ViewResolver in Spring MVC

Last Updated : 11 Aug, 2025

In Spring MVC, a ViewResolver is responsible for mapping the logical view name returned by a controller to the actual view (like a JSP file) that should be rendered.

When a controller returns a view name like "home", the ViewResolver resolves it to something like /WEB-INF/views/home.jsp.This makes view management easier and keeps the controller code clean by avoiding hardcoded paths. Let's understand this by looking at a practical implementation.

Prerequisites:

Prior to it, certain requirements are needed that are as follows:

1. Eclipse (EE version)/STS IDE
2. Spring JAR Files
3. Tomcat Apache latest version

Note: We are going to use Spring Tool Suite 4 IDE for this project. Please refer to this article to install STS in your local machine.

Step-by-Step Implementation

Step 1: Create a Dynamic Web Project in STS

Create a Dynamic Web Project in your STS IDE. You can refer to this article for how to create that.

Step 2: Add Spring JAR Files

Download the spring JARs file and place them in the **src > main > webapp > WEB-INF > lib** folder.

Step 3: Configure Apache Tomcat Server

Configure the Apache Tomcat Server and link it to the application.

Step 4: Configure Dispatcher Servlet

The `DispatcherServlet` is a critical component in Spring MVC. It acts as the front controller and manages the flow of requests. Now we are going to configure Dispatcher Servlet with our Spring MVC application, Go to the **src > main > webapp > WEB-INF > web.xml** file.

web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.h
tml"

xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/j
avaee/index.html

http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html/web-
app_4_0.xsd"
         id="WebApp_ID" version="4.0">
    <display-name>springmvc-view-resolver</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.jsp</welcome-file>
        <welcome-file>default.htm</welcome-file>
    </welcome-file-list>

    <servlet>
        <!-- Provide a Servlet Name -->
        <servlet-name>viewresolver-dispatcher</servlet-name>
        <!-- Provide a fully qualified path to the DispatcherServlet class --
>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <!-- Provide a Servlet Name that you want to map -->
        <servlet-name>viewresolver-dispatcher</servlet-name>
        <!-- Provide a URL pattern -->
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 5: Create Spring Configuration File

Now go to the **src > main > webapp > WEB-INF** and create an XML file. Actually, this is a Spring Configuration file like the beans.xml file. And the name of the file must be in this format.

YourServletName-servlet.xml

For example, for this project, the name of the file must be

viewresolver-dispatcher-servlet.xml

So either you can create a Spring Configuration File or you can just create a simple XML file and add the below lines of code inside that file.

viewresolver-dispatcher-servlet.xml:

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

  <!-- This Line is used for scanning all the packages that have controller
classes -->
  <context:component-scan base-package="com.demo.controllers">
</context:component-scan>

</beans>
```

Step 6: Creating Spring MVC Controller

Now, let's create some controllers. Go to the src/main/java and create a new controllers package (For ex. com.demo.controllers) as per your choice. And inside that create a Java class and name the class as **DemoController**. Now how to tell the Spring that this is our controller class. So the way we are going to tell the Spring is by marking it with a @Controller annotation.

> *@Controller*
>
> *public class DemoController {}*

Note: Spring will automatically initialize the class having a @Controller annotation and register that class with the spring container.

Now let us create a simple method inside the Controller class and use the @RequestMapping annotation before the method something like this.

```
// Annotation
@RequestMapping("/hello")
public String helloWorld() {
    return "";
}
```

Now in the return statement, we have to return some views (web pages), so whenever the endpoint '/hello' is invoked we can see our result on the web page. So let's create our first View.

Step 7: Creating View

Go to the **src > main > webapp > WEB-INF > right-click > New > Folder** and name the folder as **views.** Then **views > right-click > New > JSP File** and name your first view. Here we have named it a **demo.jsp** file. Below is the code for the **demo.jsp** file. We have created a simple web page inside that file.

demo.jsp:

```
<!DOCTYPE html>
<html>
<body bgcolor="green">
    <h1>Hello GeeksforGeeks!</h1>
</body>
</html>
```

Now go to the DemoController class and inside the *helloWorld() method* we have to return a value something like this.

> *return "/WEB-INF/views/demo.jsp";*

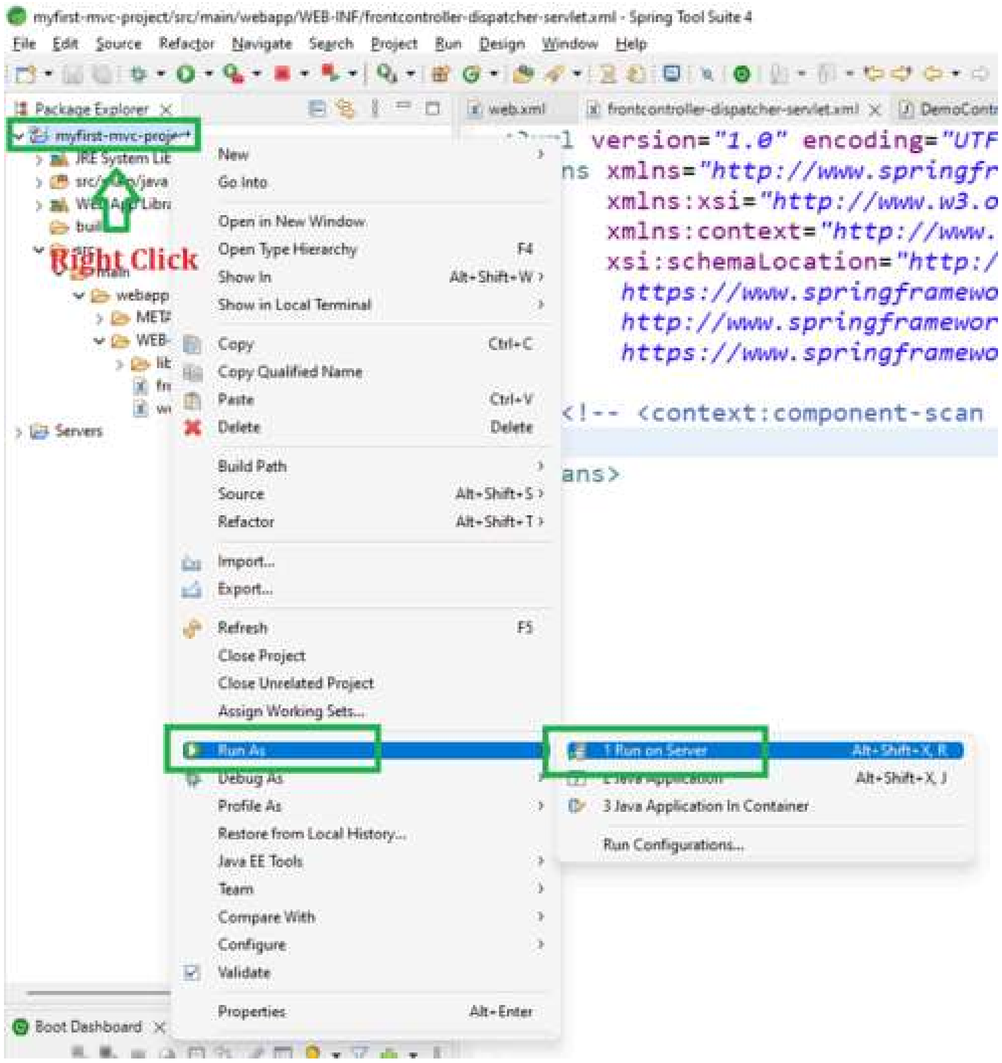
We have just been given the path for our view.

DemoController.java:

```
package com.demo.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class DemoController {

    @RequestMapping("/hello") public String helloWorld(){
        return "/WEB-INF/views/demo.jsp";
    }
}
```

Step 8: Running Spring MVC Application

To run our Spring MVC Application **right-click on your project > Run As > Run on Server.** And run your application as shown in the below image as depicted below as follows:

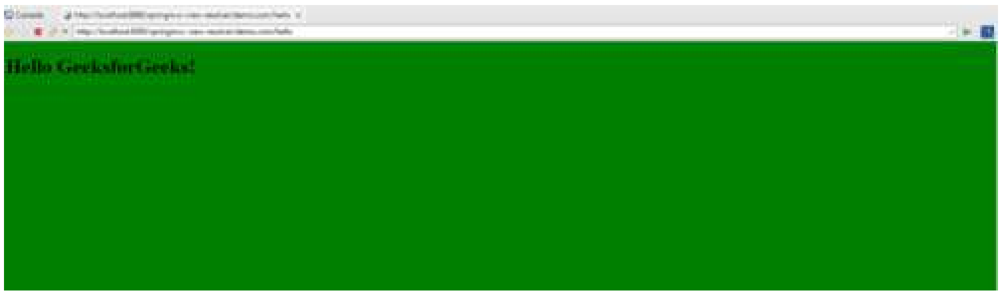


```
<?xml version="1.0" encoding="UTF
ns xmlns="http://www.springfr
   xmlns:xsi="http://www.w3.o
   xmlns:context="http://www.
   xsi:schemaLocation="http:/
     https://www.springframewo
     http://www.springframewor
     https://www.springframewo

<!-- <context:component-scan

ans>
```

After that use the following URL to run your controller

http://localhost:8080/springmvc-view-resolver/hello

Output:



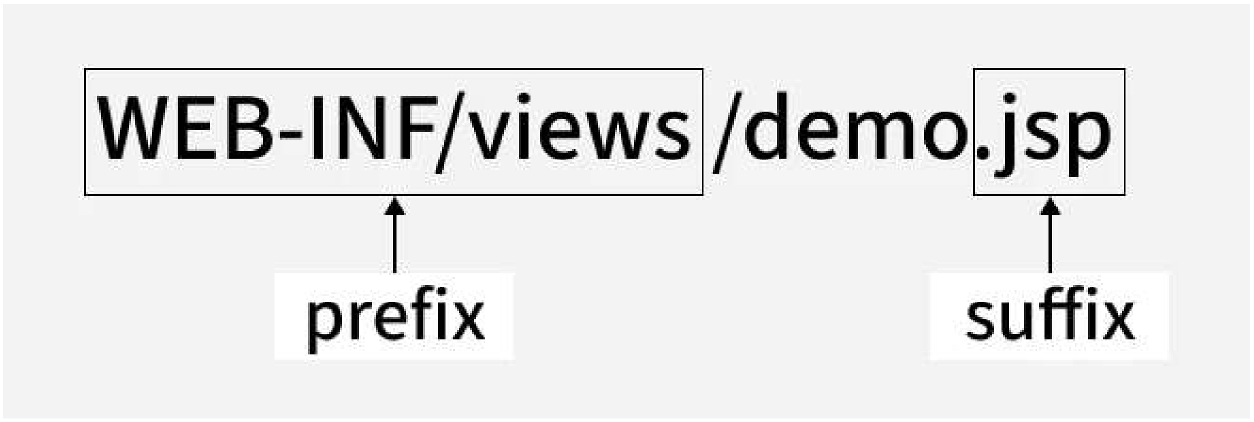
Hello GeeksforGeeks!

Understanding the Problem with Hardcoded Paths

So what's the problem with the above project? Just go to the DemoController.java file again and the problem lies in this line of code:

return "/WEB-INF/views/demo.jsp";

- Hardcoding full view paths in controllers makes the app tightly coupled. If the JSP location or extension changes, you'd need to update every controller.
- ViewResolver solves this by letting you return just the view name (e.g., "demo") and it handles the path and extension through centralized configuration—making the app easier to maintain.



Path

And we call them prefixes and suffixes accordingly.

Step 9: Update DemoController.java

Go to the DemoController.java file and modify the code as shown in the below snippet.

Modified DemoController.java:

```java
package com.demo.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class DemoController {

    @RequestMapping("/hello") public String helloWorld(){
        return "demo";
    }
}
```

Here we have only returned the page name, nothing else.

Note: The return file name and the View name must be same because we are returning that web page from the controller handler method.

Configure ViewResolver inside the Spring Configuration File (i.e. viewresolver-dispatcher-servlet.xml).

Step 10: Modify the viewresolver-dispatcher-servlet.xml

Add the below lines of code as follows:

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

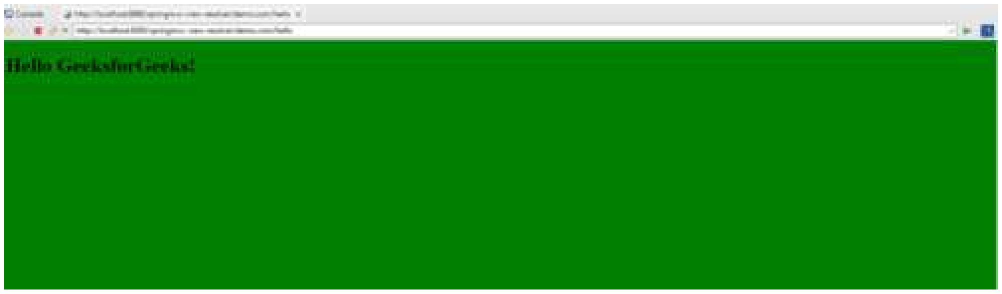
This is the thing we have discussed above and it's completely the concept of Setter Based Dependency Injection in the spring framework.

viewresolver-dispatcher-servlet.xml:

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

  <!-- This Line is used for scanning all the packages that have controller classes -->
  <context:component-scan base-package="com.demo.controllers"></context:component-scan>

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

</beans>
```

Now inside this "value" property, you can change according to the requirements as many times as you want. Now run your spring application and the output will be as follows.

Output:



```
Hello GeeksforGeeks!
```

Output

Comment More info Advertise with us