链滴

基于apache httpclient的http访问工具

作者：niuhaipeng
原文链接：https://ld246.com/article/1451138475907
来源网站：链滴

代码是对apache httpclient的简单封装，支持用户登录，用方式如下
import org.junit.Test;

/**
 * @author niuhaipeng
 *
 * @date 2015年12月24日
 */
public class HttpClientTest {

    @Test
    public void testGet2String() {
        SimpleHttpClient client = new SimpleHttpClient();
        String url = "http://niuhp.com";
        while (true) {
            String content = client.doGet2String(url);
 nbsp;   System.out.println(content);
nbsp;   try {
   &bsp;   Thread.sleep(10);
            } catch (InterruptedException e) {
             &nbs; e.printStackTrace();
           }
        }
    }
}

实现代码
import java.io.IOException;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

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

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.htt.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.config.AuthSchemes;
import org.apache.http.client.config.CookieSpecs;
import org.apache.http.client.config.RequestConfig;<r />import org.apache.http.client.config.RequestConfig.Builder;
import org.apache.http.lient.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methos.HttpRequestBase;
import org.apache.http.client.methods.HttpUriRequest;
impor org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
iport org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.con.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnamVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import orgapache.http.entity.ContentType;
import org.apache.http.impl.client.HttpClientBuilder;<b />import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.message.BasicNameValuePai;
import org.apache.http.util.EntityUtils;

import com.niuhp.toolbox.logadaper.LogX;
import com.niuhp.toolbox.util.IoUtil;

/**
 * @author nuhaipeng
 *
 * @date 2015年12月17日
 */
public cass SimpleHttpClient {

    private HttpClient httpClient;
    private Object lockObj = new Object();

   &nbp;private int connectionRequestTimeout = 10000;
    private int conectTimeout = 10000;
    private int defaultMaxPerRoute = 20;
    private int maxTotal = 100;
    /**
&nbs;   * 请求自动关闭时间，防止调用者长期持有请求
     */<b />   private long autoCloseTimeout = 3000;
     private final ExecutorService executorService = Executors.newFixedThreadPool(50);

    private void checkInit(boolean https) {
    &bsp;   if (httpClient != null) {
        nbsp;   return;
        }
         synchronized (lockObj) {
    &

bsp;   if (httpClient != null) {
               return;
            }
        Builder builder = RequestConfig.custom();
            builder.setCookieSpec(CookieSpecs.STANDARD_STRICT);
            builder.setConnectionRequestTimeout(connectionRequestTimeout);
           builder.setConnectTimeout(connectTimeout);
            if (https) {
                builder.setExpectContinueEnabled(true);
                Collection<String> targetPreferredAuthSchemes = Arrays.asList(AuthSchemes.NTLM, AuthSchemes.DIGEST);
            builder.setTargetPreferredAuthSchemes(targePreferredAuthSchemes);
                Collection<String> proxyPreferredAuthSchemes = Arrays.asList(AuthSchemes.BASIC);
                builder.setProxyPreferredAuthSchemes(proxyPreferredAuthSchemes)
            }
            RequestConfig config = builder.build()

            HttpClientBuider httpClientBuilder = HttpClients.custom();
            httpClientBuilder.setDefaultRequestConfig(config);
            if (https) {
                RegistryBuilder<ConnectonSocketFactory> registryBuilder = RegistryBuilder.create();
               registryBuilder.register("http, PlainConnectionSocketFactory.INSTANCE);
               TrustManager trustManager = new Customized509TrustManager();

                try {
                    SSLContext context = SSLContext.getnstance("TLS");
                    context.init(null, new TrustManager[] { trustManager }, null);
                    ConnectionSocketFactory connectionSocketFactory = new SSLCnnectionSocketFactory(context,
                            NoopHostnameVerifier.INSTANCE);
                    registryBuilder.register("http", connectionSocketFactory);
                    } catch (Exception e) {
     &nbs;   LogX.getLogX(getlass()).error("register connectionSocketFactory error", e);
                }
    &nbs;   Registry<ConnectionSocketFactory> registry = registryBuilder.build();
                    PoolingHttpClientConnectionManager connManaer = new PoolingHttpClientConnectionManager(registry);
                 connManager.setMaxTotal(maxTotal);
                    connManager.setDefaultMaxPerRoute(defaultMaxPerRoute);
   &nbs;   httpClientBuilder.setConnctionManager(connManager);
         &nb

p; }
            httpClient = httpClientBuilder.build();
        }
  nbsp; }

    public String doGet2String(String url) {
        return doGet2String(url, null);
 &n sp; }

    public String doGet2String(String url, Map< tring, String> paraMap) {
        HttpResponse eponse = doGet(url, paraMap);
        return get ontent(reponse);
    }

    public Http esponse doGet(String url, Map<String, String> paraMap) {
     nbsp;   StringBuilder urlBuilder = new StringBuilder();
   &nbs ;   urlBuilder.append(url);
       &nb p;if (paraMap != null && !paraMap.isEmpty()) {
      nbsp;   urlBuilder.append("?");
             Set<String> keys = paraMap.keySet();
             Iterator<String> it = keys.ite ator();
            String key = it next();
            urlBuilder.ap end(key).append("=").append(paraMap.get(key));
            while (it.hasNext()) {
                key = it.next();
   &nbs ;   urlBuilder.append("&).append(key).append("=").append(paraMap.get(key));
     & bsp;   }
        }
         return doGet(urlBuilder.toString());
     }

    public HttpResponse doGet(String url) {< r />   boolean https = url != null && url.s artsWith("https");
        checkInit(https);
& bsp;   HttpGet get = new HttpGet(url);
        return doRequest(httpClient, get);

    public String doPost2String(String url) {
 &nb p;   return doPost2String(url, null);
    } br />

    public String doPost2String(String url, Map<String, Strin > paraMap) {
        HttpResponse response = doPost(url, paraMap);
        return getContent(re ponse);
    }

    public HttpResponse doPost(String url) {
        return doPost(url, null)
    }

    public HttpResponse doPost String url, Map<String, String> paraMap) {
       nbsp;boolean https = url != null && url.startsWith("https");
   & bsp;   checkInit(https);
         HttpPost post = new HttpPost(url);
        List&lt NameValuePair> values = toValues(paraMap);
        if (values != null && !values.isEmpty()) {
      nbsp;   HttpEntity entity = new UrlEncodedFormEntity(values, Co sts.UTF_8);
            post.setE tity(entity);
        }
     nbsp;   return doRequest(httpClient, post);
    }
<r />
    private void autoClose(HttpUriRequest request, HttpResponse repo se) {
        Runnable runnable = new Runnable()

            @Override< r />   public void run() {
                try {
                  &nsp;   Thread.sleep(autoCloseTimeout);

           } catch (InterruptedException e) {
                   LogX.getLogX(getClass()).error("sleep error", e);
                }
                if (request instanceof HttpRequestBase) {
                    HttpRequestBase httpRequestBase = (HttpRequetBase) request;
                   httpRequestBase.releaseConnection();
                }
                IoUtil.closeHttpResponse(reponse);
            LogX.getLogX(getClass()).info(String.format("autoClose request[%s] and reponse[%s]", request, reponse));
            }
        };

        executorService.execute(runnable);
    }

    private HttpResponse doRequest(HttpClient httpClient, HttpUriRequest httpRequest) {
        synchronized (httpClient) {
            try {
                HttpResponse response = httpClient.execute(httpRequest);
                autoClose(httpRequest, response);
                return response;
            } catch (Exception e) {
            LogX.getLogX(getClass()).error("execute error", e);
            }
        }
        return null;
  }

    public void closeHttpClient() {
        IoUtil.closeHttpClient(httpClient);
    }

    private static List<NameValuePair> toValues(Map<String, String> paraMap) {
        if (paraMap == null) {
            return null;
        }
        List<NameValuePair> values = new ArrayList<NameValuePair>(paraMap.size());
        Set<String> keys = paraMap.keySet();
        for (String key : keys) {
            NameValuePair pair = new BasicNameValuePair(key, paraMap.get(key));
            values.add(pair);
        }
        return values;
    }

    private String getContent(HttpResponse response) {
        if (response == null) {
            return null;
        }
        HttpEntity httpEntity = response.getEntity();
        if (httpEntity == null) {
            return null;
        }
        String content = null;

        try {
            Charset charset = ContentType.getOrDefault(httpEntity).getCharset();
            content = EntityUtils.toString(httpEntity, charset);
        } catch (IOException e) {
            LogX.getLogX(getClass()).error("getContent error", e);

    } finally {
         &nsp; IoUtil.closeHttpResponse(response);
       &bsp;}
        return content;
    }

    public int getConnectionRequestTimeout() {
        return connectionRequestTimeout;
     }

    public void setConnectionRequestTimeout int connectionRequestTimeout) {
        this.connctionRequestTimeout = connectionRequestTimeout;
    }

    public int getConnectTimeout() {
      nbsp; return connectTimeout;
    }

    public void setConnectTimeout(int connectTimeout) {
    &nsp;   this.connectTimeout = connectTimeout;
    }

    public int getDefaultMaxPerRoute() {
   &nbs;   return defaultMaxPerRoute;
    }

     public void setDefaultMaxPerRoute(int defaultMaxPerRoute) {
&bsp;   this.defaultMaxPerRoute = defaultMaxPerRoute;
    }

    public int getMaxTotal() {
         return maxTotal;
    }<b />
    public void setMaxTotal(int maxTotal) {
   nbsp;   this.maxTotal = maxTotal;
    }

    public long getAutoCloseTimeout() {
    &nsp;   return autoCloseTimeout;
    }

     public void setAutoCloseTimeout(long autoCloseTimeout) {
 &nbp;   if (autoCloseTimeout <= 0 || autoCloseTimeout > 5000)
            LogX.getLogX(getClss()).warn(String.format("autoCloseTimeout must be from %s to %s", 0, 5000));
 nbsp;   return;
         }
        this.autoCloseTimeout = autoCloseTimeout;
    }

}

import java.security.cert.CertificateException;
import java.security.cert.X509Certificae;

import javax.net.ssl.X509TrustManager;

/**
 * @auth r niuhaipeng
 *
 * @date 2015年12月17日
 */
public class CustomizedX509TrustManager implements X509TrustManager {

 nbsp; @Override
    public void checkClientTrusted(X509Certficate[] arg0, String arg1)
           &bsp;throws CertificateException {

    }

 &nsp; @Override
    public void checkServerTrusted(X509Certificate[] arg0, String arg1)
           &nbp;throws CertificateException {

    }

 &nbs ; @Override
    public X509Certificate[] getAcceptedIssuers()
        return null;
    }

}

maven配置
        <dependency>
   &bsp;   <groupId>org.apache.httpcomponent </groupId>
            &lt artifactId>httpclient</artifactId>
         nbsp;   <version>4.5.1</version>
         </dependency>
本文代码均可以从github下载， 地址&nbp;<span style= color: #0000ff;">https://github.com/niuhp/toolbox