链滴

RestTemplate 指南

作者：chenlei65368
原文链接：https://ld246.com/article/1569394960005
来源网站：链滴

● 本文译自 baeldung.com

RestTemplate 指南

1 说明

在本教程中，我们将说明并很好地使用Spring REST Client – RestTemplate的各种操作。

对于所有示例的API方面，我们将从此处运行RESTful服务。

https://github.com/eugenp/tutorials/tree/master/spring-rest

2 使用GET检索资源

2.1 获取 JSON

让我们开始简单并讨论GET请求-使用getForEntity() API，示例：

```
RestTemplate restTemplate = new RestTemplate();
String fooResourceUrl
  = "http://localhost:8080/spring-rest/foos";
ResponseEntity<String> response
  = restTemplate.getForEntity(fooResourceUrl + "/1", String.class);
assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));
```

请注意，我们拥有对HTTP响应的完全访问权限-因此我们可以执行诸如检查状态代码以确保操作成功使用响应的正文之类的操作。

```
ObjectMapper mapper = new ObjectMapper();
JsonNode root = mapper.readTree(response.getBody());
JsonNode name = root.path("name");
assertThat(name.asText(), notNullValue());
```

我们在此处将响应主体作为标准String，使用Jackson（以及Jackson提供的JSON节点结构）来验证些细节。

2.2 检索POJO而不是JSON

我们还可以将响应直接映射到一个 DTO对象 –例如：

```
public class Foo implements Serializable {
    private long id;

    private String name;
    // standard getters and setters
}
```

现在，我们只需使用restTemplate.getForObject API：

```
Foo foo = restTemplate
  .getForObject(fooResourceUrl + "/1", Foo.class);
assertThat(foo.getName(), notNullValue());
```

```
assertThat(foo.getId(), is(1L));
```

3 使用HEAD检索Headers

现在，让我们快速浏览一下使用HEAD，然后再继续使用更常见的方法-我们将在这里使用headForHeders（）API：

```
HttpHeaders httpHeaders = restTemplate.headForHeaders(fooResourceUrl);
assertTrue(httpHeaders.getContentType().includes(MediaType.APPLICATION_JSON));
```

4 使用POST创建资源

为了在API中创建新的资源–我们可以充分利用postForLocation()，postForObject()或postForEntity(API。

第一个返回新创建的资源的URI，而第二个返回资源本身。

4.1 postForObject API

```
RestTemplate restTemplate = new RestTemplate();

HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));
Foo foo = restTemplate.postForObject(fooResourceUrl, request, Foo.class);
assertThat(foo, notNullValue());
assertThat(foo.getName(), is("bar"));
```

4.2 postForLocation API

让我们看一下该操作–而不是返回完整的Resource，而是返回该新创建的Resource的Location:

```
HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));
URI location = restTemplate
  .postForLocation(fooResourceUrl, request);
assertThat(location, notNullValue());
```

4.3 exchange API

让我们看一下如何使用更通用的exchange API进行POST：

```
RestTemplate restTemplate = new RestTemplate();
HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));
ResponseEntity<Foo> response = restTemplate
  .exchange(fooResourceUrl, HttpMethod.POST, request, Foo.class);

assertThat(response.getStatusCode(), is(HttpStatus.CREATED));

Foo foo = response.getBody();

assertThat(foo, notNullValue());
assertThat(foo.getName(), is("bar"));
```

4.4 提交表单

接下来，让我们看看如何使用POST方法提交表单。

首先，我们需要将``Content-Type''标头设置为application / x-www-form-urlencoded。

这样可以确保可以将一个较大的查询字符串发送到服务器，其中包含以“&”分隔的名称/值对：

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
```

我们可以将表单变量包装到LinkedMultiValueMap中：

```
MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
map.add("id", "1");
```

接下来，我们使用HttpEntity实例构建请求：

```
HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);
```

最后，我们可以通过在端点上调用restTemplate.postForEntity()连接到REST服务：/foos/form

```
ResponseEntity<String> response = restTemplate.postForEntity(
  fooResourceUrl+"/form", request , String.class);

assertThat(response.getStatusCode(), is(HttpStatus.CREATED));
```

5 使用OPTIONS获取允许的操作

接下来，我们将快速了解如何使用OPTIONS请求，并使用这种请求探索对特定URI的允许操作；

API是optionsForAllow：

```
Set<HttpMethod> optionsForAllow = restTemplate.optionsForAllow(fooResourceUrl);
HttpMethod[] supportedMethods
  = {HttpMethod.GET, HttpMethod.POST, HttpMethod.PUT, HttpMethod.DELETE};
assertTrue(optionsForAllow.containsAll(Arrays.asList(supportedMethods)));
```

6 使用PUT更新资源

接下来，我们将开始研究PUT，并且由于template.put API非常简单，因此将更具体地说明此操作的exchange API。

6.1 exchange的简单PUT

我们将从针对API的简单PUT操作开始-并记住该操作不会使任何人返回到客户端：

```
Foo updatedInstance = new Foo("newName");
updatedInstance.setId(createResponse.getBody().getId());
String resourceUrl =
  fooResourceUrl + '/' + createResponse.getBody().getId();
HttpEntity<Foo> requestUpdate = new HttpEntity<>(updatedInstance, headers);
template.exchange(resourceUrl, HttpMethod.PUT, requestUpdate, Void.class);
```

6.2 带有.exchange和请求回调的PUT

下面，我们将使用请求回调来发出PUT。

让我们确保我们准备了回调–在这里我们可以设置所需的所有标头以及请求正文：

```
RequestCallback requestCallback(final Foo updatedInstance) {
    return clientHttpRequest -> {
        ObjectMapper mapper = new ObjectMapper();
        mapper.writeValue(clientHttpRequest.getBody(), updatedInstance);
        clientHttpRequest.getHeaders().add(
          HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE);
        clientHttpRequest.getHeaders().add(
          HttpHeaders.AUTHORIZATION, "Basic " + getBase64EncodedLogPass());
    };
}
```

下一步，我们使用POST请求创建Resource：

```
ResponseEntity<Foo> response = restTemplate
  .exchange(fooResourceUrl, HttpMethod.POST, request, Foo.class);
assertThat(response.getStatusCode(), is(HttpStatus.CREATED));
```

然后我们更新资源：

```
Foo updatedInstance = new Foo("newName");
updatedInstance.setId(response.getBody().getId());
String resourceUrl =fooResourceUrl + '/' + response.getBody().getId();
restTemplate.execute(
  resourceUrl,
  HttpMethod.PUT,
  requestCallback(updatedInstance),
  clientHttpResponse -> null);
```

7 使用DELETE删除资源

要删除现有资源，我们使用delete() API：

```
String entityUrl = fooResourceUrl + "/" + existingResource.getId();
restTemplate.delete(entityUrl);
```

8 配置超时

我们可以简单地使用ClientHttpRequestFactory将RestTemplate配置超时，如下所示：

```
RestTemplate restTemplate = new RestTemplate(getClientHttpRequestFactory());

private ClientHttpRequestFactory getClientHttpRequestFactory() {
    int timeout = 5000;
    HttpComponentsClientHttpRequestFactory clientHttpRequestFactory
      = new HttpComponentsClientHttpRequestFactory();
    clientHttpRequestFactory.setConnectTimeout(timeout);
    return clientHttpRequestFactory;
}
```

我们可以将HttpClient用于其他配置选项-如下所示：

```
private ClientHttpRequestFactory getClientHttpRequestFactory() {
    int timeout = 5000;
    RequestConfig config = RequestConfig.custom()
      .setConnectTimeout(timeout)
      .setConnectionRequestTimeout(timeout)
      .setSocketTimeout(timeout)
      .build();
    CloseableHttpClient client = HttpClientBuilder
      .create()
      .setDefaultRequestConfig(config)
      .build();
    return new HttpComponentsClientHttpRequestFactory(client);
}
```

结语

我们遍历了主要的HTTP动词，使用RestTemplate来协调所有这些请求。

如果您想深入了解如何使用模板进行身份验证，请查看我关于使用RestTemplate进行基本身份验证文章。

所有这些示例和代码段的实现都可以在我的GitHub项目中找到–这是一个基于Maven的项目，因此应很容易直接导入和运行。