

Spring MVC Model Interface

In Spring MVC, the model works as a container that contains the data of the application. Here, data can be in any form such as objects, strings, information from the database, etc.

It is required to place the **Model** interface in the controller part of the application. The object of **HttpServletRequest** reads the information provided by the user and pass it to the **Model** interface. Now, a view page easily accesses the data from the model part

Methods of 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 return 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

Steps To Develop SpringMvc Based Applications:

step 1: Create A Project add The maven Dependencies

```
<!--  
https://mvnrepository.com/artifact/org.springframework/spring-webm  
vc -->  
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-webmvc</artifactId>  
  <version>4.3.9.RELEASE</version>  
</dependency>  
<!-- https://mvnrepository.com/artifact/jstl/jstl -->  
<dependency>  
  <groupId>jstl</groupId>  
  <artifactId>jstl</artifactId>  
  <version>1.1.2</version>  
</dependency>
```

Step 2: Configure Dispatcher Servlet In Web.xml

```
<servlet>  
  <servlet-name>ds</servlet-name>  
  <servlet-class>  
    org.springframework.web.servlet.DispatcherServlet  
  </servlet-class>  
  <load-on-startup>1</load-on-startup>  
</servlet>  
  
<servlet-mapping>  
  <servlet-name>ds</servlet-name>  
  <url-pattern>/</url-pattern>  
</servlet-mapping>
```

Step 3: Create Spring Configuration File based Servlet Logical Name DispatcherServlet, the framework will try to load the applicationContext from a file named [servlet-name]-servlet.xml located in the application's WebContent/WEB-INF directory. In this case, our file will be ds-servlet.xml

```
<?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"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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-4.3.xsd
                           http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd">

    <!-- Provide support for component scanning -->

    <context:component-scan base-package = "com.shiva.controller" />

    <!--Provide support for conversion, formatting and validation -->

    <mvc:annotation-driven></mvc:annotation-driven>

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

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

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

</bean>

</beans>

Note :

Following are some important points about HelloWorld-servlet.xml file –

- **The [servlet-name]-servlet.xml file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.**
- **The <context:component-scan...> tag will be used to activate the Spring MVC annotation scanning capability, which allows to make use of annotations like @Controller and @RequestMapping, etc.**

Note: If not Configured InternalResourceViewResolver it goes to rootDirectly to find out the view component.

Step:4 Create A view Component in side the webapp folder.(root directory)

<html>

<body>

<h2>Login Application</h2>

<form action="login">

**UserName : <input type="text" name="uname"/>
**

**PassWord : <input type="text" name="pass"/>
**

<input type="submit" value="Login">

</form>

</body>

</html>

Step :5 Create A Controller Class It Act as Helper Class/HandlerClass/ControllerClas

To create the controller class, we are using two annotations **@Controller** and **@RequestMapping**.

The **@Controller** annotation marks this class as Controller.

The **@RequestMapping** annotation is used to map the class with the specified URL name

In controller class:

- The **HttpServletRequest** is used to read the HTML form data provided by the user.
- The **Model** contains the request data and provides it to view page
- the **@RequestParam** is used to read the HTML form data provided by a user and bind it to the request parameter.
- The **Model** contains the request data and provides it to view page.

In Spring MVC, the **@RequestParam** annotation is used to read the form data and bind it automatically to the parameter present in the provided method. So, it ignores the requirement of **HttpServletRequest** object to read the provided data.

Including form data, it also maps the request parameter to query parameter and parts in multipart requests. If the method parameter type is **Map** and a request parameter name is specified, then the request parameter value is converted to a **Map** else the map parameter is populated with all request parameter names and values.

```
package com.shiva.controller;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
@Controller
```

```
public class LoginController {
```

```
    String view=null;
```

```
    /*
```

The `HttpServletRequest` is used to read the HTML form data provided by the user.

Even we can do one more approach using `@RequestParam`

The `@RequestParam` is used to read the HTML form data provided by a user and bind it to the request parameter

The Model contains the request data and provides it to view page

`*/`

`@RequestMapping("/login")`

```
public String checkValication(@RequestParam("uname")String
uname,@RequestParam("pass")String pass,Model model)
```

```
{
```

```
if(uname.equals("shiva")&&pass.equals("shiva"))
```

```
{
```

```
    //adding uname value to model ,to carry this information to
viewComponent...
```

```
    model.addAttribute("message",uname);
```

```
    return view="success";
```

```
}
```

```
else
```

```
{
```

```
    String msg="Sorry "+ uname+". You entered an incorrect password";
```

```
    model.addAttribute("message", msg);
```

```
}
```

```
        return view="error";
    }
}
```

**Note: Controller class giving response back view or model Object to DispatcherServlet
DispatcherServlet receives Model or view some time it can receive ModelAndView(view
and object also)it gives ViewResolver**

**ViewResolver takes the request from dispatcher Servlet it gives appropriate view
component**

If ViewResolver not configuredThen it gives to root Directory

**Step 6: View Resolver sent to view Components(pages) WEB-INF/page/
WEB-INF/page/**

**Inside page directory we hv view components We configured configuration file using
InternalResourceViewResolver**

Success.jsp

```
<%@page isELIgnored="false" %>
<html>
<body>

<h1>Welcome : ${message} </h1>
</body>
</html>
```

Error.jsp

```
<%@page isELIgnored="false" %>
<html>
<body>
<h1> ${message} </h1>
<br><br>
<jsp:include page="/index.jsp"></jsp:include>
</body>
</html>
</html>
```

- ▲ SpringWebMvcApp_03
 - ▷ Spring Elements
 - ▷ Deployment Descriptor: Archetype Created Web Application
 - ▲ Java Resources
 - ▲ src/main/java
 - ▲ com.shiva.controller
 - ▷ LoginController.java
 - ▷ src/main/resources
 - ▷ Libraries
 - ▷ JavaScript Resources
 - ▷ Deployed Resources
 - ▲ src
 - ▲ main
 - ▷ java
 - resources
 - ▲ webapp
 - ▲ WEB-INF
 - ▲ jsp
 - error.jsp
 - success.jsp
 - ds-servlet.xml
 - ▷ web.xml
 - index.jsp
 - ▷ target
 - pom.xml

