvps.add(new BasicNameValuePair("password","admin"));
                    post.setEntity(new UrlEncodedFormEntity(nvps,charset));

                    response = httpclient.execute(post);
                    entity = response.getEntity();
                    if(entity != null){
                          InputStream is =  entity.getContent();
                          //使用响应中的编码来解释响应的内容
                          BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                          String line = null;
                          while((line = br.readLine()) != null){
                                System.out.println(line);
                          }

                          is.close();
                    }

                    //******************* 请求文章查询 ********************//
```

```
                    get = new
HttpGet("http://localhost:8080/cms/backend/ArticleServlet");
                    response = httpclient.execute(get);
                    entity = response.getEntity();
                    if(entity != null){
                          InputStream is =  entity.getContent();
                          //使用响应中的编码来解释响应的内容
                          BufferedReader br = new BufferedReader(new
InputStreamReader(is,charset));
                          String line = null;
                          while((line = br.readLine()) != null){
                                System.out.println(line);
                          }

                          is.close();
                    }

                    //释放所有的链接资源，一般在所有的请求处理完成之后，才需要释放
                    httpclient.getConnectionManager().shutdown();
              } catch (Exception e) {
                    e.printStackTrace();
              }
        }
```