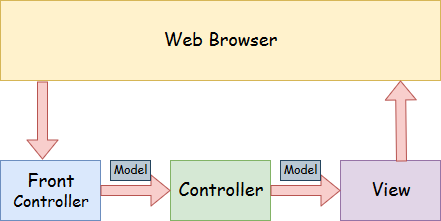
**Spring MVC**

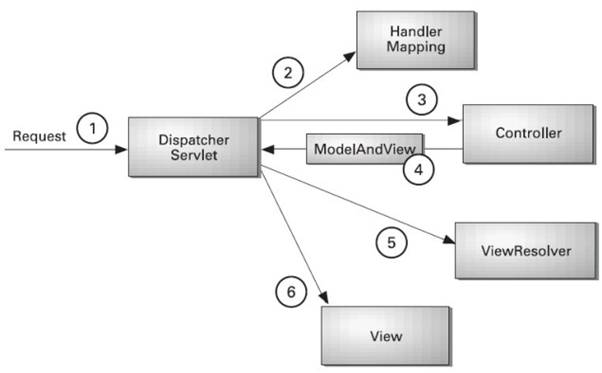
The Spring Web MVC framework provides a model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications.

The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.



* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the **DispatcherServlet** class works as the front controller. It is responsible for managing the flow of the Spring MVC application.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.



* As displayed in the figure, all the incoming requests are intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of the view resolver in the XML file and invokes the specified view component.

Spring Web MVC Framework Example

Let's see the simple example of a Spring Web MVC framework. The steps are as follows:

* Load the spring jar files or add dependencies in the case of Maven
* Create the controller class
* Provide the entry of controller in the web.xml file
* Define the bean in the separate XML file
* Display the message in the JSP page
* Start the server and deploy the project

Steps using Maven Project

1. File New Project – Maven Project – Select Archetype (maven-archetype-webapp) – next button - Specify Group Id and Artifact Id then click Finish Button
2. Open POM.xml file then add following dependencies

<!-- Spring dependencies -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.1.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->

<dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jasper</artifactId>

<version>9.0.12</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>3.0-alpha-1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

1. Add servlet-api.jar file to the application

Add servlet-api.jar from tomcat folder-lib folder to the application

* 1. Right Click on Application Folder-Build path – Configure Build path – select Servlet-api.jar from Tomcat-lib folder

1. Open Web.xml then add following code

<servlet>

<servlet-name>ds</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ds</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

1. Create a folder in WEB-INF folder with any name(viewpages) to hold all the web pages.
   1. Right Click on WEB-INF – New – Folder
2. Now add required jsp pages in that folder(viewpages)
   1. Right click on folder(viewpages) – new – JSP File
3. Now Add an xml file in WEB-INF folder(xml file name must be as follows)
   1. Servletname-servlet.xml (Ex:- ds-servlet.xml)
      1. Get Servletname from web.xml
4. Now add following code in xml file

<?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:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/mvc*

[*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd*](http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd)*"*>

<bean name=*"r"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/Pages/"*/>

<property name=*"suffix"* value=*".jsp"*/>

</bean>

</beans>

1. Create a Controller Class
   1. Right click on src/main/java – new – class – package name and class name(testContorller.java)

@Controller

**public** **class** TestController {

@RequestMapping(value="/p1", method=RequestMethod.***GET***)

**public** String ViewPage1()

{

**return** "Page1"; // here Page1 is, Page1.jsp

}

@RequestMapping(value="/p2", method=RequestMethod.***GET***)

**public** String ViewPage2()

{

**return** "Page2"; // here Page2 is, Page2.jsp

}

}

1. Now configure controller in servlet xml file as follows

<mvc:annotation-driven />

<context:component-scan base-package=*"com.testapp"* />

1. Design jsp page as per the requirement
2. New Run the application with tomcat server.
   1. To display page, on browser – <http://localhost:8080/appname/mappingurl>
   2. [localhost:8080/MyFirstWebMVCApp/](http://localhost:8080/MyFirstWebMVCApp/)p1

**Classes in MVC:**

**Model, ModelMap, and ModelAndView** :- send data to the view from controller

Model, ModelMap, and ModelAndView are used to define a model in a Spring MVC application.

**Model** defines a holder for model attributes and is primarily designed for adding attributes to the model.

**modelclassObj.addAttribute("key", value);**

**map.addAttribute(“x”, 100);**

ModelMap is an extension of Model with the ability to store attributes in a map and chain method calls.

**map.addAttribute("message", message).addAttribute("time", time);**

ModelAndView is a holder for a model and a view; it allows to return both model and view in one return value.

**public ModelAndView getMessage()**

**{**

**var mav = new ModelAndView();**

**mav.addObject("message", message);**

**mav.setViewName("show");**

**return mav;**

**}**

The final interface to pass values to a view is the*ModelAndView*.

This interface allows us to pass all the information required by Spring MVC in one return:

**ModelAndView modelAndView = new ModelAndView("viewPage"); modelAndView.addObject("message", "Baeldung");**

**return modelAndView;**