Spring MVC Tutorial

Spring MVC tutorial provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet.

In Spring Web MVC, **DispatcherServlet** class works as the front controller. It is responsible to manage the flow of the spring mvc application.

The **@Controller** annotation is used to mark the class as the controller in Spring 3.

The @RequestMapping annotation is used to map the request url. It is applied on the method.

@RequestMapping is one of the most widely used **Spring MVC** annotation. `org.springframework.web.bind.annotation.RequestMapping` annotation is used to map web requests onto specific handler classes and/or handler methods.

@RequestMapping with Class: We can use it with class definition to create the base URI. For example:

```
@Controller
@RequestMapping("/home")
public class HomeController {


}
```

Now /home is the URI for which this controller will be used. This concept is very similar to servlet context of a web application.

All the incoming request is intercepted by the DispatcherServlet that works as the front controller. The DispatcherServlet gets 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 view resolver in the xml file and invokes the specified view component.

InternalResourceViewResolver.

In Spring MVC based application, the last step of request processing is to return the logical view name. Here DispatcherServlet has to delegate control to a view template so the information is rendered. This view template decides that which view should be rendered based on returned logical view name. These view templates are one or more view resolver beans declared in the web application context. These beans have to implement the ViewResolver interface for DispatcherServlet to auto-detect them. Spring MVC comes with several ViewResolverimplementations. In this example, we will look at such a view resolver template i.e. **InternalResourceViewResolver**.

In most of the applications, views are mapped to a template's name and location directly. InternalResourceViewResolver helps in mapping the logical view names to directly view files under a certain pre-configured directory. To register InternalResourceViewResolver, you can declare a bean of this type in the web application context.

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

After above configuration, InternalResourceViewResolver will resolves the view names 'home' and 'admin/home' etc. in the following way.

LOGICAL VIEW NAME	ACTUAL VIEW FILE
home	/WEB-INF/jsp/home.jsp
admin/home	/WEB-INF/jsp/admin/home.jsp
report/main	/WEB-INF/jsp/report/main.jsp

Class DispatcherServlet

java.lang.Object

- javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet
    - org.springframework.web.servlet.HttpServletBean
      - org.springframework.web.servlet.FrameworkServlet
        - org.springframework.web.servlet.DispatcherServlet

- **All Implemented Interfaces:**
  Serializable, Servlet, ServletConfig, Aware, ApplicationContextAware, EnvironmentAware, EnvironmentCapable

DispatcherServlet is the class which manages the entire request handling process.Like a normal servlet **DispatcherServlet** also needs to be configured in the web deployement Descriptor(web.xml).By default **DispatcherServlet** will look for a name **dispatcher-servlet**.xml to load the Spring MVC configuration

EL (Expression Language)

If you want to invoke some Java code to **access and display** "backend" data inside a JSP page, then you need to use EL (Expression Language), those ${} things. E.g. redisplaying submitted input values:

```
<input type="text" name="user" value="${rs.getInt(1)}" />
```

Model AndView

The model presents a placeholder to hold the information you want to display on the view.

Example 1
If you have...

```
return new ModelAndView("welcomePage","WelcomeMessage","Welcome!");
```

... then in your jsp, to display the message, you will do:-

```
Hello Stranger! ${WelcomeMessage} // displays Hello Stranger! Welcome!
```

Example 2
If you have...

```
MyBean bean = new MyBean();
bean.setName("Mike!");
bean.setMessage("Meow!");

return new ModelAndView("welcomePage","model",bean);
```

... then in your jsp, you can do:-

```
Hello ${model.name}! {model.message} // displays Hello Mike! Meow!
```