轻松把玩HttpClient

崔成龙

目　录

前言

> 原文出处：轻松把玩HttpClient
> 作者：崔成龙

本系列文章经作者授权在看云整理发布，未经作者允许，请勿转载！

轻松把玩HttpClient

> 介绍如何使用HttpClient，通过一些简单示例，来帮助初学者快速入手。最后提供了一个非常强大的工具类，比现在网络上分享的都强大，支持插件式设置header、代理、ssl等配置信息。

HttpClient3.x之Get请求和Post请求示例

HttpClient的支持在HTTP/1.1规范中定义的所有的HTTP方法：GET, HEAD, POST, PUT, DELETE, TRACE 和 OPTIONS。每有一个方法都有一个对应的类：HttpGet，HttpHead，HttpPost，HttpPut，HttpDelete，HttpTrace和HttpOptions。所有的这些类均实现了HttpUriRequest接口，故可以作为execute的执行参数使用。请求URI是能够应用请求的统一资源标识符。 HTTP请求的URI包含一个协议计划protocol scheme，主机名host name,，可选的端口optional port，资源的路径resource path，可选的查询optional query和可选的片段optional fragment。

head，put，delete，trace HttpClient支持这些方法，
大多数浏览器不支持这些方法，原因是Html 4中对 FORM 的method方法只支持两个get和post，很多浏览器还都依然是基于html4的。

通常会在JAVA中通过代码调用URL进行远端方法调用，这些方法有的是Get请求方式的，有的是POST请求方式的，为此，总结一例，贴出以便查阅。

依赖JAR包有：commons-codec.jar,commons-httpclient.jar,commons-logging.jar。

```
package com.wujintao.httpclient;

import java.io.IOException;
import java.io.InputStream;

import org.apache.commons.httpclient.DefaultHttpMethodRetryHandler;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.NameValuePair;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.params.HttpMethodParams;
import org.junit.Test;

public class TestCase {

        @Test
        public void testGetRequest() throws IllegalStateException, IOExce
ption {
                HttpClient client = new HttpClient();
                StringBuilder sb = new StringBuilder();
                InputStream ins = null;
                // Create a method instance.
                GetMethod method = new GetMethod("http://www.baidu.com");
                // Provide custom retry handler is necessary
                method.getParams().setParameter(HttpMethodParams.RETRY_HA
NDLER,
                                new DefaultHttpMethodRetryHandler(3, fals
e));
                try {
                        // Execute the method.
                        int statusCode = client.executeMethod(method);
                        System.out.println(statusCode);
                        if (statusCode == HttpStatus.SC_OK) {
                                ins = method.getResponseBodyAsStream();
                                byte[] b = new byte[1024];
                                int r_len = 0;
                                while ((r_len = ins.read(b)) > 0) {
                                        sb.append(new String(b, 0, r_len,
 method
                                                        .getResponseCharS
et()));
                                }
                        } else {
                                System.err.println("Response Code: " + st
atusCode);
                        }
                } catch (HttpException e) {
                        System.err.println("Fatal protocol violation: " +
 e.getMessage());
                } catch (IOException e) {
                        System.err.println("Fatal transport error: " + e.
getMessage());
                } finally {
                        method.releaseConnection();
                        if (ins != null) {
                                ins.close();
                        }
                }
                System.out.println(sb.toString());
        }

        @Test
        public void testPostRequest() throws HttpException, IOException {
                HttpClient client = new HttpClient();
                PostMethod method = new PostMethod("http://www.baidu.com/
getValue");
                method.setRequestHeader("Content-Type",
                                "application/x-www-form-urlencoded;charse
t=gb2312");
                NameValuePair[] param = { new NameValuePair("age", "11"),
                                new NameValuePair("name", "jay"), };
                method.setRequestBody(param);
                int statusCode = client.executeMethod(method);
                System.out.println(statusCode);
                method.releaseConnection();
```

```
          }
  }
```

httpclient3.x中使用HTTPS的方法

HttpClient请求https的实例：

```
import javax.net.ssl.SSLContext;

import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.ResponseHandler;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.ClientConnectionManager;

import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.scheme.SchemeSocketFactory;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.ClientParamsStack;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.params.DefaultedHttpParams;
import org.apache.http.params.HttpParams;

public class HttpClientTest {

        public static void main(String args[]) {

                try {

                        HttpClient httpclient = new DefaultHttpClient();
                        //Secure Protocol implementation.
                        SSLContext ctx = SSLContext.getInstance("SSL");
                        //Implementation of a trust manager for X509 cert
ificates
                        X509TrustManager tm = new X509TrustManager() {

                                public void checkClientTrusted(X509Certif
icate[] xcs,
                                                String string) throws Cer
tificateException {

                                }

                                public void checkServerTrusted(X509Certif
icate[] xcs,
                                                String string) throws Cer
tificateException {
                                }

                                public X509Certificate[] getAcceptedIssue
rs() {
                                        return null;
                                }
                        };
                        ctx.init(null, new TrustManager[] { tm }, null);
                        SSLSocketFactory ssf = new SSLSocketFactory(ctx);

                        ClientConnectionManager ccm = httpclient.getConne
ctionManager();
                        //register https protocol in httpclient's scheme
registry
                        SchemeRegistry sr = ccm.getSchemeRegistry();
                        sr.register(new Scheme("https", 443, ssf));

                        HttpGet httpget = new HttpGet("");
                        HttpParams params = httpclient.getParams();

                        params.setParameter("param1", "paramValue1");

                        httpget.setParams(params);
                        System.out.println("REQUEST:" + httpget.getURI())
;
                        ResponseHandler responseHandler = new BasicRespon
seHandler();
                        String responseBody;

                        responseBody = httpclient.execute(httpget, respon
seHandler);


                        System.out.println(responseBody);

                        // Create a response handler

                } catch (NoSuchAlgorithmException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (ClientProtocolException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (Exception ex) {
                        ex.printStackTrace();

                }
        }
}
```

简单的利用UrlConnection，后台模拟http请求

版权声明：本文为博主原创文章，未经博主允许不得转载。如需转载请声明：【转自 http://blog.csdn.net/xiaoxian8023 】

这两天在整理看httpclient，然后想自己用UrlConnection后台模拟实现Http请求，于是一个简单的小例子就新鲜出炉了（支持代理哦）：

```
public class SimpleHttpTest {

        public static String send(String urlStr, Map<String,String> map,S
tring encoding){
                String body="";
                StringBuffer sbuf = new StringBuffer();
                if(map!=null){
                        for (Entry<String,String> entry : map.entrySet())
 {
                                sbuf.append(entry.getKey()).append("=").a
ppend(entry.getValue()).append("&");
                        }
                        if(sbuf.length()>0){
                                sbuf.deleteCharAt(sbuf.length()-1);
                        }
                }
                 // 1、重新对请求报文进行 GBK 编码
        byte[] postData = null;
        try {
            postData = sbuf.toString().getBytes(encoding);
        } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
        }

        // 2、发送 HTTP(S) 请求
        OutputStream reqStream = null;
        InputStream resStream = null;
        URLConnection request = null;
        try {
            System.out.println("交易请求地址：" + urlStr);
            System.out.println("参数：" + sbuf.toString());

            //A、与服务器建立 HTTP(S) 连接
                URL url = null;
            try {
                Proxy proxy = new Proxy(java.net.Proxy.Type.HTTP,new Inet
SocketAddress("127.0.0.1", 8087));
                url = new URL(urlStr);
                request = url.openConnection(proxy);
                request.setDoInput(true);
                request.setDoOutput(true);
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }

            //B、指定报文头【Content-type】、【Content-length】 与 【Keep-ali
ve】
            request.setRequestProperty("Content-type", "application/x-www
-form-urlencoded");
            request.setRequestProperty("Content-length", String.valueOf(p
ostData.length));
            request.setRequestProperty("Keep-alive", "false");
            request.setRequestProperty("User-Agent", "Mozilla/4.0 (compat
ible; MSIE 5.0; Windows NT; DigExt)");

            //C、发送报文至服务器
            reqStream = request.getOutputStream();
            reqStream.write(postData);
            reqStream.close();

            //D、接收服务器返回结果
            ByteArrayOutputStream ms = null;
            resStream = request.getInputStream();
            ms = new ByteArrayOutputStream();
            byte[] buf = new byte[4096];
            int count;
            while ((count = resStream.read(buf, 0, buf.length)) > 0) {
                ms.write(buf, 0, count);
            }
            resStream.close();
            body = new String(ms.toByteArray(), encoding);
        } catch (UnknownHostException e) {
            System.err.println( "服务器不可达【" + e.getMessage() + "】");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (reqStream != null)
                        reqStream.close();
                if (resStream != null)
                        resStream.close();
            } catch (Exception ex) {
            }
        }

        System.out.println("交易响应结果：");
        System.out.println(body);
        return body;
        }

        public static void main(String[] args) {
                String url="http://php.weather.sina.com.cn/iframe/index/w
_cl.php";
                Map<String, String> map = new HashMap<String, String>();
                map.put("code", "js");
                map.put("day", "0");
                map.put("city", "上海");
                map.put("dfc", "1");
                map.put("charset", "utf-8");
                send(url, map,"utf-8");
        }
}
```

结果如下：

```
交易请求地址：http://php.weather.sina.com.cn/iframe/index/w_cl.php
参数：dfc=1&charset=utf-8&day=0&code=js&city=上海
交易响应结果：
(function(){var w=[];w['上海']=[{s1:'阴',s2:'阴',f1:'yin',f2:'yin',t1:'17'
,t2:'14',p1:'≤3',p2:'≤3',d1:'东北风',d2:'东北风'}];var add={now:'2015-11-11
 19:04:33',time:'1447239873',update:'北京时间11月11日17:10更新',error:'0',to
tal:'1'};window.SWther={w:w,add:add};})();//0
```

代码中的步骤写的很明白了，如果你有心，还可以对该方法进行各种封装，方便使用。下篇我会分享一下httpclient是如何模拟后台来发送http请求的，还有配置ssl、代理、自定义header等等，敬请期待吧。

轻松把玩HttpClient之模拟post请求示例

版权声明：本文为博主原创文章，未经博主允许不得转载。如需转载请声明：【转自 http://blog.csdn.net/xiaoxian8023 】

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。当前官网最新版介绍页是：http://hc.apache.org/httpcomponents-client-4.5.x/index.html

许多需要后台模拟请求的系统或者框架都用的是httpclient。所以作为一个java开发人员，有必要学一学。本文提供了一个简单的demo，供初学者参考。

使用HttpClient发送请求、接收响应很简单，一般需要如下几步即可：

1. 创建CloseableHttpClient对象。
2. 创建请求方法的实例，并指定请求URL。如果需要发送GET请求，创建HttpGet对象；如果需要发送POST请求，创建HttpPost对象。
3. 如果需要发送请求参数，可可调用setEntity(HttpEntity entity)方法来设置请求参数。setParams方法已过时（4.4.1版本）。
4. 调用HttpGet、HttpPost对象的setHeader(String name, String value)方法设置header信息，或者调用setHeaders(Header[] headers)设置一组header信息。
5. 调用CloseableHttpClient对象的execute(HttpUriRequest request)发送请求，该方法返回一个CloseableHttpResponse。
6. 调用HttpResponse的getEntity()方法可获取HttpEntity对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容；调用CloseableHttpResponse的getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头。
7. 释放连接。无论执行方法是否成功，都必须释放连接

具体代码如下(HttpClient-4.4.1)：

```
/**
 * 简单httpclient实例
 *
 * @author arron
 * @date 2015年11月11日 下午6:36:49
 * @version 1.0
 */
```

```
public class SimpleHttpClientDemo {

        /**
         * 模拟请求
         *
         * @param url           资源地址
         * @param map   参数列表
         * @param encoding      编码
         * @return
         * @throws ParseException
         * @throws IOException
         */
        public static String send(String url, Map<String,String> map,Stri
ng encoding) throws ParseException, IOException{
                String body = "";

                //创建httpclient对象
                CloseableHttpClient client = HttpClients.createDefault();
                //创建post方式请求对象
                HttpPost httpPost = new HttpPost(url);

                //装填参数
                List<NameValuePair> nvps = new ArrayList<NameValuePair>()
;
                if(map!=null){
                        for (Entry<String, String> entry : map.entrySet()
) {
                                nvps.add(new BasicNameValuePair(entry.get
Key(), entry.getValue()));
                        }
                }
                //设置参数到请求对象中
                httpPost.setEntity(new UrlEncodedFormEntity(nvps, encodin
g));

                System.out.println("请求地址："+url);
                System.out.println("请求参数："+nvps.toString());

                //设置header信息
                //指定报文头【Content-type】、【User-Agent】
                httpPost.setHeader("Content-type", "application/x-www-for
m-urlencoded");
                httpPost.setHeader("User-Agent", "Mozilla/4.0 (compatible
; MSIE 5.0; Windows NT; DigExt)");

                //执行请求操作，并拿到结果（同步阻塞）
                CloseableHttpResponse response = client.execute(httpPost)
;
                //获取结果实体
                HttpEntity entity = response.getEntity();
                if (entity != null) {
                        //按指定编码转换结果实体为String类型
                        body = EntityUtils.toString(entity, encoding);
                }
                EntityUtils.consume(entity);
                //释放链接
                response.close();
        return body;
        }

        public static void main(String[] args) throws ParseException, IOE
xception {
                String url="http://php.weather.sina.com.cn/iframe/index/w
_cl.php";
                Map<String, String> map = new HashMap<String, String>();
                map.put("code", "js");
                map.put("day", "0");
                map.put("city", "上海");
                map.put("dfc", "1");
                map.put("charset", "utf-8");
                String body = send(url, map,"utf-8");
      System.out.println("交易响应结果：");
      System.out.println(body);
        }
}
```

在main方法中测试一下：

```
        public static void main(String[] args) throws ParseException, IOE
xception {
                String url="http://php.weather.sina.com.cn/iframe/index/w
_cl.php";
                Map<String, String> map = new HashMap<String, String>();
                map.put("code", "js");
                map.put("day", "0");
                map.put("city", "上海");
                map.put("dfc", "1");
                map.put("charset", "utf-8");
                String body = send(url, map,"utf-8");
                System.out.println("交易响应结果：");
                System.out.println(body);

                System.out.println("-----------------------------------")
;

                map.put("city", "北京");
                body = send(url, map, "utf-8");
                System.out.println("交易响应结果：");
                System.out.println(body);
        }
```

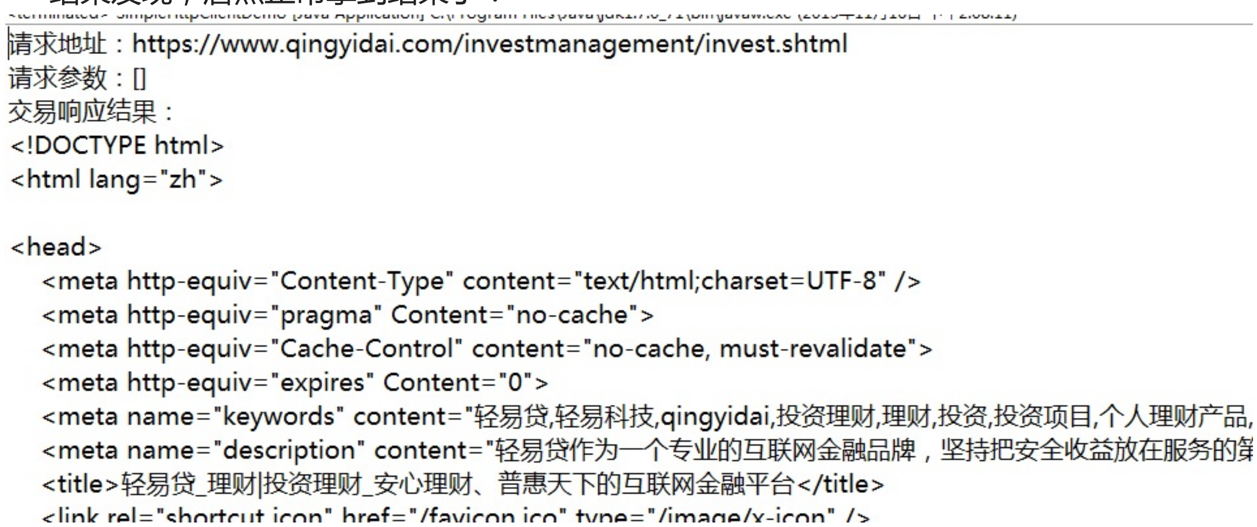
结果如下：

```
请求地址：http://php.weather.sina.com.cn/iframe/index/w_cl.php
请求参数：[dfc=1, charset=utf-8, day=0, code=js, city=上海]
交易响应结果：
(function(){var w=[];w['上海']=[{s1:'小雨',s2:'小雨',f1:'xiaoyu',f2:'xiaoyu
',t1:'21',t2:'16',p1:'≤3',p2:'≤3',d1:'南风',d2:'北风'}];var add={now:'2015
-11-16 13:16:23',time:'1447650983',update:'北京时间11月16日08:10更新',error:
'0',total:'1'};window.SWther={w:w,add:add};})();//0

-----------------------------------
请求地址：http://php.weather.sina.com.cn/iframe/index/w_cl.php
请求参数：[dfc=1, charset=utf-8, day=0, code=js, city=北京]
交易响应结果：
(function(){var w=[];w['北京']=[{s1:'多云',s2:'多云',f1:'duoyun',f2:'duoyun
',t1:'9',t2:'1',p1:'≤3',p2:'≤3',d1:'无持续风向',d2:'无持续风向'}];var add={n
ow:'2015-11-16 13:18:35',time:'1447651115',update:'北京时间11月16日08:10更新
',error:'0',total:'1'};window.SWther={w:w,add:add};})();//0
```

现在我们测试一下https链接：https://www.qingyidai.com/investmanagement/invest.shtml

```
        public static void main(String[] args) throws ParseException, IOE
xception {
                String url = "https://www.qingyidai.com/investmanagement/
invest.shtml";
                String body = send(url, null, "utf-8");
                System.out.println("交易响应结果：");
                System.out.println(body);
        }
```

结果发现，居然正常拿到结果了：



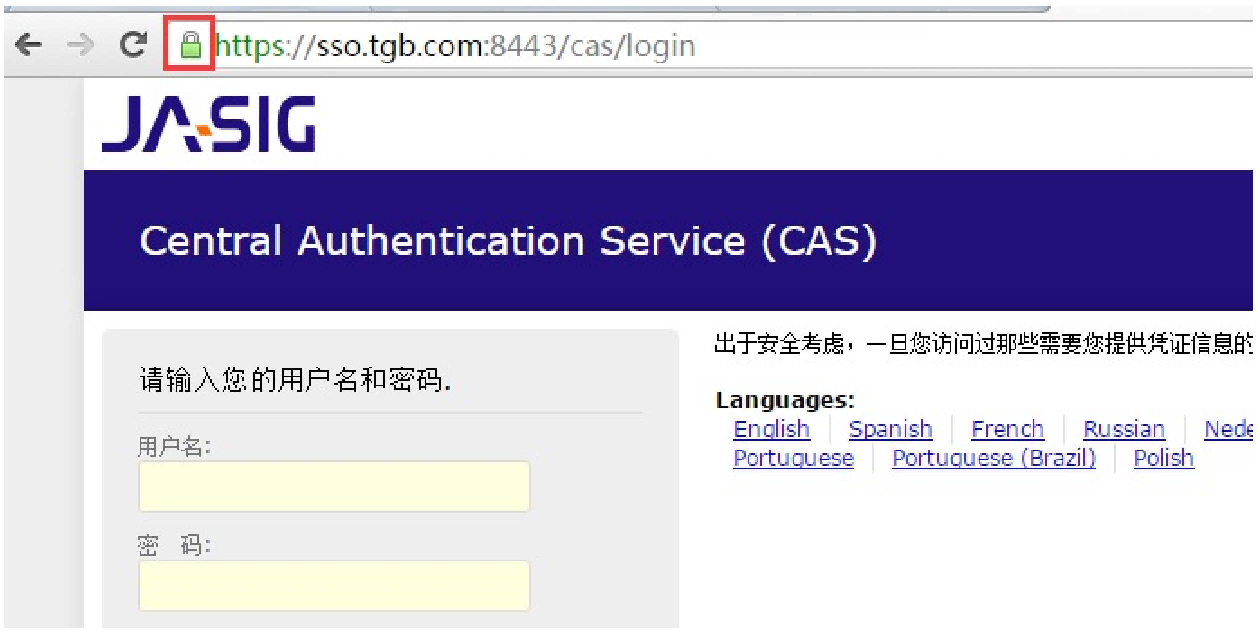
```
请求地址：https://www.qingyidai.com/investmanagement/invest.shtml
请求参数：[]
交易响应结果：
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
  <meta http-equiv="pragma" Content="no-cache">
  <meta http-equiv="Cache-Control" content="no-cache, must-revalidate">
  <meta http-equiv="expires" Content="0">
  <meta name="keywords" content="轻易贷,轻易科技,qingyidai,投资理财,理财,投资,投资项目,个人理财产品,
  <meta name="description" content="轻易贷作为一个专业的互联网金融品牌，坚持把安全收益放在服务的第
  <title>轻易贷_理财|投资理财_安心理财、普惠天下的互联网金融平台</title>
  <link rel="shortcut icon" href="/favicon.ico" type="/image/x-icon" />
```

原来如果网站的证书已经被ca机构认证通过了，那么用HttpClient来调用的话，会直接成功的。不用再单独配置htts链接了。不过如果是自生成的证书，还是需要配置https的，下篇就来配置一下吧，敬请期待。

轻松把玩HttpClient之配置ssl，采用绕过证书验证实现https

版权声明：本文为博主原创文章，未经博主允许不得转载。如需转载请声明：【转自 http://blog.csdn.net/xiaoxian8023 】

上篇文章说道httpclient不能直接访问https的资源，这次就来模拟一下环境，然后配置https测试一下。在前面的文章中，分享了一篇自己生成并在tomcat中配置ssl的文章《Tomcat配置SSL》，大家可以据此来在本地配置https。我已经配置好了，效果是这样滴：


https://sso.tgb.com:8443/cas/login

JASIG

Central Authentication Service (CAS)

请输入您的用户名和密码.

用户名:

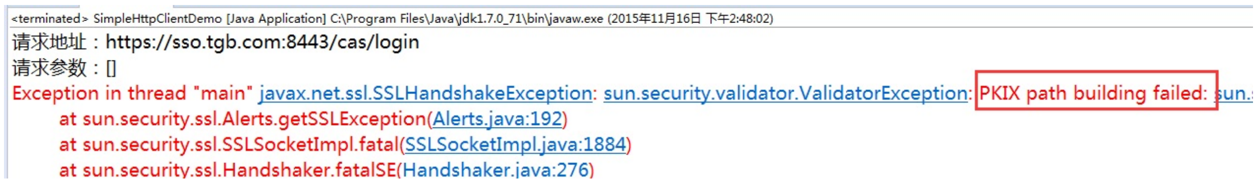
密　码:

出于安全考虑，一旦您访问过那些需要您提供凭证信息的

Languages:
English | Spanish | French | Russian | Nede
Portuguese | Portuguese (Brazil) | Polish

可以看到已经信任该证书（显示浅绿色小锁），浏览器可以正常访问。现在我们用代码测试一下：

```
	public static void main(String[] args) throws ParseException, IOE
xception, KeyManagementException, NoSuchAlgorithmException, HttpProcessEx
ception {
		String url = "https://sso.tgb.com:8443/cas/login";
		String body = send(url, null, "utf-8");
		System.out.println("交易响应结果：");
		System.out.println(body);
		System.out.println("-----------------------------------")
;
	}
```



```
<terminated> SimpleHttpClientDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2015年11月16日 下午2:48:02)
请求地址：https://sso.tgb.com:8443/cas/login
请求参数：[]
Exception in thread "main" javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.
	at sun.security.ssl.Alerts.getSSLException(Alerts.java:192)
	at sun.security.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1884)
	at sun.security.ssl.Handshaker.fatalSE(Handshaker.java:276)
```

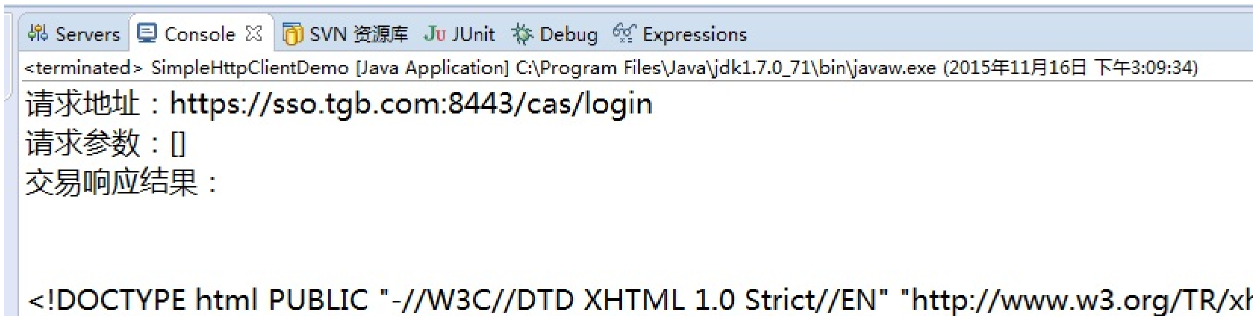
发现抛出了异常，我知道的有两种方案（也许还有我不知道的方案），这里介绍第一种方案，也是用的比较多的方案——绕过证书验证。直接看代码吧：

```
        /**
         * 绕过验证
         *
         * @return
         * @throws NoSuchAlgorithmException
         * @throws KeyManagementException
         */
        public static SSLContext createIgnoreVerifySSL() throws NoSuchAlg
orithmException, KeyManagementException {
                SSLContext sc = SSLContext.getInstance("SSLv3");

                // 实现一个X509TrustManager接口，用于绕过验证，不用修改里面的方法
                X509TrustManager trustManager = new X509TrustManager() {
                        @Override
                        public void checkClientTrusted(
                                        java.security.cert.X509Certificat
e[] paramArrayOfX509Certificate,
                                        String paramString) throws Certif
icateException {
                        }

                        @Override
                        public void checkServerTrusted(
                                        java.security.cert.X509Certificat
e[] paramArrayOfX509Certificate,
                                        String paramString) throws Certif
icateException {
                        }

                        @Override
                        public java.security.cert.X509Certificate[] getAc
ceptedIssuers() {
                                return null;
                        }
                };

                sc.init(null, new TrustManager[] { trustManager }, null);
                return sc;
        }
```

然后修改原来的send方法：

```
        /**
         * 模拟请求
         *
         * @param url           资源地址
         * @param map   参数列表
         * @param encoding      编码
         * @return
         * @throws NoSuchAlgorithmException
         * @throws KeyManagementException
         * @throws IOException
         * @throws ClientProtocolException
         */
        public static String send(String url, Map<String,String> map,Stri
ng encoding) throws KeyManagementException, NoSuchAlgorithmException, Cli
entProtocolException, IOException {
                String body = "";
                //采用绕过验证的方式处理https请求
                SSLContext sslcontext = createIgnoreVerifySSL();

        // 设置协议http和https对应的处理socket链接工厂的对象
        Registry<ConnectionSocketFactory> socketFactoryRegistry = Registr
yBuilder.<ConnectionSocketFactory>create()
            .register("http", PlainConnectionSocketFactory.INSTANCE)
            .register("https", new SSLConnectionSocketFactory(sslcontext)
)
            .build();
        PoolingHttpClientConnectionManager connManager = new PoolingHttpC
lientConnectionManager(socketFactoryRegistry);
        HttpClients.custom().setConnectionManager(connManager);

        //创建自定义的httpclient对象
                CloseableHttpClient client = HttpClients.custom().setConn
ectionManager(connManager).build();
//              CloseableHttpClient client = HttpClients.createDefault();

                //创建post方式请求对象
                HttpPost httpPost = new HttpPost(url);

                //装填参数
                List<NameValuePair> nvps = new ArrayList<NameValuePair>()
;
                if(map!=null){
                        for (Entry<String, String> entry : map.entrySet()
) {
                                nvps.add(new BasicNameValuePair(entry.get
Key(), entry.getValue()));
                        }
                }
                //设置参数到请求对象中
                httpPost.setEntity(new UrlEncodedFormEntity(nvps, encodin
g));

                System.out.println("请求地址："+url);
                System.out.println("请求参数："+nvps.toString());
                //设置header信息
                //指定报文头【Content-type】、【User-Agent】
                httpPost.setHeader("Content-type", "application/x-www-for
m-urlencoded");
                httpPost.setHeader("User-Agent", "Mozilla/4.0 (compatible
; MSIE 5.0; Windows NT; DigExt)");

                //执行请求操作，并拿到结果（同步阻塞）
                CloseableHttpResponse response = client.execute(httpPost)
;
                //获取结果实体
                HttpEntity entity = response.getEntity();
                if (entity != null) {
                        //按指定编码转换结果实体为String类型
                        body = EntityUtils.toString(entity, encoding);
                }
                EntityUtils.consume(entity);
                //释放链接
                response.close();
        return body;
        }
```

现在再进行测试，发现果然通了。



```
<terminated> SimpleHttpClientDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2015年11月16日 下午3:09:34)
请求地址：https://sso.tgb.com:8443/cas/login
请求参数：[]
交易响应结果：


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xh
```

下篇介绍另一种方案，应对自己生成的证书，敬请期待。

轻松把玩HttpClient之配置ssl，采用设置信任自签名证书实现https

在上篇文章《HttpClient配置ssl实现https简单示例——绕过证书验证》中简单分享了一下如何绕过证书验证。如果你想用httpclient访问一个网站，但是对方的证书没有通过ca认证或者其他问题导致证书不被信任，比如12306的证书就是这样的。所以对于这样的情况，你只能是选择绕过证书验证的方案了。

但是，如果是自己用jdk或者其他工具生成的证书，还是希望用其他方式认证自签名的证书，这篇文章就来分享一下如何设置信任自签名的证书。当然你也可以参考官网示例中。

要想信任自签名的证书，必须得知道密钥库的路径及密钥库的密码。然后加载到程序来才可以。具体代码如下：

```
          /**
           * 设置信任自签名证书
           *
           * @param keyStorePath          密钥库路径
           * @param keyStorepass          密钥库密码
           * @return
           */
          public static SSLContext custom(String keyStorePath, String keySt
orepass){
                  SSLContext sc = null;
                  FileInputStream instream = null;
                  KeyStore trustStore = null;
                  try {
                          trustStore = KeyStore.getInstance(KeyStore.getDef
aultType());
                          instream = new FileInputStream(new File(keyStoreP
ath));
                          trustStore.load(instream, keyStorepass.toCharArra
y());
                          // 相信自己的CA和所有自签名的证书
                          sc = SSLContexts.custom().loadTrustMaterial(trust
Store, new TrustSelfSignedStrategy()).build();
                  } catch (KeyStoreException | NoSuchAlgorithmException| Ce
rtificateException | IOException | KeyManagementException e) {
                          e.printStackTrace();
                  } finally {
                          try {
                                  instream.close();
```

```
                        } catch (IOException e) {
                        }
                }
                return sc;
        }
```

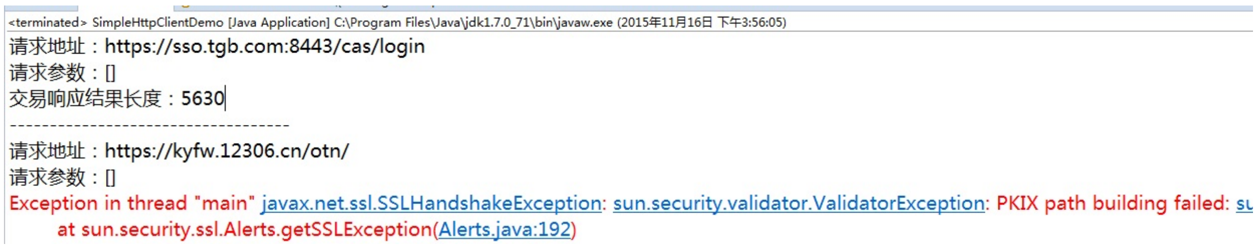
然后修改原来的send方法：

```
        /**
         * 模拟请求
         *
         * @param url           资源地址
         * @param map   参数列表
         * @param encoding      编码
         * @return
         * @throws ParseException
         * @throws IOException
         * @throws KeyManagementException
         * @throws NoSuchAlgorithmException
         * @throws ClientProtocolException
         */
        public static String send(String url, Map<String,String> map,Stri
ng encoding) throws ClientProtocolException, IOException {
                String body = "";

                //tomcat是我自己的密钥库的密码，你可以替换成自己的
                //如果密码为空，则用"nopassword"代替
                SSLContext sslcontext = custom("D:\\keys\\wsriakey", "tom
cat");

        // 设置协议http和https对应的处理socket链接工厂的对象
        Registry<ConnectionSocketFactory> socketFactoryRegistry = Registr
yBuilder.<ConnectionSocketFactory>create()
            .register("http", PlainConnectionSocketFactory.INSTANCE)
            .register("https", new SSLConnectionSocketFactory(sslcontext)
)
            .build();
        PoolingHttpClientConnectionManager connManager = new PoolingHttpC
lientConnectionManager(socketFactoryRegistry);
        HttpClients.custom().setConnectionManager(connManager);

        //创建自定义的httpclient对象
                CloseableHttpClient client = HttpClients.custom().setConn
ectionManager(connManager).build();
//              CloseableHttpClient client = HttpClients.createDefault();

                //创建post方式请求对象
                HttpPost httpPost = new HttpPost(url);

                //装填参数
                List<NameValuePair> nvps = new ArrayList<NameValuePair>()
;
                if(map!=null){
                        for (Entry<String, String> entry : map.entrySet()
) {
                                nvps.add(new BasicNameValuePair(entry.get
Key(), entry.getValue()));
                        }
                }
                //设置参数到请求对象中
                httpPost.setEntity(new UrlEncodedFormEntity(nvps, encodin
g));

                System.out.println("请求地址："+url);
                System.out.println("请求参数："+nvps.toString());

                //设置header信息
                //指定报文头【Content-type】、【User-Agent】
                httpPost.setHeader("Content-type", "application/x-www-for
m-urlencoded");
                httpPost.setHeader("User-Agent", "Mozilla/4.0 (compatible
; MSIE 5.0; Windows NT; DigExt)");

                //执行请求操作，并拿到结果（同步阻塞）
                CloseableHttpResponse response = client.execute(httpPost)
;
                //获取结果实体
                HttpEntity entity = response.getEntity();
                if (entity != null) {
                        //按指定编码转换结果实体为String类型
                        body = EntityUtils.toString(entity, encoding);
                }
                EntityUtils.consume(entity);
                //释放链接
                response.close();
        return body;
        }
```

测试一下吧：

```
        public static void main(String[] args) throws ParseException, IOE
xception, KeyManagementException, NoSuchAlgorithmException{
                String url = "https://sso.tgb.com:8443/cas/login";
                String body = send(url, null, "utf-8");
                System.out.println("交易响应结果长度："+body.length());

                System.out.println("-----------------------------------")
;

                url = "https://kyfw.12306.cn/otn/";
                body = send(url, null, "utf-8");
                System.out.println("交易响应结果长度："+body.length());
        }
```

测试结果：



```
<terminated> SimpleHttpClientDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2015年11月16日 下午3:56:05)
请求地址：https://sso.tgb.com:8443/cas/login
请求参数：[]
交易响应结果长度：5630
-----------------------------------
请求地址：https://kyfw.12306.cn/otn/
请求参数：[]
Exception in thread "main" javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: su
	at sun.security.ssl.Alerts.getSSLException(Alerts.java:192)
```

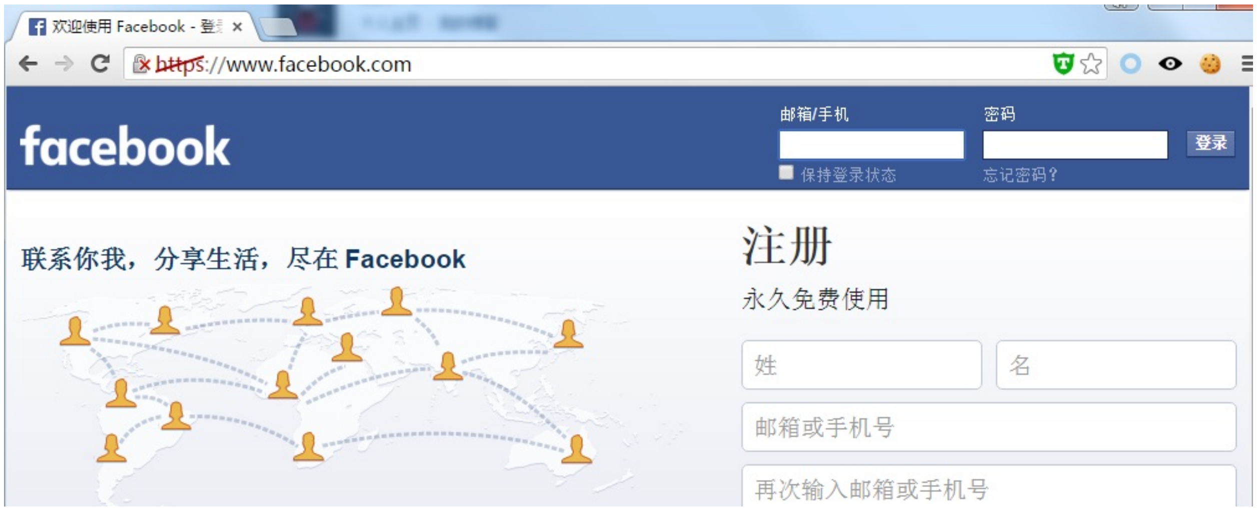
从结果中，我们很清楚的看到，使用自签名的证书，访问自签名的网站可以正常访问，访问12306则会失败。所以自签名的也只能用于自定义密钥和证书的情况下使用。而12306这种情况还是要用上一篇提到的“绕过证书验证”方案。

轻松把玩HttpClient之设置代理，可以访问FaceBook

前面的文章介绍了一些HttpClient的简单示例，本文继续采用示例的方式来演示HttpClient的功能。

在项目中我们可能会遇到这样的情况：为了保证系统的安全性，只允许使用特定的代理才可以访问，而与这些系统使用HttpClient进行交互时，只能为其配置代理。

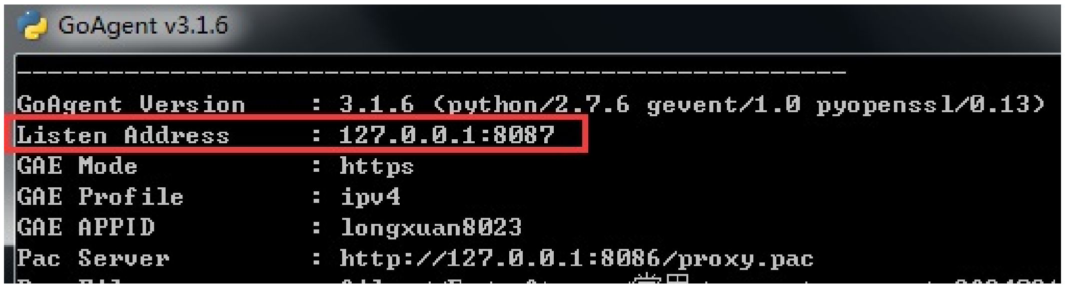
这里我们使用goagent代理访问脸书来模拟这种情况。facebook由于某些原因被封，只能通过代理来访问，所以正好也符合我们现在的演示需求。现在在浏览器上访问是可以访问的：



可以看到facebook采用的也是https的方式，而且该网站的证书不客户端被信任。所以我们要采用“绕过证书验证”的方式使用Https。那如何设置代理呢，官网有相关的示例。我采用的跟官网提供的稍微不一样，具体代码如下：

```
        /**
         * 设置代理
         * @param builder
         * @param hostOrIP
         * @param port
         */
        public static HttpClientBuilder proxy(String hostOrIP, int port){
                // 依次是代理地址，代理端口号，协议类型
                HttpHost proxy = new HttpHost(hostOrIP, port, "http");
                DefaultProxyRoutePlanner routePlanner = new DefaultProxyR
outePlanner(proxy);
                return HttpClients.custom().setRoutePlanner(routePlanner)
;
        }
```

返回值是HttpClientBuilder，这个类是用来生成HttpClient对象的，同时可以设置各种参数，这里提供返回值是为了配置代理后，继续配置ssl。打开goagent，看看代理ip的设定如图：



```
GoAgent v3.1.6
------------------------------------------------------------
GoAgent Version    : 3.1.6 (python/2.7.6 gevent/1.0 pyopenssl/0.13)
Listen Address     : 127.0.0.1:8087
GAE Mode           : https
GAE Profile        : ipv4
GAE APPID          : longxuan8023
Pac Server         : http://127.0.0.1:8086/proxy.pac
```

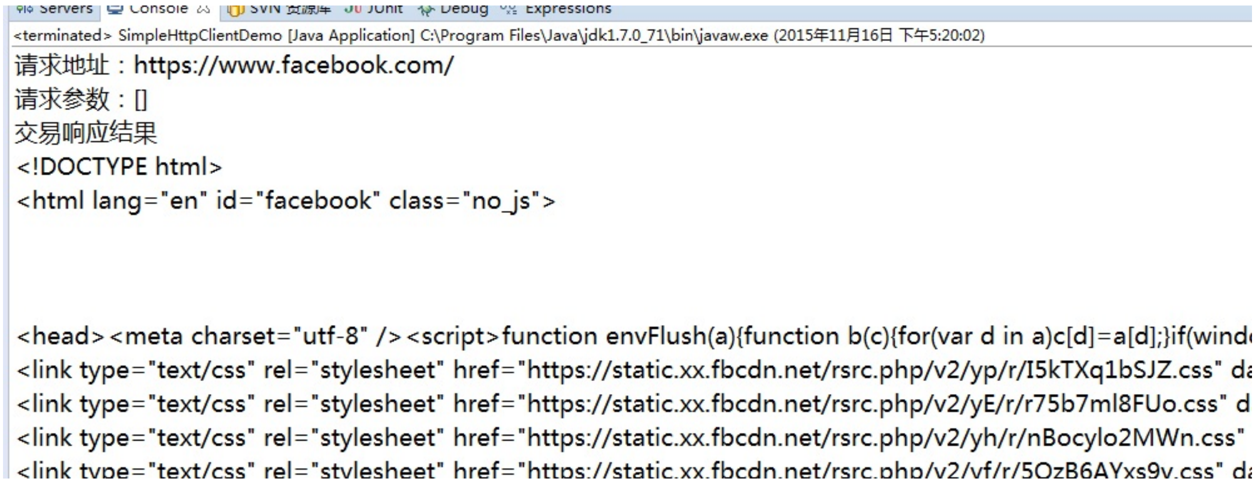
现在修改send方法：

```
        /**
         * 模拟请求
         *
         * @param url           资源地址
         * @param map    参数列表
         * @param encoding      编码
         * @return
         * @throws NoSuchAlgorithmException
         * @throws KeyManagementException
         * @throws IOException
         * @throws ClientProtocolException
         */
        public static String send(String url, Map<String,String> map,Stri
ng encoding) throws KeyManagementException, NoSuchAlgorithmException, Cli
entProtocolException, IOException {
                String body = "";

                //绕过证书验证，处理https请求
                SSLContext sslcontext = createIgnoreVerifySSL();

        // 设置协议http和https对应的处理socket链接工厂的对象
        Registry<ConnectionSocketFactory> socketFactoryRegistry = Registr
yBuilder.<ConnectionSocketFactory>create()
            .register("http", PlainConnectionSocketFactory.INSTANCE)
            .register("https", new SSLConnectionSocketFactory(sslcontext)
)
            .build();
        PoolingHttpClientConnectionManager connManager = new PoolingHttpC
lientConnectionManager(socketFactoryRegistry);
        HttpClients.custom().setConnectionManager(connManager);

        //创建自定义的httpclient对象
                CloseableHttpClient client = proxy("127.0.0.1", 8087).set
ConnectionManager(connManager).build();
//              CloseableHttpClient client = HttpClients.createDefault();

                //创建post方式请求对象
                HttpPost httpPost = new HttpPost(url);

                //装填参数
                List<NameValuePair> nvps = new ArrayList<NameValuePair>()
;
                if(map!=null){
                        for (Entry<String, String> entry : map.entrySet()
) {
                                nvps.add(new BasicNameValuePair(entry.get
Key(), entry.getValue()));
                        }
                }
                //设置参数到请求对象中
                httpPost.setEntity(new UrlEncodedFormEntity(nvps, encodin
g));

                System.out.println("请求地址："+url);
                System.out.println("请求参数："+nvps.toString());

                //设置header信息
                //指定报文头【Content-type】、【User-Agent】
                httpPost.setHeader("Content-type", "application/x-www-for
m-urlencoded");
                httpPost.setHeader("User-Agent", "Mozilla/4.0 (compatible
; MSIE 5.0; Windows NT; DigExt)");

                //执行请求操作，并拿到结果（同步阻塞）
                CloseableHttpResponse response = client.execute(httpPost)
;
                //获取结果实体
                HttpEntity entity = response.getEntity();
                if (entity != null) {
                        //按指定编码转换结果实体为String类型
                        body = EntityUtils.toString(entity, encoding);
                }
                EntityUtils.consume(entity);
                //释放链接
                response.close();
        return body;
        }
```

测试代码如下：

```
        public static void main(String[] args) throws ParseException, IOE
xception, KeyManagementException, NoSuchAlgorithmException{
                String url = "https://www.facebook.com/";
                String body = send(url, null, "utf-8");
                System.out.println("交易响应结果");
                System.out.println(body);
        }
```

运行后，结果：



```
<terminated> SimpleHttpClientDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2015年11月16日 下午5:20:02)
请求地址：https://www.facebook.com/
请求参数：[]
交易响应结果
<!DOCTYPE html>
<html lang="en" id="facebook" class="no_js">



<head><meta charset="utf-8" /><script>function envFlush(a){function b(c){for(var d in a)c[d]=a[d];}if(wind
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yp/r/I5kTXq1bSJZ.css" da
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yE/r/r75b7ml8FUo.css" d
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yh/r/nBocylo2MWn.css"
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yf/r/5OzB6AYxs9y.css" da
```

果然可以访问成功了。

好了基本的教程就到这里，下篇我将其封装的一个工具类，自认为相对于网上分享的封装类要强大很多，敬请期待吧。

轻松把玩HttpClient之封装HttpClient工具类(一)（现有网上分享中的最强大的工具类）

版权声明：本文为博主原创文章，未经博主允许不得转载。如需转载请声明：【转自 http://blog.csdn.net/xiaoxian8023 】

搜了一下网络上别人封装的HttpClient，大部分特别简单，有一些看起来比较高级，但是用起来都不怎么好用。调用关系不清楚，结构有点混乱。所以也就萌生了自己封装HttpClient工具类的想法。要做就做最好的，本工具类支持插件式配置Header、插件式配置httpclient对象，这样就可以方便地自定义header信息、配置ssl、配置proxy等。

是不是觉得说的有点悬乎了，那就先看看调用吧：

```
        public static void testSimple() throws HttpProcessException{
                String url = "http://www.oschina.net";
                //简单调用
                String resp = HttpClientUtil.send(url);
                System.out.println("请求结果内容长度："+ resp.length());
        }

        public static void testOne() throws HttpProcessException{

                String url = "https://sso.tgb.com:8443/cas/login";

                //自定义HttpClient，设置超时、代理、ssl
                //HttpClient client= HCB.custom().timeout(10000).proxy("1
27.0.0.1", 8087).ssl().build();//采用默认方式（绕过证书验证）
                HttpClient client= HCB.custom().timeout(10000).ssl("D:\\k
eys\\wsriakey","tomcat").build();

                //设置header信息
                Header[] headers=HttpHeader.custom().keepAlive("false").c
onnection("close").contentType(Headers.APP_FORM_URLENCODED).build();

                //执行请求
                String resp=HttpClientUtil.send(client, url, headers);
                System.out.println("请求结果如下：");
                System.out.println(resp);
        }
```

轻松配置了代理、自定义证书的ssl、以及各种header头信息，是不是觉得还凑合呢，那就继续看吧。

写这个工具类时，抽象了一下所有的demo，最后封装了一个最基本的方法（拆分成了2

个方法了），其所有参数列表有：HttpClient对象、url(必须有)、请求方式、请求参数parasMap、header数组、编码格式encoding。

由于封装的是工具类，所以最好是无状态的，可以支持多线程的方式调用的，所以方法都是static类型的。这也是为什么要把HttpClient对象也是作为了一个参数传入而非成员变量了，而且这样也为扩展HttpClient的配置提供了便利。

因为HTTP1.1规范中定义了6种HTTP方法：GET, HEAD, POST, PUT, DELETE, TRACE 和 OPTIONS，其实还有一个PATCH，这几个方法在HttpClient中都有一个对应的类：HttpGet，HttpHead，HttpPost，HttpPut，HttpDelete，HttpTrace、HttpOptions以及HttpPatch。所有的这些类均继承了HttpRequestBase超类，故可以作为参数使用（用枚举类作为参数，用另一个方法来创建具体的请求方法对象）。

Header头信息也是作为一个重要的参数，在请求特定网站的时候需要设置不同的Header，而header又是比较繁杂的，所以这里也是作为了一个参数传入的，也是方便扩展。

使用map来作为post方式传入参数是习惯使然，不做过多的解释。

编码这个参数主要是为了为待提交的数据和反馈结果进行转码处理。

简单说一下流程：

1. 创建请求对象request；
2. 为request设置header信息；
3. 判断当前请求对象是否是HttpEntityEnclosingRequestBase的子类，如果是，则支持setEntity方法，来设置参数。
4. 执行请求，并拿到结果（同步阻塞）；
5. 获取并解码请求结果实体；
6. 关闭链接

就是这么简单，具体来看看代码吧：

```
	/**
	 * 请求资源或服务，自定义client对象，传入请求参数，设置内容类型，并指定参数和返回数据的编码
	 * 
	 * @param client		client对象
	 * @param url			资源地址
	 * @param httpMethod	请求方法
	 * @param parasMap		请求参数
```

```
         * @param headers              请求头信息
         * @param encoding             编码
         * @return                             返回处理结果
         * @throws HttpProcessException
         */
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Map<String,String>parasMap,
                                Header[] headers, String encoding) throws
 HttpProcessException {
                String body = "";
                try {
                        //创建请求对象
                        HttpRequestBase request = getRequest(url, httpMet
hod);

                        //设置header信息
                        request.setHeaders(headers);

                        //判断是否支持设置entity(仅HttpPost、HttpPut、HttpPat
ch支持)
                        if(HttpEntityEnclosingRequestBase.class.isAssigna
bleFrom(request.getClass())){
                                List<NameValuePair> nvps = new ArrayList<
NameValuePair>();

                                //检测url中是否存在参数
                                url = Utils.checkHasParas(url, nvps);

                                //装填参数
                                Utils.map2List(nvps, parasMap);

                                //设置参数到请求对象中
                                ((HttpEntityEnclosingRequestBase)request)
.setEntity(new UrlEncodedFormEntity(nvps, encoding));

                                logger.debug("请求地址："+url);
                                if(nvps.size()>0){
                                        logger.debug("请求参数："+nvps.toSt
ring());
                                }
                        }else{
                                int idx = url.indexOf("?");
                                logger.debug("请求地址："+url.substring(0,
(idx>0 ? idx-1:url.length()-1)));
                                if(idx>0){
                                        logger.debug("请求参数："+url.subst
ring(idx+1));
                                }
                        }

                        //调用发送请求
                        body = execute(client, request, url, encoding);

                } catch (UnsupportedEncodingException e) {
                        throw new HttpProcessException(e);
                }
                return body;
        }


        /**
         * 请求资源或服务
         *
         * @param client                client对象
         * @param request               请求对象
         * @param url                   资源地址
         * @param parasMap              请求参数
         * @param encoding              编码
         * @return                              返回处理结果
         * @throws HttpProcessException
         */
        private static String execute(HttpClient client, HttpRequestBase
 request,String url, String encoding) throws HttpProcessException {
                String body = "";
                HttpResponse response =null;
                try {

                        //执行请求操作，并拿到结果（同步阻塞）
                        response = client.execute(request);

                        //获取结果实体
                        HttpEntity entity = response.getEntity();

                        if (entity != null) {
                                //按指定编码转换结果实体为String类型
                                body = EntityUtils.toString(entity, encod
 ing);

                                logger.debug(body);
                        }
                        EntityUtils.consume(entity);
                } catch (ParseException | IOException e) {
                        throw new HttpProcessException(e);
                } finally {
                        close(response);
                }

                return body;
        }
```

第一个方法中，我们看到有HttpMethods类型的参数，在创建request对象时，用到了它。它是什么呢？其实只是一个枚举类：

```
        /**
         * 枚举HttpMethods方法
         *
         * @author arron
         * @date 2015年11月17日 下午4:45:59
         * @version 1.0
         */
        public enum HttpMethods{

                /**
                 * 求获取Request-URI所标识的资源
                 */
                GET(0, "GET"),

                /**
                 * 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据
被包含在请求体中。
                 * POST请求可能会导致新的资源的建立和/或已有资源的修改
                 */
                POST(1, "POST"),

                /**
                 * 向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。
                 * 这一方法可以在不必传输整个响应内容的情况下，就可以获取包含在响应
消息头中的元信息
                 * 只获取响应信息报头
                 */
                HEAD(2, "HEAD"),

                /**
                 * 向指定资源位置上传其最新内容（全部更新，操作幂等）
                 */
                PUT     (3, "PUT"),

                /**
                 * 请求服务器删除Request-URI所标识的资源
                 */
                DELETE  (4, "DELETE"),

                /**
                 * 请求服务器回送收到的请求信息，主要用于测试或诊断
                 */
                TRACE(5, "TRACE"),

                /**
                 * 向指定资源位置上传其最新内容（部分更新，非幂等）
                 */
                PATCH   (6, "PATCH"),

                /**
                 * 返回服务器针对特定资源所支持的HTTP请求方法。
                 * 也可以利用向Web服务器发送'*'的请求来测试服务器的功能性
                 */
                OPTIONS (7, "OPTIONS"),

//              /**
//               * HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器
//               */
//              CONNECT(99, "CONNECT"),
                ;

                private int code;
                private String name;
                private HttpMethods(int code, String name){
                        this.code = code;
                        this.name = name;
                }
                public String getName() {
                        return name;
                }
                public int getCode() {
                        return code;
                }
        }
```

通过getRequest方法，来实例化对应方法的请求对象。

```
        /**
         * 根据请求方法名，获取request对象
         *
         * @param url                                       资源地址
         * @param method                            请求方式
         * @return
         */
        private static HttpRequestBase getRequest(String url, HttpMethods
method) {
                HttpRequestBase request = null;
                switch (method.getCode()) {
                        case 0:// HttpGet
                                request = new HttpGet(url);
                                break;
                        case 1:// HttpPost
                                request = new HttpPost(url);
                                break;
                        case 2:// HttpHead
                                request = new HttpHead(url);
                                break;
                        case 3:// HttpPut
                                request = new HttpPut(url);
                                break;
                        case 4:// HttpDelete
                                request = new HttpDelete(url);
                                break;
                        case 5:// HttpTrace
                                request = new HttpTrace(url);
                                break;
                        case 6:// HttpPatch
                                request = new HttpPatch(url);
                                break;
                        case 7:// HttpOptions
                                request = new HttpOptions(url);
                                break;
                        default:
                                request = new HttpPost(url);
                                break;
                }
                return request;
        }
```

当然最后的关闭链接也是一个小方法：

```
        /**
         * 尝试关闭response
         *
         * @param resp                          HttpResponse对象
         */
        private static void close(HttpResponse resp) {
                try {
                        if(resp == null) return;
                        //如果CloseableHttpResponse 是resp的父类，则支持关闭
                        if(CloseableHttpResponse.class.isAssignableFrom(r
esp.getClass())){
                                ((CloseableHttpResponse)resp).close();
                        }
                } catch (IOException e) {
                        logger.error(e);
                }
        }
```

当然各种参数的组合方法也简单提供一下（为了节约空间，已去掉注释）：

```
        public static String send(String url) throws HttpProcessException
 {
                return send(url, Charset.defaultCharset().name());
        }
        public static String send(String url, String encoding) throws Htt
pProcessException {
                return send(url, new Header[]{},encoding);
        }
        public static String send(String url, Header[] headers) throws Ht
tpProcessException {
                return send(url, headers, Charset.defaultCharset().name()
);
        }
        public static String send(String url, Header[] headers, String en
coding) throws HttpProcessException {
                return send(url, new HashMap<String,String>(), headers, e
ncoding);
        }
        public static String send(String url, Map<String,String>parasMap)
 throws HttpProcessException {
                return send(url, parasMap, Charset.defaultCharset().name(
));
        }
        public static String send(String url, Map<String,String>parasMap,
 String encoding) throws HttpProcessException {
```

```
                return send(url, parasMap, new Header[]{}, encoding);
        }
        public static String send(String url, Map<String,String>parasMap,
 Header[] headers) throws HttpProcessException {
                return send(url, parasMap, headers, Charset.defaultCharse
t().name());
        }
        public static String send(String url, Map<String,String>parasMap,
 Header[] headers, String encoding) throws HttpProcessException {
                return send(url, HttpMethods.POST, parasMap, headers, enc
oding);
        }
        public static String send(String url, HttpMethods httpMethod) thr
ows HttpProcessException {
                return send(url, httpMethod, Charset.defaultCharset().nam
e());
        }
        public static String send(String url, HttpMethods httpMethod, Str
ing encoding) throws HttpProcessException {
                return send(url, httpMethod, new Header[]{},encoding);
        }
        public static String send(String url, HttpMethods httpMethod, Hea
der[] headers) throws HttpProcessException {
                return send(url, httpMethod, headers, Charset.defaultChar
set().name());
        }
        public static String send(String url, HttpMethods httpMethod, Hea
der[] headers, String encoding) throws HttpProcessException {
                return send(url, httpMethod, new HashMap<String, String>(
), headers, encoding);
        }
        public static String send(String url, HttpMethods httpMethod, Map
<String,String>parasMap) throws HttpProcessException {
                return send(url, httpMethod, parasMap, Charset.defaultCha
rset().name());
        }
        public static String send(String url, HttpMethods httpMethod, Map
<String,String>parasMap, String encoding) throws HttpProcessException {
                return send(url, httpMethod, parasMap, new Header[]{}, en
coding);
        }
        public static String send(String url, HttpMethods httpMethod, Map
<String,String>parasMap, Header[] headers) throws HttpProcessException {
                return send(url, httpMethod, parasMap, headers, Charset.d
efaultCharset().name());
        }
        public static String send(String url, HttpMethods httpMethod, Map
<String,String>parasMap, Header[] headers, String encoding) throws HttpPr
ocessException {
                return send(create(url), url, httpMethod, parasMap, heade
rs, encoding);
        }

        public static String send(HttpClient client, String url) throws H
ttpProcessException {
                return send(client, url, Charset.defaultCharset().name())
```

```
;
        }
        public static String send(HttpClient client, String url, String e
ncoding) throws HttpProcessException {
                return send(client, url, new Header[]{}, encoding);
        }
        public static String send(HttpClient client, String url, Header[]
 headers) throws HttpProcessException {
                return send(client, url, headers, Charset.defaultCharset(
).name());
        }
        public static String send(HttpClient client, String url, Header[]
 headers, String encoding) throws HttpProcessException {
                return send(client, url, new HashMap<String, String>(), h
eaders, encoding);
        }
        public static String send(HttpClient client, String url, Map<Stri
ng,String>parasMap) throws HttpProcessException {
                return send(client, url, parasMap, Charset.defaultCharset
().name());
        }
        public static String send(HttpClient client, String url, Map<Stri
ng,String>parasMap, String encoding) throws HttpProcessException {
                return send(client, url, parasMap, new Header[]{}, encodi
ng);
        }
        public static String send(HttpClient client, String url, Map<Stri
ng,String>parasMap, Header[] headers) throws HttpProcessException {
                return send(client, url, parasMap, headers, Charset.defau
ltCharset().name());
        }
        public static String send(HttpClient client, String url, Map<Stri
ng,String>parasMap,Header[] headers,String encoding) throws HttpProcessEx
ception {
                return send(client, url, HttpMethods.POST, parasMap, head
ers, encoding);
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod) throws HttpProcessException {
                return send(client, url, httpMethod, Charset.defaultChars
et().name());
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, String encoding) throws HttpProcessException {
                return send(client, url, httpMethod, new Header[]{}, enco
ding);
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Header[] headers) throws HttpProcessException {
                return send(client, url, httpMethod, headers, Charset.def
aultCharset().name());
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Header[] headers, String encoding) throws HttpProcessExce
ption {
                return send(client, url, httpMethod, new HashMap<String,
String>(), headers, encoding);
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Map<String,String>parasMap) throws HttpProcessException {
                return send(client, url, httpMethod, parasMap, Charset.de
faultCharset().name());
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Map<String,String>parasMap, String encoding) throws HttpP
rocessException {
                return send(client, url, httpMethod, parasMap, new Header
[]{}, encoding);
        }
        public static String send(HttpClient client, String url, HttpMeth
ods httpMethod, Map<String,String>parasMap, Header[] headers) throws Http
ProcessException {
                return send(client, url, httpMethod, parasMap, headers, C
harset.defaultCharset().name());
        }
```

可以看到上面这一堆方法，其实主要分成2类，一类是传入client对象的，一组是没有传入的。也就是说该工具类提供了一种默认的client对象。这个将会在下一篇文章会有补充。

当然，为了方便操作，还是提供了get、post、put、patch、delete、head、options、trace等方法，由于推荐使用send方法，所以这几个方法只是做了一个简单的调用：

```
        public static String get(String url, Header[] headers,String enco
ding) throws HttpProcessException {
                return get(create(url), url, headers, encoding);
        }
        public static String get(HttpClient client, String url, Header[]
headers,String encoding) throws HttpProcessException {
                return send(client, url, HttpMethods.GET, headers, encodi
ng);
        }

        public static String post(String url, Map<String,String>parasMap,
Header[] headers,String encoding) throws HttpProcessException {
                return post(create(url), url, parasMap, headers, encoding
);
        }
        public static String post(HttpClient client, String url, Map<Stri
ng,String>parasMap,Header[] headers,String encoding) throws HttpProcessEx
ception {
                return send(client, url, HttpMethods.POST, parasMap, head
ers, encoding);
        }

        public static String put(String url, Map<String,String>parasMap,H
eader[] headers,String encoding) throws HttpProcessException {
                return put(create(url), url, parasMap, headers, encoding)
;
```

```
        }
        public static String put(HttpClient client, String url, Map<Strin
g,String>parasMap,Header[] headers,String encoding) throws HttpProcessExc
eption {
                return send(client, url, HttpMethods.PUT, parasMap, heade
rs, encoding);
        }

        public static String delete(String url, Header[] headers,String e
ncoding) throws HttpProcessException {
                return delete(create(url), url, headers, encoding);
        }
        public static String delete(HttpClient client, String url, Header
[] headers,String encoding) throws HttpProcessException {
                return send(client, url, HttpMethods.DELETE, headers, enc
oding);
        }

        public static String patch(String url, Map<String,String>parasMap
,Header[] headers,String encoding) throws HttpProcessException {
                return patch(create(url), url, parasMap, headers, encodin
g);
        }
        public static String patch(HttpClient client, String url, Map<Str
ing,String>parasMap, Header[] headers,String encoding) throws HttpProcess
Exception {
                return send(client, url, HttpMethods.PATCH, parasMap, hea
ders, encoding);
        }

        public static String head(String url, Header[] headers,String enc
oding) throws HttpProcessException {
                return head(create(url), url, headers, encoding);
        }
        public static String head(HttpClient client, String url, Header[]
 headers,String encoding) throws HttpProcessException {
                return send(client, url, HttpMethods.HEAD, headers, encod
ing);
        }

        public static String options(String url, Header[] headers,String
encoding) throws HttpProcessException {
                return options(create(url), url, headers, encoding);
        }
        public static String options(HttpClient client, String url, Heade
r[] headers,String encoding) throws HttpProcessException {
                return send(client, url, HttpMethods.OPTIONS, headers, en
coding);
        }

        public static String trace(String url, Header[] headers,String en
coding) throws HttpProcessException {
                return trace(create(url), url, headers, encoding);
        }
        public static String trace(HttpClient client, String url, Header[
] headers,String encoding) throws HttpProcessException {
```

```
                return send(client, url, HttpMethods.TRACE, headers, enco
ding);
        }
```

差点忘记了，最后还有一个简单的通用工具类

```
/**
 *
 * @author arron
 * @date 2015年11月10日 下午12:49:26
 * @version 1.0
 */
public class Utils {

        /**
         * 检测url是否含有参数，如果有，则把参数加到参数列表中
         *
         * @param url                                   资源地址
         * @param nvps                          参数列表
         * @return      返回去掉参数的url
         */
        public static String checkHasParas(String url, List<NameValuePair
> nvps) {
                // 检测url中是否存在参数
                if (url.contains("?") && url.indexOf("?") < url.indexOf("
=")) {
                        Map<String, String> map = buildParas(url.substrin
g(url
                                        .indexOf("?") + 1));
                        map2List(nvps, map);
                        url = url.substring(0, url.indexOf("?"));
                }
                return url;
        }

        /**
         * 参数转换，将map中的参数，转到参数列表中
         *
         * @param nvps                          参数列表
         * @param map                           参数列表（map）
         */
        public static void map2List(List<NameValuePair> nvps, Map<String,
 String> map) {
                if(map==null) return;
                // 拼接参数
                for (Entry<String, String> entry : map.entrySet()) {
                        nvps.add(new BasicNameValuePair(entry.getKey(), e
ntry
                                        .getValue()));
                }
        }
	/**
	 * 生成参数
	 * 参数格式“k1=v1&k2=v2”
	 *
	 * @param paras					参数列表
	 * @return						返回参数列
表（map）
	 */
	public static Map<String,String> buildParas(String paras){
		String[] p = paras.split("&");
		String[][] ps = new String[p.length][2];
		int pos = 0;
		for (int i = 0; i < p.length; i++) {
			pos = p[i].indexOf("=");
			ps[i][0]=p[i].substring(0,pos);
			ps[i][1]=p[i].substring(pos+1);
			pos = 0;
		}
		return buildParas(ps);
	}

	/**
	 * 生成参数
	 * 参数类型：{{"k1","v1"},{"k2","v2"}}
	 *
	 * @param paras						参数列表
	 * @return						返回参数列
表（map）
	 */
	public static Map<String,String> buildParas(String[][] paras){
		// 创建参数队列
		Map<String,String> map = new HashMap<String, String>();
		for (String[] para: paras) {
			map.put(para[0], para[1]);
		}
		return map;
	}

}
```

简单的封装就是这样了。

由于HttpClient和Header都作为参数传入，所以也可以进行扩展，比如代理、ssl等都是对HttpClient进行配置的，下面的文章就分别分享一下如何插件式配置HttpClient以及Header。敬请期待。

代码已上传至：https://github.com/Arronlong/httpclientUtil。

轻松把玩HttpClient之封装HttpClient工具类(二)，插件式配置HttpClient对象

上一篇文章中，简单分享一下封装HttpClient工具类的思路及部分代码，本文将分享如何实现插件式配置HttpClient对象。

如果你看过我前面的几篇关于HttpClient的文章或者官网示例，应该都知道HttpClient对象在创建时，都可以设置各种参数，但是却没有简单的进行封装，比如对我来说比较重要的3个：代理、ssl（包含绕过证书验证和自定义证书验证）、超时。还需要自己写。所以这里我就简单封装了一下，顺便还封装了一个连接池的配置。

其实说是插件式配置，那是高大上的说法，说白了，就是采用了建造者模式来创建HttpClient对象（级联调用）。HttpClient的jar包中提供了一个创建HttpClient对象的类HttpClientBuilder。所以我是创建该类的子类HCB，然后做了一些改动。每个配置方法的返回值都是HCB，这样就支持级联调用了。具体代码如下：

```
package com.tgb.ccl.http.httpclient.builder;

import org.apache.http.HttpHost;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.DefaultProxyRoutePlanner;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;

import com.tgb.ccl.http.common.SSLs;
import com.tgb.ccl.http.exception.HttpProcessException;

/**
 * httpclient创建者
 *
 * @author arron
 * @date 2015年11月9日 下午5:45:47
 * @version 1.0
 */
public class  HCB extends HttpClientBuilder{
        private boolean isSetPool=false;//记录是否设置了连接池
        private boolean isNewSSL=false;//记录是否设置了更新了ssl

        //用于配置ssl
        private SSLs ssls = SSLs.getInstance();

        private HCB(){}
        public static HCB custom(){
                return new HCB();
        }

        /**
         * 设置超时时间
         *
         * @param timeout               超市时间，单位-毫秒
         * @return
         */
        public HCB timeout(int timeout){
                 // 配置请求的超时设置
        RequestConfig config = RequestConfig.custom()
                .setConnectionRequestTimeout(timeout)
                .setConnectTimeout(timeout)
                .setSocketTimeout(timeout)
                .build();
                return (HCB) this.setDefaultRequestConfig(config);
        }

        /**
         * 设置ssl安全链接
         *
         * @return
         * @throws HttpProcessException
         */
        public HCB ssl() throws HttpProcessException {
                if(isSetPool){//如果已经设置过线程池，那肯定也就是https链接了
                        if(isNewSSL){
                                throw new HttpProcessException("请先设置ss
l，后设置pool");
                        }
                        return this;
                }
                Registry<ConnectionSocketFactory> socketFactoryRegistry =
 RegistryBuilder
                                .<ConnectionSocketFactory> create()
                                .register("http", PlainConnectionSocketFa
ctory.INSTANCE)
                                .register("https", ssls.getSSLCONNSF()).b
uild();
                //设置连接池大小
                PoolingHttpClientConnectionManager connManager = new Pool
ingHttpClientConnectionManager(socketFactoryRegistry);
                return (HCB) this.setConnectionManager(connManager);
        }


        /**
```

```
	 * 设置自定义sslcontext
	 * 
	 * @param keyStorePath		密钥库路径
	 * @return
	 * @throws HttpProcessException
	 */
	public HCB ssl(String keyStorePath) throws HttpProcessException{
		return ssl(keyStorePath,"nopassword");
	}
	/**
	 * 设置自定义sslcontext
	 * 
	 * @param keyStorePath		密钥库路径
	 * @param keyStorepass		密钥库密码
	 * @return
	 * @throws HttpProcessException
	 */
	public HCB ssl(String keyStorePath, String keyStorepass) throws H
ttpProcessException{
		this.ssls = SSLs.custom().customSSL(keyStorePath, keyStor
epass);
		this.isNewSSL=true;
		return ssl();
	}
	
	
	/**
	 * 设置连接池（默认开启https）
	 * 
	 * @param maxTotal					最大连接数
	 * @param defaultMaxPerRoute	每个路由默认连接数
	 * @return
	 * @throws HttpProcessException
	 */
	public HCB pool(int maxTotal, int defaultMaxPerRoute) throws Http
ProcessException{
		Registry<ConnectionSocketFactory> socketFactoryRegistry =
 RegistryBuilder
				.<ConnectionSocketFactory> create()
				.register("http", PlainConnectionSocketFa
ctory.INSTANCE)
				.register("https", ssls.getSSLCONNSF()).b
uild();
		//设置连接池大小
		PoolingHttpClientConnectionManager connManager = new Pool
ingHttpClientConnectionManager(socketFactoryRegistry);
		connManager.setMaxTotal(maxTotal);
		connManager.setDefaultMaxPerRoute(defaultMaxPerRoute);
		isSetPool=true;
		return (HCB) this.setConnectionManager(connManager);
	}
	
	/**
	 * 设置代理
	 * 
	 * @param hostOrIP		代理host或者ip
```

```
          * @param port                  代理端口
          * @return
          */
        public HCB proxy(String hostOrIP, int port){
                // 依次是代理地址，代理端口号，协议类型
                HttpHost proxy = new HttpHost(hostOrIP, port, "http");
                DefaultProxyRoutePlanner routePlanner = new DefaultProxyR
outePlanner(proxy);
                return (HCB) this.setRoutePlanner(routePlanner);
        }
}
```

大家可以看到，这个有成员变量，而且不是static类型，所以是非线程安全的。所以我为了方便使用，就效仿HttpClients（其custom方法可以创建HttpClientBuilder实例）写了一个静态的custom方法，来返回一个新的HCB实例。将构造方法设置成了private，无法通过new的方式创建实例，所以只能通过custom方法来创建。在想生成HttpClient对象的时候，调用一下build方法就可以了。于是乎就出现了这样简单、方便又明了的调用方式：

```
        HttpClient client = HCB.custom().timeout(10000).proxy("127.0.0.1"
, 8087).ssl("D:\\keys\\wsriakey","tomcat").build();
```

说到ssl，还需要另外一个封装的类，为了其他工具类有可能也会用到ssl，所以就单出来了。不多解释，直接上代码：

```
/**
 * 设置ssl
 *
 * @author arron
 * @date 2015年11月3日 下午3:11:54
 * @version 1.0
 */
public class SSLs {

    private static final SSLHandler simpleVerifier = new SSLHandler();
        private static SSLConnectionSocketFactory sslConnFactory ;
        private static SSLs sslutil = new SSLs();
        private SSLContext sc;

        public static SSLs getInstance(){
                return sslutil;
        }
        public static SSLs custom(){
                return new SSLs();
        }

    // 重写X509TrustManager类的三个方法,信任服务器证书
    private static class SSLHandler implements  X509TrustManager, Hostnam
eVerifier{
```

```
                @Override
                public java.security.cert.X509Certificate[] getAcceptedIs
suers() {
                        return null;
                }

                @Override
                public void checkServerTrusted(java.security.cert.X509Cer
tificate[] chain,
                                String authType) throws java.security.cer
t.CertificateException {
                }

                @Override
                public void checkClientTrusted(java.security.cert.X509Cer
tificate[] chain,
                                String authType) throws java.security.cer
t.CertificateException {
                }

                @Override
                public boolean verify(String paramString, SSLSession para
mSSLSession) {
                        return true;
                }
        };

        // 信任主机
    public static HostnameVerifier getVerifier() {
        return simpleVerifier;
    }

    public synchronized SSLConnectionSocketFactory getSSLCONNSF() throws
HttpProcessException {
        if (sslConnFactory != null)
                return sslConnFactory;
        try {
                SSLContext sc = getSSLContext();
                sc.init(null, new TrustManager[] { simpleVerifier }, null
);
                sslConnFactory = new SSLConnectionSocketFactory(sc, simpl
eVerifier);
                } catch (KeyManagementException e) {
                        throw new HttpProcessException(e);
                }
        return sslConnFactory;
    }

    public SSLs customSSL(String keyStorePath, String keyStorepass) throw
s HttpProcessException{
        FileInputStream instream =null;
        KeyStore trustStore = null;
                try {
                        trustStore = KeyStore.getInstance(KeyStore.getDef
aultType());
```

```
                        instream = new FileInputStream(new File(keyStoreP
ath));
                trustStore.load(instream, keyStorepass.toCharArray());
            // 相信自己的CA和所有自签名的证书
                sc= SSLContexts.custom().loadTrustMaterial(trustStore, ne
w TrustSelfSignedStrategy()) .build();
                } catch (KeyStoreException | NoSuchAlgorithmException | C
ertificateException | IOException | KeyManagementException e) {
                        throw new HttpProcessException(e);
                }finally{
                        try {
                                instream.close();
                        } catch (IOException e) {}
                }
                return this;
    }

    public SSLContext getSSLContext() throws HttpProcessException{
        try {
                if(sc==null){
                        sc = SSLContext.getInstance("SSLv3");
                }
                        return sc;
                } catch (NoSuchAlgorithmException e) {
                        throw new HttpProcessException(e);
                }
    }
}
```

基本上就是这样了。在上一篇中遗留了一个小问题，正好在这里说一下。上一篇文中说道提供一个默认的HttpClient实现，其实是2个，分别针对于http和https。方便调用。具体代码如下：

```
        //默认采用的http协议的HttpClient对象
        private static  HttpClient client4HTTP;

        //默认采用的https协议的HttpClient对象
        private static HttpClient client4HTTPS;

        static{
                try {
                        client4HTTP = HCB.custom().build();
                        client4HTTPS = HCB.custom().ssl().build();
                } catch (HttpProcessException e) {
                        logger.error("创建https协议的HttpClient对象出错：{}"
, e);
                }
        }

        /**
         * 判断url是http还是https，直接返回相应的默认client对象
         *
```

```
          * @return                                              返回对应默
认的client对象
          * @throws HttpProcessException
          */
         private static HttpClient create(String url) throws HttpProcessEx
ception  {
                 if(url.toLowerCase().startsWith("https://")){
                         return client4HTTPS;
                 }else{
                         return client4HTTP;
                 }
         }
```

这样在使用工具类的时候，如果不需要自定义HttpClient时，就直接用下面的方式调用：

```
         public static void testSimple() throws HttpProcessException{
                 String url = "http://tool.oschina.net/";
                 //简单调用
                 String resp = HttpClientUtil.send(url);
                 System.out.println("请求结果内容长度："+ resp);
         }
```

好了，插件化配置HttpClient，就是这些内容，在下一篇文章中分享如何插件式配置Header。没错，思路还是跟本文一样。敬请期待吧。

代码已上传至：https://github.com/Arronlong/httpclientUtil。

轻松把玩HttpClient之封装HttpClient工具类(三)，插件式配置Header

上篇文章介绍了插件式配置HttpClient，本文将介绍插件式配置Header。

为什么要配置header在前面已经提到了，还里再简单说一下，要使用HttpClient模拟请求，去访问各种接口或者网站资源，都有可能有各种限制，比如说java客户端模拟访问csdn博客，就必须设置User-Agent，否则就报错了。还有各种其他情况，必须的设置一些特定的Header，才能请求成功，或者才能不出问题。

好了就说这么多，本次还是采用构造者模式的级联调用方式，来完成该工具类。在该工具类中，为所有常用的Http Request Header都提供了设置方法。具体参数参考的链接是 HTTP Header 详解。

不再多废话了，看具体代码吧：

```
package com.tgb.ccl.http.common;

import java.util.HashMap;
import java.util.Map;

import org.apache.http.Consts;
import org.apache.http.Header;
import org.apache.http.message.BasicHeader;

/**
 * 创建HttpReqHead
 *
 * @author arron
 * @date 2015年11月9日 上午10:37:23
 * @version 1.0
 */
public class HttpHeader {

        private HttpHeader() {};

        public static HttpHeader custom() {
                return new HttpHeader();
        }

        //记录head头信息
        private Map<String, Header> headerMaps = new HashMap<String, Head
er>();

        /**
         * 指定客户端能够接收的内容类型
         * 例如：Accept: text/plain, text/html
         *
         * @param accept
         */
        public HttpHeader accept(String accept) {
                headerMaps.put(HttpReqHead.ACCEPT,
                                new BasicHeader(HttpReqHead.ACCEPT, accep
t));
                return this;
        }

        /**
         * 浏览器可以接受的字符编码集
         * 例如：Accept-Charset: iso-8859-5
         *
         * @param acceptCharset
         */
        public HttpHeader acceptCharset(String acceptCharset) {
                headerMaps.put(HttpReqHead.ACCEPT_CHARSET,
                                new BasicHeader(HttpReqHead.ACCEPT_CHARSE
T, acceptCharset));
                return this;
        }

        /**
         * 指定浏览器可以支持的web服务器返回内容压缩编码类型
         * 例如：Accept-Encoding: compress, gzip
         *
         * @param acceptEncoding
         */
        public HttpHeader acceptEncoding(String acceptEncoding) {
                headerMaps.put(HttpReqHead.ACCEPT_ENCODING,
                                new BasicHeader(HttpReqHead.ACCEPT_ENCODI
NG, acceptEncoding));
                return this;
        }

        /**
         * 浏览器可接受的语言
         * 例如：Accept-Language: en,zh
         *
         * @param acceptLanguage
         */
        public HttpHeader acceptLanguage(String acceptLanguage) {
                headerMaps.put(HttpReqHead.ACCEPT_LANGUAGE,
                                new BasicHeader(HttpReqHead.ACCEPT_LANGUA
GE, acceptLanguage));
                return this;
        }

        /**
```

```
	 * 可以请求网页实体的一个或者多个子范围字段
	 * 例如：Accept-Ranges: bytes
	 * 
	 * @param acceptRanges
	 */
	public HttpHeader acceptRanges(String acceptRanges) {
		headerMaps.put(HttpReqHead.ACCEPT_RANGES,
				new BasicHeader(HttpReqHead.ACCEPT_RANGES
, acceptRanges));
		return this;
	}

	/**
	 * HTTP授权的授权证书
	 * 例如：Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
	 * 
	 * @param authorization
	 */
	public HttpHeader authorization(String authorization) {
		headerMaps.put(HttpReqHead.AUTHORIZATION,
				new BasicHeader(HttpReqHead.AUTHORIZATION
, authorization));
		return this;
	}

	/**
	 * 指定请求和响应遵循的缓存机制
	 * 例如：Cache-Control: no-cache
	 * 
	 * @param cacheControl
	 */
	public HttpHeader cacheControl(String cacheControl) {
		headerMaps.put(HttpReqHead.CACHE_CONTROL,
				new BasicHeader(HttpReqHead.CACHE_CONTROL
, cacheControl));
		return this;
	}

	/**
	 * 表示是否需要持久连接（HTTP 1.1默认进行持久连接）
	 * 例如：Connection: close 短链接； Connection: keep-alive 长连接
	 * 
	 * @param connection
	 * @return
	 */
	public HttpHeader connection(String connection) {
		headerMaps.put(HttpReqHead.CONNECTION,
				new BasicHeader(HttpReqHead.CONNECTION, c
onnection));
		return this;
	}

	/**
	 * HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器
	 * 例如：Cookie: $Version=1; Skin=new;
	 * 
```

```
         * @param cookie
         */
        public HttpHeader cookie(String cookie) {
                headerMaps.put(HttpReqHead.COOKIE,
                                new BasicHeader(HttpReqHead.COOKIE, cooki
e));
                return this;
        }

        /**
         * 请求内容长度
         * 例如：Content-Length: 348
         *
         * @param contentLength
         */
        public HttpHeader contentLength(String contentLength) {
                headerMaps.put(HttpReqHead.CONTENT_LENGTH,
                                new BasicHeader(HttpReqHead.CONTENT_LENGT
H, contentLength));
                return this;
        }

        /**
         * 请求的与实体对应的MIME信息
         * 例如：Content-Type: application/x-www-form-urlencoded
         *
         * @param contentType
         */
        public HttpHeader contentType(String contentType) {
                headerMaps.put(HttpReqHead.CONTENT_TYPE,
                                new BasicHeader(HttpReqHead.CONTENT_TYPE,
 contentType));
                return this;
        }

        /**
         * 请求发送的日期和时间
         * 例如：Date: Tue, 15 Nov 2010 08:12:31 GMT
         *
         * @param date
         * @return
         */
        public HttpHeader date(String date) {
                headerMaps.put(HttpReqHead.DATE,
                                new BasicHeader(HttpReqHead.DATE, date));
                return this;
        }

        /**
         * 请求的特定的服务器行为
         * 例如：Expect: 100-continue
         *
         * @param expect
         */
        public HttpHeader expect(String expect) {
                headerMaps.put(HttpReqHead.EXPECT,
                                new BasicHeader(HttpReqHead.EXPECT, expec
t));
                return this;
        }

        /**
         * 发出请求的用户的Email
         * 例如：From: user@email.com
         *
         * @param from
         */
        public HttpHeader from(String from) {
                headerMaps.put(HttpReqHead.FROM,
                                new BasicHeader(HttpReqHead.FROM, from));
                return this;
        }

        /**
         * 指定请求的服务器的域名和端口号
         * 例如：Host: blog.csdn.net
         *
         * @param host
         * @return
         */
        public HttpHeader host(String host) {
                headerMaps.put(HttpReqHead.HOST,
                                new BasicHeader(HttpReqHead.HOST, host));
                return this;
        }

        /**
         * 只有请求内容与实体相匹配才有效
         * 例如：If-Match: “737060cd8c284d8af7ad3082f209582d”
         *
         * @param ifMatch
         * @return
         */
        public HttpHeader ifMatch(String ifMatch) {
                headerMaps.put(HttpReqHead.IF_MATCH,
                                new BasicHeader(HttpReqHead.IF_MATCH, ifM
atch));
                return this;
        }

        /**
         * 如果请求的部分在指定时间之后被修改则请求成功，未被修改则返回304代码
         * 例如：If-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT
         *
         * @param ifModifiedSince
         * @return
         */
        public HttpHeader ifModifiedSince(String ifModifiedSince) {
                headerMaps.put(HttpReqHead.IF_MODIFIED_SINCE,
                                new BasicHeader(HttpReqHead.IF_MODIFIED_S
INCE, ifModifiedSince));
                return this;
        }

        /**
         * 如果内容未改变返回304代码，参数为服务器先前发送的Etag，与服务器回应的Eta
g比较判断是否改变
         * 例如：If-None-Match: “737060cd8c284d8af7ad3082f209582d”
         *
         * @param ifNoneMatch
         * @return
         */
        public HttpHeader ifNoneMatch(String ifNoneMatch) {
                headerMaps.put(HttpReqHead.IF_NONE_MATCH,
                                new BasicHeader(HttpReqHead.IF_NONE_MATCH
, ifNoneMatch));
                return this;
        }

        /**
         * 如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。参数也为Etag
         * 例如：If-Range: “737060cd8c284d8af7ad3082f209582d”
         *
         * @param ifRange
         * @return
         */
        public HttpHeader ifRange(String ifRange) {
                headerMaps.put(HttpReqHead.IF_RANGE,
                                new BasicHeader(HttpReqHead.IF_RANGE, ifR
ange));
                return this;
        }

        /**
         * 只在实体在指定时间之后未被修改才请求成功
         * 例如：If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT
         *
         * @param ifUnmodifiedSince
         * @return
         */
        public HttpHeader ifUnmodifiedSince(String ifUnmodifiedSince) {
                headerMaps.put(HttpReqHead.IF_UNMODIFIED_SINCE,
                                new BasicHeader(HttpReqHead.IF_UNMODIFIED
_SINCE, ifUnmodifiedSince));
                return this;
        }

        /**
         * 限制信息通过代理和网关传送的时间
         * 例如：Max-Forwards: 10
         *
         * @param maxForwards
         * @return
         */
        public HttpHeader maxForwards(String maxForwards) {
                headerMaps.put(HttpReqHead.MAX_FORWARDS,
                                new BasicHeader(HttpReqHead.MAX_FORWARDS,
 maxForwards));
```

```
                return this;
        }

        /**
         * 用来包含实现特定的指令
         * 例如：Pragma: no-cache
         *
         * @param pragma
         * @return
         */
        public HttpHeader pragma(String pragma) {
                headerMaps.put(HttpReqHead.PRAGMA,
                                new BasicHeader(HttpReqHead.PRAGMA, pragm
a));
                return this;
        }

        /**
         * 连接到代理的授权证书
         * 例如：Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
         *
         * @param proxyAuthorization
         */
        public HttpHeader proxyAuthorization(String proxyAuthorization) {
                headerMaps.put(HttpReqHead.PROXY_AUTHORIZATION,
                                new BasicHeader(HttpReqHead.PROXY_AUTHORI
ZATION, proxyAuthorization));
                return this;
        }

        /**
         * 只请求实体的一部分，指定范围
         * 例如：Range: bytes=500-999
         *
         * @param range
         */
        public HttpHeader range(String range) {
                headerMaps.put(HttpReqHead.RANGE,
                                new BasicHeader(HttpReqHead.RANGE, range)
);
                return this;
        }

        /**
         * 先前网页的地址，当前请求网页紧随其后,即来路
         * 例如：Referer: http://www.zcmhi.com/archives/71.html
         *
         * @param referer
         */
        public HttpHeader referer(String referer) {
                headerMaps.put(HttpReqHead.REFERER,
                                new BasicHeader(HttpReqHead.REFERER, refe
rer));
                return this;
        }
        /**
         * 客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息
         * 例如：TE: trailers,deflate;q=0.5
         *
         * @param te
         */
        public HttpHeader te(String te) {
                headerMaps.put(HttpReqHead.TE,
                                new BasicHeader(HttpReqHead.TE, te));
                return this;
        }

        /**
         * 向服务器指定某种传输协议以便服务器进行转换（如果支持）
         * 例如：Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
         *
         * @param upgrade
         */
        public HttpHeader upgrade(String upgrade) {
                headerMaps.put(HttpReqHead.UPGRADE,
                                new BasicHeader(HttpReqHead.UPGRADE, upgr
ade));
                return this;
        }

        /**
         * User-Agent的内容包含发出请求的用户信息
         *
         * @param userAgent
         * @return
         */
        public HttpHeader userAgent(String userAgent) {
                headerMaps.put(HttpReqHead.USER_AGENT,
                                new BasicHeader(HttpReqHead.USER_AGENT, u
serAgent));
                return this;
        }

        /**
         * 关于消息实体的警告信息
         * 例如：Warn: 199 Miscellaneous warning
         *
         * @param warning
         * @return
         */
        public HttpHeader warning(String warning) {
                headerMaps.put(HttpReqHead.WARNING,
                                new BasicHeader(HttpReqHead.WARNING, warn
ing));
                return this;
        }

        /**
         * 通知中间网关或代理服务器地址，通信协议
         * 例如：Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
         *
         * @param via
         * @return
         */
        public HttpHeader via(String via) {
                headerMaps.put(HttpReqHead.VIA,
                                new BasicHeader(HttpReqHead.VIA, via));
                return this;
        }

        /**
         * 设置此HTTP连接的持续时间（超时时间）
         * 例如：Keep-Alive: 300
         *
         * @param keepAlive
         * @return
         */
        public HttpHeader keepAlive(String keepAlive) {
                headerMaps.put(HttpReqHead.KEEP_ALIVE,
                                new BasicHeader(HttpReqHead.KEEP_ALIVE, k
eepAlive));
                return this;
        }

        public String accept() {
                return get(HttpReqHead.ACCEPT);
        }

        public String acceptCharset() {
                return get(HttpReqHead.ACCEPT_CHARSET);
        }

        public String acceptEncoding() {
                return get(HttpReqHead.ACCEPT_ENCODING);
        }

        public String acceptLanguage() {
                return get(HttpReqHead.ACCEPT_LANGUAGE);
        }

        public String acceptRanges() {
                return get(HttpReqHead.ACCEPT_RANGES);
        }

        public String authorization() {
                return get(HttpReqHead.AUTHORIZATION);
        }

        public String cacheControl() {
                return get(HttpReqHead.CACHE_CONTROL);
        }

        public String connection() {
                return get(HttpReqHead.CONNECTION);
        }

        public String cookie() {
                return get(HttpReqHead.COOKIE);
        }

        public String contentLength() {
                return get(HttpReqHead.CONTENT_LENGTH);
        }

        public String contentType() {
                return get(HttpReqHead.CONTENT_TYPE);
        }

        public String date() {
                return get(HttpReqHead.DATE);
        }

        public String expect() {
                return get(HttpReqHead.EXPECT);
        }

        public String from() {
                return get(HttpReqHead.FROM);
        }

        public String host() {
                return get(HttpReqHead.HOST);
        }

        public String ifMatch() {
                return get(HttpReqHead.IF_MATCH);
        }

        public String ifModifiedSince() {
                return get(HttpReqHead.IF_MODIFIED_SINCE);
        }

        public String ifNoneMatch() {
                return get(HttpReqHead.IF_NONE_MATCH);
        }

        public String ifRange() {
                return get(HttpReqHead.IF_RANGE);
        }

        public String ifUnmodifiedSince() {
                return get(HttpReqHead.IF_UNMODIFIED_SINCE);
        }

        public String maxForwards() {
                return get(HttpReqHead.MAX_FORWARDS);
        }

        public String pragma() {
                return get(HttpReqHead.PRAGMA);
        }

        public String proxyAuthorization() {
                return get(HttpReqHead.PROXY_AUTHORIZATION);
        }

        public String referer() {
                return get(HttpReqHead.REFERER);
        }

        public String te() {
                return get(HttpReqHead.TE);
        }

        public String upgrade() {
                return get(HttpReqHead.UPGRADE);
        }

        public String userAgent() {
                return get(HttpReqHead.USER_AGENT);
        }

        public String via() {
                return get(HttpReqHead.VIA);
        }

        public String warning() {
                return get(HttpReqHead.WARNING);
        }

        public String keepAlive() {
                return get(HttpReqHead.KEEP_ALIVE);
        }


        /**
         * 获取head信息
         *
         * @return
         */
        private String get(String headName) {
                if (headerMaps.containsKey(headName)) {
                        return headerMaps.get(headName).getValue();
                }
                return null;
        }

        /**
         * 返回header头信息
         *
         * @return
         */
        public Header[] build() {
                Header[] headers = new Header[headerMaps.size()];
                int i = 0;
                for (Header header : headerMaps.values()) {
                        headers[i] = header;
                        i++;
                }
                headerMaps.clear();
                headerMaps = null;
                return headers;
        }

        /**
         * Http头信息
         *
         * @author arron
         * @date 2015年11月9日 上午11:29:04
         * @version 1.0
         */
        private static class HttpReqHead {
                public static final String ACCEPT = "Accept";
                public static final String ACCEPT_CHARSET = "Accept-Chars
et";
                public static final String ACCEPT_ENCODING = "Accept-Enco
ding";
                public static final String ACCEPT_LANGUAGE = "Accept-Lang
uage";
                public static final String ACCEPT_RANGES = "Accept-Ranges
";
                public static final String AUTHORIZATION = "Authorization
";
                public static final String CACHE_CONTROL = "Cache-Control
";
                public static final String CONNECTION = "Connection";
                public static final String COOKIE = "Cookie";
                public static final String CONTENT_LENGTH = "Content-Leng
th";
                public static final String CONTENT_TYPE = "Content-Type";
                public static final String DATE= "Date";
                public static final String EXPECT = "Expect";
                public static final String FROM = "From";
                public static final String HOST = "Host";
                public static final String IF_MATCH = "If-Match ";
                public static final String IF_MODIFIED_SINCE = "If-Modifi
ed-Since";
                public static final String IF_NONE_MATCH = "If-None-Match
";
                public static final String IF_RANGE = "If-Range";
                public static final String IF_UNMODIFIED_SINCE = "If-Unmo
dified-Since";
                public static final String KEEP_ALIVE = "Keep-Alive";
                public static final String MAX_FORWARDS = "Max-Forwards";
                public static final String PRAGMA = "Pragma";
                public static final String PROXY_AUTHORIZATION = "Proxy-A
uthorization";
                public static final String RANGE = "Range";
                public static final String REFERER = "Referer";
                public static final String TE = "TE";
                public static final String UPGRADE = "Upgrade";
                public static final String USER_AGENT = "User-Agent";
                public static final String VIA = "Via";
                public static final String WARNING = "Warning";
        }
        /**
         * 常用头信息配置
         *
         * @author arron
         * @date 2015年11月18日 下午5:30:00
         * @version 1.0
         */
        public static class Headers{
                public static final String APP_FORM_URLENCODED="applicati
on/x-www-form-urlencoded";
                public static final String TEXT_PLAIN="text/plain";
                public static final String TEXT_HTML="text/html";
                public static final String TEXT_XML="text/xml";
                public static final String TEXT_JSON="text/json";
                public static final String CONTENT_CHARSET_ISO_8859_1 = C
onsts.ISO_8859_1.name();
                public static final String CONTENT_CHARSET_UTF8 = Consts.
UTF_8.name();
                public static final String DEF_PROTOCOL_CHARSET = Consts.
ASCII.name();
                public static final String CONN_CLOSE = "close";
                public static final String KEEP_ALIVE = "keep-alive";
                public static final String EXPECT_CONTINUE = "100-continu
e";
        }
}
```

调用方式：

```
        //设置header信息
        Header[] headers=HttpHeader.custom().keepAlive("false").connectio
n("close").contentType(Headers.APP_FORM_URLENCODED).build();
```

就是这么简单。到此该工具类就完成了。下一篇将分享该工具类以及单次调用测试和多线程调用测试。

代码已上传至：https://github.com/Arronlong/httpclientUtil。

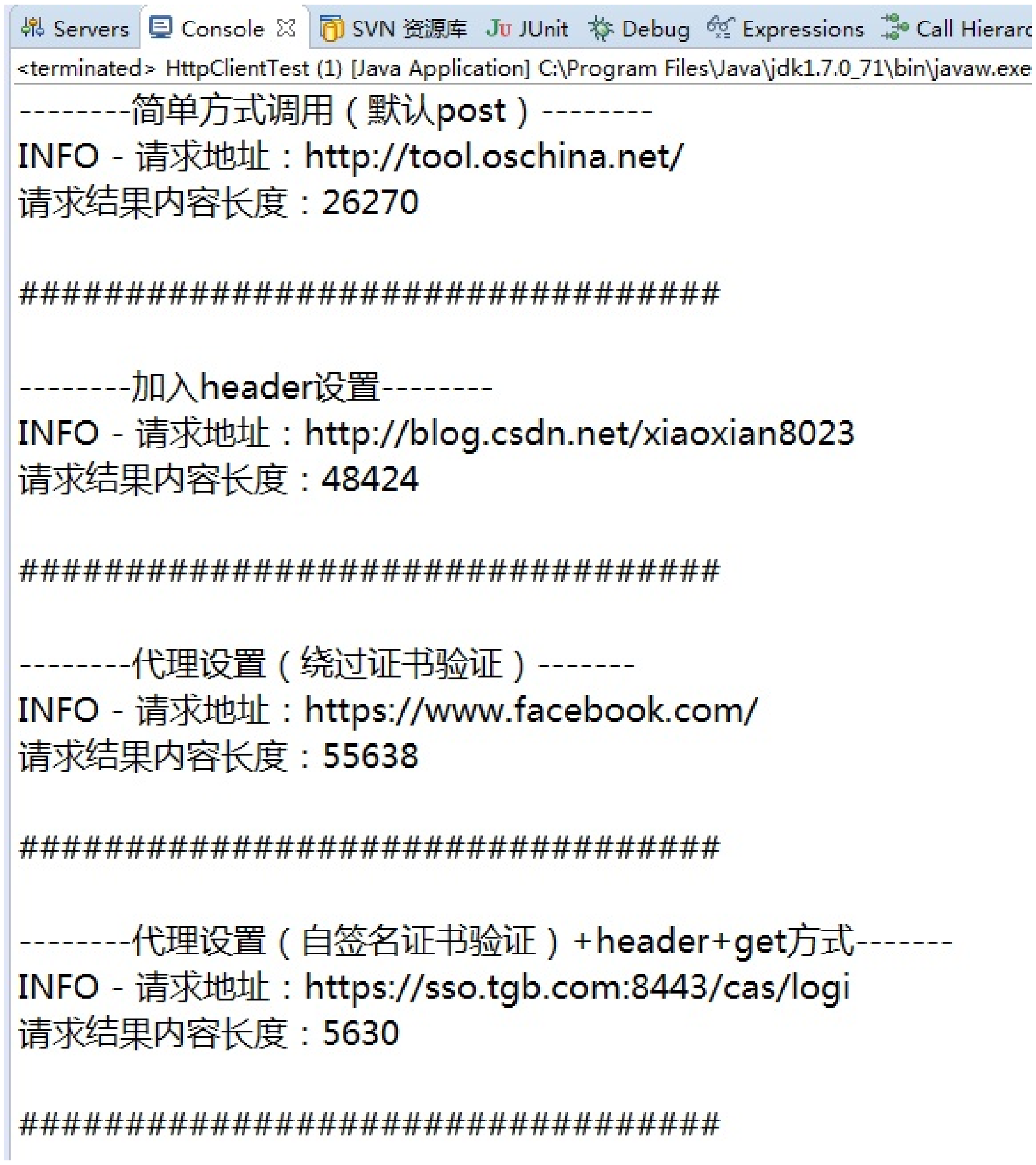
轻松把玩HttpClient之封装HttpClient工具类(四)，单线程调用及多线程批量调用测试

本文主要来分享一下该工具类的测试结果。工具类的整体源码不再单独分享，源码基本上都已经在文章中了。开始我们的测试。

单线程调用测试：

```
        public static void testOne() throws HttpProcessException{

                System.out.println("--------简单方式调用（默认post）--------
");
                String url = "http://tool.oschina.net/";
                //简单调用
                String resp = HttpClientUtil.send(url);
                System.out.println("请求结果内容长度："+ resp.length());

                System.out.println("\n#################################\n
");

                System.out.println("--------加入header设置--------");
                url="http://blog.csdn.net/xiaoxian8023";
                //设置header信息
                Header[] headers=HttpHeader.custom().userAgent("Mozilla/5
.0").build();
                //执行请求
                resp = HttpClientUtil.send(url, headers);
                System.out.println("请求结果内容长度："+ resp.length());

                System.out.println("\n#################################\n
");

                System.out.println("--------代理设置（绕过证书验证）-------")
;
                url="https://www.facebook.com/";
                HttpClient client= HCB.custom().timeout(10000).proxy("127
.0.0.1", 8087).ssl().build();//采用默认方式（绕过证书验证）
                //执行请求
                resp = HttpClientUtil.send(client,url);
                System.out.println("请求结果内容长度："+ resp.length());

                System.out.println("\n#################################\n
");

                System.out.println("--------代理设置（自签名证书验证）+header
+get方式-------");
                url = "https://sso.tgb.com:8443/cas/login";
                client= HCB.custom().timeout(10000).ssl("D:\\keys\\wsriak
ey","tomcat").build();
                headers=HttpHeader.custom().keepAlive("false").connection
("close").contentType(Headers.APP_FORM_URLENCODED).build();
                //执行请求
                resp = HttpClientUtil.send(client, url, HttpMethods.GET,
headers);
                System.out.println("请求结果内容长度："+ resp.length());

                System.out.println("\n#################################\n
");
        }
```

测试结果如下：



```
<terminated> HttpClientTest (1) [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe
--------简单方式调用（默认post）--------
INFO - 请求地址：http://tool.oschina.net/
请求结果内容长度：26270

####################################

--------加入header设置--------
INFO - 请求地址：http://blog.csdn.net/xiaoxian8023
请求结果内容长度：48424

####################################

--------代理设置（绕过证书验证）-------
INFO - 请求地址：https://www.facebook.com/
请求结果内容长度：55638

####################################

--------代理设置（自签名证书验证）+header+get方式-------
INFO - 请求地址：https://sso.tgb.com:8443/cas/logi
请求结果内容长度：5630

####################################
```

可以看到4次调用，都没有问题。

那么现在试试多线程调用吧。我定义一个数组，里面有20篇文章的地址。我启动20个线程的线程池来测试，写了一个20*50次的for循环，看看全部线程结束时有没有报错，能用多长时间：

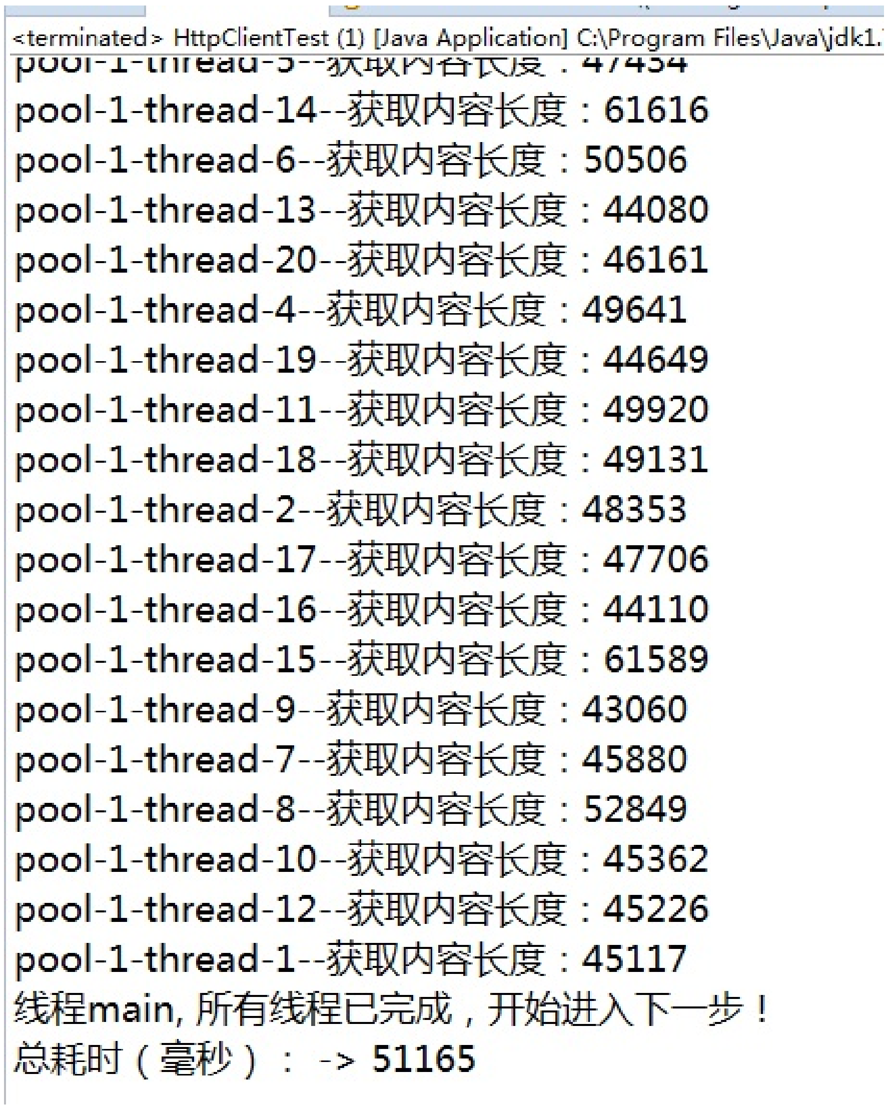
```
        public static void testMutilTask(){
                // URL列表数组
                String[] urls = {
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49862725",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49834643",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49834615",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49834589",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49785417",

                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48679609",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48681987",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48710653",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48729479",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48733249",

                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48806871",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/48826857",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49663643",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/49619777",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/47335659",

                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/47301245",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/47057573",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/45601347",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/45569441",
                                "http://blog.csdn.net/xiaoxian8023/articl
e/details/43312929",
                                };

                // 设置header信息
                Header[] headers = HttpHeader.custom().userAgent("Mozilla
/5.0").build();
                HttpClient client= HCB.custom().timeout(10000).build();

                 long start = System.currentTimeMillis();
                try {
                    int pagecount = urls.length;
                    ExecutorService executors = Executors.newFixedThreadP
ool(pagecount);
                    CountDownLatch countDownLatch = new CountDownLatch(pa
gecount*100);
                    for(int i = 0; i< pagecount*100;i++){
                        //启动线程抓取
                        executors.execute(new GetRunnable(urls[i%pagecoun
t], headers, countDownLatch).setClient(client));
                    }
                    countDownLatch.await();
                    executors.shutdown();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } finally {
                    System.out.println("线程" + Thread.currentThread().get
Name() + ", 所有线程已完成，开始进入下一步！");
                }

                long end = System.currentTimeMillis();
                System.out.println("总耗时（毫秒）： -> " + (end - start));
                //(7715+7705+7616)/3= 23 036/3= 7 678.66---150=51.2
                //(9564+8250+8038+7604+8401)/5=41 857/5=8 371.4--150
                //(9803+8244+8188+8378+8188)/5=42 801/5= 8 560.2---150
      }

       static class GetRunnable implements Runnable {
                private CountDownLatch countDownLatch;
                private String url;
                private Header[] headers;
                private HttpClient client = null;

                public GetRunnable setClient(HttpClient client){
                        this.client = client;
                        return this;
                }

                public GetRunnable(String url, Header[] headers,CountDown
Latch countDownLatch){
                        this.url = url;
                        this.headers = headers;
                    this.countDownLatch = countDownLatch;
                }
                @Override
                public void run() {
                    try {
                        String response = null;
                        if(client!=null){
                                response = HttpClientUtil.send(client, ur
l, headers);
                        }else{
                                response =  HttpClientUtil.send(url, head
ers);
                        }
                        System.out.println(Thread.currentThread().getName
()+"--获取内容长度："+response.length());
                    } catch (HttpProcessException e) {
                                        e.printStackTrace();
                                } finally {
                        countDownLatch.countDown();
                    }
                }
```

```
            }
```

定义了一个ExecutorService的线程池，使用CountDownLatch来保证所有线程都运行完毕，测试一下看看：

```
        public static void main(String[] args) throws Exception {
//              testOne();
                testMutilTask();
        }
```

测试结果如下：



```
<terminated> HttpClientTest (1) [Java Application] C:\Program Files\Java\jdk1.
pool-1-thread-5--获取内容长度：47434
pool-1-thread-14--获取内容长度：61616
pool-1-thread-6--获取内容长度：50506
pool-1-thread-13--获取内容长度：44080
pool-1-thread-20--获取内容长度：46161
pool-1-thread-4--获取内容长度：49641
pool-1-thread-19--获取内容长度：44649
pool-1-thread-11--获取内容长度：49920
pool-1-thread-18--获取内容长度：49131
pool-1-thread-2--获取内容长度：48353
pool-1-thread-17--获取内容长度：47706
pool-1-thread-16--获取内容长度：44110
pool-1-thread-15--获取内容长度：61589
pool-1-thread-9--获取内容长度：43060
pool-1-thread-7--获取内容长度：45880
pool-1-thread-8--获取内容长度：52849
pool-1-thread-10--获取内容长度：45362
pool-1-thread-12--获取内容长度：45226
pool-1-thread-1--获取内容长度：45117
线程main, 所有线程已完成，开始进入下一步！
总耗时（毫秒）： -> 51165
```

从结果中可以清楚的看到执行1000次调用，总消耗是51165，平均51ms/个，速度快，而且没有报错。

好了，本工具就分享到这里，下次会分享异步的HttpClient，敬请期待。
代码已上传至：https://github.com/Arronlong/httpclientUtil。

轻松把玩HttpAsyncClient之模拟post请求示例

版权声明：本文为博主原创文章，未经博主允许不得转载。如需转载请声明：【转自 http://blog.csdn.net/xiaoxian8023 】

如果看到过我前些天写过的《轻松把玩HttpClient之模拟post请求示例》这篇文章，你再看本文就是小菜一碟了，如果你顺便懂一些NIO，基本上是毫无压力了。因为HttpAsyncClient相对于HttpClient，就多了一个NIO，这也是为什么支持异步的原因。

不过我有一个疑问，虽说NIO是同步非阻塞IO，但是HttpAsyncClient提供了回调的机制，这点儿跟netty很像，所以可以模拟类似于AIO的效果。但是官网上的例子却基本上都是使用Future future = httpclient.execute(request, null);来同步获得执行结果。

好吧，反正我是用回调的方式实现的。代码基本上跟httpClient那篇一致。不一样的地方主要有这么2个地方：配置ssl时不一样；调用execute方式时，使用回调。具体代码如下：

```
package com.tgb.ccl.http.simpledemo;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

import org.apache.http.HttpEntity;
import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.ParseException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.concurrent.FutureCallback;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.conn.DefaultProxyRoutePlanner;
import org.apache.http.impl.nio.client.CloseableHttpAsyncClient;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.impl.nio.client.HttpAsyncClients;
import org.apache.http.impl.nio.conn.PoolingNHttpClientConnectionManager;
import org.apache.http.impl.nio.reactor.DefaultConnectingIOReactor;
import org.apache.http.impl.nio.reactor.IOReactorConfig;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.nio.conn.NoopIOSessionStrategy;
import org.apache.http.nio.conn.SchemeIOSessionStrategy;
import org.apache.http.nio.conn.ssl.SSLIOSessionStrategy;
import org.apache.http.nio.reactor.ConnectingIOReactor;
import org.apache.http.ssl.SSLContexts;
import org.apache.http.util.EntityUtils;

/**
 * HttpAsyncClient模拟post请求简单示例
 *
 * @author arron
 * @date 2015年11月1日 下午2:23:18
 * @version 1.0
 */
public class SimpleHttpAsyncClientDemo {

        /**
         * 设置信任自定义的证书
         *
         * @param keyStorePath          密钥库路径
         * @param keyStorepass          密钥库密码
         * @return
         */
        public static SSLContext custom(String keyStorePath, String keySt
orepass) {
                SSLContext sc = null;
                FileInputStream instream = null;
                KeyStore trustStore = null;
                try {
                        trustStore = KeyStore.getInstance(KeyStore.getDef
aultType());
                        instream = new FileInputStream(new File(keyStoreP
ath));
                        trustStore.load(instream, keyStorepass.toCharArra
y());
                        // 相信自己的CA和所有自签名的证书
                        sc = SSLContexts.custom().loadTrustMaterial(trust
Store, new TrustSelfSignedStrategy()).build();
                } catch (KeyStoreException | NoSuchAlgorithmException| Ce
rtificateException | IOException | KeyManagementException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                instream.close();
                        } catch (IOException e) {
                        }
                }
                return sc;
        }

        /**
         * 绕过验证
         *
         * @return
         * @throws NoSuchAlgorithmException
         * @throws KeyManagementException
         */
        public static SSLContext createIgnoreVerifySSL() throws NoSuchAlg
orithmException, KeyManagementException {
                SSLContext sc = SSLContext.getInstance("SSLv3");

                // 实现一个X509TrustManager接口，用于绕过验证，不用修改里面的方法
                X509TrustManager trustManager = new X509TrustManager() {
                        @Override
                        public void checkClientTrusted(
                                        java.security.cert.X509Certificat
e[] paramArrayOfX509Certificate,
                                        String paramString) throws Certif
icateException {
                        }

                        @Override
                        public void checkServerTrusted(
                                        java.security.cert.X509Certificat
e[] paramArrayOfX509Certificate,
                                        String paramString) throws Certif
icateException {
                        }

                        @Override
                        public java.security.cert.X509Certificate[] getAc
ceptedIssuers() {
                                return null;
                        }
                };
                sc.init(null, new TrustManager[] { trustManager }, null);
                return sc;
        }

        /**
         * 设置代理
         * @param builder
         * @param hostOrIP
         * @param port
         */
        public static HttpAsyncClientBuilder proxy(String hostOrIP, int p
ort){
                // 依次是代理地址，代理端口号，协议类型
                HttpHost proxy = new HttpHost(hostOrIP, port, "http");
                DefaultProxyRoutePlanner routePlanner = new DefaultProxyR
outePlanner(proxy);
                return HttpAsyncClients.custom().setRoutePlanner(routePla
nner);
        }

        /**
         * 模拟请求
         *
         * @param url                           资源地址
         * @param map                   参数列表
         * @param encoding      编码
         * @param handler               结果处理类
         * @return
         * @throws NoSuchAlgorithmException
         * @throws KeyManagementException
         * @throws IOException
         * @throws ClientProtocolException
         */
        public static void send(String url, Map<String,String> map,final
String encoding, final AsyncHandler handler) throws KeyManagementExceptio
n, NoSuchAlgorithmException, ClientProtocolException, IOException {

                //绕过证书验证，处理https请求
                SSLContext sslcontext = createIgnoreVerifySSL();

        // 设置协议http和https对应的处理socket链接工厂的对象
                Registry<SchemeIOSessionStrategy> sessionStrategyRegistry
 = RegistryBuilder.<SchemeIOSessionStrategy>create()
                .register("http", NoopIOSessionStrategy.INSTANCE)
                .register("https", new SSLIOSessionStrategy(sslcontext))
                .build();
                //配置io线程
        IOReactorConfig ioReactorConfig = IOReactorConfig.custom().setIoT
hreadCount(Runtime.getRuntime().availableProcessors()).build();
                //设置连接池大小
        ConnectingIOReactor ioReactor;
                ioReactor = new DefaultConnectingIOReactor(ioReactorConfi
g);
        PoolingNHttpClientConnectionManager connManager = new PoolingNHtt
pClientConnectionManager(ioReactor, null, sessionStrategyRegistry, null);

        //创建自定义的httpclient对象
                final CloseableHttpAsyncClient client = proxy("127.0.0.1"
, 8087).setConnectionManager(connManager).build();
//              CloseableHttpAsyncClient client = HttpAsyncClients.create
Default();

                //创建post方式请求对象
                HttpPost httpPost = new HttpPost(url);

                //装填参数
                List<NameValuePair> nvps = new ArrayList<NameValuePair>()
;
                if(map!=null){
                        for (Entry<String, String> entry : map.entrySet()
) {
                                nvps.add(new BasicNameValuePair(entry.get
Key(), entry.getValue()));
                        }
                }
                //设置参数到请求对象中
                httpPost.setEntity(new UrlEncodedFormEntity(nvps, encodin
g));

                System.out.println("请求地址："+url);
                System.out.println("请求参数："+nvps.toString());

                //设置header信息
                //指定报文头【Content-type】、【User-Agent】
                httpPost.setHeader("Content-type", "application/x-www-for
m-urlencoded");
                httpPost.setHeader("User-Agent", "Mozilla/4.0 (compatible
; MSIE 5.0; Windows NT; DigExt)");

                // Start the client
                client.start();
                //执行请求操作，并拿到结果（异步）
                client.execute(httpPost, new FutureCallback<HttpResponse>
() {

                        @Override
                        public void failed(Exception ex) {
                                handler.failed(ex);
                                close(client);
                        }

                        @Override
                        public void completed(HttpResponse resp) {
                                String body="";
                                //这里使用EntityUtils.toString()方式时会大概
率报错，原因：未接受完毕，链接已关
                                try {
                                        HttpEntity entity = resp.getEntit
y();
                                        if (entity != null) {
                                                final InputStream instrea
m = entity.getContent();
                                                try {
                                                        final StringBuild
er sb = new StringBuilder();
                                                        final char[] tmp
= new char[1024];
                                                        final Reader read
er = new InputStreamReader(instream,encoding);
                                                        int l;
                                                        while ((l = reade
r.read(tmp)) != -1) {
                                                                sb.append
(tmp, 0, l);
                                                        }
                                                        body = sb.toStrin
g();
                                                } finally {
                                                        instream.close();
                                                        EntityUtils.consu
me(entity);
                                                }
                                        }
                                } catch (ParseException | IOException e)
{
                                        e.printStackTrace();
                                }
                                handler.completed(body);
                                close(client);
                        }

                        @Override
                        public void cancelled() {
                                handler.cancelled();
                                close(client);
                        }
                });
        }

        /**
         * 关闭client对象
         *
         * @param client
         */
        private static void close(CloseableHttpAsyncClient client) {
                try {
                        client.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        static class AsyncHandler implements IHandler{

                @Override
                public Object failed(Exception e) {
                        System.err.println(Thread.currentThread().getName
()+"--失败了--"+e.getClass().getName()+"--"+e.getMessage());
                        return null;
                }
                @Override
                public Object completed(String respBody) {
                        System.out.println(Thread.currentThread().getName
()+"--获取内容："+respBody);
                        return null;
                }
                @Override
                public Object cancelled() {
                        System.out.println(Thread.currentThread().getName
()+"--取消了");
                        return null;
                }
```

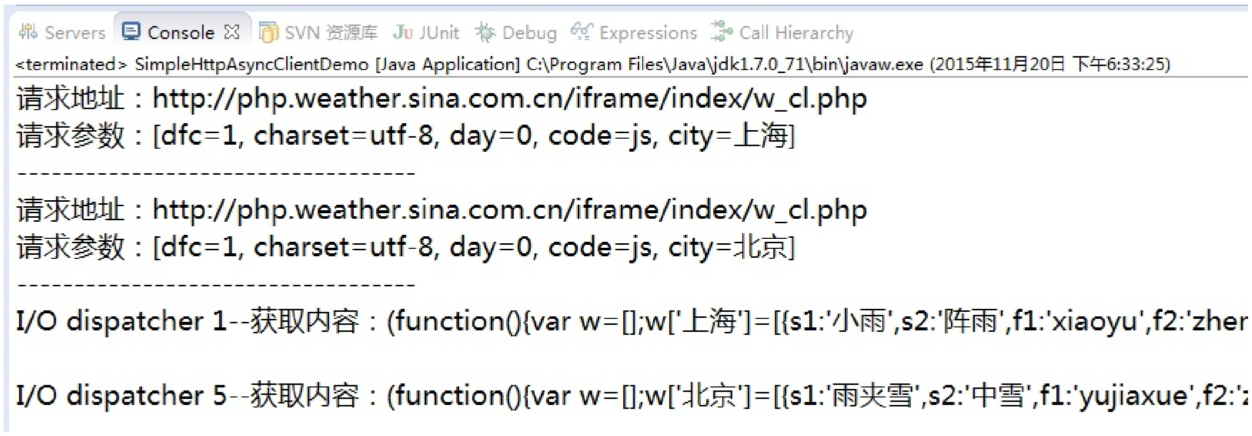
```
        }

        /**
         * 回调处理接口
         *
         * @author arron
         * @date 2015年11月10日 上午10:05:40
         * @version 1.0
         */
        public interface IHandler {

                /**
                 * 处理异常时，执行该方法
                 * @return
                 */
                Object failed(Exception e);

                /**
                 * 处理正常时，执行该方法
                 * @return
                 */
                Object completed(String respBody);

                /**
                 * 处理取消时，执行该方法
                 * @return
                 */
                Object cancelled();
        }

}
```

来一个测试类：

```
        public static void main(String[] args) throws KeyManagementExcept
ion, NoSuchAlgorithmException, ClientProtocolException, IOException {
                AsyncHandler handler = new AsyncHandler();
                String url = "http://php.weather.sina.com.cn/iframe/index
/w_cl.php";
                Map<String, String> map = new HashMap<String, String>();
                map.put("code", "js");
                map.put("day", "0");
                map.put("city", "上海");
                map.put("dfc", "1");
                map.put("charset", "utf-8");
                send(url, map, "utf-8", handler);

                System.out.println("-----------------------------------")
;

                map.put("city", "北京");
                send(url, map, "utf-8", handler);
```

```
                    System.out.println("-----------------------------------")
;
          }
```

测试结果如下：



```
<terminated> SimpleHttpAsyncClientDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2015年11月20日 下午6:33:25)
请求地址：http://php.weather.sina.com.cn/iframe/index/w_cl.php
请求参数：[dfc=1, charset=utf-8, day=0, code=js, city=上海]
-----------------------------------
请求地址：http://php.weather.sina.com.cn/iframe/index/w_cl.php
请求参数：[dfc=1, charset=utf-8, day=0, code=js, city=北京]
-----------------------------------
I/O dispatcher 1--获取内容：(function(){var w=[];w['上海']=[{s1:'小雨',s2:'阵雨',f1:'xiaoyu',f2:'zher

I/O dispatcher 5--获取内容：(function(){var w=[];w['北京']=[{s1:'雨夹雪',s2:'中雪',f1:'yujiaxue',f2:'z
```

很简单吧，其实基于HttpAsyncClient的工具类我也进行了封装，跟HttpClient工具类差不多。代码都已经提交至：https://github.com/Arronlong/httpclientUtil。有兴趣的自行下载，博客中就不再分享了。

轻松把玩HttpClient之封装HttpClient工具类(五)，携带Cookie的请求

最近更新了一下HttpClientUtil工具类代码，主要是添加了一个参数HttpContext，这个是用来干嘛的呢？其实是用来保存和传递Cookie所需要的。因为我们有很多时候都需要登录，然后才能请求一些想要的数据。而在这以前使用HttpClientUtil工具类，还不能办到。现在更新了以后，终于可以了。

先说一下思路：本次的demo，就是获取csdn中的c币，要想获取c币，必须先登录。而每次登录需要5个参数。其中2个必不可少的参数是用户名和密码，其他的3个参数，是需要从登录页面获取的。在第一次请求登录页面，只要设置了CookieStore，就可以自动获取cookie了，然后从返回的html源码中获取参数，再组装添加用户名密码，然后第二次登录，如果返回的html源码中有“帐号登录”这几个字，就说明登录失败了。否则登录成功。可以打印一下cookie（已注释）。然后再访问c币查询的页面，就可以从返回的html源码中解析到c币的值了。登录时需要注意的是：直接提交用户名密码或者第二次登录不携带context参数，是不能登录成功的。

具体代码如下：

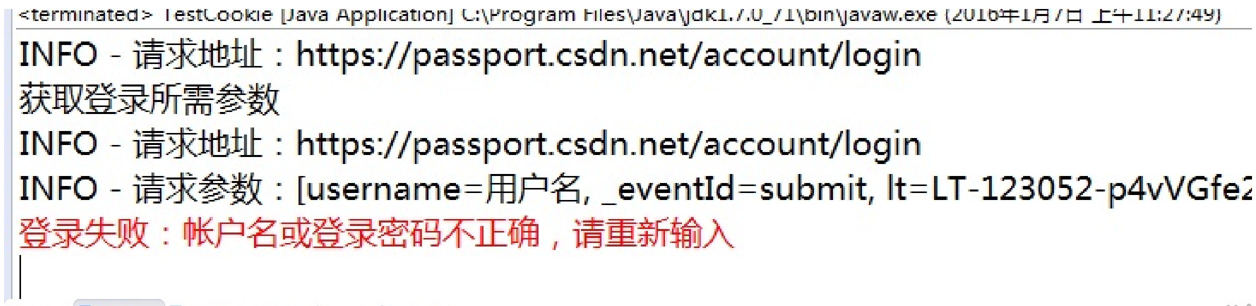
```
	public static void main(String[] args) throws HttpProcessExceptio
n {
		//登录地址
		String loginUrl = "https://passport.csdn.net/account/logi
n";
		//C币查询
		String scoreUrl = "http://my.csdn.net/my/score";

		HttpClientContext context = new HttpClientContext();
		CookieStore cookieStore = new BasicCookieStore();
		context.setCookieStore(cookieStore);
		//获取参数
		String loginform = HttpClientUtil.send(loginUrl, context)
;
//		System.out.println(loginform);
		System.out.println("获取登录所需参数");
		String lt = regex("\"lt\" value=\"([^\"]*)\"", loginform)
[0];
		String execution = regex("\"execution\" value=\"([^\"]*)\
"", loginform)[0];
```

```
                String _eventId = regex("\"_eventId\" value=\"([^\"]*)\""
, loginform)[0];

                //组装参数
                Map<String, Object> map = new HashMap<String, Object>();
                map.put("username", "用户名");
                map.put("password", "密码");
                map.put("lt", lt);
                map.put("execution", execution);
                map.put("_eventId", _eventId);

                //发送登录请求
                String result = HttpClientUtil.send(loginUrl, map, contex
t);
//              System.out.println(result);
                if(result.contains("帐号登录")){//如果有帐号登录，则说明未登录
成功
                        String errmsg = regex("\"error-message\">([^<]*)<
", result)[0];
                        System.err.println("登录失败："+errmsg);
                        return;
                }
                System.out.println("----登录成功----");

//              //打印参数，可以看到cookie里已经有值了。
//              cookieStore = context.getCookieStore();
//              for (Cookie cookie : cookieStore.getCookies()) {
//                      System.out.println(cookie.getName()+"--"+cookie.g
etValue());
//              }

                //访问积分管理页面
                Header[] headers = HttpHeader.custom().userAgent("Mozilla
/5.0").build();
                result = HttpClientUtil.send(scoreUrl, headers, context);
                //获取C币
                String score = regex("\"last-img\"><span>([^<]*)<", resul
t)[0];
                System.out.println("您当前有C币："+score);

        }
```

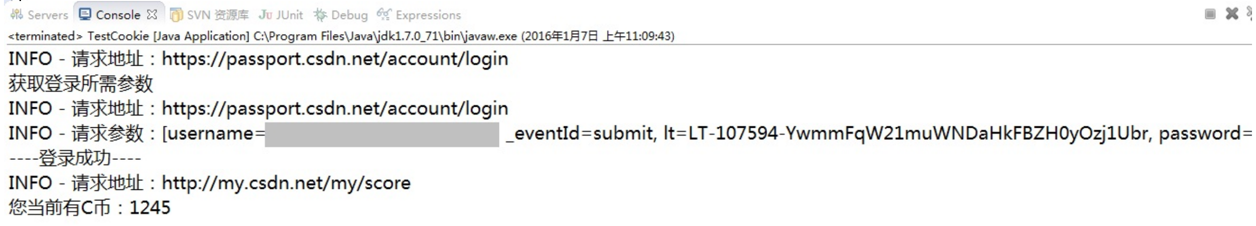
从html源码中解析参数和c币值所用到的一个方法：

```
        /**
         * 通过正则表达式获取内容
         *
         * @param regex          正则表达式
         * @param from           原字符串
         * @return
         */
        public static String[] regex(String regex, String from){
                Pattern pattern = Pattern.compile(regex);
                Matcher matcher = pattern.matcher(from);
                List<String> results = new ArrayList<String>();
                while(matcher.find()){
                        for (int i = 0; i < matcher.groupCount(); i++) {
                                results.add(matcher.group(i+1));
                        }
                }
                return results.toArray(new String[]{});
        }
```

测试结果：



```
<terminated> TestCookie [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2016年1月7日 上午11:27:49)
INFO - 请求地址：https://passport.csdn.net/account/login
获取登录所需参数
INFO - 请求地址：https://passport.csdn.net/account/login
INFO - 请求参数：[username=用户名, _eventId=submit, lt=LT-123052-p4vVGfe2
登录失败：帐户名或登录密码不正确，请重新输入
```



```
<terminated> TestCookie [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (2016年1月7日 上午11:09:43)
INFO - 请求地址：https://passport.csdn.net/account/login
获取登录所需参数
INFO - 请求地址：https://passport.csdn.net/account/login
INFO - 请求参数：[username=[redacted] _eventId=submit, lt=LT-107594-YwmmFqW21muWNDaHkFBZH0yOzj1Ubr, password=
----登录成功----
INFO - 请求地址：http://my.csdn.net/my/score
您当前有C币：1245
```

最重要的就是context这个参数了，给它设置了cookiestore，那么会在每次请求时将cookie带入请求中。或者也可以在header中手动设置cookie参数，也是可以做到的。

轻松把玩HttpClient之封装HttpClient工具类(六)，封装输入参数，简化工具类

在写这个工具类的时候发现传入的参数太多，以至于方法泛滥，只一个send方法就有30多个，所以对工具类进行了优化，把输入参数封装在一个对象里，这样以后再扩展输入参数，直接修改这个类就ok了。

不多说了，先上代码：

```
/**
 * 请求配置类
 *
 * @author arron
 * @date 2016年2月2日 下午3:14:32
 * @version 1.0
 */
public class HttpConfig {

        private HttpConfig(){};

        /**
         * 获取实例
         * @return
         */
        public static HttpConfig custom(){
                return new HttpConfig();
        }

        /**
         * HttpClient对象
         */
        private HttpClient client;

        /**
         * CloseableHttpAsyncClient对象
         */
        private CloseableHttpAsyncClient asynclient;

        /**
         * 资源url
         */
        private String url;

        /**
         * Header头信息
         */
        private Header[] headers;

        /**
         * 请求方法
         */
        private HttpMethods method=HttpMethods.GET;

        /**
         * 请求方法名称
         */
        private String methodName;

        /**
         * 用于cookie操作
         */
        private HttpContext context;

        /**
         * 传递参数
         */
        private Map<String, Object> map;

        /**
         * 输入输出编码
         */
        private String encoding=Charset.defaultCharset().displayName();

        /**
         * 输入编码
         */
        private String inenc;

        /**
         * 输出编码
         */
        private String outenc;

        /**
         * 输出流对象
         */
        private OutputStream out;

        /**
         * 异步操作回调执行器
         */
        private IHandler handler;

        /**
         * HttpClient对象
         */
        public HttpConfig client(HttpClient client) {
                this.client = client;
                return this;
        }

        /**
```

```
         * CloseableHttpAsyncClient对象
         */
        public HttpConfig asynclient(CloseableHttpAsyncClient asynclient)
 {
                this.asynclient = asynclient;
                return this;
        }

        /**
         * 资源url
         */
        public HttpConfig url(String url) {
                this.url = url;
                return this;
        }

        /**
         * Header头信息
         */
        public HttpConfig headers(Header[] headers) {
                this.headers = headers;
                return this;
        }

        /**
         * 请求方法
         */
        public HttpConfig method(HttpMethods method) {
                this.method = method;
                return this;
        }

        /**
         * 请求方法
         */
        public HttpConfig methodName(String methodName) {
                this.methodName = methodName;
                return this;
        }

        /**
         * cookie操作相关
         */
        public HttpConfig context(HttpContext context) {
                this.context = context;
                return this;
        }

        /**
         * 传递参数
         */
        public HttpConfig map(Map<String, Object> map) {
                this.map = map;
                return this;
        }
        /**
         * 输入输出编码
         */
        public HttpConfig encoding(String encoding) {
                //设置输入输出
                inenc(encoding);
                outenc(encoding);
                this.encoding = encoding;
                return this;
        }

        /**
         * 输入编码
         */
        public HttpConfig inenc(String inenc) {
                this.inenc = inenc;
                return this;
        }

        /**
         * 输出编码
         */
        public HttpConfig outenc(String outenc) {
                this.outenc = outenc;
                return this;
        }

        /**
         * 输出流对象
         */
        public HttpConfig out(OutputStream out) {
                this.out = out;
                return this;
        }

        /**
         * 异步操作回调执行器
         */
        public HttpConfig handler(IHandler handler) {
                this.handler = handler;
                return this;
        }


        public HttpClient client() {
                return client;
        }

        public CloseableHttpAsyncClient asynclient() {
                return asynclient;
        }

        public Header[] headers() {
                return headers;
        }
```

```
		public String url() {
			return url;
		}

		public HttpMethods method() {
			return method;
		}

		public String methodName() {
			return methodName;
		}

		public HttpContext context() {
			return context;
		}

		public Map<String, Object> map() {
			return map;
		}

		public String encoding() {
			return encoding;
		}

		public String inenc() {
			return inenc == null ? encoding : inenc;
		}

		public String outenc() {
			return outenc == null ? encoding : outenc;
		}

		public OutputStream out() {
			return out;
		}

		public IHandler handler() {
			return handler;
		}
}
```

将构造方法设置为private，然后提供一个custom()方法来获取新的实例，所有的set方法，都是返回HttpConfig，这样就支持链式调用（创建者模式）了。

工具类的核心方法如下：

```

		/**
		 * 请求资源或服务
		 *
		 * @param config
		 * @return
		 * @throws HttpProcessException
		 */
```

```
        public static String send(HttpConfig config) throws HttpProcessEx
ception {
                return fmt2String(execute(config), config.outenc());
        }

        /**
         * 请求资源或服务
         *
         * @param client                                                client对象
         * @param url                                                   资源地址
         * @param httpMethod    请求方法
         * @param parasMap              请求参数
         * @param headers                       请求头信息
         * @param encoding              编码
         * @return                                                              返回处理结
果
         * @throws HttpProcessException
         */
        private static HttpResponse execute(HttpConfig config) throws Htt
pProcessException {
                if(config.client()==null){//检测是否设置了client
                        config.client(create(config.url()));
                }
                HttpResponse resp = null;
                try {
                        //创建请求对象
                        HttpRequestBase request = getRequest(config.url()
, config.method());

                        //设置header信息
                        request.setHeaders(config.headers());

                        //判断是否支持设置entity(仅HttpPost、HttpPut、HttpPat
ch支持)
                        if(HttpEntityEnclosingRequestBase.class.isAssigna
bleFrom(request.getClass())){
                                List<NameValuePair> nvps = new ArrayList<
NameValuePair>();

                                //检测url中是否存在参数
                                config.url(Utils.checkHasParas(config.url
(), nvps, config.inenc()));

                                //装填参数
                                HttpEntity entity = Utils.map2List(nvps,
config.map(), config.inenc());

                                //设置参数到请求对象中
                                ((HttpEntityEnclosingRequestBase)request)
.setEntity(entity);

                                logger.info("请求地址："+config.url());
                                if(nvps.size()>0){
                                        logger.info("请求参数："+nvps.toStr
ing());
                                }
                        }else{
                                int idx = config.url().indexOf("?");
                                logger.info("请求地址："+config.url().subst
ring(0, (idx>0 ? idx : config.url().length())));
                                if(idx>0){
                                        logger.info("请求参数："+config.url
().substring(idx+1));
                                }
                        }
                        //执行请求操作，并拿到结果（同步阻塞）
                        resp = (config.context()==null)?config.client().e
xecute(request) : config.client().execute(request, config.context()) ;

                        //获取结果实体
                        return resp;

                } catch (IOException e) {
                        throw new HttpProcessException(e);
                }
        }

        //-----------华----丽----分----割----线--------------
        //-----------华----丽----分----割----线--------------
        //-----------华----丽----分----割----线--------------

        /**
         * 转化为字符串
         *
         * @param entity                        实体
         * @param encoding      编码
         * @return
         * @throws HttpProcessException
         */
        public static String fmt2String(HttpResponse resp, String encodin
g) throws HttpProcessException {
                String body = "";
                try {
                        if (resp.getEntity() != null) {
                                // 按指定编码转换结果实体为String类型
                                body = EntityUtils.toString(resp.getEntit
y(), encoding);

                                logger.debug(body);
                        }
                        EntityUtils.consume(resp.getEntity());
                } catch (ParseException | IOException e) {
                        throw new HttpProcessException(e);
                }finally{
                        close(resp);
                }
                return body;
        }

        /**
         * 转化为流
         *
         * @param entity                        实体
```

```
	 * @param out				输出流
	 * @return
	 * @throws HttpProcessException
	 */
	public static OutputStream fmt2Stream(HttpResponse resp, OutputSt
ream out) throws HttpProcessException {
		try {
			resp.getEntity().writeTo(out);
			EntityUtils.consume(resp.getEntity());
		} catch (ParseException | IOException e) {
			throw new HttpProcessException(e);
		}finally{
			close(resp);
		}
		return out;
	}
```

再附上测试代码：

```
public static void testOne() throws HttpProcessException{
		System.out.println("--------简单方式调用（默认post）--------
");
		String url = "http://tool.oschina.net/";
		HttpConfig  config = HttpConfig.custom();
		//简单调用
		String resp = HttpClientUtil.get(config.url(url));

		System.out.println("请求结果内容长度："+ resp.length());

		System.out.println("\n#################################\n
");

		System.out.println("--------加入header设置--------");
		url="http://blog.csdn.net/xiaoxian8023";
		//设置header信息
		Header[] headers=HttpHeader.custom().userAgent("Mozilla/5
.0").build();
		//执行请求
		resp = HttpClientUtil.get(config.headers(headers));
		System.out.println("请求结果内容长度："+ resp.length());

		System.out.println("\n#################################\n
");

		System.out.println("--------代理设置（绕过证书验证）-------")
;
		url="https://www.facebook.com/";
		HttpClient client= HCB.custom().timeout(10000).proxy("127
.0.0.1", 8087).ssl().build();//采用默认方式（绕过证书验证）
		//执行请求
		resp = HttpClientUtil.get(config.client(client));
		System.out.println("请求结果内容长度："+ resp.length());
```

```
                System.out.println("\n####################################\n
");

//              System.out.println("--------代理设置（自签名证书验证）+header
+get方式-------");
//              url = "https://sso.tgb.com:8443/cas/login";
//              client= HCB.custom().timeout(10000).ssl("D:\\keys\\wsriak
ey","tomcat").build();
//              headers=HttpHeader.custom().keepAlive("false").connection
("close").contentType(Headers.APP_FORM_URLENCODED).build();
//              //执行请求
//              resp = CopyOfHttpClientUtil.get(config.method(HttpMethods
.GET));
//              System.out.println("请求结果内容长度："+ resp.length());
                try {
                        System.out.println("--------下载测试-------");
                        url="http://ss.bdimg.com/static/superman/img/logo
/logo_white_fe6da1ec.png";
                        FileOutputStream out = new FileOutputStream(new F
ile("d://aaa//000.png"));
                        HttpClientUtil.down(HttpConfig.custom().url(url).
out(out));
                        out.flush();
                        out.close();
                        System.out.println("--------下载测试+代理-------");

                        out = new FileOutputStream(new File("d://aaa//001
.png"));
                        HttpClientUtil.down(HttpConfig.custom().client(cl
ient).url(url).out(out));
                        out.flush();
                        out.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }

                System.out.println("\n####################################\n
");
        }
```

可以看到这样调用会更显得清晰明了。以后再添加功能时，改起来也会比较方便了。工具类也提供了输出流的功能，可以用于下载文件或者加载验证码图片，非常方便。

最新的完整代码请到GitHub上进行下载：https://github.com/Arronlong/httpclientUtil 。