Spring MVC Request and Response

Contents

[Request Handler 2](#_Toc80057152)

[Controller 2](#_Toc80057153)

[Request Handlers 2](#_Toc80057154)

[Step 4: The handler method receives the request 2](#_Toc80057155)

[Response 2](#_Toc80057156)

[MVC Components 2](#_Toc80057157)

[Model 2](#_Toc80057158)

[ModelMap class 3](#_Toc80057159)

[ModelAndView 4](#_Toc80057160)

[@RequestParam 6](#_Toc80057161)

# Request Handler

As we observed in the control flow section, whenever a request is triggered by the client, it reaches the dispatcher servlet and is then delegated to the appropriate controller.

Here we are going to learn the mechanism through which we can create controllers and handler methods.

## Controller

## Request Handlers

handler method via dispatcher servlet and then through the controller.

### Step 4: The handler method receives the request

* The handler method receives the request
* In case there are request parameters, the handler method extracts the data with the help of:

1. HttpServletRequest object
2. @RequestParams annotation
3. @ModelAttribute annotation

We will study about these concepts in the later sections

# Response

## MVC Components

In Spring MVC, the model works as a container for data of the application. Here, the data can be in any form such as an object, string or information from the database, etc.

The model is represented by:

1. Model Interface
2. ModelMap class
3. ModelAndView class

### Model

Model is an interface. Model can be used to transport data from the controller (request handler method) to the view. Here is an example:

|  |
| --- |
| @RequestMapping("/get-employees")  public String getEmployees(Model model) {  // Data access code to fetch the list of employees from the database  List<Employees> employeeList = employeeService.getEmployees();  // Setting the data in the model  **model.addAttribute("employees", employeeList);**  **model.addAttribute(“message”, employeeList.size() + “ Employees found”)**  return "display-employees";  } |

Get the data in a JSP file with the help of implicit object: request and its method getAttribute(“key”). The method will return an Object so we need to properly cast it according to target object

List<Employee> employees = (List<Employee>) request.getAttribute(key);

#### Methods in Model Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| Model addAllAttributes(Collection<?> arg) | It adds all the attributes in the provided Collection into this Map. |
| Model addAllAttributes(Map<String,?> arg) | It adds all the attributes in the provided Map into this Map. |
| Model addAllAttribute(Object arg) | It adds the provided attribute to this Map using a generated name. |
| Model addAllAttribute(String arg0, Object arg1) | It binds the attribute with the provided name. |
| Map<String, Object> asMap() | It returns the current set of model attributes as a Map. |
| Model mergeAttributes(Map< String,?> arg) | It adds all attributes in the provided Map into this Map, with existing objects of the same name taking precedence. |
| boolean containsAttribute(String arg) | It indicates whether this model contains an attribute of the given name |

### ModelMap class

Just like the Model interface, ModelMap is also used to pass values to render a view.

The advantage of ModelMap is it gives us the ability to pass a collection of values and treat these values as if they were within a Map.

Example:

|  |
| --- |
| @RequestMapping("/get-employees")  public String getEmployees(ModelMap map) {  map.addAttribute("welcomeMessage", "welcome");  return "viewPage";  } |

Get the data in a JSP file with the help of implicit object: request and its method getAttribute(“key”). The method will return an Object so we need to properly cast ir according to target object

String msg = (String)request.getAttribute(key);

### ModelAndView

ModelAndView is a holder for both Model and View in the web MVC framework. These two classes are distinct; ModelAndView merely holds both to make it possible for a controller to return both model and view in a single return value.

Example:

|  |
| --- |
| // Using ModelAndView to send the data to view  @RequestMapping("/about")  public ModelAndView about() {  System.out.println("About Us");  // Creating a ModelAndView Object  ModelAndView modelAndView = new ModelAndView();  // Setting the data  modelAndView.addObject("instituteName", "ComputoPlus");  modelAndView.addObject("institueLocation", "Indore");    LocalDateTime now = LocalDateTime.now();  modelAndView.addObject("today", now);    // Setting the view name  modelAndView.setViewName("aboutus");    return modelAndView;  } |

Receiving Request Parameters

Get the data in a JSP file with the help of implicit object: request and its method getAttribute(“key”). The method will return an Object so we need to properly cast it according to target object

String instituteName = (String)request.getAttribute("instituteName");

String instituteLocation = (String)request.getAttribute("institueLocation ");

LocalDateTime todaysDateTime = (String)request.getAttribute("today");

However, in Spring MVC we will use @RequestParam to get single values and @ModelAttribute that is in sync with form fields data to fetch and store data in an object

@RequestMapping

@RequestMapping relates/maps a class/method to a particular url/request

@RequestMapping can be used on both - class and method

@RequestMapping ("/main")

class HomeController{

@RequestMapping ("/home")

public String home(){

}

}

Now when there is a request with url = /main, the whole class will be mapped with it

Inorder to execute home(), the url/request must be /main/home and not just /home

When we execute a request from a client to a controller by default Get method gets executed - to change this we can use properties in @RequestMapping

We can specify different properties in @RequestMapping such as url and method:

@RequestMapping (path="/main", method=RequestMethod.POST)

Now if we make a normal request, the request will be sent as a GET request and the method will not receive it as it is set to receive POST request-error

### @RequestParam

Name=”kaustubh”, username=”kchoudhary”, password=”12345”

#### @ModelAttribute

The basic requirement of @ModelAttribute is that the names of form elements (the name parameters must be same as that of the entity class property names)