(/)

Spring Web Annotations

Last updated: January 8, 2024

Written by: Attila Fejér (https://www.baeldung.com/author/attila-fejer)

Reviewed by: Michal Aibin (https://www.baeldung.com/editor/michal-author)

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

reference

Spring Annotations (https://www.baeldung.com/tag/spring-annotations)

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

This article is part of a series: (/)

1. Overview

In this tutorial, we'll explore Spring Web annotations from the *org.springframework.web.bind.annotation* package.

2. *@RequestMapping*

Simply put, *@RequestMapping* (/spring-requestmapping) **marks request handler methods** inside *@Controller* classes; it can be configured using:

path, or its aliases, *name,* and *value:* which URL the method is mapped to
method: compatible HTTP methods
params: filters requests based on presence, absence, or value of HTTP parameters
headers: filters requests based on presence, absence, or value of HTTP headers
consumes: which media types the method can consume in the HTTP request body
produces: which media types the method can produce in the HTTP response body

Here's a quick example of what that looks like:

```
@Controller
class VehicleController {

    @RequestMapping(value = "/vehicles/home", method =
RequestMethod.GET)
    String home() {
        return "home";
    }
}
```

We can provide **default settings for all handler methods in a *@Controller* class** if we apply this annotation on the class level. The only **exception is the URL which Spring won't override** with method level settings but appends the two path parts.

For example, the following configuration has the same effect as the one above:

```
@Controller
@RequestMapping(value = "/vehicles", method = RequestMethod.GET)
class VehicleController {

    @RequestMapping("/home")
    String home() {
        return "home";
    }
}
```

Moreover, *@GetMapping*, *@PostMapping*, *@PutMapping*, *@DeleteMapping*, and *@PatchMapping* are different variants of *@RequestMapping* with the HTTP method already set to GET, POST, PUT, DELETE, and PATCH respectively.

These are available since Spring 4.3 release.

3. *@RequestBody*

Let's move on to *@RequestBody* (/spring-request-response-body) – which maps the **body of the HTTP request to an object**:

```
@PostMapping("/save")
void saveVehicle(@RequestBody Vehicle vehicle) {
    // ...
}
```

The deserialization is automatic and depends on the content type of the request.

4. @PathVariable (/)

Next, let's talk about *@PathVariable*.

This annotation indicates that a **method argument is bound to a URI template variable**. We can specify the URI template with the *@RequestMapping* annotation and bind a method argument to one of the template parts with *@PathVariable*.

We can achieve this with the *name* or its alias, the *value* argument:

```
@RequestMapping("/{id}")
Vehicle getVehicle(@PathVariable("id") long id) {
    // ...
}
```

If the name of the part in the template matches the name of the method argument, we don't have to specify it in the annotation:

```
@RequestMapping("/{id}")
Vehicle getVehicle(@PathVariable long id) {
    // ...
}
```

Moreover, we can mark a path variable optional by setting the argument *required* to false:

```
@RequestMapping("/{id}")
Vehicle getVehicle(@PathVariable(required = false) long id) {
    // ...
}
```

5. @RequestParam

We use *@RequestParam* for **accessing HTTP request parameters**:

```
@RequestMapping("/")
Vehicle getVehicleByParam(@RequestParam("id") long id) {
    // ...
}
```

It has the same configuration options as the *@PathVariable* annotation.

In addition to those settings, with *@RequestParam* we can specify an injected value when Spring finds no or empty value in the request. To achieve this, we have to set the *defaultValue* argument.

Providing a default value implicitly sets *required* to *false:*

```
@RequestMapping("/buy")
Car buyCar(@RequestParam(defaultValue = "5") int seatCount) {
    // ...
}
```

Besides parameters, there are **other HTTP request parts we can access: cookies and headers**. We can access them with the annotations ***@CookieValue* and *@RequestHeader*** respectively.

We can configure them the same way as *@RequestParam*.

6. Response Handling Annotations

In the next sections, we will see the most common annotations to manipulate HTTP responses in Spring MVC.

6.1. *@ResponseBody*

If we mark a request handler method with *@ResponseBody (/spring-request-response-body),* **Spring treats the result of the method as the response itself**:

```
@ResponseBody                    (/)
@RequestMapping("/hello")
String hello() {
    return "Hello World!";
}
```

If we annotate a *@Controller* class with this annotation, all request handler methods will use it.

6.2. *@ExceptionHandler*

With this annotation, we can declare a **custom error handler method**. Spring calls this method when a request handler method throws any of the specified exceptions.

The caught exception can be passed to the method as an argument:

```
@ExceptionHandler(IllegalArgumentException.class)
void onIllegalArgumentException(IllegalArgumentException exception) {
    // ...
}
```

6.3. *@ResponseStatus*

We can specify the **desired HTTP status of the response** if we annotate a request handler method with this annotation. We can declare the status code with the *code* argument, or its alias, the *value* argument.

Also, we can provide a reason using the *reason* argument.

We also can use it along with *@ExceptionHandler*:

```
@ExceptionHandler(IllegalArgumentException.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
void onIllegalArgumentException(IllegalArgumentException exception) {
    // ...
}
```

For more information about HTTP response status, please visit this article (/spring-mvc-controller-custom-http-status-code).

7. Other Web Annotations

Some annotations don't manage HTTP requests or responses directly. In the next sections, we'll introduce the most common ones.

7.1. *@Controller*

We can define a Spring MVC controller with *@Controller*. For more information, please visit our article about Spring Bean Annotations (/spring-bean-annotations).

7.2. *@RestController*

The *@RestController* **combines *@Controller* and *@ResponseBody***.

Therefore, the following declarations are equivalent:

```
@Controller
@ResponseBody
class VehicleRestController {
    // ...
}
```

```
@RestController
class VehicleRestController {
    // ...
}
```

7.3. *@ModelAttribute*

With this annotation we can **access elements that are already in the model** of an MVC *@Controller*, by providing the model key:

```
@PostMapping("/assemble")
void assembleVehicle(@ModelAttribute("vehicle") Vehicle
vehicleInModel) {
    // ...
}
```

Like with *@PathVariable* and *@RequestParam*, we don't have to specify the model key if the argument has the same name:

```
@PostMapping("/assemble")
void assembleVehicle(@ModelAttribute Vehicle vehicle) {
    // ...
}
```

Besides, *@ModelAttribute* has another use: if we annotate a method with it, Spring will **automatically add the method's return value to the model**:

```
@ModelAttribute("vehicle")
Vehicle getVehicle() {
    // ...
}
```

Like before, we don't have to specify the model key, Spring uses the method's name by default:

```
@ModelAttribute
Vehicle vehicle() {
    // ...
}
```

Before Spring calls a request handler method, it invokes all *@ModelAttribute* annotated methods in the class.

More information about *@ModelAttribute* can be found in this article (/spring-mvc-and-the-modelattribute-annotation).

7.4. @CrossOrigin (/)

@CrossOrigin **enables cross-domain communication** for the annotated request handler methods:

```
@CrossOrigin
@RequestMapping("/hello")
String hello() {
    return "Hello World!";
}
```

If we mark a class with it, it applies to all request handler methods in it.

We can fine-tune CORS behavior with this annotation's arguments.

For more details, please visit this article (/spring-cors).

8. Conclusion

In this article, we saw how we can handle HTTP requests and responses with Spring MVC.

As usual, the examples are available over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-boot-modules/spring-boot-annotations).

Next »
Spring Boot Annotations (/spring-boot-annotations)

« Previous
Spring Core Annotations (/spring-core-annotations)

(/)

Get started with Spring and Spring Boot, through the *Learn Spring* course:

>> CHECK OUT THE COURSE (/ls-course-end)

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)