微笑面对生活
2018年08月30日 阅读 7355

关注

Springboot -- 用更优雅的方式发HTTP请求(RestTemplate详解)

> 本文已授权"后端技术精选"独家发布。

> `RestTemplate` 是 `Spring` 提供的用于访问Rest服务的客户端， `RestTemplate` 提供了多种便捷访问远程Http服务的方法,能够大大提高客户端的编写效率。

我之前的HTTP开发是用apache的HttpClient开发，代码复杂，还得操心资源回收等。代码很复杂，冗余代码多，稍微截个图，这是我封装好的一个post请求工具：

```
public String getJsonByParam(String url, List<BasicNameValuePair> formParams) throws IOEx
    CloseableHttpClient httpclient = HttpClients.createDefault();
    HttpPost httpPost = new HttpPost(url);

    httpPost.setHeader( name: "Accept",  value: "application/json");
    httpPost.setHeader( name: "Content-Type",  value: "application/x-www-form-urlencoded")
    httpPost.setHeader( name: "User-Agent",  value: "Mozilla/5.0 (Windows NT 10.0; Win64; x
    UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(formParams,  charset: "UTF-8
    httpPost.setEntity(formEntity);
    CloseableHttpResponse response = httpclient.execute(httpPost);
    HttpEntity entity = response.getEntity();
    String resultPre = EntityUtils.toString(entity,  defaultCharset: "UTF-8");
    Document doc = Jsoup.parse(resultPre);
    return doc.text();
}
```

本教程将带领大家实现Spring生态内RestTemplate的 `Get请求` 和 `Post请求` 还有 `exchange指定请求类型` 的实践和 `RestTemplate` 核心方法源码的分析，看完你就会用优雅的方式来发HTTP请求。

> 是 `Spring` 用于同步client端的核心类，简化了与 `http` 服务的通信，并满足 `RestFul` 原则，程序代码可以给它提供URL，并提取结果。默认情况下， `RestTemplate` 默认依赖jdk的HTTP连接工具。当然你也可以 通过 `setRequestFactory` 属性切换到不同的HTTP源，比如 `Apache HttpComponents` 、 `Netty` 和 `OkHttp` 。

RestTemplate能大幅简化了提交表单数据的难度，并且附带了自动转换JSON数据的功能，但只有理解了HttpEntity的组成结构（header与body），且理解了与uriVariables之间的差异，才能真正掌握其用法。这一点在Post请求更加突出，下面会介绍到。

该类的入口主要是根据HTTP的六个方法制定：

HTTP method	RestTemplate methods
DELETE	delete
GET	getForObject
	getForEntity
HEAD	headForHeaders
OPTIONS	optionsForAllow
POST	postForLocation
	postForObject
PUT	put
any	exchange
	execute

此外，exchange和excute可以通用上述方法。

册其他的转换器。(其实这点在使用的时候是察觉不到的，很多方法有一个responseType 参数，它让你传入一个响应体所映射成的对象，然后底层用HttpMessageConverter将其做映射)

```
HttpMessageConverterExtractor<T> responseExtractor =
                                new HttpMessageConverterExtractor<>(responseType, getMessageCo
```

HttpMessageConverter.java 源码:

```
public interface HttpMessageConverter<T> {
        //指示此转换器是否可以读取给定的类。
        boolean canRead(Class<?> clazz, @Nullable MediaType mediaType);

        //指示此转换器是否可以写给定的类。
        boolean canWrite(Class<?> clazz, @Nullable MediaType mediaType);

        //返回List<MediaType>
        List<MediaType> getSupportedMediaTypes();

        //读取一个inputMessage
        T read(Class<? extends T> clazz, HttpInputMessage inputMessage)
                        throws IOException, HttpMessageNotReadableException;

        //往output message写一个Object
        void write(T t, @Nullable MediaType contentType, HttpOutputMessage outputMessage)
                        throws IOException, HttpMessageNotWritableException;

}
```

在内部， RestTemplate 默认使用 SimpleClientHttpRequestFactory 和 DefaultResponseErrorHandler 来分别处理 HTTP 的创建和错误，但也可以通过 setRequestFactory 和 setErrorHandler 来覆盖。

2. get请求实践

2.1. getForObject()方法

```
public <T> T getForObject(URI url, Class<T> responseType)
```

`getForObject()` 其实比 `getForEntity()` 多包含了将HTTP转成POJO的功能，但是 `getForObject` 没有处理 `response` 的能力。因为它拿到手的就是成型的 `pojo` 。省略了很多 `response` 的信息。

2.1.1 POJO:

```
public class Notice {
    private int status;
    private Object msg;
    private List<DataBean> data;
}
public  class DataBean {
  private int noticeId;
  private String noticeTitle;
  private Object noticeImg;
  private long noticeCreateTime;
  private long noticeUpdateTime;
  private String noticeContent;
}
```

示例：2.1.2 不带参的get请求

```
/**
 * 不带参的get请求
 */
@Test
public void restTemplateGetTest(){
        RestTemplate restTemplate = new RestTemplate();
        Notice notice = restTemplate.getForObject("http://xxx.top/notice/list/1/5"
                        , Notice.class);
        System.out.println(notice);
}
```

控制台打印：

```
Notice{status=200, msg=null, data=[DataBean{noticeId=21, noticeTitle='aaa', noticeImg=null,
noticeCreateTime=1525292723000, noticeUpdateTime=1525292723000, noticeContent='<p>aaa</p>'},
DataBean{noticeId=20, noticeTitle='ahaha', noticeImg=null, noticeCreateTime=1525291492000,
noticeUpdateTime=1525291492000, noticeContent='<p>ah.......'
```

示例：2.1.3 带参数的get请求1

```
Notice notice = restTemplate.getForObject("http://fantj.top/notice/list/{1}/{2}"
                                , Notice.class,1,5);
```

明眼人一眼能看出是用了占位符 `{1}` 。

示例：2.1.4 带参数的get请求2

```
Map<String,String> map = new HashMap();
map.put("start","1");
map.put("page","5");
Notice notice = restTemplate.getForObject("http://fantj.top/notice/list/"
                , Notice.class,map);
```

明眼人一看就是利用map装载参数，不过它默认解析的是 `PathVariable` 的url形式。

2.2 getForEntity()方法

```
public <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Object... uriVari
public <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Map<String, ?> ur
public <T> ResponseEntity<T> getForEntity(URI url, Class<T> responseType){}
```

与getForObject()方法不同的是返回的是 `ResponseEntity` 对象，如果需要转换成pojo，还需要json工具类的引入，这个按个人喜好用。不会解析json的可以百度 `FastJson` 或者 `Jackson` 等工具类。然后我们研究一下 `ResponseEntity` 下面有啥方法。

ResponseEntity.java

```
public HttpStatus getStatusCode(){}
public int getStatusCodeValue(){}
public boolean equals(@Nullable Object other) {}
public String toString() {}
public static BodyBuilder status(HttpStatus status) {}
public static BodyBuilder ok() {}
public static <T> ResponseEntity<T> ok(T body) {}
public static BodyBuilder created(URI location) {}
...
```

HttpStatus.java

```
public enum HttpStatus {
public boolean is1xxInformational() {}
public boolean is2xxSuccessful() {}
public boolean is3xxRedirection() {}
public boolean is4xxClientError() {}
public boolean is5xxServerError() {}
public boolean isError() {}
}
```

BodyBuilder.java

```
public interface BodyBuilder extends HeadersBuilder<BodyBuilder> {
    //设置正文的长度，以字节为单位，由Content-Length标头
      BodyBuilder contentLength(long contentLength);
    //设置body的MediaType 类型
      BodyBuilder contentType(MediaType contentType);
    //设置响应实体的主体并返回它。
      <T> ResponseEntity<T> body(@Nullable T body);
}
```

可以看出来，ResponseEntity包含了HttpStatus和BodyBuilder的这些信息，这更方便我们处理resp
原生的东西。

```
@Test
public void rtGetEntity(){
                RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<Notice> entity = restTemplate.getForEntity("http://fantj.top/notice/lis
                , Notice.class);

        HttpStatus statusCode = entity.getStatusCode();
        System.out.println("statusCode.is2xxSuccessful()"+statusCode.is2xxSuccessful());

        Notice body = entity.getBody();
        System.out.println("entity.getBody()"+body);


        ResponseEntity.BodyBuilder status = ResponseEntity.status(statusCode);
        status.contentLength(100);
        status.body("我在这里添加一句话");
        ResponseEntity<Class<Notice>> body1 = status.body(Notice.class);
        Class<Notice> body2 = body1.getBody();
        System.out.println("body1.toString()"+body1.toString());
    }
```

```
statusCode.is2xxSuccessful()true
entity.getBody()Notice{status=200, msg=null, data=[DataBean{noticeId=21, noticeTitle='aaa', ..
body1.toString()<200 OK,class com.waylau.spring.cloud.weather.pojo.Notice,{Content-Length=[100
```

当然，还有 `getHeaders()` 等方法没有举例。

3. post请求实践

> 同样的,post请求也有 `postForObject` 和 `postForEntity` 。

```
public <T> T postForObject(String url, @Nullable Object request, Class<T> responseType, Object
                        throws RestClientException {}
public <T> T postForObject(String url, @Nullable Object request, Class<T> responseType, Map<St
                        throws RestClientException {}
public <T> T postForObject(URI url, @Nullable Object request, Class<T> responseType) thro
```

> 我用一个验证邮箱的接口来测试。

```
@Test
public void rtPostObject(){
    RestTemplate restTemplate = new RestTemplate();
    String url = "http://47.xxx.xxx.96/register/checkEmail";
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
    MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
    map.add("email", "844072586@qq.com");

    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);
    ResponseEntity<String> response = restTemplate.postForEntity( url, request , String.class
    System.out.println(response.getBody());
}
```

执行结果：

```
{"status":500,"msg":"该邮箱已被注册","data":null}
```

```
 INFO 9220 --- [           main] c.w.s.c.w.c.HelloControllerTe
{"status":500,"msg":"该邮箱已被注册","data":null}
 INFO 9220 --- [       Thread-2] o.s.w.c.s.GenericWebApplicati
```

代码中， `MultiValueMap` 是 `Map` 的一个子类，它的一个 `key` 可以存储多个 `value` ， 简单的看下这个接口：

```
public interface MultiValueMap<K, V> extends Map<K, List<V>> {...}
```

为什么用 `MultiValueMap` ?因为 `HttpEntity` 接受的request类型是它。

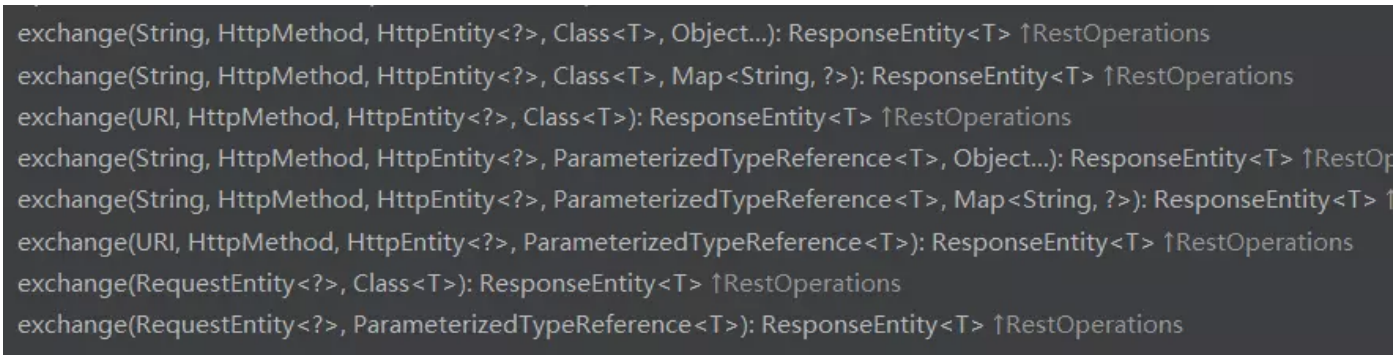
为什么用 `HttpEntity` 是因为 `restTemplate.postForEntity` 方法虽然表面上接收的request是 `@Nullable Object request` 类型，但是你追踪下去会发现，这个 `request` 是用 `HttpEntity` 来解析。核心代码如下：

```
if (requestBody instanceof HttpEntity) {
        this.requestEntity = (HttpEntity<?>) requestBody;
}else if (requestBody != null) {
        this.requestEntity = new HttpEntity<>(requestBody);
}else {
        this.requestEntity = HttpEntity.EMPTY;
}
```

我曾尝试用map来传递参数，编译不会报错，但是执行不了，是无效的url request请求(400 ERROR)。其实这样的请求方式已经满足post请求了，cookie也是属于header的一部分。可以按需求设置请求头和请求体。其它方法与之类似。

4. 使用exchange指定调用方式

> exchange()方法跟上面的getForObject()、getForEntity()、postForObject()、postForEntity()等方法不同之处在于它可以指定请求的HTTP类型。



但是你会发现exchange的方法中似乎都有 `@Nullable HttpEntity<?> requestEntity` 这个参数，这就意味着我们至少要用HttpEntity来传递这个请求体，之前说过源码所以建议就使用HttpEntity提高性能

示例

```
        RestTemplate restTemplate = new RestTemplate();
        String url = "http://xxx.top/notice/list";
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        JSONObject jsonObj = new JSONObject();
        jsonObj.put("start",1);
        jsonObj.put("page",5);

        HttpEntity<String> entity = new HttpEntity<>(jsonObj.toString(), headers);
        ResponseEntity<JSONObject> exchange = restTemplate.exchange(url,
                                        HttpMethod.GET, entity, JSONObject.class);
        System.out.println(exchange.getBody());
    }
```

这次可以看到，我使用了 `JSONObject` 对象传入和返回。

当然，HttpMethod方法还有很多，用法类似。

5. excute()指定调用方式

`excute()` 的用法与 `exchange()` 大同小异了，它同样可以指定不同的 `HttpMethod` ，不同的是它返回的对象是响应体所映射成的对象 `<T>` ，而不是 `ResponseEntity<T>` 。

需要强调的是， `execute()` 方法是以上所有方法的底层调用。随便看一个：

```
        @Override
        @Nullable
        public <T> T postForObject(String url, @Nullable Object request, Class<T> responseType
                throws RestClientException {

            RequestCallback requestCallback = httpEntityCallback(request, responseType);
            HttpMessageConverterExtractor<T> responseExtractor =
                    new HttpMessageConverterExtractor<>(responseType, getMessageCo
            return execute(url, HttpMethod.POST, requestCallback, responseExtractor, uriVa
        }
```

关注公众号，回复 java架构 获取架构视频资源。

关注下面的标签，发现更多相似文章

后端 Java HTTP Apache

微笑面对生活 java工程师
获得点赞 1,534 次 · 文章被阅读 42,436 次

关注

安装掘金浏览器插件
打开新标签页发现好内容，掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧！

评论

输入评论...

Vincentz
更优雅的方式应该是用Feign Client
5月前 回复

二哥很猛 java @ 挖坑师

Vincentz

回复 Vincentz: 不需要的，了解一下

5月前

小茂 HR @ 威富通科技有限...

想来鹅厂的程序员可加我微信wml19941128xiaomao

6月前 回复

liangzzz Java工程师 @ 珠海

httpRequest这个类库不错，可以参考一下

6月前 回复

本历年加油！ Java工程师

RestTemplate和AsyncRestTemplate都out了。现在是webclient

6月前 回复

本历年加油！ Java工程师

RestTemplate和AsyncRestTemplate都out了。现在是webclient

6月前 3 回复

daqzi java Python 大数据 ai...

还是我的工具类更强大

6月前 1 回复

查看更多 >

相关推荐

专栏 · 石杉的架构笔记 · 18分钟前 · Java / 架构

【真实案例分享】面对BAT大厂的竞争对手时，小公司Java工程师是如何败北的？【石杉的架构笔记】

3 2

专栏 · crossoverJie · 23分钟前 · Java / JVM

一个线程罢工的诡异事件

5

为什么阿里巴巴禁止在 foreach 循环里进行元素的 remove/add 操作

52 9

专栏 · 王磊的博客 · 1天前 · 面试 / Java

Java 200+ 面试题补充③ Dubbo 模块

46 3

芋道源码_以德服人_不服就干 · 2小时前 · Java

配置中心 Apollo 源码解析 —— 服务自身配置 ServerConfig

2

专栏 · 小姐姐味道 · 1分钟前 · Java

JAVA多线程使用场景和注意事项

热 · 专栏 · 胡七筒 · 1天前 · JavaScript

程序猿生存指南-58 悲喜之间

61 53

专栏 · 张风捷特烈 · 12小时前 · Android / HTTP

网络篇：协天子令诸侯[-Http-]

11

专栏 · Sophie May · 16小时前 · Java

JVM内存结构

5

专栏 · 预流 · 16小时前 · 后端 / 大数据

大数据技术简介

8