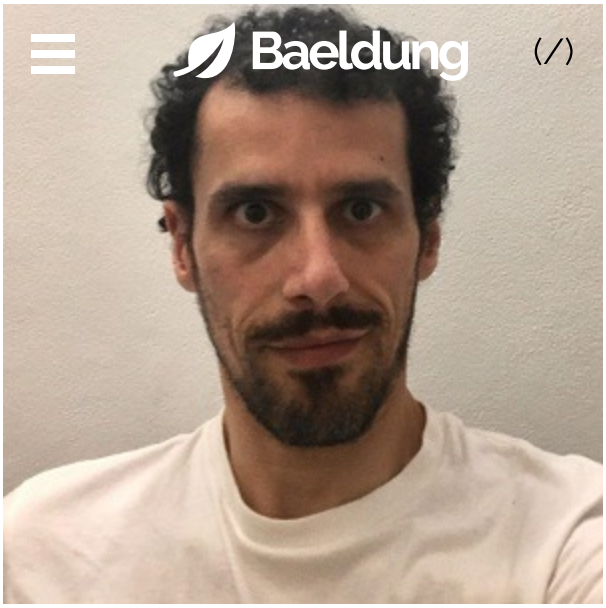
(/)

Spring @PathVariable Annotation

Last updated: January 8, 2024

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



> Reviewed by: Bruno Fontana (https://www.baeldung.com/editor/brunofontana)

Spring MVC (https://www.baeldung.com/category/spring/spring-web/spring-mvc)

Spring MVC Basics (https://www.baeldung.com/tag/spring-mvc-basics)

1. Overview

In this quick tutorial, we'll explore Spring's *@PathVariable* annotation.

Simply put, **the *@PathVariable* annotation can be used to handle template variables in the request URI mapping**, and set them as method parameters.

Let's see how to use *@PathVariable* and its various attributes.

(/)

Further reading:

Spring @RequestParam vs @PathVariable Annotations (/spring-requestparam-vs-pathvariable)

Understand the differences between Spring's @RequestParam and @PathVariable annotations.

Read more (/spring-requestparam-vs-pathvariable) →

Validating RequestParams and PathVariables in Spring (/spring-validate-requestparam-pathvariable)

Learn how to validate request parameters and path variables with Spring MVC

Read more (/spring-validate-requestparam-pathvariable) →

Spring MVC @PathVariable with a dot (.) gets truncated (/spring-mvc-pathvariable-dot)

Learn how to handle path variables that contain a dot in Spring MVC request mappings.

Read more (/spring-mvc-pathvariable-dot) →

2. A Simple Mapping

A simple use case of the *@PathVariable* annotation would be an endpoint that identifies an entity with a primary key:

```
@GetMapping("/api/employees/{id}")
@ResponseBody
public String getEmployeesById(@PathVariable String id) {
    return "ID: " + id;
}
```

In this example, we use the *@PathVariable* annotation to extract the templated part of the URI, represented by the variable *{id}*.

A simple *GET request* to */api/employees/{id}* will invoke *getEmployeesById* with the extracted id value:

```
http://localhost:8080/api/employees/111
----
ID: 111
```

Now let's further explore this annotation, and have a look at its attributes.

3. Specifying the Path Variable Name

In the previous example, we skipped defining the name of the template path variable since the names for the method parameter and the path variable were the same.

However, if the path variable name is different, we can specify it in the argument of the *@PathVariable* annotation:

```
@GetMapping("/api/employeeswithvariable/{id}")
@ResponseBody
public String getEmployeesByIdWithVariableName(@PathVariable("id")
String employeeId) {
    return "ID: " + employeeId;
}
```

```
http://localhost:8080/api/employeeswithvariable/1
----
ID: 1
```

We can also define the path variable name as *@PathVariable(value="id")* instead of *PathVariable("id")* for clarity.

4. Multiple Path Variables in a Single Request

Depending on the use case, **we can have more than one path variable in our request URI for a controller method, which also has multiple method parameters**:

```
@GetMapping("/api/employees/{id}/{name}")
@ResponseBody
public String getEmployeesByIdAndName(@PathVariable String id,
@PathVariable String name) {
    return "ID: " + id + ", name: " + name;
}
```

```
http://localhost:8080/api/employees/1/bar
----
ID: 1, name: bar
```

We can also handle more than one *@PathVariable* parameter using a method parameter of type *java.util.Map<String, String>*:

```
@GetMapping("/api/employeeswithmapvariable/{id}/{name}")
@ResponseBody
public String getEmployeesByIdAndNameWithMapVariable(@PathVariable
Map<String, String> pathVarsMap) {
    String id = pathVarsMap.get("id");
    String name = pathVarsMap.get("name");
    if (id != null && name != null) {
        return "ID: " + id + ", name: " + name;
    } else {
        return "Missing Parameters";
    }
}
```

```
http://localhost:8080/api/employees/1/bar
----
ID: 1, name: bar
```

There is, however, a small catch while handling multiple *@PathVariable* parameters when the path variable string contains a dot(.) character. We've discussed those corner cases in detail here (/spring-mvc-pathvariable-dot).

5. Optional Path Variables

In Spring, method parameters annotated with *@PathVariable* are required by default:

```
@GetMapping(value = { "/api/employeeswithrequired",
"/api/employeeswithrequired/{id}" })
@ResponseBody
public String getEmployeesByIdWithRequired(@PathVariable String id) {
    return "ID: " + id;
}
```

Given how it looks, the above controller should handle both */api/employeeswithrequired* and */api/employeeswithrequired/1* request paths. However, since method parameters annotated by *@PathVariables* are mandatory by default, it doesn't handle the requests sent to the */api/employeeswithrequired* path:

```
http://localhost:8080/api/employeeswithrequired
----
{"timestamp":"2020-07-08T02:20:07.349+00:00","status":404,"error":"Not
Found","message":"","path":"/api/employeeswithrequired"}


http://localhost:8080/api/employeeswithrequired/1
----
ID: 111
```

We can handle this in two different ways.

5.1. Setting *@PathVariable* as Not Required

We can set the *required* property of *@PathVariable* to *false* to make it optional. Thus, modifying our previous example, we can now handle the URI versions with and without the path variable:

```
@GetMapping(value = { "/api/employeeswithrequiredfalse",
"/api/employeeswithrequiredfalse/{id}" })
@ResponseBody
public String getEmployeesByIdWithRequiredFalse(@PathVariable(required
= false) String id) {
    if (id != null) {
        return "ID: " + id;
    } else {
        return "ID missing";
    }
}
```

```
http://localhost:8080/api/employeeswithrequiredfalse
----
ID missing
```

5.2. Using *java.util.Optional*

Since the introduction of Spring 4.1, we can also use *java.util.Optional<T>* (/java-optional) (available in Java 8+) to handle a non-mandatory path variable:

```
@GetMapping(value = { "/api/employeeswithoptional",
"/api/employeeswithoptional/{id}" })
@ResponseBody
public String getEmployeesByIdWithOptional(@PathVariable
Optional<String> id) {
    if (id.isPresent()) {
        return "ID: " + id.get();
    } else {
        return "ID missing";
    }
}
```

Now if we don't specify the path variable *id* in the request, we get the default response:

```
http://localhost:8080/api/employeeswithoptional
----
ID missing
```

5.3. Using a Method Parameter of Type *Map<String, String>*

As shown earlier, we can use a single method parameter of type *java.util.Map* to handle all the path variables in the request URI. **We can also use this strategy to handle the optional path variables case:**

```
@GetMapping(value = { "/api/employeeswithmap/{id}",
"/api/employeeswithmap" })
@ResponseBody
public String getEmployeesByIdWithMap(@PathVariable Map<String,
String> pathVarsMap) {
    String id = pathVarsMap.get("id");
    if (id != null) {
        return "ID: " + id;
    } else {
        return "ID missing";
    }
}
```

6. Default Value for *@PathVariable*

Out of the box, there isn't a provision to define a default value for method parameters annotated with *@PathVariable*. However, we can use the same strategies discussed above to satisfy the default value case for *@PathVariable*, we just need to check for *null* on the path variable.

For instance, using *java.util.Optional<String, String>*, we can identify if the path variable is *null* or not. If it is *null*, then we can just respond to the request with a default value:

```
@GetMapping(value = { "/api/defaultemployeeswithoptional",
"/api/defaultemployeeswithoptional/{id}" })
@ResponseBody
public String getDefaultEmployeesByIdWithOptional(@PathVariable
Optional<String> id) {
    if (id.isPresent()) {
        return "ID: " + id.get();
    } else {
        return "ID: Default Employee";
    }
}
```

7. Conclusion

In this article, we discussed how to use Spring's *@PathVariable* annotation. We also identified the various ways to effectively use the *@PathVariable* annotation to suit different use cases, such as optional parameters and dealing with default values.

The code example shown in this article is also available over on Github (https://github.com/eugenp/tutorials/tree/master/spring-web-modules/spring-mvc-java).

Learning to build your API
with Spring?

Download the E-book (/rest-api-spring-guide)

Comments are closed on this article!

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)

APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)

REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)

SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)

SECURITY WITH SPRING (/SECURITY-SPRING)

SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)

THE FULL ARCHIVE (/FULL_ARCHIVE)

EDITORS (/EDITORS)

JOBS (/TAG/ACTIVE-JOB/)

OUR PARTNERS (/PARTNERS)

PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)

PRIVACY POLICY (/PRIVACY-POLICY)

COMPANY INFO (/BAELDUNG-COMPANY-INFO)

CONTACT (/CONTACT)