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.
```

```
        // 响应模型
        CloseableHttpResponse response = null;
        try {
            // 由客户端执行(发送)Get请求
            response = httpClient.execute(httpGet);
            // 从响应模型中获取响应实体
            HttpEntity responseEntity = response.getEntity();
            System.out.println("响应状态为:" + response.getStatusLine());
            if (responseEntity != null) {
                System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
                System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 释放资源
                if (httpClient != null) {
                    httpClient.close();
                }
                if (response != null) {
                    response.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

对应接收示例：

```java
/**
 * 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：

```java
/**
 * GET---有参测试 (方式一:手动在url后面加上参数)
 *
 * @date 2018年7月13日 下午4:19:23
 */
@Test
public void doGetTestWayOne() {
    // 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)
    CloseableHttpClient httpClient = HttpClientBuilder.create().build();

    // 参数
    StringBuffer params = new StringBuffer();
    try {
        // 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)
        params.append("name=" + URLEncoder.encode("&", "utf-8"));
        params.append("&");
        params.append("age=24");
    } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
    }

    // 创建Get请求
```

```
        HttpGet httpGet = new HttpGet("http://localhost:12345/doGetControllerTwo" + "?" + params);
        // 响应模型
        CloseableHttpResponse response = null;
        try {
            // 配置信息
            RequestConfig requestConfig = RequestConfig.custom()
                    // 设置连接超时时间(单位毫秒)
                    .setConnectTimeout(5000)
                    // 设置请求超时时间(单位毫秒)
                    .setConnectionRequestTimeout(5000)
                    // socket读写超时时间(单位毫秒)
                    .setSocketTimeout(5000)
                    // 设置是否允许重定向(默认为true)
                    .setRedirectsEnabled(true).build();

            // 将上面的配置信息 运用到这个Get请求里
            httpGet.setConfig(requestConfig);

            // 由客户端执行(发送)Get请求
            response = httpClient.execute(httpGet);

            // 从响应模型中获取响应实体
            HttpEntity responseEntity = response.getEntity();
            System.out.println("响应状态为:" + response.getStatusLine());
            if (responseEntity != null) {
                System.out.println("响应内容长度为:" + responseEntity.getContentLength());
                System.out.println("响应内容为:" + EntityUtils.toString(responseEntity));
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 释放资源
                if (httpClient != null) {
                    httpClient.close();
                }
                if (response != null) {
                    response.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

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!";
}
```


GET有参(方式二：使用URI获得HttpGet)：

HttpClient发送示例：

```
    /**
     * 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)
     *
```

```
4.      * @date 2018年7月13日 下午4:19:23
5.      */
6.     @Test
7.     public void doGetTestWayTwo() {
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.
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)
```

```
                    // 设置请求超时时间(单位毫秒)
                    .setConnectionRequestTimeout(5000)
                    // socket读写超时时间(单位毫秒)
                    .setSocketTimeout(5000)
                    // 设置是否允许重定向(默认为true)
                    .setRedirectsEnabled(true).build();

            // 将上面的配置信息 运用到这个Get请求里
            httpGet.setConfig(requestConfig);

            // 由客户端执行(发送)Get请求
            response = httpClient.execute(httpGet);

            // 从响应模型中获取响应实体
            HttpEntity responseEntity = response.getEntity();
            System.out.println("响应状态为:" + response.getStatusLine());
            if (responseEntity != null) {
                System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
                System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 释放资源
                if (httpClient != null) {
                    httpClient.close();
                }
                if (response != null) {
```

```
                response.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

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!";
}
```


POST无参：

HttpClient发送示例：

```
/**
 * POST---无参测试
 *
 * @date 2018年7月13日 下午4:18:50
 */
@Test
public void doPostTestOne() {

    // 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)
    CloseableHttpClient httpClient = HttpClientBuilder.create().build();

    // 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.             }
```

```
        }
    }
```

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!";
}
```


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：

```
    /**
     * POST---有参测试(普通参数)
     *
     * @date 2018年7月13日 下午4:18:50
     */
    @Test
    public void doPostTestFour() {

        // 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)
        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        // 参数
        StringBuffer params = new StringBuffer();
        try {
            // 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)
            params.append("name=" + URLEncoder.encode("&", "utf-8"));
            params.append("&");
            params.append("age=24");
        } catch (UnsupportedEncodingException e1) {
```

```
                e1.printStackTrace();
            }

            // 创建Post请求
            HttpPost httpPost = new
HttpPost("http://localhost:12345/doPostControllerFour" + "?" + params);

            // 设置ContentType(注:如果只是传普通参数的话,ContentType不一定非
要用application/json)
            httpPost.setHeader("Content-Type",
"application/json;charset=utf8");

            // 响应模型
            CloseableHttpResponse response = null;
            try {
                // 由客户端执行(发送)Post请求
                response = httpClient.execute(httpPost);
                // 从响应模型中获取响应实体
                HttpEntity responseEntity = response.getEntity();

                System.out.println("响应状态为:" + response.getStatusLine());
                if (responseEntity != null) {
                    System.out.println("响应内容长度为:" +
responseEntity.getContentLength());
                    System.out.println("响应内容为:" +
EntityUtils.toString(responseEntity));
                }
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (ParseException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
```

```
                    // 释放资源
                    if (httpClient != null) {
                        httpClient.close();
                    }
                    if (response != null) {
                        response.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
```

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!!!";
}
```


POST有参(对象参数)：

先给出User类

```java
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 + "!!!";
    }

}
```

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)
10.     CloseableHttpClient httpClient = HttpClientBuilder.create().build();
11.
12.     // 创建Post请求
13.     HttpPost httpPost = new HttpPost("http://localhost:12345/doPostControllerTwo");
14.     User user = new User();
15.     user.setName("潘晓婷");
16.     user.setAge(18);
17.     user.setGender("女");
18.     user.setMotto("姿势要优雅~");
19.     // 我这里利用阿里的fastjson，将Object转换为json字符串;
20.     // (需要导入com.alibaba.fastjson.JSON包)
21.     String jsonString = JSON.toJSONString(user);
22.
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请求体中
26.     httpPost.setEntity(entity);
27.
28.     httpPost.setHeader("Content-Type", "application/json;charset=utf8");
29.
30.     // 响应模型
```

```
        CloseableHttpResponse response = null;
        try {
            // 由客户端执行(发送)Post请求
            response = httpClient.execute(httpPost);
            // 从响应模型中获取响应实体
            HttpEntity responseEntity = response.getEntity();

            System.out.println("响应状态为:" + response.getStatusLine());
            if (responseEntity != null) {
                System.out.println("响应内容长度为:" + responseEntity.getContentLength());
                System.out.println("响应内容为:" + EntityUtils.toString(responseEntity));
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 释放资源
                if (httpClient != null) {
                    httpClient.close();
                }
                if (response != null) {
                    response.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

对应接收示例：

```java
/**
 * 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();
}
```


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：

```java
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 + "!!!";
    }

}
```

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)
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.             }
```

```
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 释放资源
                if (httpClient != null) {
                    httpClient.close();
                }
                if (response != null) {
                    response.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

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;
}
```


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。