(/)

# **Download a Large File** Through a Spring RestTemplate

Last modified: June 16, 2019

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

**Spring (https://www.baeldung.com/category/spring/)** +

**RestTemplate (https://www.baeldung.com/tag/resttemplate/)**

## 1. Overview

In this tutorial, we're going to show different techniques on how to download large files (https://www.baeldung.com/java-download-file) with *RestTemplate*.

## 2. *RestTemplate*

*RestTemplate (https://www.baeldung.com/rest-template)* is a blocking and synchronous HTTP Client introduced in Spring 3. According to the Spring documentation (https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html), it'll be deprecated in the future since they've introduced *WebClient (https://www.baeldung.com/spring-5-webclient)* as a reactive nonblocking HTTP client in version 5.

## 3. Pitfalls

Usually, when we download a file, we store it on our file system or load it into memory as a byte array. But when it's a large file, in-memory loading may lead to an *OutOfMemoryError (https://www.baeldung.com/java-gc-overhead-limit-exceeded)*. Hence, we have to store data in a file as we read chunks of response.

Let's first look at a couple of ways that don't work:

First, what happens if we return a *Resource* as our return type:

```
Resource download() {
    return new ClassPathResource(locationForLargeFile);
}
```

The reason this doesn't work is that *ResourceHttpMesssageConverter* **will load the entire response body into a *ByteArrayInputStream*** still adding the memory pressure we wanted to avoid.

Second, what if we return an *InputStreamResource* and configure *ResourceHttpMessageConverter#supportsReadStreaming*? Well, this doesn't work either since **by the time we can call *InputStreamResource.getInputStream()*, we get a "*socket closed"* error!** This is because the *"execute*" closes the response input stream before the exit.

So what can we do to solve the problem? Actually, there are two things here, too:

- **Write a custom *HttpMessageConverter* (https://www.baeldung.com/spring-httpmessageconverter-rest)** that supports *File* as a return type
- Use *RestTemplate.execute* with **a custom *ResponseExtractor*** to store the input stream in a *File*

In this tutorial, we'll use the second solution because it is more flexible and also needs less effort.

## 4. Download Without Resume

Let's implement a *ResponseExtractor* to write the body to a temporary file*:*

```
File file = restTemplate.execute(FILE_URL, HttpMethod.GET, null, clie
    File ret = File.createTempFile("download", "tmp");
    StreamUtils.copy(clientHttpResponse.getBody(), new FileOutputStre
    return ret;
});

Assert.assertNotNull(file);
Assertions
  .assertThat(file.length())
  .isEqualTo(contentLength);
```

Here we have used the *StreamUtils.copy* to copy the response input stream in a *FileOutputStream*, but other techniques and libraries (https://www.baeldung.com/convert-input-stream-to-a-file) are also available.

## 5. Download with Pause and Resume

As we're going to download a large file, it's reasonable to consider downloading after we've paused for some reason.

So first let's check if the download URL supports resume:

```
HttpHeaders headers = restTemplate.headForHeaders(FILE_URL);

Assertions
  .assertThat(headers.get("Accept-Ranges"))
  .contains("bytes");
Assertions
  .assertThat(headers.getContentLength())
  .isGreaterThan(0);
```

Then we can implement a *RequestCallback* to set “Range” header and resume the download:

```
restTemplate.execute(
  FILE_URL,
  HttpMethod.GET,
  clientHttpRequest -> clientHttpRequest.getHeaders().set(
    "Range",
    String.format("bytes=%d-%d", file.length(), contentLength)),
    clientHttpResponse -> {
        StreamUtils.copy(clientHttpResponse.getBody(), new FileOutput
    return file;
});

Assertions
  .assertThat(file.length())
  .isLessThanOrEqualTo(contentLength);
```

If we don't know the exact content length, we can set the *Range* header value using *String.format*:

```
String.format("bytes=%d-", file.length())
```

## 6. Conclusion

We've discussed problems that can arise when downloading a large file. We also presented a solution while using *RestTemplate*. Finally, we've shown how we can implement a resumable download.

As always the code is available in our GitH (https://github.com/eugenp/tutorials/tree/master/spring-resttemplate)ub (https://github.com/eugenp/tutorials/tree/master/spring-resttemplate).

Learning to build your API

**with Spring**?

Enter your email address

**>> Get the eBook**

## Leave a Reply

Start the discussion...

Subscribe

## CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

## SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

## ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)