VaanNila

Menu

- Struts 1
- Struts 2
- Spring
- Hibernate
- Ant
- Log4j

Subscribe

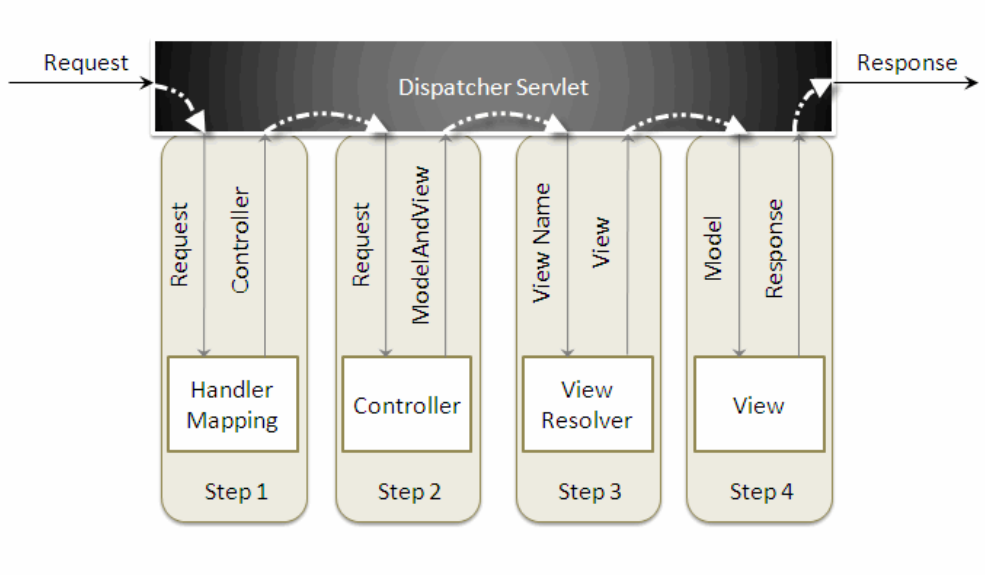
- RSS
- Email

Spring > Spring MVC

Spring MVC

Spring MVC helps in building flexible and loosely coupled web applications. The Model-view-controller design pattern helps in seperating the business logic, presentation logic and navigation logic. Models are responsible for encapsulating the application data. The Views render response to the user with the help of the model object . Controllers are responsible for receiving the request from the user and calling the back-end services.

The figure below shows the flow of request in the Spring MVC Framework.

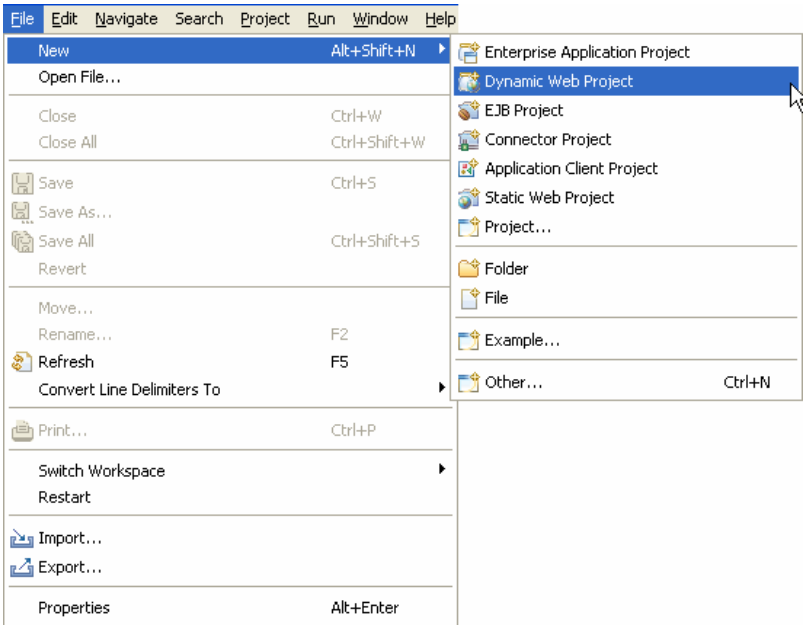


When a request is sent to the Spring MVC Framework the following sequence of events happen.

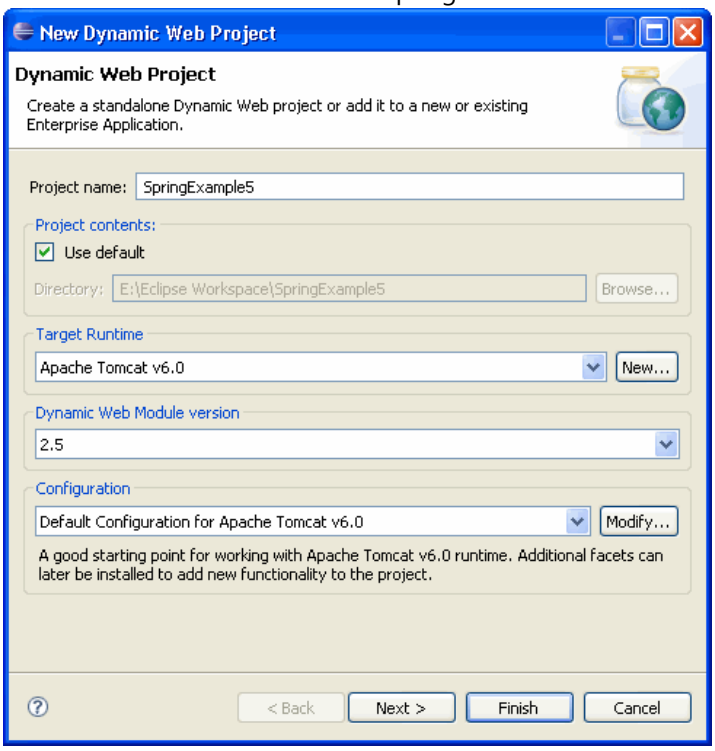
- The `DispatcherServlet` first receives the request.
- The `DispatcherServlet` consults the `HandlerMapping` and invokes the `Controller` associated with the request.
- The `Controller` process the request by calling the appropriate service methods and returns a `ModeAndView` object to the `DispatcherServlet`. The `ModeAndView` object contains the model data and the view name.
- The `DispatcherServlet` sends the view name to a `ViewResolver` to find the actual `View` to invoke.
- Now the `DispatcherServlet` will pass the model object to the `View` to render the result.
- The `View` with the help of the model data will render the result back to the user.

To understand the Spring MVC Framework we will now create a simple hello world example using the Eclipse IDE. I am using Exclipse IDE 3.4 , Spring IDE plugin, Tomcat 6.0 and Spring 3.0 to demonstrate this example.

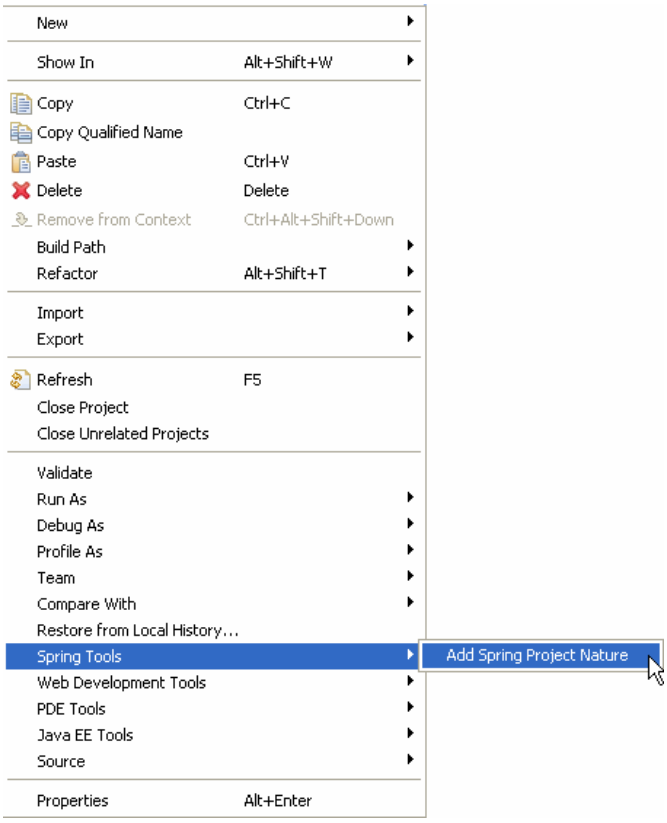
Go to `File -> New -> Dynamic Web Project`, to create a web project.



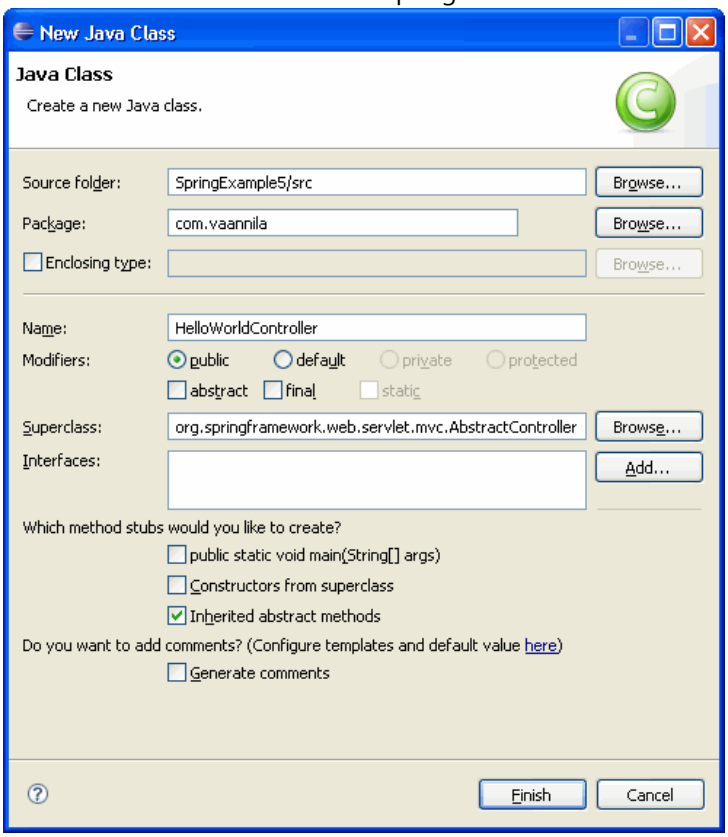
Enter the project name and click the `Finish` button.



Right click the project folder, and select `Spring Tools -> Add Spring Project Nature`, to add Spring capabilities to the web project. This feature will be available once you install the Spring IDE.



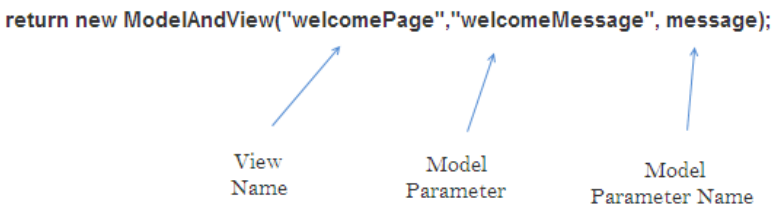
Create a new package `com.vaannila` inside the `src` directory. The Spring controller class extends `org.springframework.web.servlet.mvc.AbstractController` class. To create a new controller class right click the `src` directory and create a new java class, enter the controller class name and super class name and the `Finish` button.



Copy the following code inside the `HelloWorldController` class.

```
01. import javax.servlet.http.HttpServletRequest;
02. import javax.servlet.http.HttpServletResponse;
03.
04. import org.springframework.web.servlet.ModelAndView;
05. import org.springframework.web.servlet.mvc.AbstractController;
06.
07. public class HelloWorldController extends AbstractController {
08.
09.     private String message;
10.
11.     @Override
12.     protected ModelAndView handleRequestInternal(HttpServletRequest request,
           HttpServletResponse response) throws Exception {
13.         return new ModelAndView("welcomePage","welcomeMessage", message);
14.     }
15.
16.     public void setMessage(String message) {
17.         this.message = message;
18.     }
19.
20. }
```

The `HelloWorldController` class has a `message` property that is set thru the setter injection. The `HelloWorldController` class should override the `handleRequestInternal()` method to process the request. After processing the request the `handleRequestInternal()` method returns a `ModelAndView` object back to the `DispatcherServlet`.



The `DispatcherSevlet`, as the name indicates, is a single servlet that manages the entire request-handling process. When a request is sent to the `DispatcherServlet` it delegates the job by invoking the appropriate controllers to process the request. Like any other servlet the `DispatcherServlet` need to be configured in the web deployment descriptor as shown.

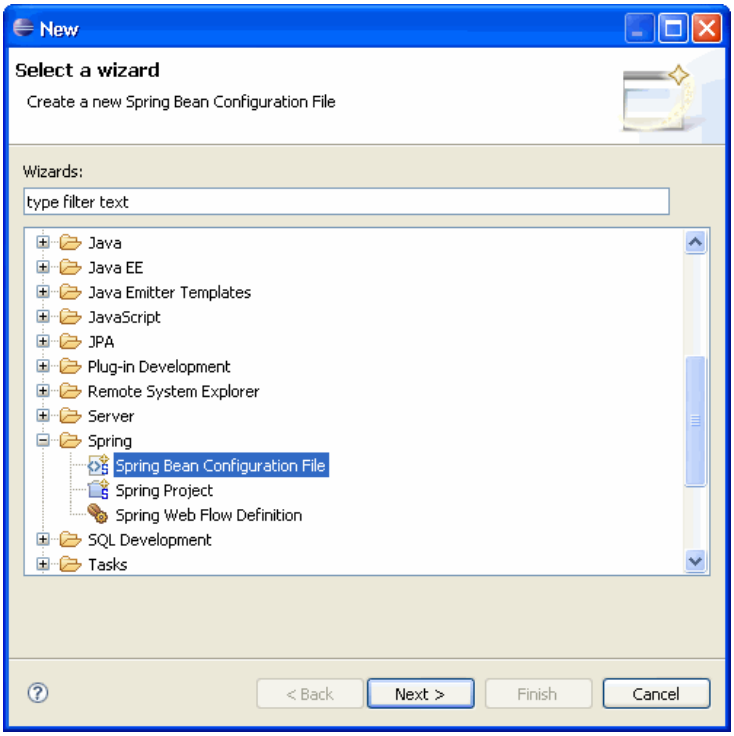
```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
      version="2.5">
03.   <servlet>
04.       <servlet-name>dispatcher</servlet-name>
05.       <servlet-class> org.springframework.web.servlet.DispatcherServlet
```

```
            </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

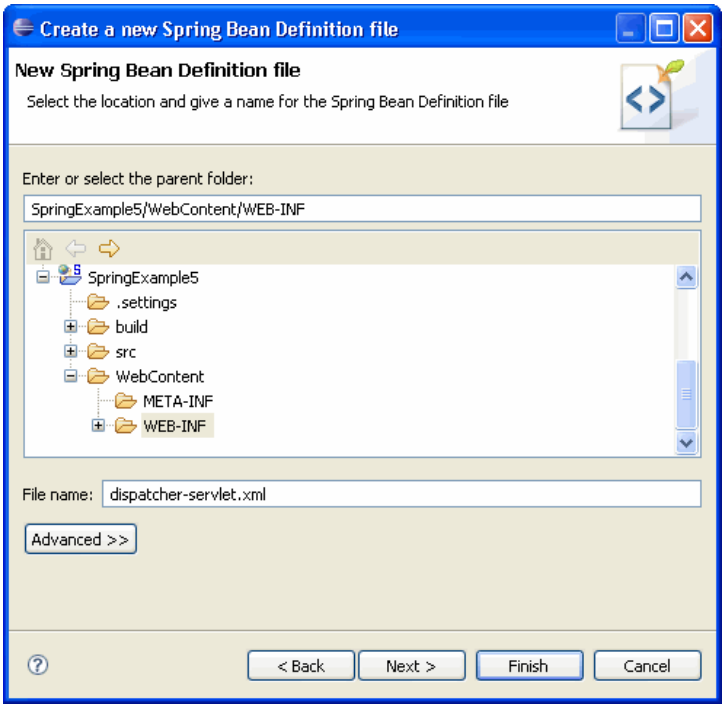
Here the servlet name is `dispatcher`. By default the `DispatcherServlet` will look for a file name `dispatcher-servlet.xml` to load the Spring MVC configuration. This file name is formed by concatenating the servlet name ("`dispatcher`") with "`-servlet.xml`". Here we user the the `url-pattern` as "`.htm`" inorder to hide the implementations technology to the users.

The `redirect.jsp` will be invoked first when we execute the Spring web application. This is the only `jsp` file outside the `WEB-INF` directory and it is here to provide a redirect to the `DispatcherServlet`. All the other views should be stored under the `WEB-INF` directory so that they can be invoked only through the controller process.

To create a bean configuration file right click the `WebContent` folder and select `New -> Other`. The following dialog box appears.



Select the Spring Bean Configuration file and click Next.



Enter the file name as "`dispatcher-servlet.xml`" and click the `Finish` button.

Now the Spring bean configuration file is created, we need to configure the `Controller` and the `ViewResolver` classes. The following code shows how to do this.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
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.xsd">

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

    <bean name="/welcome.htm" class="com.vaannila.HelloWorldController" >
        <property name="message" value="Hello World!" />
    </bean>

</beans>
```

First let's understand how to configure the controller.

```
<bean name="/welcome.htm" class="com.vaannila.HelloWorldController" >
    <property name="message" value="Hello World!" />
</bean>
```

Here the `name` attribute of the `bean` element indicates the URL pattern to map the request. Since the `id` attribute can't contain special characters like "/" , we specify the URL pattern using the `name` attribute of the `bean` element. By default the `DispatcherServlet` uses the `BeanNameUrlHandlerMapping` to map the incoming request. The `BeanNameUrlHandlerMapping` uses the bean name as the URL pattern. Since `BeanNameUrlHandlerMapping` is used by default, you need not do any seperate configuration for this.

We set the message attribute of the `HelloWorldController` class thru setter injection. The `HelloWorldController` class is configured just like an another JavaBean class in the Spring application context, so like any other JavaBean we can set values to it through Dependency Injection(DI).

The `redirect.jsp` will redirect the request to the `DispatcherServlet`, which inturn consults with the `BeanNameUrlHandlerMapping` and invokes the `HelloWorldController`. The `handleRequestInternal()` method in the HelloWorldController class will be invoked. Here we return the `message` property under the name `welcomeMessage` and the view name `welcomePage` to the `DispatcherServlet`. As of now we only know the view name, and to find the actual view to invoke we need a `ViewResolver`.

The `ViewResolver` is configured using the following code.

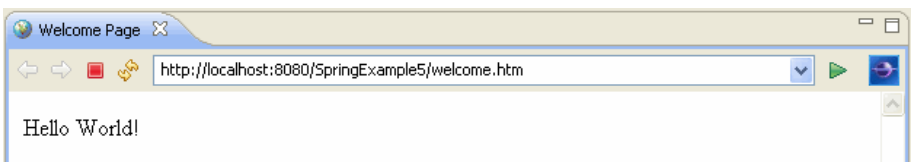
```
<bean id="viewResolver"
class=" org.springframework.web.servlet.view.InternalResourceViewResolver" >
    <property name="prefix">
        <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

Here the `InternalResourceViewResolver` is used to resolve the view name to the actual view. The `prefix value + view name + suffix value` will give the actual view location. Here the actual view location is `/WEB-INF/jsp/welcomePage.jsp`

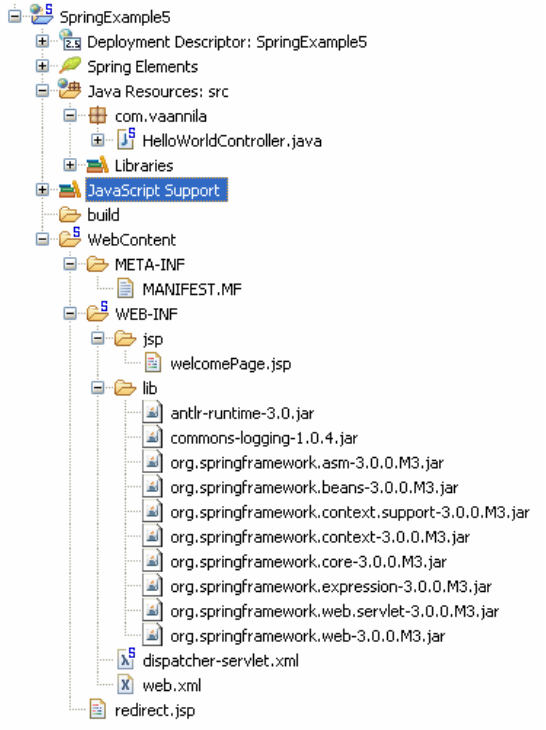
The following library files are needed to run the example.

```
antlr-runtime-3.0
commons-logging-1.0.4
org.springframework.asm-3.0.0.M3
org.springframework.beans-3.0.0.M3
org.springframework.context-3.0.0.M3
org.springframework.context.support-3.0.0.M3
org.springframework.core-3.0.0.M3
org.springframework.expression-3.0.0.M3
org.springframework.web-3.0.0.M3
org.springframework.web.servlet-3.0.0.M3
```

To execute the example run the `redirect.jsp` file. The following page will be displayed.



The directory structure of the example is shown below.



You can download and try the Spring MVC example by clicking the Download link below.

Source :**Download**

War :**Download**