链滴

使用 lambda 表达式对 HttpClient 进行封装

作者：xiaoting
原文链接：https://ld246.com/article/1507980589561
来源网站：链滴

功能点

- 1、使用jdk lambda 表达式对HttpClient进行封装
- 2、支持http和https

要求:HttpClient的版本为4.5.1

```
public class HttpClientUtil {
/**
* 日志
/
private static final Logger log = Logger.getLogger("lavasoft");
/*
* 请求编码
/
private static final String CHARSET_UTF_8 = "UTF-8";
/*
* json头
/
private static final String CONTENT_TYPE_JSON = "application/json";
/*
* xml请求
/
private static final String CONTENT_TYPE_XML = "text/xml";
/*
* 连接池
/
private static PoolingHttpClientConnectionManager connMgr;
/*
* 请求配置
*/
private static RequestConfig requestConfig;


/**
 * 超时时间
 */
private static final int MAX_TIMEOUT = 7000;

static {
    // 设置连接池
    connMgr = new PoolingHttpClientConnectionManager();
```

```
        // 设置连接池大小
        connMgr.setMaxTotal(100);
        connMgr.setDefaultMaxPerRoute(connMgr.getMaxTotal());
        /*校验链接*/
        connMgr.setValidateAfterInactivity(1000);
        RequestConfig.Builder configBuilder = RequestConfig.custom();
        // 设置连接超时
        configBuilder.setConnectTimeout(MAX_TIMEOUT);
        // 设置读取超时
        configBuilder.setSocketTimeout(MAX_TIMEOUT);
        // 设置从连接池获取连接实例的超时
        configBuilder.setConnectionRequestTimeout(MAX_TIMEOUT);
        requestConfig = configBuilder.build();
    }

    /**
     * 发送get请求
     *
     * @param url url
     * @return
     */
    public static String get(String url) {
        return doGet(url, new HashMap<>());
    }

    /**
     * 发送get请求
     *
     * @param url url
     * @return
     */
    public static String getHttps(String url) {
        return doGet(url, new HashMap<>(), false);
    }

    /**
     * 发送get请求 https
     *
     * @param url url
     * @return
     */
    public static String getHttps(String url, Map<String, Object> params) {
        return doGet(url, params, false);
    }

    public static String doGet(String url, Map<String, Object> params, boolean bl) {
        log.info("____请求参数:" + params.toString());
        String param = getStringBuffer(params);
        String finalApiUrl = url + param;
        return HttpClient.domain(httpClient -> {
            HttpGet httpGet = new HttpGet(finalApiUrl);
            return execute(httpClient, httpGet);
        }, bl);
    }
```

```java
/**
 * 发送 GET 请求（HTTP），K-V形式
 *
 * @param url
 * @param params
 * @return
 */
public static String doGet(String url, Map<String, Object> params) {
    return doGet(url, params, true);
}

private static String getStringBuffer(Map<String, Object> params) {
    if (MapUtil.isNotNull(params)) {
        StringBuffer param = new StringBuffer();
        int i = 0;
        for (String key : params.keySet()) {
            if (i == 0) {
                param.append("?");
            } else {
                param.append("&");
            }
            param.append(key).append("=").append(params.get(key));
            i++;
        }
        return param.toString();
    } else {
        return "";
    }
}

/**
 * 发送post请求
 *
 * @param url    post url
 * @param params post参数
 * @return
 */
public static String post(String url, Map<String, Object> params) {
    log.info("___请求参数:" + params.toString());
    return HttpClient.domain(httpClient -> {
        HttpPost httpPost = httpPostHandler(url, params);
        return execute(httpClient, httpPost);
    }, true);
}

/**
 * 发送post请求
 *
 * @param url post url
 * @return
 */
public static String post(String url) {
    return post(url, new HashMap<>());
```

```
}

/**
 * post json数据
 *
 * @param url
 * @param jsonStr
 * @return
 */
public static String postJson(String url, String jsonStr) {
    log.info("___请求参数:" + jsonStr);
    return HttpClient.domain(httpClient -> {
        HttpPost httpPost = new HttpPost(url);
        StringEntity stringEntity;
        try {
            stringEntity = new StringEntity(jsonStr);
        } catch (UnsupportedEncodingException e) {
            return null;
        }
        httpPost.setHeader("Content-Type", CONTENT_TYPE_JSON);
        httpPost.setEntity(stringEntity);
        return execute(httpClient, httpPost);
    }, true);
}

/**
 * 发送 SSL POST 请求（HTTPS），K-V形式
 *
 * @param apiUrl API接口URL
 * @param params 参数map
 * @return
 */
public static String doPostSSL(String url, Map<String, Object> params) {
    log.info("___请求参数:" + params.toString());
    return HttpClient.domain(httpClient -> {
        HttpPost httpPost = httpPostHandler(url, params);
        return execute(httpClient, httpPost);
    }, false);
}

/**
 * 发送 SSL POST 请求（HTTPS），JSON形式
 *
 * @param apiUrl API接口URL
 * @param json   JSON对象
 * @return
 */
public static String doPostSSL(String apiUrl, Object json) {
    log.info("___请求参数:" + json);
    return HttpClient.domain(httpClient -> {
        HttpPost httpPost = new HttpPost(apiUrl);
        StringEntity stringEntity;
        stringEntity = new StringEntity(json.toString(), CHARSET_UTF_8);
        stringEntity.setContentEncoding(CHARSET_UTF_8);
```

```
            httpPost.setHeader("Content-Type", CONTENT_TYPE_JSON);
            httpPost.setEntity(stringEntity);
            return execute(httpClient, httpPost);
        }, false);
    }


    private static HttpPost httpPostHandler(String url, Map<String, Object> params) {
        HttpPost httpPost = new HttpPost(url);
        List<NameValuePair> naps = new ArrayList<>();
        if (MapUtil.isNotNull(params)) {
            for (Map.Entry<String, Object> entry : params.entrySet()) {
                naps.add(new BasicNameValuePair(entry.getKey(), entry.getValue().toString()));
            }
        }
        try {
            httpPost.setEntity(new UrlEncodedFormEntity(naps, CHARSET_UTF_8));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return httpPost;
    }

    private static String execute(CloseableHttpClient httpClient, HttpRequestBase httpGetOrPost)

        String res = null;
        CloseableHttpResponse response = null;
        try {
            httpGetOrPost.setConfig(requestConfig);
            log.info("_______请求url:" + httpGetOrPost.getURI());
            response = httpClient.execute(httpGetOrPost);
            log.info("_______返回code:" + response.getStatusLine().getStatusCode());
            if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
                return null;
            }
            HttpEntity entity = response.getEntity();
            res = EntityUtils.toString(entity, CHARSET_UTF_8);
            log.info("_______返回值:" + res);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            doResponseClose(response);
        }
        return res;
    }

    private static void doHttpClientClose(CloseableHttpClient httpClient) {
        if (httpClient != null) {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
}

private static void doResponseClose(CloseableHttpResponse response) {
    if (response != null) {
        try {
            response.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public interface HttpClientInterface<T> {
    T domain(CloseableHttpClient httpClient);
}

static class HttpClient {
    /**
     * @param interfaces 具体的操作接口
     * @param bl         是否是http请求,false https true http
     * @param <T>        返回泛型
     * @return
     */
    public static <T extends Object> T domain(HttpClientInterface<T> interfaces, boolean bl) {
        // 返回值
        T object;
        CloseableHttpClient httpClient;
        if (bl) {
            httpClient = HttpClients.createDefault();
        } else {
            httpClient = HttpClients.custom().setSSLSocketFactory(createSSLConnSocketFactory()).
                    setConnectionManager(connMgr).setDefaultRequestConfig(requestConfig).
                    setConnectionManagerShared(true).build();
        }
        try {
            // 业务操作
            object = interfaces.domain(httpClient);
        } finally {
            doHttpClientClose(httpClient);
        }
        return object;
    }
}

/**
 * 创建SSL安全连接
 *
 * @return
 */
private static SSLConnectionSocketFactory createSSLConnSocketFactory() {
    SSLConnectionSocketFactory sslsf = null;
    try {
        SSLContext sslContext = new SSLContextBuilder().loadTrustMaterial(null, (chain, authTyp
) -> true).build();
```

```
            sslsf = new SSLConnectionSocketFactory(sslContext, (arg0, agr1) -> true);
        } catch (GeneralSecurityException e) {
            e.printStackTrace();
        }
        return sslsf;
    }

    }
```

原文链接：使用 lambda 表达式对 HttpClient 进行封装