(/)

Spring MVC and the @ModelAttribute Annotation

Last updated: February 23, 2022

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

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

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

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

(/)

1. Overview

One of the most important Spring MVC (http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html) annotations is the @ModelAttribute (http://docs.spring.io/spring/docs/3.1.x/javadoc-api/org/springframework/web/bind/annotation/ModelAttribute.html) annotation.

@ModelAttribute is an annotation that binds a method parameter or method return value to a named model attribute, and then exposes it to a web view.

In this tutorial, we'll demonstrate the usability and functionality of this annotation through a common concept, a form submitted from a company's employee.

Further reading:

Model, ModelMap, and ModelAndView in Spring MVC (/spring-mvc-model-model-map-model-view)

Learn about the interfaces Model, ModelMap and ModelAndView provided by Spring MVC.

Read more (/spring-mvc-model-model-map-model-view) →

Spring @RequestParam Annotation (/spring-request-param)

A detailed guide to Spring's @RequestParam annotation

Read more (/spring-request-param) →

2. *@ModelAttribute* in Depth

As the introductory paragraph revealed, we can use *@ModelAttribute* either as a method parameter or at the method level.

2.1. At the Method Level

When we use the annotation at the method level, it indicates the purpose of the method is to add one or more model attributes. Such methods support the same argument types as @RequestMapping (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html) methods, but they can't be mapped directly to requests.

Let's look at a quick example here to understand how this works:

```
@ModelAttribute
public void addAttributes(Model model) {
    model.addAttribute("msg", "Welcome to the Netherlands!");
}
```

In the above example, we see a method that adds an attribute named *msg* to all *model*s defined in the controller class.

Of course, we'll see this in action later in the article.

In general, Spring MVC will always make a call to that method first, before it calls any request handler methods. Basically, ***@ModelAttribute* methods are invoked before the controller methods annotated with *@RequestMapping* are invoked.** This is because the model object has to be created before any processing starts inside the controller methods.

It's also important that we annotate the respective class as @ControllerAdvice (http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html). Thus, we can add values in *Model* that'll be identified as global. This actually means that for every request, a default value exists for every method in the response.

2.2. As a Method Argument

When we use the annotation as a method argument, it indicates to retrieve the argument from the model. When the annotation isn't present, it should first be instantiated, and then added to the model. Once present in the model, the arguments fields should populate from all request parameters that have matching names.

In the following code snippet, we'll populate the *employee* model attribute with data from a form submitted to the *addEmployee* endpoint. Spring MVC does this behind the scenes before invoking the submit method:

```
@RequestMapping(value = "/addEmployee", method = RequestMethod.POST)
public String submit(@ModelAttribute("employee") Employee employee) {
    // Code that uses the employee object

    return "employeeView";
}
```

Later in this article, we'll see a complete example of how to use the *employee* object to populate the *employeeView* template.

It binds the form data with a bean. **The controller annotated with *@RequestMapping* can have custom class argument(s) annotated with *@ModelAttribute*.**

In Spring MVC, we refer to this as data binding, a common mechanism that saves us from having to parse each form field individually.

3. Form Example

In this section, we'll look at the example outlined in the overview section, a very basic form that prompts a user (specifically a company employee) to enter some personal information (specifically *name* and *id*). After the submission is complete, and without any errors, the user expects to see the previously submitted data displayed on another screen.

3.1. The View

Let's first create a simple form with id and name fields:

```
<form:form method="POST" action="/spring-mvc-basics/addEmployee"
  modelAttribute="employee">
    <form:label path="name">Name</form:label>
    <form:input path="name" />

    <form:label path="id">Id</form:label>
    <form:input path="id" />

    <input type="submit" value="Submit" />
</form:form>
```

3.2. The Controller

Here's the controller class, where we'll implement the logic for the aforementioned view:

```
@Controller
@ControllerAdvice
public class EmployeeController {

    private Map<Long, Employee> employeeMap = new HashMap<>();

    @RequestMapping(value = "/addEmployee", method =
RequestMethod.POST)
    public String submit(
      @ModelAttribute("employee") Employee employee,
      BindingResult result, ModelMap model) {
        if (result.hasErrors()) {
            return "error";
        }
        model.addAttribute("name", employee.getName());
        model.addAttribute("id", employee.getId());

        employeeMap.put(employee.getId(), employee);

        return "employeeView";
    }

    @ModelAttribute
    public void addAttributes(Model model) {
        model.addAttribute("msg", "Welcome to the Netherlands!");
    }
}
```

In the *submit()* method, we have an *Employee* object bound to our *View*. We can map our form fields to an object model as simply as that. In the method, we're fetching values from the form and setting them to *ModelMap (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/ui/ModelMap.html)*.

In the end, we return *employeeView*, which means that we call the respective JSP file as a *View* representative.

Furthermore, there's also an *addAttributes()* method. Its purpose is to add values in the *Model* that'll be identified globally. That is, every request to every controller method will return a default value as a response. We also have to annotate the specific class as *@ControllerAdvice*.

3.3. The Model

As previously mentioned, the *Model* object is very simple and contains all that the "front-end" attributes require. Now let's have a look at an example:

```
@XmlRootElement
public class Employee {

    private long id;
    private String name;

    public Employee(long id, String name) {
        this.id = id;
        this.name = name;
    }

    // standard getters and setters removed
}
```

3.4. Wrap Up

@ControllerAdvice assists a controller, and in particular, *@ModelAttribute* methods that apply to all *@RequestMapping* methods. Of course, our *addAttributes()* method will be the very first to run, prior to the rest of the *@RequestMapping* methods.

Keeping that in mind, and after both *submit()* and *addAttributes()* are run, we can refer to them in the *View* returned from the *Controller* class by mentioning their given name inside a dollarized curly-braces duo, like *${name}*.

3.5. Results View

Now let's print what we received from the form:

```
<h3>${msg}</h3>
Name : ${name}
ID : ${id}
```

4. Conclusion

In this article, we investigated the use of the *@ModelAttribute* annotation for both method arguments and method level use cases.

The implementation of this article can be found in the github (https://github.com/eugenp/tutorials/tree/master/spring-web-modules/spring-mvc-basics-5) project.

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)

(/)