SPRING MVC

[DISPATCHER SERVLET:](#_37af5ge7jnqm)

[REQUIRED CONFIGURATION:](#_91nb2nibibto)

[DEFINING A CONTROLLER:](#_2b8u22m2t81g)

[EXERCISE:](#_id1gj8ckb73y)

[STEPS:](#_gzaplbqbw11k)

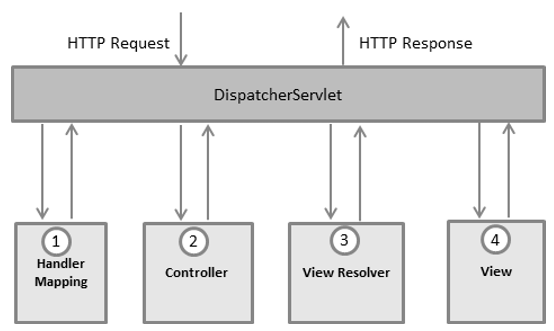
[SOURCE CODE:](#_2rep7v6r5bvl)

The Spring Web MVC framework provides Model-View-Controller (MVC) 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.

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

# DISPATCHER SERVLET:

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses.



* After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to **call the appropriate *Controller*.**
* The ***Controller*** takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and **returns view name to the *DispatcherServlet*.**
* The *DispatcherServlet* will take help from ***ViewResolver* to pickup the defined view** for the request.
* Once view is finalized, The *DispatcherServlet* **passes the model data to the view** which is finally rendered on the browser.

# REQUIRED CONFIGURATION:

You need to map requests that you want the *DispatcherServlet* to handle, by using a URL mapping in the web.xml file. The web.xml file will be kept in the WebContent/WEB-INF directory of your web application.

SAMPLE web.xml file:

Upon initialization of dispatcherservlet DispatcherServlet, the framework will try to load the application context from a file named [servlet-name]-servlet.xml located in the application's WebContent/WEB-INF directory.Thus dispatcher servlet file name should be dispatcherservlet-servlet.xml .<servlet-mapping> tag indicates what URLs will be handled by which DispatcherServlet. Here all the HTTP requests ending with .jsp will be handled by the dispatcherservlet DispatcherServlet.

<web-app id = "WebApp\_ID" version = "2.4"

xmlns = "http://java.sun.com/xml/ns/j2ee"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>SpringTask</display-name>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>dispatcherservlet</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherservlet</servlet-name>

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

</servlet-mapping>

</web-app>

The following are the contents of dispatcherservlet-servlet.xml:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 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">

<context:component-scan base-package="com.zilker.springmvc" />

<context:annotation-config />

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

<property name="prefix" value="/pages/" />

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

</bean>

</beans>

* 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 use to activate Spring MVC annotation scanning capability which allows to make use of annotations like @Controller and @RequestMapping etc.
* The ***InternalResourceViewResolver*** will have rules defined to resolve the view names. As per the above defined rule, a logical view named hello is delegated to a view implementation located at */WEB-INF/jsp/hello.jsp* .

# DEFINING A CONTROLLER:

The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it. The @Controllerannotation indicates that a particular class serves the role of a controller. The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

# EXERCISE:

Use Spring MVC to implement login flow and registration flow with connection to database

## STEPS:

* Create a Maven project
* You will be getting a pom.xml file by default on creation of a maven project.
* Add spring dependencies in this pom.xml files , all these dependendencies will be downloaded and stored under maven dependencies.
* Under source package, create 6 different packages:
  + Com.zilker.springmvc.beans
  + Com.zilker.springmvc.constants
  + Com.zilker.springmvc.controller
  + Com.zilker.springmvc.dao
  + Com.zilker.springmvc.delegate
  + com.zilker.springmvc.utils

Create web.xml inside WEB-INF folder and define the configurations for dispatcher servlet as above.

Create dispatcher servlet and define the viewResolver configurations in dispatcherservlet-servlet.xml

Now create separate forms for register and login and redirect to those pages from index.jsp.Give the url mapping of the controller class in the form action of register and login.

The controller redirects the flow to delegate then to dao.

Define the business logic for login and register in dao and returns the respective values to delegate which in turn returns it to controller.

Controller wraps the value in a model object and returns the result to a jsp page,the name of this page is mentioned in return “name\_of\_jsp\_page”.

The view resolver inside the dispatcherservlet takes care of redirecting the value in model to the respective jsp page.

The value set in model object can be used with the help of JSTL tags in JSP pages.

## SOURCE CODE:

<https://github.com/sruthiviswanathan/ServletTasks/tree/master/SpringTask>