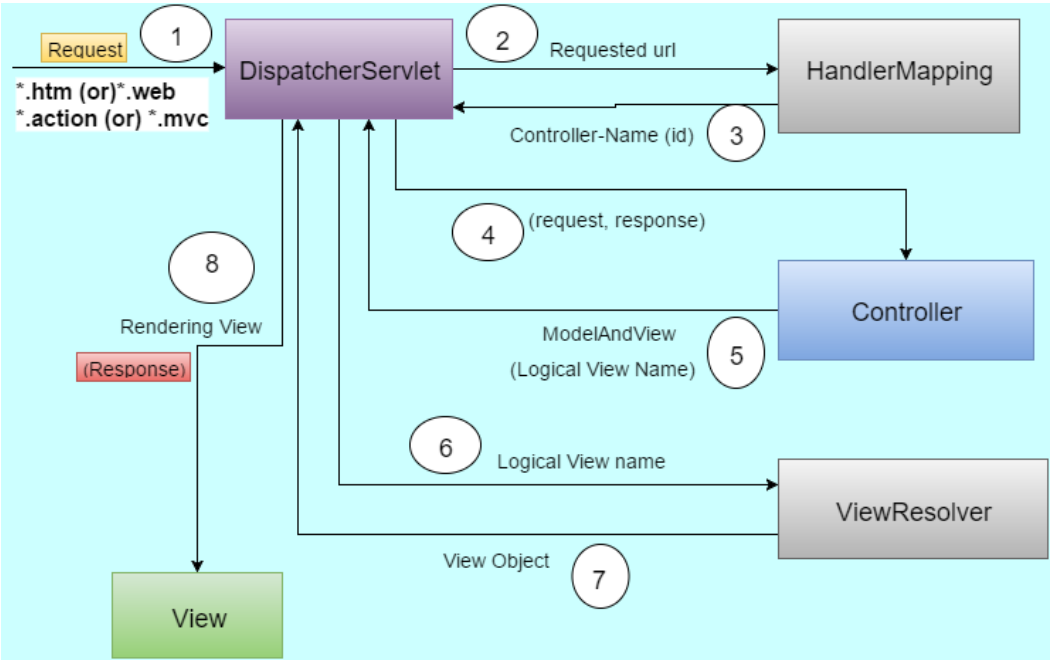
SPRING MVC FLOW

SpringTutors February 22, 2016 Features 116 Views

Spring MVC Flow



Let's understand overall how spring MVC works-

In case of Spring MVC user has to send the request with extension *.htm or *.mvc or *.web or *.action depends on the DispatcherServlet configuration wild card pattern. When user send the request it will be received by DispatcherServlet. Now DispatcherServlet asks the HandlerMapping to give the Controller information to whom he has to send the request by giving incoming url. Now HandlerMapping will take the url and map the appropriate Controller and return the Controller information to DispatcherServlet. Now DispatcherServlet will forward the request to Controller for processing the request. Now controller will process the request and return back the logical view name to the DispatcherServlet. After getting the logical view name DispatcherServlet will ask the ViewResolver to resolve the view by passing logical view name. View resolver will take the logical view and creates the object of the view which should be rendered by DispatcherServlet and return the View Object to DispatcherServlet. Now by getting the View Object DispatcherServlet will render the appropriate view. In this way Spring MVC flow works. In the above explanation we have understoond the overall flow of spring MVC.

Let's see how each and every component declared above in spring mvc perform their task to full fill the users request through small web application-

Project Directory Structure For Spring MVC-

```
FirstMvcWeb
 |-src
    |-WebContent
        |-WEB-INF
           |-lib
           |-web.xml
           |-home.jsp
```

In above project directory structure FirstMvcWeb is the root directory of our application. This application will show how to render the home.jsp. In a typical web application we put jsp pages in WebContent directory but in above application we have put the home.jsp inside the WEB-INF. In case of Spring MVC it is recommended to keep View inside the WEB-INF. Let's see what happens when we keep view in WebContent directory in case of Servlet and Jsp or Struts framework-

1. WebContent is a public directory, views in this directory can be accessed by any users directly. In this case our presentation layer component get exposed to End-User. It means if users of our application is technically strong then they might hack our application by using session and knowing the type of view (jsp/html).

2. As End-User can directly access the view he or she may send the request to access the view which expect some data to render. In this case user will get an ugly error message which will give bad experience of using our application to the user.

3. Let's suppose our application is for booking movie ticket a user comes and book ticket and live the system without logout to the page another user comes and he request the same view page with same url and in same session then he will also gets the same ticket with same information as that is booked by first user. It will be the biggest problem.

In case of Servlet and Jsp or Struts or any other framework except spring Mvc views are either Jsp or Html.There is no point in placing the views in WEB-INF to abstract it from user. That's why it is not recommended to place the views inside WEB-INF.

In case of Spring Mvc views may be any thing like jsp, html, xml, pdf, Excel Sheet etc. so, due to above reasons it is recommended to place the views inside WEB-INF directory only.

WEB-INF is the protected directory in the project directory structure. Any thing placed inside the WEB-INF directory can be used by application component only. It cannot be used by user directly from outside. In this way our views will be abstracted from end user and our application does not suffer the above explained problems.

Now Let's first create the home.jsp page and put it in the WEB-INF directory-

home.jsp

```
<html>
 <head>
 <title>Home</title>
 </head>
 <body>
 <p>Welcome to FirstMvcWeb Home. </p>
 </body>
 </html>
```

Now we want to access home.jsp to access it we cannot send the request directly to the home.jsp because it is inside the WEB-INF directory. It can be accessed by application internal component only which is Controller. Controller will access home.jsp but the problem is we cannot send request directly to the controller because it is not JEE component. To send the request to the Controller DispatcherServlet comes into picture.

DispatcherServlet is the component which is provided by spring.It is the single entry point to our application. It is the front controller in our application which will forward the request to controller to handle the request. Now to forward the request to Controller we need DispatcherServlet so, let's configure the DispatcherServlet in descriptor file (web.xml) as follows-

web.xml(Deployment Descriptor File)

```
<?xml version="1.0" encoding="UTF-8"?>
 <web-app>
 <servlet>
 <servlet-name>dispatcher</servlet-name>
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>dispatcher</servlet-name>
 <url-pattern>*.htm</url-pattern>
 </servlet-mapping>
 </web-app>
```

Now when user send the request with url /home.htm it will be received by servlet container and it will go to deployment descriptor file (web.xml) as there url-pattern is *.htm it will match with that and forward the request to DispatcherServlet. Now DispatcherServlet will forward the request to Controller. But, DispatcherServlet does not know to which controller it has to forward the request. To solve this problem of DispatcherServlet HandlerMapping comes into picture.

HandlerMapping is a component which will map the incoming request to corresponding Controller. It will take the incoming request from DispatcherServlet and finds the proper Controller for that request. We can write our own HandllerMapping class and configure it as a bean, but the problem is DispatcherServlet will not know which method of our class has to be called. So, when we write our own HandlerMapping class then we have to implement it from HandllerMapping Interface and override the

getHandler(HttpServletResquest request) method provided by Spring like following-

```
class MyHandlerMapping implements HandlerMapping
 {
 HandlerExecutionChain getHandler(HttpServletrequest request)
 {
 //logic for mapping the handler based on request and return the handler.
 }
 }
```

Instead of writing our Own HandlerMapping spring has provided lot of implementation class for HandlerMapping which we can use to map the Controller. Some of the HandlerMapping implementation are-

1.SimpleUrlHandlerMapping

2.BeanNameUrlHandlerMapping

3.ControllerBeanNameHandlerMapping

4.ControllerClassNameHandlerMapping

In our first application let's use SimpleUrlHandlerMapping.We will discuss about various HandlerMapping implementaion classes in later section. As this implementation class is provided by spring DispatcherServlet knows which method it has to call to get the handler (Controller). Now to call the getHandler(…) method DispatcherServlet needs object of SimpleUrlHandlerMapping but it cannot create because, it will get tightly coupled with it. To solve this problem spring has provided dependency injection mechanism. DispatcherServlet will create IOC container by reading the spring bean configuration file.Configuration file name should be "<Servlet-name>-servlet.xml" by default DispatcherServlet will looks for the dispatcher-servlet.xml file to create the IOC container. Spring bean configuration file name should starts with <servlet-name> configured in web.xml then append -servlet.xml to it. Let's configure SimpleUrlHandlerMapping in spring bean configuration file-

dispatcher-servlet.xml

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
 <property name="mappings">
 <props>
 <prop key="/hello.htm">homeViewController
</prop>
</props>
</property>
</bean>
```

In the above configuration file homeViewController will be the bean id of View Controller which we will configure in dispatcher-servlet.xml later. We have configured the SimpleUrlHandlerMapping without any bean id because DispatcherServlet always takes any HandllerMapping implentations as HandllerMapping only. We will understand it when we discuss internals of DispatcherServlet. Now DispatcherServlet will get the SimpleUrlHandllerMapping bean from IOC container and keep it as HandllerMapping reference

because it will not know which HandllerMapping is configured by programmer. After that DispatcherServlet will invoke getHandller(HttpServletRequest request) method of SimpleUrlHandllerMapping to get the Controller bean id. SimpleUrlHandlerMapping will take the request and try to match the incoming url to the configured property key of mappings url if matched then it will return the value which is the bean id of the Controller to DispatcherServlet. Now by getting the bean id of the Controller DispatcherServlet will get the Controller from the IOC container and forward the request to it by calling handle(request,respose) method.We will discuss about internals of DispatcherServlet in later section.

Now Let's create a Controller class which will handle the request and then configure it in spring bean configuration file (dispatcher-servlet.xml). Just creating any class will not become Controller class because DispatcherServlet will not know which method it has to call to forward the request so, Spring has given an interface called Controller and its multiple implementation class. to make our class as Controller class we have to either implements from Controller interface or extends from its implementation classes. We will discuss about various implementation classes in later section. At this time let's extends our class from AbstractController class which is one of the implementaion class provided by spring.

Controller

```
class HomeViewController extends AbstractController
{
public ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletResponse responce)
{
ModelAndView mav=null;
mav.setViewName("home");
return mav;
}
}
```

AbstractController class contains contains handleRequestInternal(req,res) method which will return ModelAndView to the DispatcherServlet. To process the request we have to override this method . In this application we have nothing to process we just want to render home.jsp page so create the ModelandView object and set the logical view name to it and return the object to DispatcherServlet. Here Model means data which we want to send to the user after processing the request but at present we have not any data so, we are not setting any data.

dispatcher-servlet.xml

```
<!-->HandllerMapping Configuration<-->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/hello.htm">homeViewController</prop>
</props>
</property>
</bean>
```

```
<!-->Cotroller Configuration<-->
<bean id="homeViewController" class="HomeViewController"/>
```

Now After getting the ModelAndView object from Controller DispatcherServlet has to render the view but it does not know which type of view he has to render. To solve this problem ViewResolver comes into picture. ViewResolver is the component which will take the logical view name and resolve it to the View class object which is to be rendered. Now DispatcherServlet has logical view name so it will call the ViewResolver by passing the logicla view name. To get the ViewResolver Dispatcher Servlet will have to find it in IOC container. So, let's configure the ViewResolver in spring bean configuration file(dispatcher-servlet.xml). Again tere are lots of ViewResolver impementations are provided by spring so we not need to create our own ViewResolver. Let's use one of the ViewResolver implementations provided by Spring. Here we are using InternalResourceViewResolver.

dispatcher-servlet.xml

```
<!-->HandllerMapping Configuration<-->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/hello.htm">homeViewController</prop>
</props>
</property>
</bean>
<!-->Cotroller Configuration<-->
<bean id="homeViewController" class="HomeViewController"/>
<!-->ViewResolver Configuration<-->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/"/>
<property name="suffix" value=".jsp"/>
</bean>
```

We have not configured InternalResourceViewresolver with bean id because DispatcherSerrvlet treat all Implementations of ViewResolver as ViewResolver only. Now ViewResolver will takes the logical view name and appends prefix and suffix to it to get the view location after finding the view it will creates the View Type(Jstl for jsp) class and its object which is to be rendered and returns the object of that class to DispatcherServlet. Now DispatcherServlet will call the render(model) method of that View class by passing model (data) to render the respective view to the user. In this case there is no data so model will be null and home.jsp page will be rendered.

In this way various components of Spring MVC communicate with each other to process the request. We will discuss each and every component in details in later section.