# The Guide to RestTemplate

Last modified: October 25, 2017

by baeldung (http://www.baeldung.com/author/baeldung/)

**HttpClient (http://www.baeldung.com/category/http/)**

I just announced the new *Spring 5* modules in REST With Spring:

**>> CHECK OUT THE COURSE (/rest-with-spring-course#new-modules)**

## 1. Overview

In this tutorial, we're going to illustrate the broad range of operations where the Spring REST Client – *RestTemplate* – can be used, and used well.

For the API side of all examples, we'll be running the RESTful service from here (https://github.com/eugenp/tutorials/tree/master/spring-rest).

### Further reading:

### Basic Authentication with the RestTemplate (http://www.baeldung.com/how-to-use-resttemplate-with-basic-authentication-in-spring)

How to do Basic Authentication with the Spring RestTemplate.

**Read more (http://www.baeldung.com/how-to-use-resttemplate-with-basic-authentication-in-spring) →**

### RestTemplate with Digest Authentication (http://www.baeldung.com/how-to-set-up-digest-authentication)

How to set up Digest Authentication for the Spring RestTemplate using HttpClient 4.

**Read more (http://www.baeldung.com/resttemplate-digest-authentication) →**

### Exploring the Spring Boot TestRestTemplate (http://www.baeldung.com/spring-boot-testresttemplate)

Learn how to use the new TestRestTemplate in Spring Boot to test a simple API.

**Read more (http://www.baeldung.com/spring-boot-testresttemplate) →**

# 2. Use GET to Retrieve Resources

## 2.1. Get Plain JSON

Let's start simple and talk about GET requests – with a quick example using the *getForEntity()* API:

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

Notice that we have full access to the HTTP response – so we can do things like checking the status code to make sure the operation was actually successful, or work with the actual body of the response:

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

We're working with the response body as a standard String here – and using Jackson (and the JSON node structure that Jackson provides) to verify some details.

## 2.1. Retrieving POJO Instead of JSON

We can also map the response directly to a Resource DTO – for example:

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

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

Now – we can simply use the *getForObject* API in the template:

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

# 3. Use HEAD to Retrieve Headers

Let's now have a quick look at using HEAD before moving on to the more common methods – we're going to be using the *headForHeaders()* API here:

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

# 4. Use POST to Create a Resource

In order to create a new Resource in the API – we can make good use of the *postForLocation()*, *postForObject()* or *postForEntity()* APIs.

The first returns the URI of the newly created Resource while the second returns the Resource itself.

## 4.1. The *postForObject* API

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

HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));
```

```
5  Foo foo = restTemplate.postForObject(fooResourceUrl, request, Foo.class);
6  assertThat(foo, notNullValue());
7  assertThat(foo.getName(), is("bar"));
```

## 4.2. The *postForLocation* API

Similarly, let's have a look at the operation that – instead of returning the full Resource, just returns the *Location* of that newly created Resource:

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

## 4.3. The *exchange* API

Finally, let's have a look at how to do a POST with the more generic *exchange* API:

```
1  RestTemplate restTemplate = new RestTemplate();
2  HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));
3  ResponseEntity<Foo> response = restTemplate
4    .exchange(fooResourceUrl, HttpMethod.POST, request, Foo.class);
5
6  assertThat(response.getStatusCode(), is(HttpStatus.CREATED));
7
8  Foo foo = response.getBody();
9
10 assertThat(foo, notNullValue());
11 assertThat(foo.getName(), is("bar"));
```

# 5. Use OPTIONS to get Allowed Operations

Next, we're going to have a quick look at using an OPTIONS request and exploring the allowed operations on a specific URI using this kind of request; the API is *optionsForAllow*:

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

# 6. Use PUT to Update a Resource

Next, we'll start looking at PUT – and more specifically the *exchange* API for this operation, because of the *template.put* API is pretty straightforward.

## 6.1. Simple PUT with *.exchange*

We'll start with a simple PUT operation against the API – and keep in mind that the operation isn't returning anybody back to the client:

```
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. PUT with *.exchange* and a Request Callback

Next, we're going to be using a request callback to issue a PUT.

Let's make sure we prepare the callback – where we can set all the headers we need as well as a request body:

```
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());
    };
}
```

Next, we create the Resource with POST request:

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

And then we update the Resource:

```
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. Use DELETE to Remove a Resource

To remove an existing Resource we'll make short work of the *delete()* API:

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

# 8. Configure Timeout

We can configure *RestTemplate* to time out by simply using *ClientHttpRequestFactory* – as follows:

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

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

And we can use *HttpClient* for further configuration options – as follows:

```
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);
}
```

# 9. Conclusion

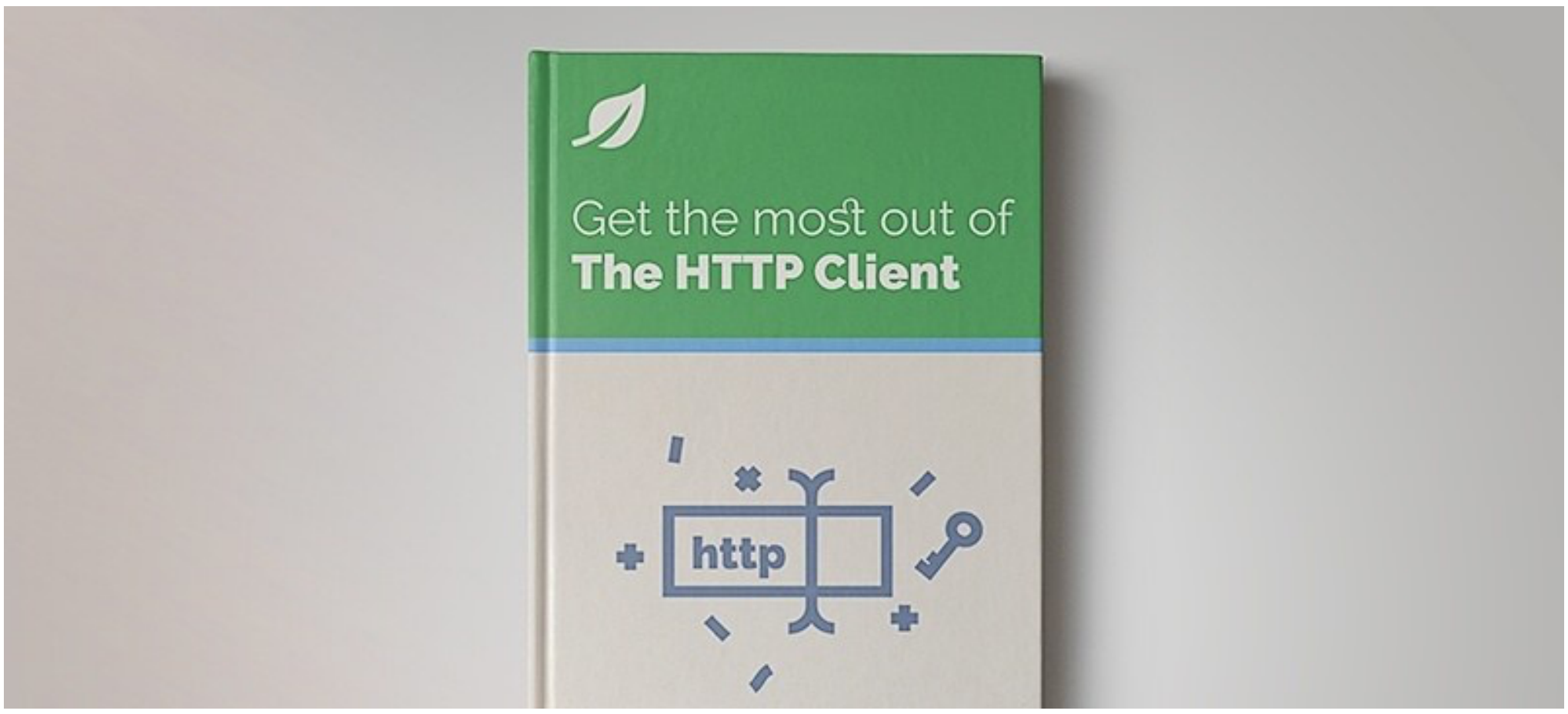
We went over the main HTTP Verbs, using *RestTemplate* to orchestrate requests using all of these.

If you want to dig into how to do authentication with the template – check out my write-up on Basic Auth with RestTemplate (/how-to-use-resttemplate-with-basic-authentication-in-spring).

The implementation of all these examples and code snippets **can be found in my GitHub project (https://github.com/eugenp/tutorials/tree/master/spring-rest-full)** – this is a Maven-based project, so it should be easy to import and run as it is.

## I just announced the new Spring 5 modules in REST With Spring:

**>> CHECK OUT THE LESSONS (/rest-with-spring-course#new-modules)**





**Download the free e-book**

# Get the most out of the HTTP Client

Email Address

Download

Subscribe ▾ ▲ newest ▲ oldest ▲ most voted

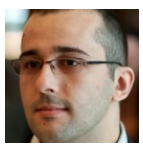
Guest

## springusr

Lets say we called exchange method, now that is returning 401, but response has body which has some status information. Current implementation throws exception – so how to parse that body. How to make/force rest tempate to make transform response for different response codes if possible.

+ 0 −

1 year ago



Guest

### Eugen Paraschiv (http://www.baeldu...

That behavior comes from the default error handler: *DefaultResponseErrorHandler* – which you can replace with your own custom implementation via the *setErrorHandler* API.
Hope that helps. Cheers,
Eugen.

+ 0 −

1 year ago

Guest

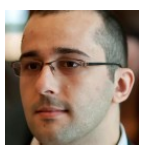
## Terry

I needed to write a client that did chunked uploading of arbitrarily large files. But I could not find an example of that anywhere on the internet. I ended up writing it in a non-spring way. Could something like your request callback example do that?

I suppose a followup question is how would one write the server side for chunked processing.

+ 0 −

1 year ago



Guest

### Eugen Paraschiv (http://www.baeldu...

Hey Terry, That sounds like an interesting scenario, but unfortunately one that's a bit to open-ended to be able to answer here in the comments. At a high level, one suggestion I have is – if you're dealing with lower level operations, you might want to have a look at the HttpClient as well. As for the server side – if I understand your question correctly, you're basically asking how do you do streaming. There's really no one right answer here. You can have a look at various arguments supported in Spring MVC methods, WebSockets, etc. Hope that points you… Read more »

+ 0 −

1 year ago

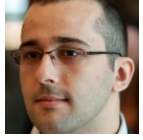
Guest

## Chris Borghmans

We need 2 rest templates in our app, with different timeout settings would it be best to create 2 beans in our spring config for restTemplate and restTemplateLongTimeout or should we always create a new template in the service were we need it?

+ 0 −

1 year ago



Guest

Eugen Paraschiv (http://www.baeldu...

Hey Chris,

RestTemplate isn't complex enough to make the timeout easily configurable per request, so yes, the easiest solution here would be to create two separate beans, and just wire the one you know you need.

Hope that helps. Cheers,

Eugen.

+ 1 −

1 year ago


(HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)

(HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

(HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)

C

(HTTP://WWW.BAELDUNG.COM/CONTACT)

C

I

(HTTP://WWW.BAELDUNG.COM/BAELDUNG-

COMPANY-

INFO)

T

C

S

(HTTP://WWW.BAELDUNG.COM/TERMS-

OF-

SERVICE)

P

P

(HTTP://WWW.BAELDUNG.COM/PRIVACY-

POLICY)