

Spring boot with Rest Template and Configuration

Check this out >> <https://github.com/fishyimm/spring-boot-with-rest-template>

1. Create spring boot project with simple dependencies (or you can create project spring boot by IDE)

The screenshot shows the 'Generate a' section of the Spring Boot Project Generator. It includes a 'Maven Project' dropdown, a 'with' dropdown set to 'Java', and a version dropdown set to '1.5.12'. Below this, the 'Project Metadata' section has input fields for 'Group' (com.example) and 'Artifact' (demo). The 'Dependencies' section on the right has a search bar with 'Web, Security, JPA, Actuator, DevTools...' and a 'Selected Dependencies' list containing 'Web' and 'DevTools'.

2. Create simple rest template configuration as below:

The screenshot shows an IDE with a project structure on the left and the code for RestConfig.java on the right. The project structure includes 'demo' with sub-projects 'src/main/java' and 'src/test/java'. The code for RestConfig.java is as follows:

```
package com.example.demo.config;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@Configuration
@EnableAutoConfiguration
public class RestConfig {

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
        RestTemplate template = restTemplateBuilder.build();
        return template;
    }
}
```

3. Create simple controller and inject rest template into controller.

thank you for the API from >> <https://spring.io/guides/gs/consuming-rest/>

The screenshot shows the code for TestController.java. The code is as follows:

```
@RestController
public class TestController {

    @Autowired
    RestTemplate restTemplate;

    public Quote test() {}

    @RequestMapping(value="/testString", method = RequestMethod.GET)
    public String testString() {

        HttpHeaders header = new HttpHeaders();
        header.set("test", "value");

        HttpEntity<String> request = new HttpEntity<>(header);

        ResponseEntity<String> resp = restTemplate.exchange("http://gturnquist-quoters.cfapps.io/api/random", HttpMethod.GET, request, String.class);
        return resp.getBody().toString();
    }
}
```

4. Try to access <http://localhost:8080/testString> and see the result.

The screenshot shows a web browser with the address bar set to 'localhost:8080/testString'. The response body is a JSON object: {"type": "success", "value": {"id": 4, "quote": "Previous to Spring Boot, I remember XML hell, confusing set up, and many hours of frustration."}}

5. Next try to response as JSON object create bean to handle the response

demo [boot] [devtools]
src/main/java
com.example.demo
com.example.demo.bean
Quote.java
Value.java
com.example.demo.config
CustomInterceptor.java
RestConfig.java
WebConfig.java
com.example.demo.controller
TestController.java
src/main/resources
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
bin
src
target
mvnw

```

package com.example.demo.bean;

public class Quote {
    private String type;
    private Value value;
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public Value getValue() {
        return value;
    }
    public void setValue(Value value) {
        this.value = value;
    }
    @Override
    public String toString() {
        return "Quote [type=" + type + ", value=" + value + "]";
    }
}

```

demo [boot] [devtools]
src/main/java
com.example.demo
com.example.demo.bean
Quote.java
Value.java
com.example.demo.config
CustomInterceptor.java
RestConfig.java
WebConfig.java
com.example.demo.controller
TestController.java
src/main/resources
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
bin
src
target
mvnw

```

package com.example.demo.bean;

public class Value {
    private Long id;
    private String quote;
    @Override
    public String toString() {
        return "Value [id=" + id + ", quote=" + quote + "]";
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getQuote() {
        return quote;
    }
    public void setQuote(String quote) {
        this.quote = quote;
    }
}

```

6. Add new method to same controller we have but this time response as JSON object as picture below:

```

@RestController
public class TestController {

    @Autowired
    RestTemplate restTemplate;

    @RequestMapping(value="/test", method = RequestMethod.GET)
    public Quote test() {
        HttpHeaders header = new HttpHeaders();
        header.set("test", "value");

        Map<String, String> map = new HashMap<String, String>();
        map.put("key", "value");

        HttpEntity<Map<String, String>> request = new HttpEntity<>(map, header);

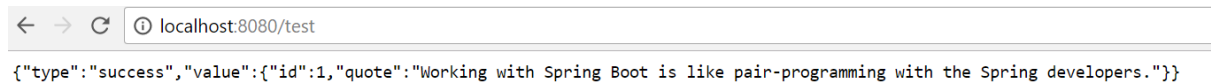
        ResponseEntity<Quote> resp = restTemplate.exchange("http://gturquist-quoters.cfapps.io/api/random", HttpMethod.GET, request, Quote.class);
        return resp.getBody();
    }

    public String testString() {}

    public Quote testquote() {}
}

```

7. Try to access <http://localhost:8080/test> and see the result, it may look the same as previous endpoint but the response type is difference.




```
{"type":"success","value":{"id":1,"quote":"Working with Spring Boot is like pair-programming with the Spring developers."}}}
```

8. Now we try to add some additional configuration to rest template, try add message converter to convert response message to JSON on every response

```
@Configuration
@EnableAutoConfiguration
public class RestConfig {
    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
        RestTemplate template = restTemplateBuilder
            .messageConverters(setMappingJackson2HttpMessageConverter())
            .build();
        return template;
    }

    private MappingJackson2HttpMessageConverter setMappingJackson2HttpMessageConverter() {
        MappingJackson2HttpMessageConverter mappingJackson2HttpMessageConverter = new MappingJackson2HttpMessageConverter();
        mappingJackson2HttpMessageConverter.getForObjectMapper().setSerializationInclusion(JsonInclude.Include.NON_NULL);
        mappingJackson2HttpMessageConverter.getForObjectMapper().setSerializationInclusion(JsonInclude.Include.NON_EMPTY);
        return mappingJackson2HttpMessageConverter;
    }
}
```

- now try to access <http://localhost:8080/testString> and see the result again. What happen here is method response type is string but we set rest template to convert message to JSON so the result will be cannot deserialize the message as error message in picture below:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Apr 13 23:06:24 ICT 2018
There was an unexpected error (type=Bad Request, status=400).
JSON parse error: Can not deserialize instance of java.lang.String out of START_OBJECT token; nested exception is com.fasterxml.jackson.databind.JsonMappingException: Can not deserialize instance of java.lang.String out of START_OBJECT token at [Source: java.io.PushbackInputStream@61e90583; line: 1, column: 1]

so when you want to configure rest template you need to understand your api purpose, if your api only return JSON format then you can configure rest template like this but if not better use simple configure is enough.

9. You can add connection, read timeout and message interceptor in rest template to log request and response as well.

```
package com.example.demo.config;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@Configuration
@EnableAutoConfiguration
public class RestConfig {

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
        RestTemplate template = restTemplateBuilder
            .interceptors(new CustomInterceptor())
            .setConnectTimeout(10000)
            .setReadTimeout(10000)
            .build();
        return template;
    }
}

package com.example.demo.config;

import java.io.BufferedReader;

public class CustomInterceptor implements ClientHttpRequestInterceptor {

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution)
        throws IOException {
        traceRequest(request, body);
        ClientHttpResponse response = execution.execute(request, body);
        traceResponse(response);
        return response;
    }

    private void traceRequest(HttpRequest request, byte[] body) throws IOException {
        System.out.println("=====request begin=====");
        System.out.println("URI          : " + request.getURI());
        System.out.println("Method       : " + request.getMethod());
        System.out.println("Headers      : " + request.getHeaders());
        System.out.println("Request body: " + new String(body, "UTF-8"));
        System.out.println("=====request end=====");
    }

    private void traceResponse(ClientHttpResponse response) throws IOException {
        StringBuilder inputStringBuilder = new StringBuilder();
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(response.getBody(), "UTF-8"));
        String line = bufferedReader.readLine();
        while (line != null) {
            inputStringBuilder.append(line);
            inputStringBuilder.append('\n');
            line = bufferedReader.readLine();
        }
        System.out.println("=====response begin=====");
        System.out.println("Status code  : {" + response.getStatusCode());
        System.out.println("Status text  : {" + response.getStatusText());
        System.out.println("Headers      : {" + response.getHeaders());
        System.out.println("Response body: {" + inputStringBuilder.toString());
        System.out.println("=====response end=====");
    }
}
```

10. Now try <http://localhost:8080/testString> and see the result. You gonna see in console log that request and response has been print out in console but it got some error as well

```
demo - DemoApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_162\bin\javaw.exe (Apr 13, 2018, 10:49:56 PM)
=====request begin=====
URI      : http://gturnquist-quoters.cfapps.io/api/random
Method    : GET
Headers   : {Accept=[text/plain, text/plain, application/json, application/json, application/*+json, application/*+json, */*, */*], test=[value], Content-Length=[0]}
Request body:
=====request end=====
=====response begin=====
Status code  : {}200
Status text  : {}OK
Headers      : {}Content-Type=[application/json;charset=UTF-8], Date=[Fri, 13 Apr 2018 16:19:09 GMT], Server=[Apache-Coyote/1.1], X-Vcap-Request-Id=[ab483980-bd09-4
Response body: {}{"type":"success","value":{"id":12,"quote":"@springboot with @springframework is pure productivity! Who said in #java one has to write double the co
=====response end=====
2018-04-13 23:19:10.475 ERROR 13824 --- [nio-8080-exec-7] o.a.c.c.C.[.][.].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context wit

java.lang.NullPointerException: null
    at com.example.demo.controller.TestController.testString(TestController.java:48) ~[classes/:na]
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:1.8.0_162]
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) ~[na:1.8.0_162]
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) ~[na:1.8.0_162]
    at java.lang.reflect.Method.invoke(Method.java:498) ~[na:1.8.0_162]
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:205) ~[spring-web-4.3.16.RELEASE.jar:4.3.16.RELEASE]
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:133) ~[spring-web-4.3.16.RELEASE.jar:4.3.16.RELEASE]
```

let check it out, The response stream is being read in logging interceptor already and it can be read once since it received in stream format that why `resp.getBody()` is null here.

```
39 @RequestMapping(value="/testString", method = RequestMethod.GET)
40 public String testString() {
41
42     HttpHeaders header = new HttpHeaders();
43     header.set("test", "value");
44
45     HttpEntity<String> request = new HttpEntity<>(header);
46
47     ResponseEntity<String> resp = restTemplate.exchange("http://gturnquist-quoters.cfapps.io/api/random", HttpMethod.GET, request, String.class);
48     return resp.getBody().toString();
49 }
```

So the solution to solve this issue is add `.requestFactory(new BufferingClientHttpRequestFactory(new SimpleClientHttpRequestFactory()))` in rest template and try it again as picture below:

```
1 @Configuration
2 @EnableAutoConfiguration
3 public class RestConfig {
4     @Bean
5     public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
6         RestTemplate template = restTemplateBuilder
7             .requestFactory(new BufferingClientHttpRequestFactory(new SimpleClientHttpRequestFactory()))
8             .interceptors(new CustomInterceptor())
9             .setConnectTimeout(10000)
10            .setReadTimeout(10000)
11            .build();
12         return template;
13     }
14 }
```

now we got the response already without any error.

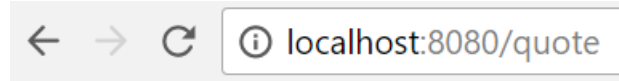
```
< - - - - - localhost:8080/testString
{"type": "success", "value": {"id": 1, "quote": "Working with Spring Boot is like pair-programming with the Spring developers."}}
```

11. Try access <http://localhost:8080/quote> and see the result.

```
@RequestMapping(value="/quote", method = RequestMethod.GET)
public Quote testquote() {

    Quote Quote = new Quote();
    Quote.setType("type");
    return Quote;
}
```

as you can see there is some null value in response JSON message

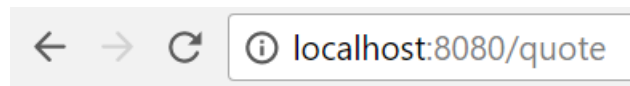


```
{"type":"type","value":null}
```

in order to remove null value from it you can use **@JsonIgnoreProperties** for every bean class you created or use **configureMessageConverters** to remove empty and null value in response message if you want to make it simple try this messageConverter as picture below:

```
5 @Configuration
6 @EnableAutoConfiguration
7 @EnableWebMvc
8 public class WebConfig extends WebMvcConfigurerAdapter {
9
10     @Override
11     public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {
12         final MappingJackson2HttpMessageConverter converter = new MappingJackson2HttpMessageConverter();
13         final ObjectMapper objectMapper = new ObjectMapper();
14         objectMapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);
15         objectMapper.setSerializationInclusion(JsonInclude.Include.NON_EMPTY);
16
17         converter.setObjectMapper(objectMapper);
18         converters.add(converter);
19         super.configureMessageConverters(converters);
20     }
21 }
22 }
```

now try <http://localhost:8080/quote> and see the result again there is no null value in response message as picture below:



```
{"type":"type"}
```

to use this configure it depend on whether you want to return null value to client or you don't want to.

Reference:

- <https://spring.io/guides/gs/consuming-rest/>
- <https://stackoverflow.com/questions/7952154/spring-resttemplate-how-to-enable-full-debugging-logging-of-requests-responses>
- <https://stackoverflow.com/questions/33372859/spring-resttemplate-bufferingclienthttprequestfactory-simpleclienthttprequestfactory>