<http://blog.csdn.net/lmb55/article/details/70246185>

HTTP协议时Internet上使用的很多也很重要的一个协议，越来越多的java应用程序需要通过HTTP协议来访问网络资源。   
**HTTPClient提供的主要功能：**

1、实现了所有HTTP的方法（GET、POST、等PUT、HEAD）；   
2、支持自动转向；   
3、支持HTTPS协议；   
4、支持代理服务器等。

**使用HttpClient需要以下6个步骤：**

1. 创建HttpClient的实例
2. 创建某种连接方法的实例，GetMethod/PostMethod。在 GetMethod/PostMethod的构造函数中传入待连接的地址
3. 调用第一步中创建好的实例的 execute 方法来执行第二步中创建好的 method 实例
4. 读response
5. 释放连接。无论执行方法是否成功，都必须释放连接
6. 对得到后的内容进行处理

**HTTP GET方法应用：**   
**应用一：**

String url = "";

//构造HttpClient实例

HttpClient httpClient = new HttpClient();

//创建Get方法实例

GetMethod getMethod = new GetMethod(url);

//使用系统提供的默认的恢复策略

getMethod.getParams().setParameter(HttpMethodParams.RETRY\_HANDLER,

new DefaultHttpMethodRetryHandler( ));

try{

int statusCode = httpClient.executeMethod(getMethod);

if (statusCode == HttpStatus.SC\_OK) {

//方法一：

byte[] respByte = getMethod.getResponseBody();

logger.info("返回信息：" + new String(respByte));

//方法二：

String respStr = getMethod.getResponseBodyAsString();

logger.info("返回信息：" + respStr);

}

logger.error("Method failed: "+ getMethod.getStatusLine());

}catch(HttpException e){

logger.error("发生致命的异常，可能是协议不对或者返回的内容有问题",e);

}catch(IOException e){

logger.error("网络异常",e);

}finally{

//释放连接

getMethod.releaseConnection();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28

**应用二：**

public String getRequest(String url, String requestStr) {

logger.debug("the getRest'params : url = " + "http://" + url + "?" + requestStr);

String respStr = null;

try {

// 定义HttpClient

DefaultHttpClient client = new DefaultHttpClient();

// 实例化HTTP方法

HttpGet request = new HttpGet();

url = url + "?" + requestStr;

url = URLEncoder.encode(url, "UTF-8");

request.setURI(new URI("http://" + url));

HttpResponse response = client.execute(request);

logger.debug("response status ： " + response.getStatusLine().getStatusCode());

/\*\*请求发送成功，并得到响应\*\*/

if (response.getStatusLine().getStatusCode() == HttpStatus.SC\_OK) {

/\*\*读取服务器返回过来的数据\*\*/

respStr = EntityUtils.toString(response.getEntity());

} else {

logger.error("get request fail:" + url);

}

}catch(Exception e){

logger.error("get request fail:" + url, e);

}

return respStr;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

**HTTP POST方法应用:**   
POST方法用来向目的服务器发出请求，要求它接受被附在请求后的实体，并把它当作请求队列（Request-Line）中请求URI所指定资源的附加新子项。   
**应用一：**

String url = "";

String requestStr = "";

//创建HttpClient实例

HttpClient httpClient = new HttpClient();

//创建Post方法实例

PostMethod postMethod = new PostMethod(url);

//\*\*\*\*\*\*方法一：创建请求实体,发送请求start\*\*\*\*\*\*\*\*\*\*\*\*\*

byte[] b = requestStr.getBytes();

InputStream is = new ByteArrayInputStream(b, 0, b.length);

RequestEntity re = new InputStreamRequestEntity(is, b.length, "text/xml; charset=UTF-8");//GBK

//设置请求头

postMethod.addRequestHeader("Content-Type", "application/x-www-form-urlencoded;charset=utf-8");

//设置请求体

postMethod.setRequestEntity(re);

//\*\*\*\*\*\*创建请求实体，发送请求end\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*方法二：创建请求体,发送请求start\*\*\*\*\*\*\*\*\*\*\*\*\*

// 构造名称值对节点类对象

NameValuePair[] data = {

new NameValuePair("id", "yourUserName"),

new NameValuePair("passwd", "yourPwd") };

// 设置请求体

postMethod.setRequestBody(data);

//\*\*\*\*\*\*创建请求体，发送请求end\*\*\*\*\*\*\*\*\*\*\*\*\*

try {

int statusCode=httpClient.executeMethod(postMethod);

if (statusCode == HttpStatus.SC\_OK) {

//方法一：

String responseStr = postMethod.getResponseBodyAsString();

logger.info("返回信息：" + responseStr);

//方法二：

byte[] responseByte = postMethod.getResponseBody();

logger.info("返回信息：" + new String(responseByte));

}

logger.error("Method failed: "+ postMethod.getStatusLine());

} catch(HTTPException e){

logger.error("发生致命的异常，可能是协议不对或者返回的内容有问题",e);

}catch (IOException e) {

logger.error("网络异常",e);

}finally{

//释放连接

postMethod.releaseConnection();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45

**注意：（自动转向问题的代码实现）**

// HttpClient对于要求接受后继服务的请求，象POST和PUT等不能自动处理转发

// 301（永久移走）或者302（暂时转向）

if (statusCode == HttpStatus.SC\_MOVED\_PERMANENTLY || statusCode == HttpStatus.SC\_MOVED\_TEMPORARILY) {

// 从头中取出转向的地址

Header locationHeader = (Header) postMethod.getResponseHeader("location");

if (locationHeader != null) {

String location = locationHeader.getValue();

logger.info("The page was redirected to:" + location);

} else {

logger.info("Location field value is null.");

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

**应用二：**

public static JSONObject httpPost(String url,JSONObject jsonParam, boolean noNeedResponse){

//post请求返回结果

DefaultHttpClient httpClient = new DefaultHttpClient();

JSONObject jsonResult = null;

HttpPost method = new HttpPost(url);

try {

if (null != jsonParam) {

//解决中文乱码问题

StringEntity entity = new StringEntity(jsonParam.toString(), "utf-8");

entity.setContentEncoding("UTF-8");

entity.setContentType("application/json");

method.setEntity(entity);

}

HttpResponse result = httpClient.execute(method);

url = URLDecoder.decode(url, "UTF-8");

/\*\*请求发送成功，并得到响应\*\*/

if (result.getStatusLine().getStatusCode() == 200) {

String str = "";

try {

/\*\*读取服务器返回过来的json字符串数据\*\*/

str = EntityUtils.toString(result.getEntity());

if (noNeedResponse) {

return null;

}

/\*\*把json字符串转换成json对象\*\*/

jsonResult = JSONObject.fromObject(str);

} catch (Exception e) {

logger.error("post请求提交失败:" + url, e);

}

}

} catch (IOException e) {

logger.error("post请求提交失败:" + url, e);

}

return jsonResult;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35

**使用HttpClient调用webservice服务：**

import java.nio.charset.Charset;

import org.apache.http.HttpEntity;

import org.apache.http.client.config.RequestConfig;

import org.apache.http.client.methods.CloseableHttpResponse;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClientBuilder;

import org.apache.http.util.EntityUtils;

import org.apache.log4j.Logger;

/\*\*

\* 使用HttpClient调用webservice服务

\* @author lmb

\* @date 2017-4-18

\*/

public class HttpClientCallSoapUtil {

private static final Logger logger = Logger.getLogger(HttpClientCallSoapUtil.class);

static int socketTimeout = 60000;

static int connectTimeout = 60000;

public static void main(String[] args) {

String soapXml = "<?xml version = \"1.0\" ?>"

+ "<soapenv:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:web=\"http://webservices.b.com\">"

+ " <soapenv:Header/>"

+ " <soapenv:Body>"

+ " <web:query soapenv:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">"

+ " <in0 xsi:type=\"web:QueryRequest\">"

+ " <endTime xsi:type=\"xsd:dateTime\">?</endTime>"

+ " <mobile xsi:type=\"soapenc:string\" xmlns:soapenc=\"http://schemas.xmlsoap.org/soap/encoding/\">?</mobile>"

+ " <startTime xsi:type=\"xsd:dateTime\">?</startTime>"

+ " </in0>" + " </web:query>"

+ " </soapenv:Body>" + "</soapenv:Envelope>";

String postUrl = "http://localhost:8381/services/WebService";

doPostSoap(postUrl, soapXml, "");

}

/\*\*

\* HttpClient发送soap请求

\* @param postUrl 请求webservice地址

\* @param soapXml 请求报文

\* @param soapAction

\* @return

\*/

public static String doPostSoap(String postUrl, String soapXml, String soapAction) {

String retStr = "";

// 创建HttpClientBuilder

HttpClientBuilder httpClientBuilder = HttpClientBuilder.create();

// HttpClient

CloseableHttpClient closeableHttpClient = httpClientBuilder.build();

HttpPost httpPost = new HttpPost(postUrl);

// 设置请求和传输超时时间

RequestConfig requestConfig = RequestConfig.custom()

.setSocketTimeout(socketTimeout)

.setConnectTimeout(connectTimeout).build();

httpPost.setConfig(requestConfig);

try {

httpPost.setHeader("Content-Type", "text/xml;charset=UTF-8");

httpPost.setHeader("SOAPAction", soapAction);

StringEntity data = new StringEntity(soapXml,

Charset.forName("UTF-8"));

httpPost.setEntity(data);

CloseableHttpResponse response = closeableHttpClient

.execute(httpPost);

HttpEntity httpEntity = response.getEntity();

if (httpEntity != null) {

// 打印响应内容

retStr = EntityUtils.toString(httpEntity, "UTF-8");

logger.info("response:" + retStr);

}

// 释放资源

closeableHttpClient.close();

} catch (Exception e) {

logger.error("exception in doPostSoap1\_1", e);

}

return retStr;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77

**HttpClient常用方法总结：**

import java.io.File;

import java.io.FileInputStream;

import java.security.KeyStore;

import java.util.ArrayList;

import java.util.List;

import org.apache.http.Header;

import org.apache.http.HttpEntity;

import org.apache.http.HttpResponse;

import org.apache.http.NameValuePair;

import org.apache.http.client.CookieStore;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.conn.scheme.Scheme;

import org.apache.http.conn.ssl.SSLSocketFactory;

import org.apache.http.cookie.Cookie;

import org.apache.http.entity.InputStreamEntity;

import org.apache.http.impl.client.BasicCookieStore;

import org.apache.http.impl.client.DefaultHttpClient;

import org.apache.http.protocol.BasicHttpContext;

import org.apache.http.protocol.HttpContext;

/\*\*

\* HttpClient常用方法总结

\* @param args

\* @author lmb

\* @date 2017-4-18

\*/

public class HttpClientUtil {

//总结一：\*\*\*\*\*\*\*\*\*\*当httpClient的示例不在需要时，可以使用连接管理器关闭\*\*\*\*\*\*\*\*\*\*

httpClient.getConnectionManager().shutdown();

//总结二：\*\*\*\*\*\*\*\*\*\*针对HTTPs的协议的HttpClient请求必须用户和密码 \*\*\*\*\*\*\*\*\*\*

httpclient.getCredentialsProvider()

.setCredentials(new AuthScope("localhost", 443),

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

//总结三：\*\*\*\*\*\*\*\*\*\*如果不想获取HTTPClient返回的信息\*\*\*\*\*\*\*\*\*\*

httpclient.abort();

//总结四：\*\*\*\*\*\*\*\*\*\*httpclient传送文件的方式\*\*\*\*\*\*\*\*\*\*

HttpClient httpclient = new DefaultHttpClient();

HttpPost httppost = new HttpPost("http://www.apache.org");

File file = new File("");

InputStreamEntity reqEntity = new InputStreamEntity(

new FileInputStream(file), -1);

reqEntity.setContentType("binary/octet-stream");

reqEntity.setChunked(true);

// It may be more appropriate to use FileEntity class in this particular

// instance but we are using a more generic InputStreamEntity to demonstrate

// the capability to stream out data from any arbitrary source

//

// FileEntity entity = new FileEntity(file, "binary/octet-stream");

httppost.setEntity(reqEntity);

System.out.println("executing request " + httppost.getRequestLine());

HttpResponse response = httpclient.execute(httppost);

//总结五:\*\*\*\*\*\*\*\*\*\*获取Cookie的信息\*\*\*\*\*\*\*\*\*\*

HttpClient httpclient = new DefaultHttpClient();

// 创建一个本地Cookie存储的实例

CookieStore cookieStore = new BasicCookieStore();

//创建一个本地上下文信息

HttpContext localContext = new BasicHttpContext();

//在本地上下问中绑定一个本地存储

localContext.setAttribute(ClientContext.COOKIE\_STORE, cookieStore);

//设置请求的路径

HttpGet httpget = new HttpGet("http://www.google.com/");

//传递本地的http上下文给服务器

HttpResponse response = httpclient.execute(httpget, localContext);

//获取本地信息

HttpEntity entity = response.getEntity();

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

if (entity != null) {

System.out.println("Response content length: " + entity.getContentLength());

}

//获取cookie中的各种信息

List<Cookie> cookies = cookieStore.getCookies();

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

System.out.println("Local cookie: " + cookies.get(i));

}

//获取消息头的信息

Header[] headers = response.getAllHeaders();

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

System.out.println(headers[i]);

}

//总结六：\*\*\*\*\*\*\*\*\*\*针对典型的SSL请求的处理\*\*\*\*\*\*\*\*\*\*

DefaultHttpClient httpclient = new DefaultHttpClient();

//获取默认的存储密钥类

KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());

//加载本地的密钥信息

FileInputStream instream = new FileInputStream(new File("my.keystore"));

try {

trustStore.load(instream, "nopassword".toCharArray());

} finally {

instream.close();

}

//创建SSLSocketFactory，创建相关的Socket

SSLSocketFactory socketFactory = new SSLSocketFactory(trustStore);

//设置协议的类型和密钥信息，以及断开信息

Scheme sch = new Scheme("https", socketFactory, 443);

//在连接管理器中注册中信息

httpclient.getConnectionManager().getSchemeRegistry().register(sch);

//总结七：\*\*\*\*\*\*\*\*\*\*设置请求的参数的几种方式\*\*\*\*\*\*\*\*\*\*

//A.在请求的路径中以查询字符串格式传递参数

//B.在请求的实体中添加参数

List <NameValuePair> nvps = new ArrayList <NameValuePair>();

nvps.add(new BasicNameValuePair("IDToken1", "username"));

nvps.add(new BasicNameValuePair("IDToken2", "password"));

httpost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF\_8));

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111

以下内容来自百度百科：   
**下面介绍在使用HttpClient过程中常见的一些问题。**

**1、字符编码**   
某目标页的编码可能出现在两个地方，第一个地方是服务器返回的http头中，另外一个地方是得到的html/xml页面中。

在http头的Content-Type字段可能会包含字符编码信息。例如可能返回的头会包含这样子的信息：Content-Type: text/html; charset=UTF-8。这个头信息表明该页的编码是UTF-8，但是服务器返回的头信息未必与内容能匹配上。比如对于一些双字节语言国家，可能服务器返回的编码类型是UTF-8，但真正的内容却不是UTF-8编码的，因此需要在另外的地方去得到页面的编码信息；但是如果服务器返回的编码不是UTF-8，而是具体的一些编码，比如gb2312等，那服务器返回的可能是正确的编码信息。通过method对象的getResponseCharSet()方法就可以得到http头中的编码信息。

对于象xml或者html这样的文件，允许作者在页面中直接指定编码类型。比如在html中会有<meta http-equiv="Content-Type" content="text/html; charset=gb2312"/>这样的标签；或者在xml中会有<?xml version="1.0" encoding="gb2312"?>这样的标签，在这些情况下，可能与http头中返回的编码信息冲突，需要用户自己判断到底那种编码类型应该是真正的编码。

**2、自动转向**   
根据RFC2616中对自动转向的定义，主要有两种：301和302。301表示永久的移走（Moved Permanently），当返回的是301，则表示请求的资源已经被移到一个固定的新地方，任何向该地址发起请求都会被转到新的地址上。302表示暂时的转向，比如在服务器端的servlet程序调用了sendRedirect方法，则在客户端就会得到一个302的代码，这时服务器返回的头信息中location的值就是sendRedirect转向的目标地址。

HttpClient支持自动转向处理，但是象POST和PUT方式这种要求接受后继服务的请求方式，暂时不支持自动转向，因此如果碰到POST方式提交后返回的是301或者302的话需要自己处理。就像刚才在POSTMethod中举的例子：如果想进入登录BBS后的页面，必须重新发起登录的请求，请求的地址可以在头字段location中得到。不过需要注意的是，有时候location返回的可能是相对路径，因此需要对location返回的值做一些处理才可以发起向新地址的请求。

另外除了在头中包含的信息可能使页面发生重定向外，在页面中也有可能会发生页面的重定向。引起页面自动转发的标签是：<meta http-equiv="refresh" content="5; url=....">。如果你想在程序中也处理这种情况的话得自己分析页面来实现转向。需要注意的是，在上面那个标签中url的值也可以是一个相对地址，如果是这样的话，需要对它做一些处理后才可以转发。

HTTPClient详细教程参看：[HTTPClient教程](http://www.yeetrack.com/?p=779)

版权声明：本文为博主原创文章，无需授权即可转载，甚至无需保留以上版权声明，转载时请务必注明作者