2、TestRestTemplate进行单元测试

- https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/web/client/TestRestTemplate.html
- https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.testing.utilities.test-rest-template

（1）概念

- TestRestTemplate就是对RestTemplate的封装，RestTemplate 是从 Spring3.0 开始支持的一个 HTTP 请求的客户端工具，它提供了常见的REST请求方案的模版，例如 GET 请求、POST 请求、PUT 请求、DELETE 请求以及一些通用的请求执行方法 exchange 以及 execute。在Spring中有许多类似功能的类，如JdbcTemplate, JmsTemplate等。RestTemplate 继承自 InterceptingHttpAccessor 并且实现了 RestOperations 接口，其中 RestOperations 接口定义了基本的 RESTful 操作，这些操作在 RestTemplate 中都得到了实现。在实际项目中往往使用单例模式来设计RestTemplate。

（2）为什么使用TestRestTemplate？

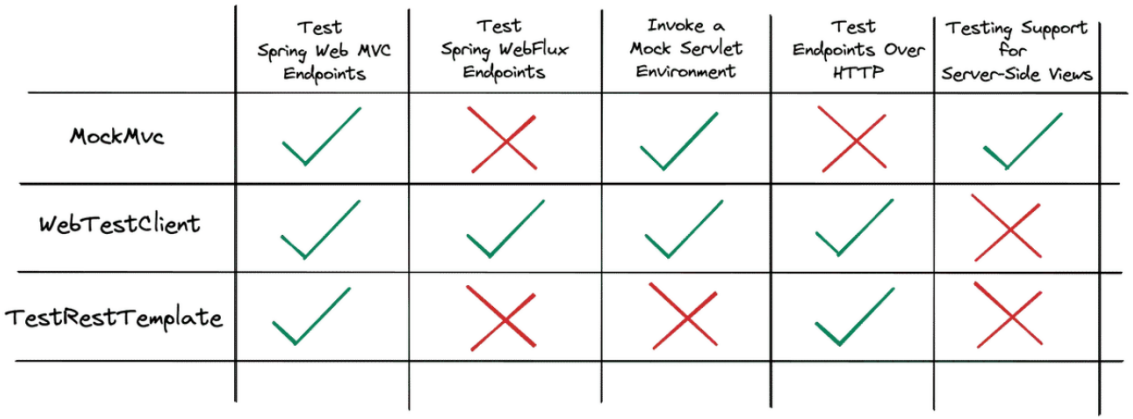
- 大家之前都用过apache的HTTPClient类，逻辑繁琐，代码复杂，还要自己编写使用类HttpClientUtil，封装对应的post，get，delete等方法。RestTemplate的行为可以通过callback回调方法和配置HttpMessageConverter 来定制，用来把对象封装到HTTP请求体，将响应信息放到一个对象中。RestTemplate提供更高等级的符合HTTP的六种主要方法，可以很简单的调用RESTful服务。

（2）简单介绍

- 相同点
  - TestRestTemplate同MockMvc一样，也是测试RestFul接口的测试工具。
- 不同点

笔记目录

- MockMvc是模拟出一个SpringMVC的运行环境（只模拟了上下文环境），并没有真正地运行一个完整的Servlet容器，而RestTemplate是启动一个完整的server环境（启动tomcat服务器，并且模拟上下文环境）。
- MockMVC无法做到将返回json数据反序列化为特定的对象，只能将返回的Json数据转化为String，TestRestTemplate可以将返回json数据反序列化为特定的对象；
- MockMvc支持测试时数据的回滚，这适用于测试一个涉及修改数据库数据的方法；而TestRestTemplate不支持回滚，因为运行的测试方法和server服务不在同一个线程，所以无法控制server服务。
- MockMVC并没有真正的启动一个完整的server服务，故要测试一些会返回error page（4开头的状态码）的可检查异常时，因为error page是由Servlet容器提供的，所以MockMVC无法测试这些方法，他还是会返回200状态码。而TestRestTemplate可以测试会抛出可检查异常的方法。



（3）依赖

复制代码

```
testImplementation("org.springframework.boot:spring-boot-starter-test")
```

内部依赖是spring-boot-test-3.0.3.jar

java

（4）底层源码梳理

（4-1）TestRestTemplate是对RestTemplate的封装

复制代码

```
public class TestRestTemplate {

    private final RestTemplateBuilder builder;

    private final HttpClientOption[] httpClientOptions;

    private final RestTemplate restTemplate;

    public TestRestTemplate(RestTemplateBuilder
restTemplateBuilder) {
        this(restTemplateBuilder, null, null);
    }

    public TestRestTemplate(HttpClientOption...
httpClientOptions) {
        this(null, null, httpClientOptions);
    }

    public TestRestTemplate(
      String username,
      String password,
      HttpClientOption... httpClientOptions
    ) {
       this(new
RestTemplateBuilder(),username,password,httpClientOptions);
    }
```

```java
    public TestRestTemplate(
      RestTemplateBuilder builder,
      String username,
      String password,
      HttpClientOption... httpClientOptions
    ) {
        this.restTemplate = builder.build();
    }
}
```

java

（4-2）TestRestTemplate的postForObject、postForEntity、postForLocation方法的入参和返回值

- 入参
  - url： 请求的url路径
  - @Nullable Object request： 可为空的请求对象
  - Class<T> responseType： 返回类型的类定义
  - Object... uriVariables： 动态的uri参数，可传递多个
  - Map<String, ?> uriVariables： 以Map方式存储的uri参数
- 返回值
  - postForObject()返回值：是HTTP协议的响应体
  - postForEntity()返回值：是ResponseEntity，ResponseEntity是对HTTP响应体的封装，除了包含响应体，还包含HTTP状态码、contentType、contentLength、Header等信息。
  - postForLocation()返回值：是URI对象，如果需要在执行完毕后返回URL，就可以用这个方法.
- 公共的源码部分
  - 源码的第一个参数requestBody，就是postForObject/postForEntity/postForLocation三个方法入参的request参数，如果requestBody是HttpEntity类型，则直接使用；如果requestBody不是HttpEntity类型，则会将其封装成HttpEntity类型。

```java
public HttpEntityRequestCallback(@Nullable Object requestBody,
@Nullable Type responseType) {
    super(responseType);
    if (requestBody instanceof HttpEntity) {
        this.requestEntity = (HttpEntity)requestBody;
    } else if (requestBody != null) {
        this.requestEntity = new HttpEntity(requestBody);
    } else {
        this.requestEntity = HttpEntity.EMPTY;
    }
}
```

- 注意点
  - postforxxx请求
    - ①直接使用一个对象作为参数，可以这么做的原因就是因为公共源码部分，进行了对象的类型检测和转换为HttpEntity。
    - ②直接使用HttpEntity作为参数，HttpEntity中携带着数据。
    - ③如果需要一个对象是以JSON形式存储，直接使用HttpEntity作为参数，HttpEntity中携带着json串数据。
  - post请求/get请求
    - 需要指定请求头时或者返回结果除了需要响应体，还需要响应码，响应头，响应体类型，响应体长度等，使用post/getForEntity方法。否则使用post/getForObject方法
    - 需要返回URI，使用post/getForLocation方法
  - 请求方式想要更灵活，使用exchange方法

复制代码

```
//只列出post方式
public class TestRestTemplate {

    public <T> ResponseEntity<T> postForEntity(
      URI url,
      Object request,
      Class<T> responseType
    ) {
        return this.restTemplate.postForEntity(
          applyRootUriIfNecessary(url),
          request,
          responseType
        );
    }

    public <T> ResponseEntity<T> postForEntity(
      String url,
      Object request,
      Class<T> responseType,
      Object... urlVariables
    ) {
        return this.restTemplate.postForEntity(
          url,
          request,
          responseType,
          urlVariables
        );
    }

    public <T> ResponseEntity<T> postForEntity(
      String url,
      Object request,
```

```
      Class<T> responseType,
      Map<String, ?> urlVariables
    ) {
        return this.restTemplate.postForEntity(
          url,
          request,
          responseType,
          urlVariables
        );
    }

    public <T> T postForObject(
      URI url,
      Object request,
      Class<T> responseType
    ) {
        return this.restTemplate.postForObject(
          applyRootUriIfNecessary(url),
          request,
          responseType
        );
    }

    public <T> T postForObject(
      String url,
      Object request,
      Class<T> responseType,
      Object... urlVariables
    ) {
        return this.restTemplate.postForObject(
          url,
          request,
```

```
65            responseType,
66            urlVariables
67          );
68      }
69
70      public <T> T postForObject(
71        String url,
72        Object request,
73        Class<T> responseType,
74        Map<String, ?> urlVariables
75      ) {
76          return this.restTemplate.postForObject(
77            url,
78            request,
79            responseType,
80            urlVariables
81          );
82      }
83
84
85      public URI postForLocation(URI url, Object request) {
86          return this.restTemplate.postForLocation(
87            applyRootUriIfNecessary(url),
88            request
89          );
90      }
91
92      public URI postForLocation(
93        String url,
94        Object request,
95        Object... urlVariables
96      ) {
```

```
97          return this.restTemplate.postForLocation(
98            url,
99            request,
100           urlVariables
101         );
102     }
103
104     public URI postForLocation(
105       String url,
106       Object request,
107       Map<String, ?> urlVariables
108     ) {
109         return this.restTemplate.postForLocation(url, request,
urlVariables);
110     }
111
112
113     public <T> ResponseEntity<T> exchange(
114       URI url,
115       HttpMethod method,
116       HttpEntity<?> requestEntity,
117       Class<T> responseType) {
118         return
this.restTemplate.exchange(applyRootUriIfNecessary(url), method,
requestEntity, responseType);
119     }
120
121
122     public <T> ResponseEntity<T> exchange(
123       String url,
124       HttpMethod method,
125       HttpEntity<?> requestEntity,
```

```java
        Class<T> responseType,
        Map<String, ?> urlVariables
    ) {
        return this.restTemplate.exchange(url, method,
requestEntity, responseType, urlVariables);
    }


    public <T> ResponseEntity<T> exchange(
      String url,
      HttpMethod method,
      HttpEntity<?> requestEntity,
      Class<T> responseType,
      Object... urlVariables
    ) {
        return this.restTemplate.exchange(url, method,
requestEntity, responseType, urlVariables);
    }

}
```

java

（5）使用案例

- 测试类上加上@SpringBootTest注解，并且设置了webEnvironment属性为WebEnvironment.RANDOM_PORT或WebEnvironment.DEFINED_PORT，才会自动配置TestRestTemplate，这就意味着webEnvironment属性会开启tomcat容器，初始化web上下文环境，监听http请求。
- 一个简单的案例

复制代码

```
1
2  import org.junit.jupiter.api.Test
3  import org.springframework.beans.factory.annotation.Autowired
4  import org.springframework.boot.test.context.SpringBootTest
5  import org.springframework.boot.test.web.client.TestRestTemplate
6  import org.springframework.http.HttpEntity
7  import org.springframework.http.HttpHeaders
8
9  @SpringBootTest(webEnvironment =
   SpringBootTest.WebEnvironment.RANDOM_PORT)
10 class xxxControllerTest {
11
12     @Autowired
13     lateinit var restTemplate: TestRestTemplate
14
   ////////////////////////////////////////////////////////////////
   ///////////
15     //getForEntity
16     //可以用一个数字做占位符，最后是一个可变长度的参数，来一一替换前面的占位
   符
17     @Test
18     public String sayHello() {
19         ResponseEntity<String> responseEntity =
   restTemplate.getForEntity("http://HELLO-SERVICE/sayhello?name=
   {1}", String.class, "张三");
20         return responseEntity.getBody();
21     }
22
23
24
   ////////////////////////////////////////////////////////////////
```

```
//
25
26    //getForEntity
27    //也可以前面使用name={name}这种形式，最后一个参数是一个map，map的key即为前边占位符的名字，map的value为参数值
28    @Test
29    public String sayHello2() {
30        Map<String, String> map = new HashMap<>();
31        map.put("name", "李四");
32        ResponseEntity<String> responseEntity = restTemplate.getForEntity("http://HELLO-SERVICE/sayhello?name={name}", String.class, map);
33        return responseEntity.getBody();
34    }
35
36 ////////////////////////////////////////////////////////////////////
37
38  //getForObject
39    @Test
40    public Book book2() {
41        Book book = restTemplate.getForObject("http://HELLO-SERVICE/getbook1", Book.class);
42        return book;
43    }
44
45 ////////////////////////////////////////////////////////////////////
46  //postForEntity
47    @Test
48    public Book book3() {
49        Book book = new Book();
```

```
        book.setName("红楼梦");
        ResponseEntity<Book> responseEntity =
restTemplate.postForEntity("http://HELLO-SERVICE/getbook2", book,
Book.class);
        return responseEntity.getBody();
    }


////////////////////////////////////////////////////////////////
///

   //如果你只关注，返回的消息体，可以直接使用postForObject。用法和
getForObject一致。


////////////////////////////////////////////////////////////////
//

   //postForLocation也是提交新资源，提交成功之后，返回新资源的URI，
postForLocation的参数和前面两种的参数基本一致，只不过该方法的返回值为Uri，
这个只需要服务提供者返回一个Uri即可，该Uri表示新资源的位置。


////////////////////////////////////////////////////////////////
////////

   //PUT请求可以通过put方法调用，put方法的参数和前面介绍的postForEntity方
法的参数基本一致，只是put方法没有返回值而已
    @Test
    public void put() {
```

```java
        Book book = new Book();
        book.setName("红楼梦");
        restTemplate.put("http://HELLO-SERVICE/getbook3/{1}",
book, 99);
    }


////////////////////////////////////////////////////////////////
///
    //DELETE请求
    @Test
    public void delete() {
        restTemplate.delete("http://HELLO-SERVICE/getbook4/{1}",
100);
    }
}
```

java

- https://cloud.tencent.com/developer/article/1703375

复制代码

```java
//postForObject发送JSON格式请求
@SpringBootTest
class PostTests {

    @Resource
```

```
    private RestTemplate restTemplate;

    @Test
    void testSimple()  {
        // 请求地址
        String url = "http://jsonplaceholder.typicode.com/posts";

        // 要发送的数据对象
        PostDTO postDTO = new PostDTO();
        postDTO.setUserId(110);
        postDTO.setTitle("zimug 发布文章");
        postDTO.setBody("zimug 发布文章 测试内容");

        // 发送post请求(内部自动会将postDTO转成json串)，并输出结果
        PostDTO result = restTemplate.postForObject(url, postDTO,
PostDTO.class);
        System.out.println(result);
    }
}

////////////////////////////////////////////////////////////////
//////
//postForObject模拟表单数据提交

@Test
public void testForm() {
    // 请求地址
    String url = "http://jsonplaceholder.typicode.com/posts";

    // 请求头设置,x-www-form-urlencoded格式的数据
    HttpHeaders headers = new HttpHeaders();
```

```
 headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    //提交参数设置
    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("title", "zimug 发布文章第二篇");
    map.add("body", "zimug 发布文章第二篇 测试内容");

    // 组装请求体
    HttpEntity<MultiValueMap<String, String>> request =
                new HttpEntity<MultiValueMap<String, String>>(map, headers);

    // 发送post请求，并打印结果，以String类型接收响应结果JSON字符串
    String result = restTemplate.postForObject(url, request, String.class);
    System.out.println(result);
}

////////////////////////////////////////////////////////////////////////////

//postForEntity发送JSON格式请求
@PostMapping(
  path= "/", consumes = "application/json", produces = "application/json")

public ResponseEntity<Object> addEmployee (
  @RequestHeader(name = "X-COM-PERSIST", required = true) String headerPersist,
```

```
  @RequestHeader(name = "X-COM-LOCATION", defaultValue = "ASIA")
String headerLocation,
  @RequestBody Employee employee ) throws Exception {
  //
}

@Test
public void testEntity() {
    // 请求地址
    String url = "http://jsonplaceholder.typicode.com/posts";

    // 要发送的数据对象
    PostDTO postDTO = new PostDTO();
    postDTO.setUserId(110);
    postDTO.setTitle("zimug 发布文章");
    postDTO.setBody("zimug 发布文章 测试内容");

    // 发送post请求(内部自动会将postDTO转成json串)，并输出结果
    ResponseEntity<String> responseEntity
                = restTemplate.postForEntity(url, postDTO,
String.class);
    String body = responseEntity.getBody(); // 获取响应体
    System.out.println("HTTP 响应body: " + postDTO.toString());

    //以下是postForEntity比postForObject多出来的内容
    HttpStatus statusCode = responseEntity.getStatusCode(); // 获
取响应码
    int statusCodeValue = responseEntity.getStatusCodeValue(); //
获取响应码值
    HttpHeaders headers = responseEntity.getHeaders(); // 获取响应
头

```

```
    System.out.println("HTTP 响应状态: " + statusCode);
    System.out.println("HTTP 响应状态码: " + statusCodeValue);
    System.out.println("HTTP Headers信息: " + headers);
}


//postForEntity发送JSON格式请求（携带headers）
@Test
public void testEntityJson() {
    URI uri = new
URI("http://localhost:"+randomServerPort+"/employees/");
    Employee employee = new Employee("Adam", "Gilly",
"test@email.com");
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

 headers.setAccept(Collections.singletonList(MediaType.APPLICATI
ON_JSON));
    headers.set("X-COM-PERSIST", "true");
    headers.set("X-COM-LOCATION", "USA");
    HttpEntity<Employee> httpEntity = new HttpEntity<>(employee,
headers);
    ResponseEntity<String> result =
restTemplate.postForEntity(uri, httpEntity, String.class);
    Assertions.assertEquals(201, result.getStatusCodeValue());
}

////////////////////////////////////////////////////////////////
/////////////

//postForLocation发送JSON格式请求

```

```java
@Test
public void testURI() {
    // 请求地址
    String url = "http://jsonplaceholder.typicode.com/posts";

    PostDTO postDTO = new PostDTO();
    postDTO.setUserId(110);
    postDTO.setTitle("zimug 发布文章");
    postDTO.setBody("zimug 发布文章 测试内容");

    // 发送post请求，并输出结果
    URI uri = restTemplate.postForLocation(url,postDTO);
    System.out.println(uri);
}
```

java