First Spring MVC Project

Contents

[First Spring MVC Web Application 2](#_Toc55999600)

[Setup Development Environment 2](#_Toc55999601)

[Create the Project 2](#_Toc55999602)

[Configure the Project 2](#_Toc55999603)

[Step 1: Configure the Front Controller 2](#_Toc55999604)

[Step 2: Create Spring Configuration File 3](#_Toc55999605)

[Step 3: Create Controller 3](#_Toc55999606)

[Step 4: Configure Annotation Scanner in Config 4](#_Toc55999607)

[Step 5: Create views 4](#_Toc55999608)

[Step 6: Configure View Resolver 4](#_Toc55999609)

[Control Flow in Spring MVC Project 5](#_Toc55999610)

[Step 1: The Client initiates the Request 5](#_Toc55999611)

[Step 2: The Role of the Front Controller 5](#_Toc55999612)

[Step 3: Controller hands the request to handler method 6](#_Toc55999613)

[Step 4: The handler method receives the request 6](#_Toc55999614)

[Step 5: The handler method processes the data 7](#_Toc55999615)

[Step 6: The handler method returns data and view 7](#_Toc55999616)

[Step 7: DispatcherServlet sends data and view to the ViewResolver 7](#_Toc55999617)

[Step 8: Front Controller sets data in View 7](#_Toc55999618)

[Full control flow 7](#_Toc55999619)

# First Spring MVC Web Application

Let us create a minimalistic Spring Web MVC application to learn the configurations and the control flow:

## Setup Development Environment

Install and configure latest

* 1. JDK
  2. Eclipse for Java EE Developers
  3. Apache Tomcat Server
  4. Apache Maven
  5. MySQL Server and Workbench if you wish to work with database

## Create & Configure the Project

### Step 1: Create and Setup a Maven Project

1. Create a new Maven project and select
   1. – Catalog: “**Internal**”
   2. Filter: “**maven-archetype-webapp**” and
   3. other related parameters for project setup
2. Configure **Tomcat Server**
3. Configure **the build path**
4. Include “**Spring Web MVC**” Dependency in POM.xml

### Step 2: Configure the Front Controller

Configure the front controller (**DispatcherServlet**) in web.xml (present in WEB-INF folder)

|  |
| --- |
| <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping> |

#### Points to remember

* The servlet name can be anything – a name of your choice. However, it must be relatable.
* The servlet class must be a fully qualified name, i.e. the class name along with the package name. We can get the class definition by pressing Ctrl + Shift + T and then searching the class by name.

### Step 3: Create Spring Configuration File

1. Create an XML file in the same folder (WEB-INF) where web.xml is.
2. The name of the file must be: <DispatcherServlet name> “-” <servlet>.

For example: In our case **dispatcher-servlet.xml**

1. Include the <beans></beans> tags with proper namespaces

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  </beans> |

### Step 4: Create Controller

In the java folder, in a proper package, create a java class with **@Controller** annotation

For a starter, define a method inside this class that will help us in displaying our first JSP page

|  |
| --- |
| package employee.manager.controller  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  @RequestMapping("/employee")  public class EmployeeController {  @RequestMapping("/")  public String redirectToEmpHome() {  return "emp-home";  }  @RequestMapping("/add")  public String addEmployee() {  // Add Employee code  return "add-employee";  }  } |

### Step 5: Configure Annotation Scanner in Config

Include the following line as the first line inside <beans> tag in dispatcher-servlet.xml file:

|  |
| --- |
| <context:component-scan base-package=" employee.manager.controller"/> |

#### Points to Remember:

* <**context**:**component**-**scan**> detects the annotations by package scanning. It tells Spring which packages need to be **scanned** to look for the annotated beans or **components**.
* As Controller class has @Controller and @RequestMapping annotations, the controller’s package must be mentioned in context:component-scan for it to work properly

### Step 6: Create views

1. Create a folder under WEB-INF directory with the name: “views”.
2. Create 2 files: “home.jsp” & “add-employee.jsp” inside “views” folder

### Step 7: Configure View Resolver

Configure the InternalResourceViewResolver bean inside the configuration file along with prefix and suffix properties

|  |
| --- |
| <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" name="viewResolver">  <property name="prefix" value="/WEB-INF/views/"/>  <property name="suffix" value=".jsp"/>  </bean> |

#### Points to remember:

* InternalResourceViewResolver class prepares a full path to the view page
* Set the path to JSP view files in prefix property
* Set the extension of JSP view files in suffix property
* During runtime, the view resolver will prepare the view file name by concatenating prefix & suffix with the file name returned by the controller & present it to the user.

For example: “/WEB-INF/views/” + emp-home + ".jsp", i.e. “/WEB-INF/views/emp-home.jsp"

## Control Flow in Spring MVC Project

We can say that there are following important processes in a Spring Web MVC application:

1. The client triggers the request
2. The front controller delegates the request to the appropriate controller
3. The controller sends the request to the appropriate request handler
4. The request handler:
   1. Receives the request
   2. Collects the request parameters
   3. Processes the data
   4. Sends the processed data and the target view to the front controller

Let us see how Spring MVC achieves the above flow

### Step 1: The Client initiates the Request

There are different ways through which a request can be triggered by the client such as:

1. When user clicks on a link
2. When a form is submitted
3. When some control passes from one server component to another
4. Through redirections

Every time any such thing happens, a request is triggered and every request is represented by a URL that specifies the request name

For example:

1. [http://localhost:8080/<Project-Name>/](http://localhost:8080/%3cProject-Name%3e/)employee/
2. [http://localhost:8080/<Project-Name>/employee/add](http://localhost:8080/%3cProject-Name%3e/employee/add-employee)

### Step 2: The Role of the Front Controller

As you can see in the above section, the requests triggered are “/employee/” and “/employee/add”, in order handle them, we have to create some handler controllers and methods.

|  |
| --- |
| @Controller  @RequestMapping("/employee")  public class EmployeeController {  @RequestMapping("/")  public String redirectToEmpHome() {  return "emp-home";  }  @RequestMapping("/add")  public String addEmployee() {  // Add Employee code  return "add-employee";  }  } |

* The request reaches the DispatcherServlet (Front Controller)
* The request will be received by DispatcherServlet (Front Controller)
* DispatcherServlet will take the help of Handler Mapping and will delegate the request to the Controller class (EmployeeController) associated with the given request: “/employee” in the above case.

### Step 3: Controller hands the request to handler method

* The Controller (EmployeeController) passes the request to the appropriate handler method:

1. “/” to “redirectToEmpHome()” and
2. “/add/” to “addEmployee()” in the above case.

### 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

### Step 5: The handler method processes the data

The handler method performs following actions on the data

1. Authentication
2. File IO
3. Database transaction
4. etc.

### Step 6: The handler method returns data and view

The handler method after processing the data returns the following back to the DispatcherServlet:

1. The processed data
2. The name of the view to render the data, here: ” emp-home” and “add-employee”

The handler can send the data and view name to the front controller with the help of:

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

We will study about these concepts in the later sections

### Step 7: DispatcherServlet sends data and view to the ViewResolver

1. The view resolver will prepare the path of the file by concatenating prefix and suffix

* “emp-home” becomes “/WEB-INF/views/emp-home.jsp”
* “add-employee” becomes “/WEB-INF/views/add-employee.jsp”

1. The view resolver will then return the full path to the front controller.

### Step 8: Front Controller sets data in View

The Dispatcher Servlet will pass the data/model object to the View page to display the result

## Full control flow

After configuring the project as described in the above section, run the application on server

1. When the project will be executed, observe the address bar – it will show [http://localhost:8080/<Project-Name>/](http://localhost:8080/%3cProject-Name%3e/)
2. This request will originate from the client and travel to the front controller (DispatcherServlet)
3. DispatcherServlet will delegate this request to the controller class (that has @Controller annotation), in our case to the EmployeeController class.
4. The request will go to the handler method that is annotated with @RequestMapping annotation and configured to handle the “/” request.

For example: In our case: @RequestMapping (value = "/", method = RequestMethod.GET) redirectToIndex method

1. The method will receive the “/” request and return the name: “emp-home” to the front controller
2. The front controller will call the view resolver declared in spring config file
3. The view resolver will prepare the path of the file by concatenating prefix and suffix
4. The view resolver will then return the full path to the front controller.
5. In case there is some data, the front controller will set it in the view
6. The front controller will send the view back to the client