

HttpClient详细使用示例

2018年07月14日 12:11:04 [justry_deng](#) 阅读数 45759 标签: [HttpClientJavaPOSTGET](#) [更多](#)

个人分类: [HTTP/HTTPS](#)

HTTP 协议可能是现在 Internet 上使用得最多、最重要的协议了，越来越多的 Java 应用程序需要直接通过 HTTP 协议来访问网络资源。虽然在 JDK 的 java net包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，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。

HTTP和浏览器有点像，但却不是浏览器。很多人觉得既然HttpClient是一个HTTP客户端编程工具，很多人把他当做浏览器来理解，但是其实HttpClient不是浏览器，它是一个HTTP通信库，因此它只提供一个通用浏览器应用程序所期望的功能子集，最根本的区别是HttpClient中没有用户界面，浏览器需要一个渲染引擎来显示页面，并解释用户输入，例如鼠标点击显示页面上的某处，有一个布局引擎，计算如何显示HTML页面，包括级联样式表和图像。javascript解释器运行嵌入HTML页面或从HTML页面引用的javascript代码。来自用户界面的事件被传递到javascript解释器进行处理。除此之外，还有用于插件的接口，可以处理Applet，嵌入式媒体对象（如pdf文件，Quicktime电影和Flash动画）或ActiveX控件（可以执行任何操作）。HttpClient只能以编程的方式通过其API用于传输和接受HTTP消息。

HttpClient的主要功能:

- 实现了所有 HTTP 的方法（GET、POST、PUT、HEAD、DELETE、HEAD、OPTIONS 等）
- 支持 HTTPS 协议
- 支持代理服务器（Nginx等）等
- 支持自动（跳转）转向
-

进入正题

环境说明: Eclipse、JDK1.8、SpringBoot

准备环节

第一步: 在pom.xml中引入HttpClient的依赖

```

<!-- httpClient -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpClient</artifactId>
    <version>4.5.5</version>
</dependency>

```

第二步：引入fastjson依赖

```

<!-- fastjson -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.47</version>
</dependency>

```

注：本人引入此依赖的目的是，在后续示例中，会用到“将对象转化为json字符串的功能”，也可以引其他有此功能的依赖。

注：SpringBoot的基本依赖配置，这里就不再多说了。

详细使用示例

声明：此示例中，以JAVA发送HttpClient(在test里面单元测试发送的)；也是以JAVA接收的（在controller里面接收的）。

声明：下面的代码，本人亲测有效。

GET无参：

HttpClient发送示例：

```

1. /**
2.  * GET---无参测试
3.  */
4. * @date 2018年7月13日 下午4:18:50
5. */
6. @Test
7. public void doGetTestOne() {
8.     // 获得Http客户端(可以理解为:你得先有一个浏览器;注意:实际上
    HttpClient与浏览器是不一样的)
9.     CloseableHttpClient httpClient = HttpClientBuilder.create().build();
10.    // 创建Get请求
11.    HttpGet httpGet = new
    HttpGet("http://localhost:12345/doGetControllerOne");
12.

```

```

13. // 响应模型
14. CloseableHttpResponse response = null;
15. try {
16. // 由客户端执行(发送)Get请求
17. response = httpClient.execute(httpGet);
18. // 从响应模型中获取响应实体
19. HttpEntity responseEntity = response.getEntity();
20. System.out.println("响应状态为:" + response.getStatusLine());
21. if (responseEntity != null) {
22. System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
23. System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
24. }
25. } catch (ClientProtocolException e) {
26. e.printStackTrace();
27. } catch (ParseException e) {
28. e.printStackTrace();
29. } catch (IOException e) {
30. e.printStackTrace();
31. } finally {
32. try {
33. // 释放资源
34. if (httpClient != null) {
35. httpClient.close();
36. }
37. if (response != null) {
38. response.close();
39. }
40. } catch (IOException e) {
41. e.printStackTrace();
42. }
43. }
44. }

```

对应接收示例：

```

/**
 * GET无参
 *
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:44
 */
@RequestMapping("/doGetControllerOne")
public String doGetControllerOne() {
    return "123";
}

```

GET有参(方式一：直接拼接URL):

HttpClient发送示例:

```

1. /**
2.  * GET---有参测试 (方式一:手动在url后面加上参数)
3.  *
4.  * @date 2018年7月13日 下午4:19:23
5.  */
6. @Test
7. public void doGetTestWayOne() {
8.     // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
    HttpClient与浏览器是不一样的)
9.     CloseableHttpClient httpClient = HttpClientBuilder.create().build();
10.
11.     // 参数
12.     StringBuffer params = new StringBuffer();
13.     try {
14.         // 字符数据最好encoding以下;这样一来, 某些特殊字符才能传过去
        (如:某人的名字就是 "&" ,不encoding的话,传不过去)
15.         params.append("name=" + URLEncoder.encode("&", "utf-8"));
16.         params.append("&");
17.         params.append("age=24");
18.     } catch (UnsupportedEncodingException e1) {
19.         e1.printStackTrace();
20.     }
21.
22.     // 创建Get请求

```

```
23.     HttpGet httpGet = new
HttpGet("http://localhost:12345/doGetControllerTwo" + "?" + params);
24.     // 响应模型
25.     CloseableHttpResponse response = null;
26.     try {
27.         // 配置信息
28.         RequestConfig requestConfig = RequestConfig.custom()
29.             // 设置连接超时时间(单位毫秒)
30.             .setConnectTimeout(5000)
31.             // 设置请求超时时间(单位毫秒)
32.             .setConnectionRequestTimeout(5000)
33.             // socket读写超时时间(单位毫秒)
34.             .setSocketTimeout(5000)
35.             // 设置是否允许重定向(默认为true)
36.             .setRedirectsEnabled(true).build();
37.
38.         // 将上面的配置信息 运用到这个Get请求里
39.         httpGet.setConfig(requestConfig);
40.
41.         // 由客户端执行(发送)Get请求
42.         response = httpClient.execute(httpGet);
43.
44.         // 从响应模型中获取响应实体
45.         HttpEntity responseEntity = response.getEntity();
46.         System.out.println("响应状态为:" + response.getStatusLine());
47.         if (responseEntity != null) {
48.             System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
49.             System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
50.         }
51.     } catch (ClientProtocolException e) {
52.         e.printStackTrace();
53.     } catch (ParseException e) {
54.         e.printStackTrace();
```

```

55.         } catch (IOException e) {
56.             e.printStackTrace();
57.         } finally {
58.             try {
59.                 // 释放资源
60.                 if (httpClient != null) {
61.                     httpClient.close();
62.                 }
63.                 if (response != null) {
64.                     response.close();
65.                 }
66.             } catch (IOException e) {
67.                 e.printStackTrace();
68.             }
69.         }
70.     }

```

对应接收示例：

```

/**
 * GET有参
 *
 * @param name
 *         姓名
 * @param age
 *         年龄
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:47
 */
@RequestMapping("/doGetControllerTwo")
public String doGetControllerTwo(String name, Integer age) {
    return "没想到[" + name + "]都" + age + "岁了!";
}
https://blog.csdn.net/justry\_deng

```

GET有参(方式二：使用URI获得HttpGet)：

HttpClient发送示例：

```

1. /**
2.  * GET---有参测试 (方式二:将参数放入键值对类中,再放入URI中,从而通过URI
   得到HttpGet实例)
3.  */

```

```
4.  * @date 2018年7月13日 下午4:19:23
5.  */
6.  @Test
7.  public void doGetTestWayTwo() {
8.      // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
      // HttpClient与浏览器是不一样的)
9.      CloseableHttpClient httpClient = HttpClientBuilder.create().build();
10.
11.      // 参数
12.      URI uri = null;
13.      try {
14.          // 将参数放入键值对类NameValuePair中,再放入集合中
15.          List<NameValuePair> params = new ArrayList<>();
16.          params.add(new BasicNameValuePair("name", "&"));
17.          params.add(new BasicNameValuePair("age", "18"));
18.          // 设置uri信息,并将参数集合放入uri
19.          // 注:这里也支持一个键值对一个键值对地往里面放
          setParameter(String key, String value)
20.          uri = new URIBuilder().setScheme("http").setHost("localhost")
21.
          .setPort(12345).setPath("/doGetControllerTwo")
22.          .setParameters(params).build();
23.      } catch (URISyntaxException e1) {
24.          e1.printStackTrace();
25.      }
26.      // 创建Get请求
27.      HttpGet httpGet = new HttpGet(uri);
28.
29.      // 响应模型
30.      CloseableHttpResponse response = null;
31.      try {
32.          // 配置信息
33.          RequestConfig requestConfig = RequestConfig.custom()
34.          // 设置连接超时时间(单位毫秒)
35.          .setConnectTimeout(5000)
```

```
36. // 设置请求超时时间(单位毫秒)
37. .setConnectionRequestTimeout(5000)
38. // socket读写超时时间(单位毫秒)
39. .setSocketTimeout(5000)
40. // 设置是否允许重定向(默认为true)
41. .setRedirectsEnabled(true).build();
42.
43. // 将上面的配置信息 运用到这个Get请求里
44. httpGet.setConfig(requestConfig);
45.
46. // 由客户端执行(发送)Get请求
47. response = httpClient.execute(httpGet);
48.
49. // 从响应模型中获取响应实体
50. HttpEntity responseEntity = response.getEntity();
51. System.out.println("响应状态为:" + response.getStatusLine());
52. if (responseEntity != null) {
53.     System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
54.     System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
55. }
56. } catch (ClientProtocolException e) {
57.     e.printStackTrace();
58. } catch (ParseException e) {
59.     e.printStackTrace();
60. } catch (IOException e) {
61.     e.printStackTrace();
62. } finally {
63.     try {
64.         // 释放资源
65.         if (httpClient != null) {
66.             httpClient.close();
67.         }
68.         if (response != null) {
```



```

69.         response.close();
70.     }
71.     } catch (IOException e) {
72.         e.printStackTrace();
73.     }
74. }
75. }

```

对应接收示例：

```

/**
 * GET有参
 *
 * @param name
 *         姓名
 * @param age
 *         年龄
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:47
 */
@RequestMapping("/doGetControllerTwo")
public String doGetControllerTwo(String name, Integer age) {
    return "没想到[" + name + "]都" + age + "岁了!";
}
https://blog.csdn.net/justry\_deng

```

POST无参：

HttpClient发送示例：

```

1.  /**
2.     * POST---无参测试
3.     *
4.     * @date 2018年7月13日 下午4:18:50
5.     */
6.     @Test
7.     public void doPostTestOne() {
8.     }
9.     // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
    HttpClient与浏览器是不一样的)
10.    CloseableHttpClient httpClient = HttpClientBuilder.create().build();
11.
12.    // 创建Post请求

```

```
13.     HttpPost httpPost = new
HttpPost("http://localhost:12345/doPostControllerOne");
14.     // 响应模型
15.     CloseableHttpResponse response = null;
16.     try {
17.         // 由客户端执行(发送)Post请求
18.         response = httpClient.execute(httpPost);
19.         // 从响应模型中获取响应实体
20.         HttpEntity responseEntity = response.getEntity();
21.
22.         System.out.println("响应状态为:" + response.getStatusLine());
23.         if (responseEntity != null) {
24.             System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
25.             System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
26.         }
27.     } catch (ClientProtocolException e) {
28.         e.printStackTrace();
29.     } catch (ParseException e) {
30.         e.printStackTrace();
31.     } catch (IOException e) {
32.         e.printStackTrace();
33.     } finally {
34.         try {
35.             // 释放资源
36.             if (httpClient != null) {
37.                 httpClient.close();
38.             }
39.             if (response != null) {
40.                 response.close();
41.             }
42.         } catch (IOException e) {
43.             e.printStackTrace();
44.         }
```

45. `}`

46. `}`

对应接收示例:

```
/**
 * POST无参
 *
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:49
 */
@RequestMapping(value = "/doPostControllerOne", method = RequestMethod.POST)
public String doPostControllerOne() {
    return "这个post请求没有任何参数!";
}
```

https://blog.csdn.net/justry_deng

POST有参(普通参数):

注: POST传递普通参数时, 方式与GET一样即可, 这里以直接在url后缀上参数的方式示例。

HttpClient发送示例:

```
1. /**
2.  * POST---有参测试(普通参数)
3.  *
4.  * @date 2018年7月13日 下午4:18:50
5.  */
6. @Test
7. public void doPostTestFour() {
8.
9.     // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
    HttpClient与浏览器是不一样的)
10.     CloseableHttpClient httpClient = HttpClientBuilder.create().build();
11.
12.     // 参数
13.     StringBuffer params = new StringBuffer();
14.     try {
15.         // 字符数据最好encoding以下;这样一来, 某些特殊字符才能传过去
        (如:某人的名字就是 "&" ,不encoding的话,传不过去)
16.         params.append("name=" + URLEncoder.encode("&", "utf-8"));
17.         params.append("&");
18.         params.append("age=24");
19.     } catch (UnsupportedEncodingException e1) {
```

```
20.         e1.printStackTrace();
21.     }
22.
23.     // 创建Post请求
24.     HttpPost httpPost = new
HttpPost("http://localhost:12345/doPostControllerFour" + "?" + params);
25.
26.     // 设置ContentType(注:如果只是传普通参数的话,ContentType不一定非
要用application/json)
27.     httpPost.setHeader("Content-Type",
"application/json;charset=utf8");
28.
29.     // 响应模型
30.     CloseableHttpResponse response = null;
31.     try {
32.         // 由客户端执行(发送)Post请求
33.         response = httpClient.execute(httpPost);
34.         // 从响应模型中获取响应实体
35.         HttpEntity responseEntity = response.getEntity();
36.
37.         System.out.println("响应状态为:" + response.getStatusLine());
38.         if (responseEntity != null) {
39.             System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
40.             System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
41.         }
42.     } catch (ClientProtocolException e) {
43.         e.printStackTrace();
44.     } catch (ParseException e) {
45.         e.printStackTrace();
46.     } catch (IOException e) {
47.         e.printStackTrace();
48.     } finally {
49.         try {
```

```

50. // 释放资源
51. if (httpClient != null) {
52.     httpClient.close();
53. }
54. if (response != null) {
55.     response.close();
56. }
57. } catch (IOException e) {
58.     e.printStackTrace();
59. }
60. }
61. }

```

对应接收示例：

```

/**
 * POST有参(普通参数)
 *
 * @return 测试数据
 * @date 2018年7月14日 上午10:54:29
 */
@RequestMapping(value = "/doPostControllerFour", method = RequestMethod.POST)
public String doPostControllerThree1(String name, Integer age) {
    return "[" + name + "]"居然才[" + age + "]"岁!!!";
}

```

https://blog.csdn.net/justry_deng

POST有参(对象参数):

先给出User类

```

package com.aspire.model;

/**
 * 用户实体类模型
 *
 * @author JustryDeng
 * @date 2018年7月13日 下午5:27:58
 */
public class User {

    /** 姓名 */
    private String name;

    /** 年龄 */
    private Integer age;

    /** 性别 */
    private String gender;

    /** 座右铭 */
    private String motto;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getMotto() {
        return motto;
    }

    public void setMotto(String motto) {
        this.motto = motto;
    }

    @Override
    public String toString() {
        return age + "岁" + gender + "人[" + name + "]的座右铭居然是: " + motto + "!!!";
    }
}

```

https://blog.csdn.net/justry_deng

HttpClient发送示例:

```
1.  /**
2.   * POST---有参测试(对象参数)
3.   */
4.   * @date 2018年7月13日 下午4:18:50
5.   */
6.   @Test
7.   public void doPostTestTwo() {
8.   |
9.   |   // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
10.  |   HttpClient与浏览器是不一样的)
11.  |
12.  |   // 创建Post请求
13.  |   HttpPost httpPost = new
14.  |   HttpPost("http://localhost:12345/doPostControllerTwo");
15.  |   User user = new User();
16.  |   user.setName("潘晓婷");
17.  |   user.setAge(18);
18.  |   user.setGender("女");
19.  |   user.setMotto("姿势要优雅~");
20.  |   // 我这里利用阿里的fastjson, 将Object转换为json字符串,
21.  |   // (需要导入com.alibaba.fastjson.JSON包)
22.  |   String jsonString = JSON.toJSONString(user);
23.  |   StringEntity entity = new StringEntity(jsonString, "UTF-8");
24.  |
25.  |   // post请求是将参数放在请求体里面传过去的;这里将entity放入post请求
26.  |   体中
27.  |   httpPost.setEntity(entity);
28.  |
29.  |   httpPost.setHeader("Content-Type",
30.  |   "application/json;charset=utf8");
31.  |
32.  |   // 响应模型
```

```

31. CloseableHttpResponse response = null;
32. try {
33.     // 由客户端执行(发送)Post请求
34.     response = httpClient.execute(httpPost);
35.     // 从响应模型中获取响应实体
36.     HttpEntity responseEntity = response.getEntity();
37.
38.     System.out.println("响应状态为:" + response.getStatusLine());
39.     if (responseEntity != null) {
40.         System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
41.         System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
42.     }
43. } catch (ClientProtocolException e) {
44.     e.printStackTrace();
45. } catch (ParseException e) {
46.     e.printStackTrace();
47. } catch (IOException e) {
48.     e.printStackTrace();
49. } finally {
50.     try {
51.         // 释放资源
52.         if (httpClient != null) {
53.             httpClient.close();
54.         }
55.         if (response != null) {
56.             response.close();
57.         }
58.     } catch (IOException e) {
59.         e.printStackTrace();
60.     }
61. }
62. }

```

对应接收示例：


```
/**
 * POST有参(对象参数)
 *
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:52
 */
@RequestMapping(value = "/doPostControllerTwo", method = RequestMethod.POST)
public String doPostControllerTwo(@RequestBody User user) {
    return user.toString();
}
```

https://blog.csdn.net/justry_deng

POST有参(普通参数 + 对象参数):

注：POST传递普通参数时，方式与GET一样即可，这里以通过URI获得HttpPost的方式为例。

先给出User类：

```

package com.aspire.model;

/**
 * 用户实体类模型
 *
 * @author JustryDeng
 * @date 2018年7月13日 下午5:27:58
 */
public class User {

    /** 姓名 */
    private String name;

    /** 年龄 */
    private Integer age;

    /** 性别 */
    private String gender;

    /** 座右铭 */
    private String motto;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getMotto() {
        return motto;
    }

    public void setMotto(String motto) {
        this.motto = motto;
    }

    @Override
    public String toString() {
        return age + "岁" + gender + "人[" + name + "]的座右铭居然是: " + motto + "!!!";
    }
}

```

https://blog.csdn.net/justry_deng

HttpClient发送示例:

```
1. /**
2.  * POST---有参测试(普通参数 + 对象参数)
3.  */
4. * @date 2018年7月13日 下午4:18:50
5. */
6. @Test
7. public void doPostTestThree() {
8.
9.     // 获得HttpClient(可以理解为:你得先有一个浏览器;注意:实际上
    HttpClient与浏览器是不一样的)
10.     CloseableHttpClient httpClient = HttpClientBuilder.create().build();
11.
12.     // 创建Post请求
13.     // 参数
14.     URI uri = null;
15.     try {
16.         // 将参数放入键值对类NameValuePair中,再放入集合中
17.         List<NameValuePair> params = new ArrayList<>();
18.         params.add(new BasicNameValuePair("flag", "4"));
19.         params.add(new BasicNameValuePair("meaning", "这是什么
    鬼? "));
20.         // 设置uri信息,并将参数集合放入uri
21.         // 注:这里也支持一个键值对一个键值对地往里面放
        setParameter(String key, String value)
22.         uri = new
        URIBuilder().setScheme("http").setHost("localhost").setPort(12345)
23.
        .setPath("/doPostControllerThree").setParameters(params).build();
24.     } catch (URISyntaxException e1) {
25.         e1.printStackTrace();
26.     }
27.
28.     HttpPost httpPost = new HttpPost(uri);
29.     // HttpPost httpPost = new
```

```
30. // HttpPost("http://localhost:12345/doPostControllerThree1");
31.
32. // 创建user参数
33. User user = new User();
34. user.setName("潘晓婷");
35. user.setAge(18);
36. user.setGender("女");
37. user.setMotto("姿势要优雅~");
38.
39. // 将user对象转换为json字符串, 并放入entity中
40. StringEntity entity = new StringEntity(JSON.toJSONString(user),
"UTF-8");
41.
42. // post请求是将参数放在请求体里面传过去的;这里将entity放入post请求
体中
43. httpPost.setEntity(entity);
44.
45. httpPost.setHeader("Content-Type",
"application/json;charset=utf8");
46.
47. // 响应模型
48. CloseableHttpResponse response = null;
49. try {
50. // 由客户端执行(发送)Post请求
51. response = httpClient.execute(httpPost);
52. // 从响应模型中获取响应实体
53. HttpEntity responseEntity = response.getEntity();
54.
55. System.out.println("响应状态为:" + response.getStatusLine());
56. if (responseEntity != null) {
57. System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
58. System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
59. }
```

```

60.         } catch (ClientProtocolException e) {
61.             e.printStackTrace();
62.         } catch (ParseException e) {
63.             e.printStackTrace();
64.         } catch (IOException e) {
65.             e.printStackTrace();
66.         } finally {
67.             try {
68.                 // 释放资源
69.                 if (httpClient != null) {
70.                     httpClient.close();
71.                 }
72.                 if (response != null) {
73.                     response.close();
74.                 }
75.             } catch (IOException e) {
76.                 e.printStackTrace();
77.             }
78.         }
79.     }

```

对应接收示例：

```

/**
 * POST有参(普通参数 + 对象参数)
 *
 * @return 测试数据
 * @date 2018年7月13日 下午5:29:52
 */
@RequestMapping(value = "/doPostControllerThree", method = RequestMethod.POST)
public String doPostControllerThree(@RequestBody User user, Integer flag, String meaning) {
    return user.toString() + "\n" + flag + ">>>" + meaning;
}

```

https://blog.csdn.net/justry_deng

提示：使用HttpClient时，可以视情况将其写为工具类。如：Github上Star非常多的一个HttpClient的工具类是

httpclientutil。本人在这里也**推荐使用该工具类**，因为该工具类的编写者封装了很多功能在里面，如果

不是有什么特殊的需求的话，完全可以不用造轮子，可以直接使用该工具类。使用方式很简单，可详

见<https://github.com/Arronlong/httpclientutil>。