(/)

Access a File from the Classpath in a Spring Application

Last modified: June 18, 2018

by baeldung (/author/baeldung/)

Spring (/category/spring/) +

1. Introduction

In this tutorial, we'll see various ways to access and load the contents of a file that's on the classpath using Spring.

2. Using *Resource*

The *Resource* interface helps in abstracting access to low-level resources. In fact, it supports handling of all kinds of file resources in a uniform manner.

Let's start by looking at various methods to obtain a *Resource* instance.

2.1. Manually

For accessing a resource from the classpath, we can simply use *ClassPathResource*:

```
public Resource loadEmployees() {
    return new ClassPathResource("data/employees.dat");
}
```

By default, *ClassPathResource* removes some boilerplate for selecting between the thread's context classloader and the default system classloader.

However, we can also indicate the classloader to use either directly:

```
return new ClassPathResource("data/employees.dat", this.getClass().get
```

Or indirectly through a specified class:

```
return new ClassPathResource(
  "data/employees.dat",
  Employee.class.getClassLoader());
```

Note that from *Resource*, we can easily jump to Java standard representations like *InputStream* or *File*.

2.2. Using *@Value*

We can also inject a *Resource* with *@Value*:

```
@Value("classpath:data/resource-data.txt")
Resource resourceFile;
```

And *@Value* supports other prefixes, too, like *file:* and *url:*.

2.3. Using ResourceLoader

Or, if we want to lazily load our resource, we can use *ResourceLoader*:

```
@Autowired
ResourceLoader resourceLoader;
```

And then we retrieve our resource with *getResource*:

```
public Resource loadEmployees() {
    return resourceLoader.getResource(
      "classpath:data/employees.dat");
}
```

Note, too that *ResourceLoader* is implemented by all concrete *ApplicationContext*s, which means that we can also simply depend on *ApplicationContext* if that suits our situation better:

```
ApplicationContext context;

public Resource loadEmployees() {
    return context.getResource("classpath:data/employees.dat");
}
```

3. Using *ResourceUtils*

As a caveat, there is another way to retrieve resources in Spring, but the *ResourceUtils* Javadoc (https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/util/ResourceUtils.html) is clear that the class is mainly for internal use.

If we see usages of *ResourceUtils* in our code:

```
public File loadEmployeesWithSpringInternalClass()
  throws FileNotFoundException {
    return ResourceUtils.getFile(
      "classpath:data/employees.dat");
}
```

We should carefully consider the rationale as **it's probably better to use one of the standard approaches above**.

4. Reading Resource Data

Once we have a *Resource,* it's easy for us to read the contents. As we have already discussed, we can easily obtain a *File* or an *InputStream* reference from the *Resource*.

Let's imagine we have the following file, *data/employees.dat*, on the classpath*:*

```
Joe Employee,Jan Employee,James T. Employee
```

4.1. Reading as a *File*

Now, we can read its contents by calling *getFile:*

```
@Test
public void whenResourceAsFile_thenReadSuccessful()
  throws IOException {

    File resource = new ClassPathResource(
      "data/employees.dat").getFile();
    String employees = new String(
      Files.readAllBytes(resource.toPath()));
    assertEquals(
      "Joe Employee,Jan Employee,James T. Employee",
      employees);
}
```

Although, note that this approach **expects the resource to be present in the filesystem and not within a jar file.**

4.2. Reading as an *InputStream*

Let's say, though, that our resource *is* inside a jar.

Then, we can instead read a *Resource* as an *InputStream*:

```
@Test
public void whenResourceAsStream_thenReadSuccessful()
  throws IOException {
    InputStream resource = new ClassPathResource(
      "data/employees.dat").getInputStream();
    try ( BufferedReader reader = new BufferedReader(
      new InputStreamReader(resource)) ) {
        String employees = reader.lines()
          .collect(Collectors.joining("\n"));

        assertEquals("Joe Employee,Jan Employee,James T. Employee", e
    }
}
```

5. Conclusion

In this quick article, we've seen a few ways to access and read a resource from the classpath using Spring including eager and lazy loading and on the filesystem or in a jar.

And, as always, I've posted all these examples over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-core).

Leave a Reply

Start the discussion...

Subscribe

CATEGORIES

SPRING (/CATEGORY/SPRING/)
REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JACKSON/)
HTTPCLIENT (/CATEGORY/HTTP/)
KOTLIN (/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 (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
CONTACT (/CONTACT)
EDITORS (/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-
+MEDIA+KIT.PDF)