Spring Material: Part-5
Spring MVC

Spring Model View Controller

➔Spring MVC is the module which use for developing web application.
➔In this module we will see different features & advantages when compared with other web based frameworks.
➔Spring MVC has overcome the problem which is available with other frameworks. Almost all the web-based require component has provided by spring people.

Differences between spring MVC and Other web-based Framework?

1st Difference

➔Many of existing web MVC frame work that are there it could be struts, it could be JSF or any other web MVC modules or web MVC frame work that are there in the market.
➔If you see most of MVC framework that are there they support the view component as JSP or HTML only.
➔They never support any view or any other technology will be a view component to rendering a view as a output to the end user.
➔So that is one of the serious problems that are there in many of the web MVC framework that is there in the market.
➔ Why because any of the web MVC frame work that is there in the market will always assumes the view as a concrete component that is never be change and always default view technology to be JSP or HTML only .
➔Unlike existing web MVC frame work likes comes to the Spring, Spring never assumes the view as an concrete component.
➔Spring always assume that view as an Interface type.
➔That does not mean that Interface is the view .That means it always has to design to have a swapping and placing anything as a view.
➔Spring never model as the view as a concrete component rather it has always has been design keeping in view as Interface because in spring you can assume or imagine view as a interface for which any number of implementation can be render.
➔In comes to the spring MVC module a view can be not only the JSP or else HTML a view can be retrieving any other technology it could be PDF,XML,Flex,Flax or a simple image also which can be capable of rendering the output to the user can be use as a view.
➔So it is not a concrete component rather view always treated as a interface .So that spring MVC has flexibility of supporting multiple technology consider as a view .

2nd Difference

➔In other web-based framework they made request wrapping process automated, but they fail in handling
Attribute value/conversation.
Ex: If one of the fields is taking int values but user has entered wrong input while mapping it will generate a big stack trace to the user, which user cannot understand?
➔In **spring** if any attribute conversion problem is there then spring will not generate a big stacks trace rather it will render same page to user along with validation message to render correct data even it will reserve previous data also.

3rd Difference

➔In struts to perform request wrapping programmer has to provide one VO class which contain all the attributes as String & attribute names & input field name should be same.
➔Moreover programmer has to write struct-config.xml to tell to ActionServlet and that VO class extends from ActionForm class.

➔Spring has one command class which help in spring user input in original form there is no need to type conversion spring has one great feature ie. properrtyEditer which automatically build corresponding values to particular attributes .

4th Difference

➔In struts validation logic written by programmer but it become boiler plate logic if there are multiple view available with same fields then for every view we have to write validation logic .Validation logic tight to each & every ActionFrom.
➔We cannot use same validation logic for multiple view/ActionForms.
➔Unlike to struts, **Spring** has provided validators which are separate from any of the form. we can easily use those validators to validate forms.

5th Difference

➔In struts ActionForm contains total String attributes again programmer has to write VO class to receive all the actionFrom values to Vo class after that business form to perform that operations.
➔**Spring** has provided command class there is no need to write any conversation logic rather command is the simple POJO class with original values we can directly use this class to perform business operations.

6th Difference

➔To apply common prepossessing logic or post processing logic functionality struts has provided filter as web component.
➔Unlike struts, **Spring** has provide more advanced feature i.e. interceptors which apply common functionality.
➔These Interceptor will not check in entry level these Interceptor are sit in spring MVC level and called along with controller class those are not called with servlet.
➔So which controller you want to apply common prepossessing logic or post processing logic you can apply.
➔So if url also change there is no effect of Interceptor.

7th Difference

➔Many of the frameworks like Struts doesn't has support for themes resolver, Time zone Locales and so on.
➔**Spring** has provided a greater support for them resolver Time zone & so on .It also support internalization.

8th Difference

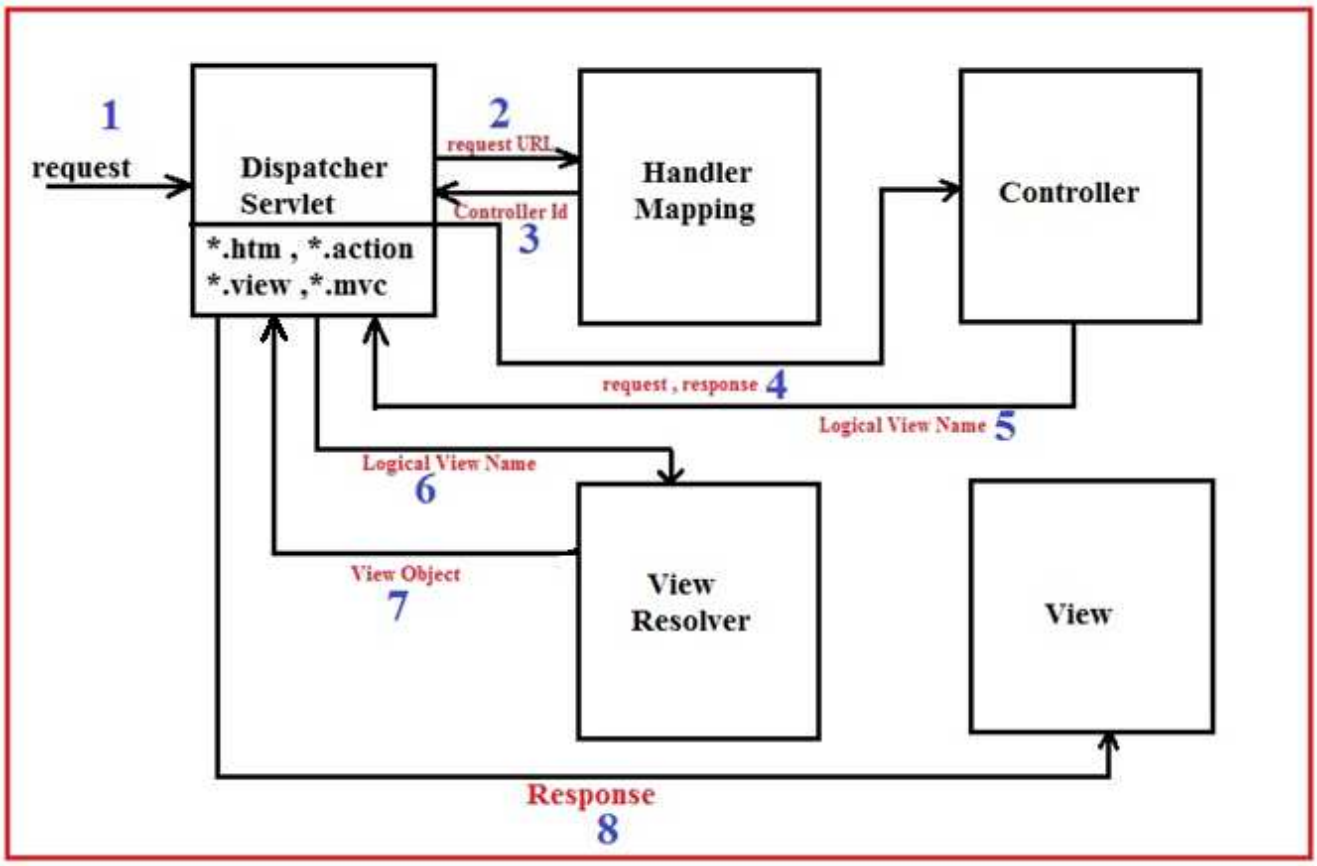
➔In struts writing strut-config.xml is critical job because programmer has to write mapping configuration i.e. Form mapping as well as action mapping.
➔If there is change in front URL then again we have to change struts-config.xml, maintenance cost more.
➔Unlike struts, **Spring** doesn't provided an configuration file, it's automatic process in spring if there is change so there is no need change anything.

9th Difference

➔In struts ActionClass read the data to the JSP inform of ActionForm .
➔In Action class execute() method will be called as a parameter request, response, ActionForm, ActionMapping as a return type as a ActionForward .

➔I can read the data from JSP page in terms of ActionForm .but we never use req.getparameter() then why I write the req.
➔The request, responses are not required but we are writing un-necessary.
➔But when it comes to spring we can model our controller in such a flexible way , what is been required as an input we can take only those parameter in the pojo classes that is added in spring 3.2.

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 whome 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.

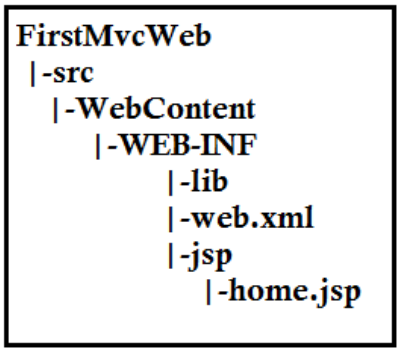
➔View resolver will take the logical view and creates the object of the view which should be rendered by DispatcherServlet and return the View Object to DispatcherServlet.

➔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 understood 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



➔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 to 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-

1st Problem

➔WebContent is a public directory.
➔Views in this directory can be accessed by any users directly.
➔In this case our presentation layer technology will get exposed to End-User.

2nd Problem

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

3rd Problem

➔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).

Ex: 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.

4th Problem

➔Whenever every time change the URL Google search engine will not able to recognize the URL.

Advantages in place the views inside WEB-INF

➔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 anything 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.
➔Anything 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

Internals

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

home.jsp

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>First Spring MVC App</title>
    </head>
    <body>
        <h1 style="color: green; font-family: fantasy; ">Welcome to my First Spring MVC Application !!!</h1>
    </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)

```
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

➔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 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-

```
public abstract interface HandlerMapping {
        public abstract HandlerExecutionChain getHandler(
                        HttpServletRequest paramHttpServletRequest) throws Exception;
}
```

➔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 in which of them is SimpleUrlHandlerMapping.

➔In our first application let's use SimpleUrlHandlerMapping.

➔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

```
<!-- Handler Mapping -->
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
      <property name="mappings">
            <props>
                  <prop key="/home.htm">viewHomeController</prop>
            </props>
      </property>
</bean>
```

➔In the above configuration file viewHomeController will be the bean id of View Controller which we will configure in dispatcher-servlet.xml later.

➔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 handleRequest(request,respose) method.

➔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

➔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 there 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

dispatcher-servlet.xml

```
<?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">

       <!-- Controller  -->
       <bean id="viewHomeController" class="com.fm.controller.ViewHomeController"/>

       <!-- Handler Mapping -->
       <bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
            <property name="mappings">
                 <props>
                      <prop key="/home.htm">viewHomeController</prop>
                 </props>
            </property>
       </bean>

       <!--View Resolver -->
       <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

            <property name="prefix" value="/WEB-INF/jsp/"></property>
                 <property name="suffix" value=".jsp"/>
       </bean>
</beans>
```

➔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 should create an extra directory for all the view under WEB_INF…

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

➔The outside people who are not part of the application those are not able to access the content of WEB-INF

➔Because WEB-INF are the protected DIRECTORY in the Web-Application..

➔The only who are internal part they are only able to access so controller able to access The WEB-INF

➔ The controller will tell to the dispatcher servlet which view have to render...after complete the processing the request controller tell the dispatcher to render the view...

➔Dispatcher servlet is an servlet so we have to configure it under the web.xml....

➔Its recommended to give the Servlet name as dispatcher servlet. The class name of the servlet is org.springframework.web.servlet.DispatcherServlet

➔The Dispatcher servlet don't know how to processing the request...programmer know how to processing the request.

➔So he configure an HandlerMapping to identify the appropriate Controller so we have to write a HandlerMapping in somewhere then dispatcher servlet will go to the HandlerMapping and takes the request then as per the request its identify an Controller .

➔So we have to configure in spring-beans configuration file that dispatcher servlet able to identify the Handle Mapping.

➔The dispatcher servlet create the IOC container For one time with in the lifetime of the application so its create in init().

➔The name of the servlet we give in the web.xml its look in the application config file as 'dispatcher-servlet.xml' and its create the IOC and we have to create it in WEB_INF

➔Whenever request came into DispactherServlet its don't know how to process it....so the programmer know how to process it
➔So we are configuration the handler mapping in Spring Bean configuration file

➔With in a typical Web application there are some other part rather than the presentation layer....so we want HandlerMapping, Controller also required.
➔As well as the Business tier component and persistence logic also required... Like Service and dao.....
➔Controller is going to call the service. dispatcher servlet will call the Controller.

➔Service has to inject to controller and dao has to inject into the service and we have to configure in spring bean config file.
➔I have more than one controller. I have bunch of controller are there with in mvc application. if all are have to config into dispacther-servlet.xml.
➔But the problem is there if we don't want tomorrow, the spring Framework in presentation layer then there is a problem to extract the business component and persistence component.
➔If dispatcher has to create the IOC with including the Service and dao then we can't not separate them.
➔So we need an extra component who is able to create the IOC container for the Business and persistence tier of application
➔So I have to write and different spring bean config file and create an extra IOC so tomorrow if we change our framework to struts then no need to writes DAO and Service again
➔The service has to inject into the Controller so service and dao are created before controller has created.

➔So find someone who able to read mah spring config file and create and IOC container during the start-up of application that is ContextListner
➔its take service and dao and create an ioc container ….
➔So i have to create an contextListner ……every application has needed of listener so spring has provide a listner class ContextLoaderListener…..
➔It creates an ioc container just the application start-up time before servlet has been Instantiated by servlet-config..
➔So we have to pass the config file the contextLoaderListener…when we pass the servlet values as init-param so to listener we pass context-param

```
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/application-context.xml</param-value>
</context-param>
```

➔its create an IOC where as bean service and dao we configure
➔If we are not write load-on-startup its will try to create the servlet at the time of 1st request at the time might be discover some in consistency may occur so we much before to detect them we have create the Servlet…so avoid the handling the1st req so we have to create it much before… Even its is optional we its recommended to configure load-on-startup
➔But we have to create as <load-on-startup>2</load-on-startup>
➔Because the service has to create by the listener then only it make use of the service and create another IOC
➔But application server are create listener first and then servlet
➔if one component is in one IOC and talk to another IOC container then go for nested IOC container .
➔So dao and service is have to parent bean factory and dispatcher have to child bean factory.

Example#1

home.jsp

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>First Spring MVC App</title>
    </head>
    <body>
        <h1 style="color: green; font-family: fantasy; ">Welcome to my First Spring MVC Application !!!</h1>
    </body>
</html>
```

Controller

```
public class ViewHomeController implements Controller {
        @Override
        public ModelAndView handleRequest(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
                ModelAndView mav=new ModelAndView();
                mav.setViewName("home");
                return mav;
        }
}
```

dispatcher-servlet.xml

```
<?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">

       <!-- Controller  -->
       <bean id="viewHomeController" class="com.fm.controller.ViewHomeController"/>

       <!-- Handler Mapping -->
       <bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
            <property name="mappings">
                 <props>
                      <prop key="/home.htm">viewHomeController</prop>
                 </props>
            </property>
       </bean>

       <!--View Resolver -->
       <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

            <property name="prefix" value="/WEB-INF/jsp/"></property>
                 <property name="suffix" value=".jsp"/>
       </bean>
</beans>
```

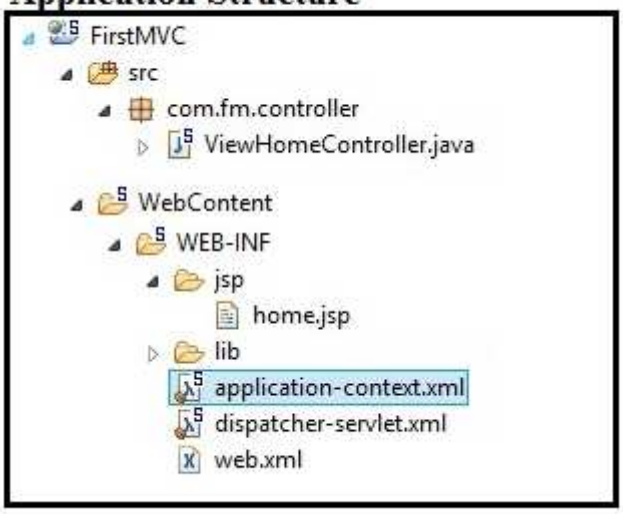
application-context.xml

```
<?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">
</beans>
```

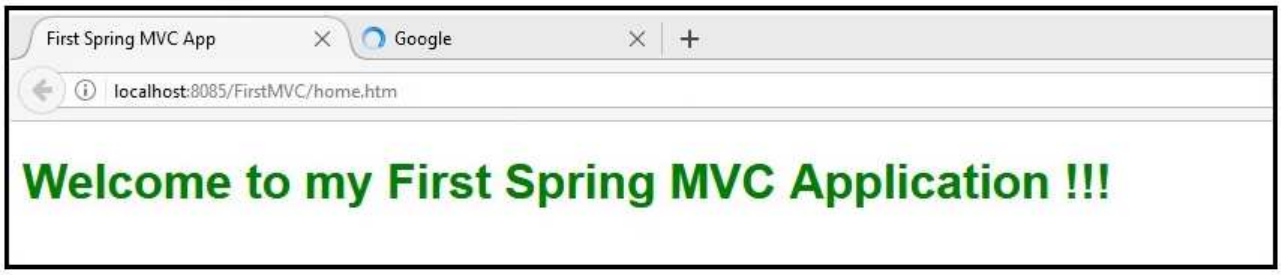
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>FirstMVC</display-name>
  <welcome-file-list>
    <welcome-file>home.htm</welcome-file>
  </welcome-file-list>
       <listener>
            <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
       </listener>
       <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/application-context.xml</param-value>
       </context-param>
       <servlet>
            <servlet-name>dispatcher</servlet-name>
            <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
            <load-on-startup>2</load-on-startup>
       </servlet>
       <servlet-mapping>
            <servlet-name>dispatcher</servlet-name>
            <url-pattern>*.htm</url-pattern>
       </servlet-mapping>
</web-app>
```

Application Structure



Output



Internal of Request processing in spring MVC?

➔Once the request comes to DispatcherServlet then it will passes to controller for processing further processing by help of HandlerMappling.

➔But to receive each and every request first we have to configure DispatcherServlet into web.xml with standard url pattern.

➔For a servlet-name we can use any name but it is recommended to use dispatcher and the servlet-class should be fully qualified name.

➔Once DispatcherServlet has been configured in web.xml now we have to look for other component which help Dispatcher to fulfil the request processing.

➔Now other components are HandlerMapping ,ViewResolver, controller these component written by programmer because programmer is the who know for which url pattern which controller has to map and once controller return the logical view name to the DispatcherServlet which view has to be rendered.

➔These things know to the programmer but we have to tell to the DispatcherServlet these are the component which help you to fulfil the request processing so to tell to the Dispatcher we have to write one spring bean configuration file which contains all the additional components as bean.

➔But how DispatcherServlet going to read this spring bean configuration file ,for that we have to follow some standard to writing that bean file .

➔Spring bean file name should be “dispatcher-servlet.xml”(<servlet-name>-servlet.xml).Then only DispatcherServlet will able to read that file.

➔While servlet loading in init method it will read the dispatcher-servlet.xml file and place as a IOC container in logical memory, means load-on-startup all the beans going to be created and place into logical memory of the IOC container.

➔Now my DispatcherServlet can easily talk to the other component to accomplish the request processing because all are the part of same IOC container.

➔Rather than controller ,viewResolver other component also will be there for example service, DAO and so on ,there are 100’s controller services and DAO’s will be there then it is very hard to place all into one spring bean configuration file.

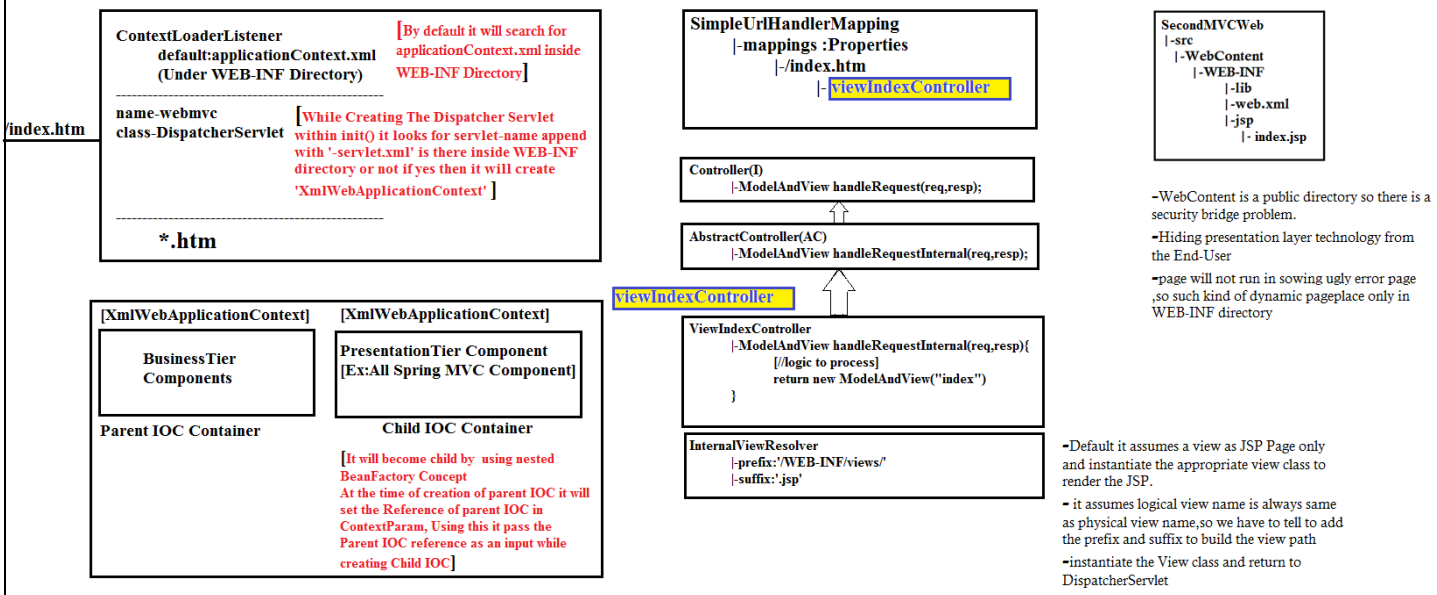
➔Some of them presentation specific beans and some of them Business and persistency also so it is very difficult to identify each other, and if there is change in one of the beans we have to check

entire file ,means readability is not there and more over my presentation logic mix with business logic and persistency logic.
➔If i want to change presentation tier then all my business and persistency gets collapsed because of all are part of one spring bean configuration file.
➔To keep my Business and Persistency tier safe at any situation we not mix with presentation tier.
➔Write one more configuration file and it contains Business tier and Persistency tier bean only .But writing configuration file is not enough we have to tell to the DispatcherServlet /some one to read this file into IOC container .
➔Now check the dependency if any service and DAO's are there then only controller going to speak to service and service is speak to DAO's.
➔If service is there then we can inject service into controller if DAO's are then we can inject DAO's into service.
➔Means before loading my DispatcherServlet my other bean file has to be loaded then only we can inject service to the controller and DAO's to the service.
➔So what is the name of the new bean file i.e. **application-context.xml.**
➔So which is the component who get's loaded at first moment once the servletContext object has been created i-e. contextListeners.
➔Listeners are the classes which get's loaded more early then any of the other component so to load application-context.xml spring has provided one Listener called **ContextLoaderListener** .
➔So we have to configure this listener into web.xml file with application-context.xml file to tell to the ContextLoaderListener.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>FirstMVC</display-name>
  <welcome-file-list>
    <welcome-file>home.htm</welcome-file>
  </welcome-file-list>
      <listener>
            <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
      </listener>
      <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/application-context.xml</param-value>
      </context-param>
      <servlet>
            <servlet-name>dispatcher</servlet-name>
            <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
            <load-on-startup>2</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>dispatcher</servlet-name>
            <url-pattern>*.htm</url-pattern>
      </servlet-mapping>
</web-app>
```

➔Then we have to configure context parameter which going to point our application-context.xml and ContextLoaderListener can easily read that file and create logical memory by placing IOC container which contains all beans.
➔Now First my application-context.xml going to load by Listener after that my DispatcherServlet going to create another IOC container by loading dispatcher-servlet.xml.
➔Now both the IOC container are the part of one project if they wanted to talk to each other then spring has to used nested IOC container concept.
➔ContextLoaderListener is the web component and it is the first component of servlet container which gets called by servletContainer .
➔Actually ContextLoaderListener is the spring specific listener but it has been extend from **ContextListener** which is servlet container part **.**



What is the reason to create two IOC container and how they going to inject with each other.

➔First of all spring is the non-invasiveness framework to keep that promise spring allows programmer to create two IOC containers.
➔dispatcher-servlet.xml one for spring specific component.
➔applicationContext.xml one is for Business layer component means in future if we don't want to use spring mvc then we can get read out from spring mvc easily without disturbing the existing business tire logic and we can replace any web-based frameworks to work with rest of the component easily.

How internally both IOC container gets created and interact with each other ?

➔First ContextLoaderListener will create an IOC container by reading application-context.xml which we have given input by configuration into web.xml.
➔It will read spring bean configuration file and it will create IOC container by placing all beans into IOC container.
➔Once it created IOC container, It will place the reference of that IOC container into Servlet context when my DispatcherServlet start to create IOC container before that it will check any parent IOC reference available into servlet context or not if it is available.
➔Thin it will take an reference and pass to the DispatcherServlet created IOC container to create IOC container along with parent reference(nested bean factory concept) means both can easily can interact with each other.
like this not exactly…

```
ApplicationContext parent=new ClassPathXmlApplicationContext("applicatioContext.xml");
ApplicationContext child=new WebApplicationContext("dispatcher-servlet.xml",parent);
```

Spring application has two types of context configuration files for Spring MVC module:

1. **ApplicationContext (default name for this file is applicationContext.xml)**
2. **WebApplicationContext (default name for this file is xxx-servlet.xml where xxx is the DispatcherServlet name in web.xml)**

ApplicationContext

➔ applicationContext.xml one is for Business layer component means in future if we don't want to use spring mvc then we can get read out from spring mvc easily without disturbing the existing business tire logic and we can replace any web-based frameworks to work with rest of the component easily.
➔Spring loads applicationContext.xml file and creates the ApplicationContext for the whole application.
➔There will be only one application context per web application.
➔If you are not explicitly declaring the context configuration file name in web.xml using the contextConfigLocation param,
➔Spring will search for the applicationContext.xml under WEB-INF folder and throw Exception if it could not find this file.
➔We can add multiple configuration.

WebApplicationContext

➔Apart from **ApplicationContext**, there can be multiple **WebApplicationContext** in a single web application.
➔In simple words, each DispatcherServlet associated with single WebApplicationContext. xxx-servlet.xml file is specific to the DispatcherServlet.
➔Spring will by default load file named "xxx-servlet.xml" from your webapps WEB-INF folder where xxx is the servlet name in web.xml.
➔If you want to change the name of that file name or change the location, add initi-param with namespace as param name and value is filename .
➔If you are not explicitly declaring the context configuration file name in web.xml using the contextConfigLocation param,

```
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/application-context.xml;/WEB-INF/security-bean.xml/</param-value>
</context-param>
```

```
<servlet>
        <servlet-name>webmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>namespace</param-name>
                <param-value>webmvc</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>webmvc</servlet-name>
        <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

ContextLoaderListener

➔Performs the actual initialization work for the root application context.
➔Reads a "contextConfigLocation" context-param and passes its value to the context instance, parsing it into potentially multiple file paths which can be separated by any number of commas and spaces, e.g.

"WEB-INF/applicationContext1.xml;WEB-INF/applicationContext2.xml".

➔ContextLoaderListener is optional. Just to make a point here: you can boot up a Spring application without ever configuring ContextLoaderListener, just a basic minimum web.xml with DispatcherServlet.

What is HandlerMapping? What is the purpose of HandlerMapping

➔First the user send the request to the Application, Once the application get deployed.
➔ContextLoaderListener create IOC container after that before executing a particular DispatcherServlet, init() method gets called.
➔And Dispactcher-servlet.xml file going to read by that init() method and create another IOC container.
➔Once it finish DispatcherServlet takes the request and apply the common processing logic, system resources and plumbing logic and all once is finish it has to send to one of the controller to handle the request processing.
➔But DispatcherServlet don't no to which controller he has to send that particular request.
➔So it is known to programmer means programmer has to tell to the DispatcherServlet to which controller he has to send that request .so to configure number of controller HandlerMapping comes to picture .

How HandlerMapping is the Component which Keep track the urlpatterns and corresponding controllers to handle the request ?

➔**T**he programme has to write one class which going to implements from HandlerMapping and he can manually write the url and controller into one map to handle it.
➔But spring has provided some predefined HandlerMapping which programmer has to use one of them as

```
|SimpleUrlHandlerMapping ->class
        |-mappings
                |-props
                        |-key➔url
                            |-values➔controller
```

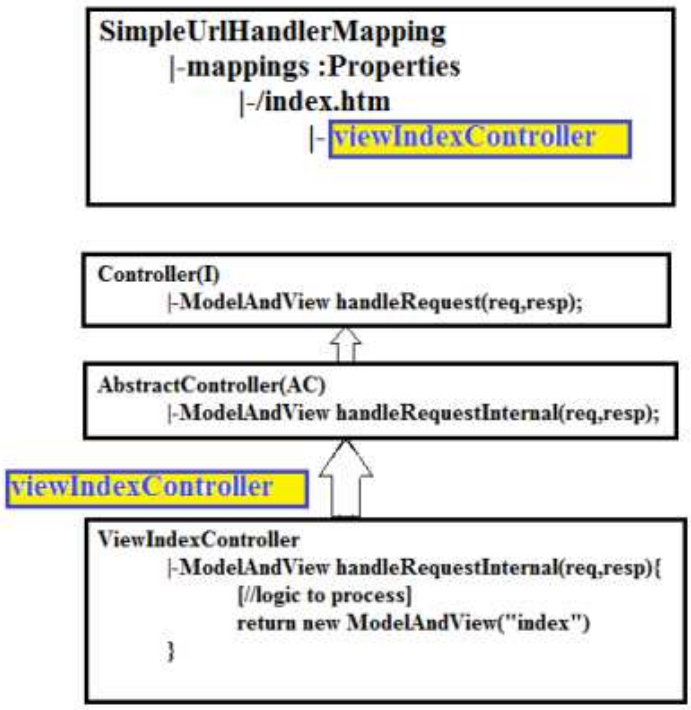
➔Just we need to configure as bean into dispatcher-servlet.xml and provide corresponding controller and url information.
➔Once DispatcherServlet get corresponding controller it will forword/call the controller to process the request.
➔But to call that particular controller DispatcherServlet has to know whom he has to call or which method he has to call .
➔You cannot write any class and we can't say this is a controller.
➔For this we have to write controllers by extending or implementing by spring specific interface or class.
➔To write an controller spring has provided one interface and one abstract class which contains some method, those overridden by our class .

Controller (Interface)
|-ModelAndView handleRequest(req,,resp)

➔We can write one our own class by implementing from controller and we can override that method for handling the request.
➔One more abstract class has to provided by spring to works with controller, we can use that abstract class to create an controller and i.e.

AbstractController(Abstract class)
ModelAndView handleRequestInternal(req,resp)

Let's see if we extending our class from AbstractController what is the flow in diagrammatic representation.



How dispatcher servlet know which HandlerMapping its has to go?

➔Its directly go to the IOC because it's have the ref of WebApplctionContext as attribute ..
➔Spring people has provide an interface all the handlerMappping are implementation from its so its always look for an Interface not the implementation class.
➔Whatever you have wrote its search for Interface so we have to flexible of our own HanlerMapping also we able to create. it is called polymorphism.
HandlerMapping mapping=webApplicationContext.getBean(SimpleUrlHandlerMapping.class);

How dispacther servlet come to know its bean id?

➔getBean has a Method every handlerMapping impl from HandlerMapping Interface By the help of class we get the Bean ,Handler mapping always talk to the Class its identified by the Class....

Example#2

Controller Class

```
public class ViewIndexController extends AbstractController {
	@Override
	public ModelAndView handleRequestInternal(HttpServletRequest request,
			HttpServletResponse response) throws Exception {
			//logic
		return new ModelAndView("index");
	}
}
```

web.xml

```
<web-app>
  <display-name>FirstMVC</display-name>
  <welcome-file-list>
    <welcome-file>index.htm</welcome-file>
  </welcome-file-list>
        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>
        <servlet>
                <servlet-name>webmvc</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <load-on-startup>2</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>webmvc</servlet-name>
                <url-pattern>*.htm</url-pattern>
        </servlet-mapping>
</web-app>
```

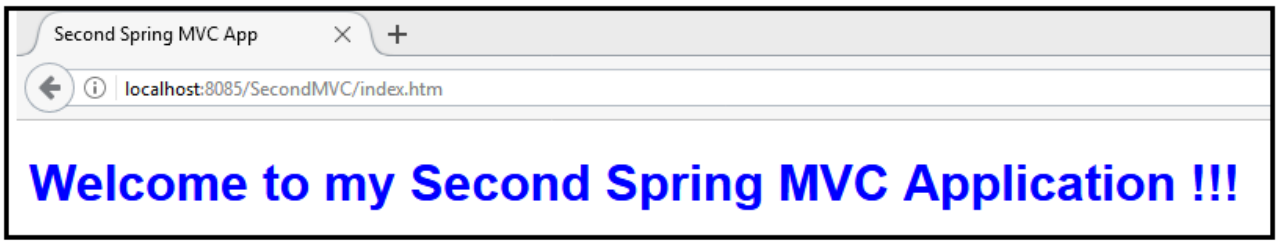
application-context.xml

```
<?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">


</beans>
```

webmvc-servlet.xml

```
<beans>
        <!-- Controller  -->
        <bean id="viewIndexController" class="com.sm.controller.ViewIndexController"/>

        <!-- Handler Mapping -->
        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
                <property name="mappings">
                        <props>
                                <prop key="/index.htm">viewIndexController</prop>
                        </props>
                </property>
        </bean>
        <!--View Resolver -->
        <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

                <property name="prefix" value="/WEB-INF/jsp/"></property>
                        <property name="suffix" value=".jsp"/>
        </bean>
</beans>
```

http://localhost:8085/SecondMVC/index.htm



HandlerMappling

What is HandlerMapping ? What is the purpose of HandlerMapping ?

➔Whenever DispathcerServlet has received the request, He has to map the request to controller.
➔But dispatcher servlet will not know for the given request which controller class the programmer has to write in HandlerMapping ,for which request which is the controller to the DispatcherServlet.
➔So the DispatcherServlet is the one who is receiving the request but as it don't know which controller it need to call that's why the dispatcher servlet will talk to the HandlerMapping .

How does the HanderMapping will know for which request what is the controller that has to delegate the request to ?

➔That is something that has to be configure by the programmer For the information related to the request and the controller to this the request has to be mapped to the hendlerMapping .
➔So that upon receiving the request the DispatcherServlet Quickly approach to the HandlerMapping in identifying the controller based on the configuration we have given .
➔So that the HandlerMapping give the name of the controller to the DispatcherServlet .

How do I need to work with HandlerMapping ?How can i use the HandlerMapping ?

➔You can write your own HandlerMapping but you can't write any class and you can't call as a HandlerMapping class.
➔If you are writing your own HandlerMapping then your class must should have implement HandlerMapping Interface and should override one method i.e.**HandlerExecutionChain getHandler(HttpServletRequest request)** which is going to take the request and going to returns the HandlerExecutionChain.
➔**But it not required to defined our own HandlerMapping ,**because Spring itself has provided lot of implementation classes for the HandlerMapping interface.
➔So you never need to find a requirement with creating your own HandlerMapping by implementing HandlerMapping interface Such a use case will not come .
➔Because Bunch of HanderMapping implementation class has provided by the spring as part of the Spring MVC.
➔You can use one of the HandlerMapping rather than creating your own handlerMapping.

How Many Implementation Spring has to provided by the HandlerMapping?

➔Handler Mapping is a component that maps incoming request to controller/handler class.
➔All Handler Mapping class are implementation of "HandlerMapping interface.
The implementation Handler mapping classes are :

1. **BeanNameUrlHandlerMapping**
2. **SimpleUrlHandlerMapping**
3. **ControllerClassNameHandlerMapping**
4. **ControllerBeanNameHandlerMapping (3.0 v)**
5. **DefaultAnnotationHandlerMapping (2.0v)**
6. **RequestMappingHandlerMapping(3.2v)**

➔So the spring mvc has provided total 6-HandlerMapping to us so we can use one of this HandlerMapping to work with our application.
➔But out of six last two handlerMappings are used in annotation approach only
➔Initially the spring doesn't have these many number of HandlerMapping .Initially as part of spring framework there are only 3 handler Mapping are there.

1. BeanNameUrlHandlerMapping
2. SimpleUrlHandlermapping
3. ControllerClassNameHandlerMapping

➔As part of spring 3.0 They have added ControllerBeanNameHandlerMapping and 2.0 DefaultAnnotationHandlerMapping is there .
➔After 3.2 the RequestMappingHandlerMapping has to be added .
➔So all the HandlerMapping are not there in the Day-1, Those are being added base on the implementation requirement the people are getting.
➔So finally we have big 6-number of HandlerMapping that are available.
Out of which DefaultAnnotationHandlerMapping and RequestMappingHandlerMapping we have to discussed when we have work with Annotations.
➔If you are using spring mvc with annotation then will the presence the two HandlerMapping will come .
➔When you have working with configuration approach there 4-you have to work.

Let's take one Handler Mapping after one Handler Mapping :-

SimpleUrl HandlerMapping

➔SimpleUrlHandlerMapping is called as the most easiest and flexible url handler mapping in the spring mvc.
➔The name itself will tells that it is a url handler mapping bean where the incoming url is directly taken as an input and base on the url it is going to map the request.
➔So give me the url based on the url SimpleUrlHandlerMapping can quickly find out the handler component .
➔This Handler mapping will takes url as an input by considering the url they map the request to the controller .I.e Programmer has to tell which url explicitly what is the controller that has to map the request based on the url.
➔It is the responsible to the programmer for which request what is the corresponding controller
➔This is useful to map the incoming request with controller class based on valid paths of request url and bean ids/names of the controller class.
➔We need to supply the details of values of "mappings" property of type (java.util.Properties)

Ex:

```
<!-- Controller -->
<bean id="viewHomeController" class="com.fm.controller.ViewHomeController"/>

<!-- Handler Mapping -->
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/home.htm">viewHomeController</prop>
        </props>
    </property>
</bean>
```

➔If the request url is contains "/home.htm" then it passes the request to ViewHomeController class whose bean id/name as "ViewHomeController"
➔Here any controller can map with any url .your classes are not tight with url the id of the class is being map with url .if you want to change it you can independently change the controller .
➔**The benefit to going with SimpleUrlHandlerMapping is what it even permit to have wild card character also as part the uri(hm*.htm)**
➔You can map multiple url also in one controller .
➔Here programmer gets flexibility to map controller and he can map multiple url to to one controller also.

```
<!-- Controller -->
<bean id="viewHomeController" class="com.fm.controller.ViewHomeController"/>

<!-- Handler Mapping -->
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/home.htm">viewHomeController</prop>
            <prop key="/in*.htm">viewHomeController</prop>
        </props>
    </property>
</bean>
```

BeanNameUrlHandlerMapping:

➔BeanNameUrlHandlerMapping is the default handler mapping .
➔If you don't provided any HandlerMapping then spring by default expose BeanNameUrlHandlerMapping.
➔It is like SimpleUrlHandlerMapping only the different is here programmer going to configure beanName as url pattern which is match to incoming request url.

It is looks like

```
<bean name="/home.htm" class="com.fm.controller.ViewHomeController"/>
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

➔It will match incoming request url and corresponding bean name if any of HandlerMapping going to match .It will call corresponding controller .

ControllerClassNameHandlerMapping

➔This controller falls under **conversion over configuration**.
➔There is no need to configuring additional beans or url mapping into config file.
➔The name itself tells you by using controller class name the HandlerMapping going to happen.
Ex: **ControllerClassNameHandlerMapping**

URL	Controller
/viewwelcome.htm	ViewWelcomeController
/viewindex.htm	ViewIndexController
/viewhome.htm	ViewHomeController

➔The ControllerClassNameHandlerMapping will take class name as url by removing 'Controller' word from the class because it is common to each and every controller class.
➔Addition to that we have to configure some additional properties also.
➔if it is required we have to configure a property caseSensitive=true/false

```
<bean class="com.fm.controller.ViewHomeController"/>
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
    <property name="caseSensitive" value="false"></property>
</bean>
```

http://localhost:8085/FirstMVC/viewhome.htm

```
<bean class="com.fm.controller.ViewHomeController"/>
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
    <property name="caseSensitive" value="true"></property>
</bean>
```

http://localhost:8085/FirstMVC/viewHome.htm

ControllerBeanNameHandlerMapping

➔The corresponding controller going to indentified by the BeanName i.e id.
➔But incoming url is different and configure id is different.
➔How it will going to match.For this we have to configure additional property in ControllerBeanNameHandlerMapping i.e

Suffix=.htm /.action/.mvc/ .view

➔It will works with both the configuration i.e *.htm /*
➔When you use *.htm then just configure that class with additional property name that class with additional property name 'suffix' and value is .htm

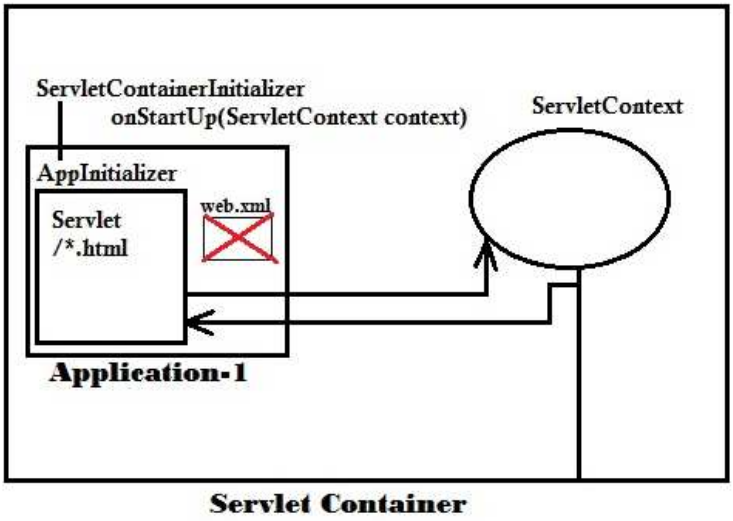
```
<bean name="vh" class="com.fm.controller.ViewHomeController"/>
<bean class="org.springframework.web.servlet.mvc.support.ControllerBeanNameHandlerMapping">
      <property name="urlSuffix" value=".htm"></property>
</bean>
```

http://localhost:8085/FirstMVC/vh.htm

Multiple Configuration in DispatcherServlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
 <display-name>FirstMVC</display-name>
 <welcome-file-list>
   <welcome-file>home.htm</welcome-file>
 </welcome-file-list>
      <listener>
            <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
      </listener>
      <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/application-context.xml;/WEB-INF/security-bean.xml/</param-value>
      </context-param>
      <servlet>
            <servlet-name>webmvc</servlet-name>
            <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
            <init-param>
                  <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/dispatcher-servlet.xml;/WEB-INF/webmvc.xml/</param-value>
            </init-param>
            <load-on-startup>2</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>webmvc</servlet-name>
            <url-pattern>*.htm</url-pattern>
      </servlet-mapping>
</web-app>
```

How can we register ContextLoaderListener and Dispatcher Servlet in ServletContext Object without using web.xml ?



WebApplicationContextInitializer

```
public class RedBusInitializer implements WebApplicationInitializer{
    @Override
    public void onStartup(ServletContext context)
                                throws ServletException {

        ContextLoaderListener contextLoaderListner=null;
        contextLoaderListner=new ContextLoaderListener();
        context.addListener(contextLoaderListner);
        DispatcherServlet dispatcher=null;
        dispatcher=new DispatcherServlet();
        ServletRegistration.Dynamic
                dy=context.addServlet("dispatcher".
                                                dispatcher);
        dy.addMapping("*.htm");
    }
}
```

Default : Listener class =applicationContext.xml
DispatcherServlet class=dispatcher-servlet.xml

If you want to change default file name

```
public class RedBusInitializer implements WebApplicationInitializer{
    @Override
    public void onStartup(ServletContext context) throws ServletException {
        XmlWebApplicationContext appContext=null;
        XmlWebApplicationContext webContext=null;
        ContextLoaderListener contextLoaderListner=null;
        DispatcherServlet dispatcher=null;

        appContext=new XmlWebApplicationContext();
        appContext.setConfigLocation("/WEB-INF/application-context.xml");
        contextLoaderListner=new ContextLoaderListener(appContext);
        context.addListener(contextLoaderListner);


        webContext=new XmlWebApplicationContext();
        webContext.setConfigLocation("/WEB-INF/webbeans.xml");
        dispatcher=new DispatcherServlet(webContext);
        dispatcher=new DispatcherServlet();
        ServletRegistration.Dynamic dy=context.addServlet("dispatcher", dispatcher);

        dy.addMapping("*.htm");
    }
}
```

Controllers

➔As we know controller is not a web component, so controller will not get call directly by container.
➔First request comes to container and container pass it to DispatcherServlet to serve it,but DispatcherServlet will not perform request processing completely.
➔So we need one more component called controller which able to process the request and return the Logical View Name.
➔Spring has provided multiple controllers as per our requirement we can use one of them:

1. **ParameterizableViewController(C)**
2. **UrlFileNameViewController(C)**
3. **AbstractController**
4. **AbstractCommandController**
5. **MultiActionController**
6. **SimpleFormController**
7. **AbstractWizadController**

Q. When we should go for ParameterizableViewController / UrlFilenameView Controller?

➔When we want render only static target page (like about-us.jsp /contact-us.jsp)as part of incoming request then we can use above controller.
➔These two controller specifically address the static component.

Q. When we should go for AbstractController?

➔ Upon clicking a hyperlink/link we wanted to display a dynamic data then we can easily use above controller.

Q. When we should go for Abstract Command Controller?

➔This controller can be used at many places, it is specific to searching, as part of request, request data is there and request wrapping also there but FormHandling is not there then we can use this Controller .

Q. When we should go for SimpleForm Controller?

➔If we wanted to perform all the aspect for request processing Like Request contains Data, Request Wrapping ,Form Handling then we can easily choose the above component/Controller.

Q. When we should go for MultiActionController?

➔In general if we wanted to handle a request then we have to write one controller for multiple action multiple controller has to write but spring has provided one controller which allows us to bind multiple action to single controller. For example page navigation form can have multiple action next, previus...

Q. When we should go for AbstractWizardController:

➔It is the controller which help the developer to gather the data using multiple wizard.
➔We cannot gather 100 of records or columns of data using jsp pages .it may load bad user experience to make good and to gather the data mostly developer follow Wizard conversion and flow driven approach .

ParameterizableViewController

➔This controller specifically for rendering static component/view.
➔The name itself tells the upon taking parameters it will return view to perform father processing.
➔ParameterizableViewController is a concrete class we can directly use and configure inside configure file. Just it will take one parameter and upon taking it, it will return logical view name for further processing.
➔For multiple static components there is no need to write multiple controllers just configure ParameterizableViewController with multiple id and map inside the HanlderMapping.
➔But the problem is if 10 static components are there 10 beans id we have to configure.
It is not a good programming rather we can use another controller
//Example SimpleUralHandlermapping
➔To avoid the above problem one more controller has provided by spring UrlFileNameViewController.

ParameterizableViewController Configuration Ex-1

```
<?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="viewHomeController" class="com.fm.controller.ViewHomeController"/>
      <bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
            <property name="mappings">
                  <props>
                        <prop key="/about-us.htm">vauc</prop>
                        <prop key="/contact-us*.htm">vcuc</prop>
                  </props>
            </property>
      </bean>

       <bean id="vauc" class="org.springframework.web.servlet.mvc.ParameterizableViewController">
            <property name="viewName" value="about-us"></property>
       </bean>

      <bean id="vcuc" class="org.springframework.web.servlet.mvc.ParameterizableViewController">
            <property name="viewName" value="contact-us"></property>
       </bean>

      <!--View Resolver -->
      <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

            <property name="prefix" value="/WEB-INF/jsp/"></property>
                  <property name="suffix" value=".jsp"/>
      </bean>
</beans>
```

UrlFilenameViewController

➔It is also one of the spring controller and it is also a concrete class ,just we need to configure it .
➔It works open taking an request url and matching that url with filenames and returning a logical view Name for Father processing.
➔Just we have to configure one time and we can map that controller to multiple url, So UrlFileNameViewController will manage it.
➔Rather configuring multiple ParameterizableViewControllers just use UrlFileNameViewController. We can easily map multiple url to that bean to handle it.

UrlFilenameViewController Configuration Ex-2

```
<?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="viewHomeController" class="com.fm.controller.ViewHomeController"/>
	<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
		<property name="mappings">
			<props>
				<prop key="/about-us.htm">vc</prop>
				<prop key="/contact-us*.htm">vc</prop>
			</props>
		</property>
	</bean>
	 <bean id="vc" class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>

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

Abstract Controller

What is the Benefits when we use Abstract Controller and why we should not use controller interface?

➔Actually, before processing any request and controller has to check is it GET method or POST method.
➔It is valid method or not, after that he verify other data and so on then only it will call our class overridden method to further processing.
➔If we write validation method checking and other verification in controller then we end up with duplication in controller then we end up with duplicating same logic across each and every controller because of that AbstractController implements controller and the provided implementation for handlerRequest() method.
➔In handlerRequest of Abstract class contains verification logic after verification handlerRequest method will call our class handlerRequestInternal for further processes.
➔We use AbstractController used when we wanted to rendered dynamic data.
➔There are some characteristics for when to use AbstractController

1. No Request data
2. No Request Wrapping
3. No form Handling

➔But just clicking on link it has to perform the further processing.

Example#1

➔ **Upon clicking a hyperlink/link we wanted to display a dynamic data then we can easily use above controller.**

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
	<bean id="wvc" class="com.redbus.controller.QuickJourneyController">
		<property name="service" ref="mjs"/>
	</bean>

	<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
		<property name="mappings">
			<props>
				<prop key="/quick-journey.htm">wvc</prop>
			</props>
		</property>
	</bean>

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

Controller

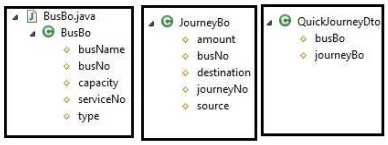
```
public class QuickJourneyController extends AbstractController{
	private ManageJourneyService service;
	public void setService(ManageJourneyService service) {
		this.service = service;
	}
	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest request,
			HttpServletResponse response) throws Exception {
		List<QuickJourneyDto> qDto=service.getRecentJourney();
		ModelAndView mav=new ModelAndView();
		mav.addObject("qDto", qDto);
		mav.setViewName("quick-journey");
		return  mav;
	}
}
```

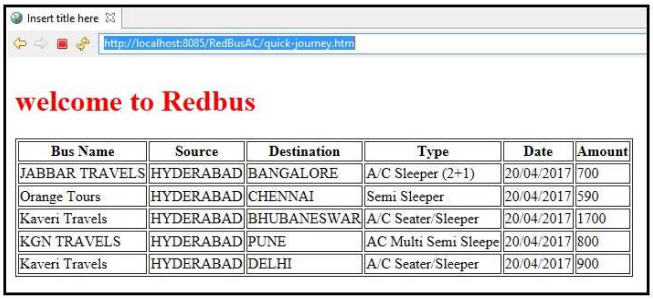
Service

```
public class ManageJourneyService {
	private JourneyDao dao;
	public void setDao(JourneyDao dao) {
		this.dao = dao;
	}
	public List<QuickJourneyDto> getRecentJourney(){
		return dao.getJourneys();
	}
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
	<import resource="config/persistancy-bean.xml"/>
	<import resource="config/service-bean.xml"/>
	<import resource="config/aop-bean.xml"/>
</beans>
```





Bus Name	Source	Destination	Type	Date	Amount
JABBAR TRAVELS	HYDERABAD	BANGALORE	A/C Sleeper (2+1)	20/04/2017	700
Orange Tours	HYDERABAD	CHENNAI	Semi Sleeper	20/04/2017	590
Kaveri Travels	HYDERABAD	BHUBANESWAR	A/C Seater/Sleeper	20/04/2017	1700
KGN TRAVELS	HYDERABAD	PUNE	AC Multi Semi Sleepe	20/04/2017	800
Kaveri Travels	HYDERABAD	DELHI	A/C Seater/Sleeper	20/04/2017	900

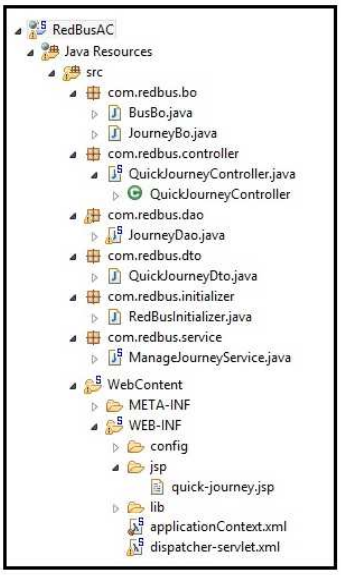
Database tables

BUS	
BUS_NO	NUMBER
BUS_NAME	VARCHAR2(50 BYTE)
SERVICE_NO	VARCHAR2(50 BYTE)
TYPE	VARCHAR2(50 BYTE)
CAPACITY	NUMBER(38,0)

JOURNEY	
JOURNEY_NO	NUMBER
SOURCE	VARCHAR2(20 BYTE)
DESTINATION	VARCHAR2(20 BYTE)
BUS_NO	NUMBER
AMOUNT	VARCHAR2(20 BYTE)

DAO

```
public class JourneyDao {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_GET_RECENT_JOURNEY=
				"select * from bus b inner join journey j on b.bus_no=j.bus_no";

	public JourneyDao(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public List<QuickJourneyDto> getJourneys(){

		return jdbcTemplate.query(SQL_GET_RECENT_JOURNEY, new RowMapper(){

			@Override
			public Object mapRow(ResultSet rs, int row) throws SQLException {

				QuickJourneyDto dto=null;
				BusBo bus=new BusBo();
				bus.setBusNo(rs.getInt("BUS_NO"));
				bus.setBusName(rs.getString("BUS_NAME"));
				bus.setServiceNo(rs.getString("SERVICE_NO"));
				bus.setType(rs.getString("TYPE"));
				bus.setCapacity(rs.getString("CAPACITY"));

				JourneyBo journey=new JourneyBo();
				journey.setJourneyNo(rs.getString("JOURNEY_NO"));
				journey.setSource(rs.getString("SOURCE"));
				journey.setDestination(rs.getString("DESTINATION"));
				journey.setBusNo(rs.getString("BUS_NO"));
				journey.setAmount(rs.getString("AMOUNT"));

				dto=new QuickJourneyDto();
				dto.setBusBo(bus);
				dto.setJourneyBo(journey);

				return dto;
			}});
	}
}
```

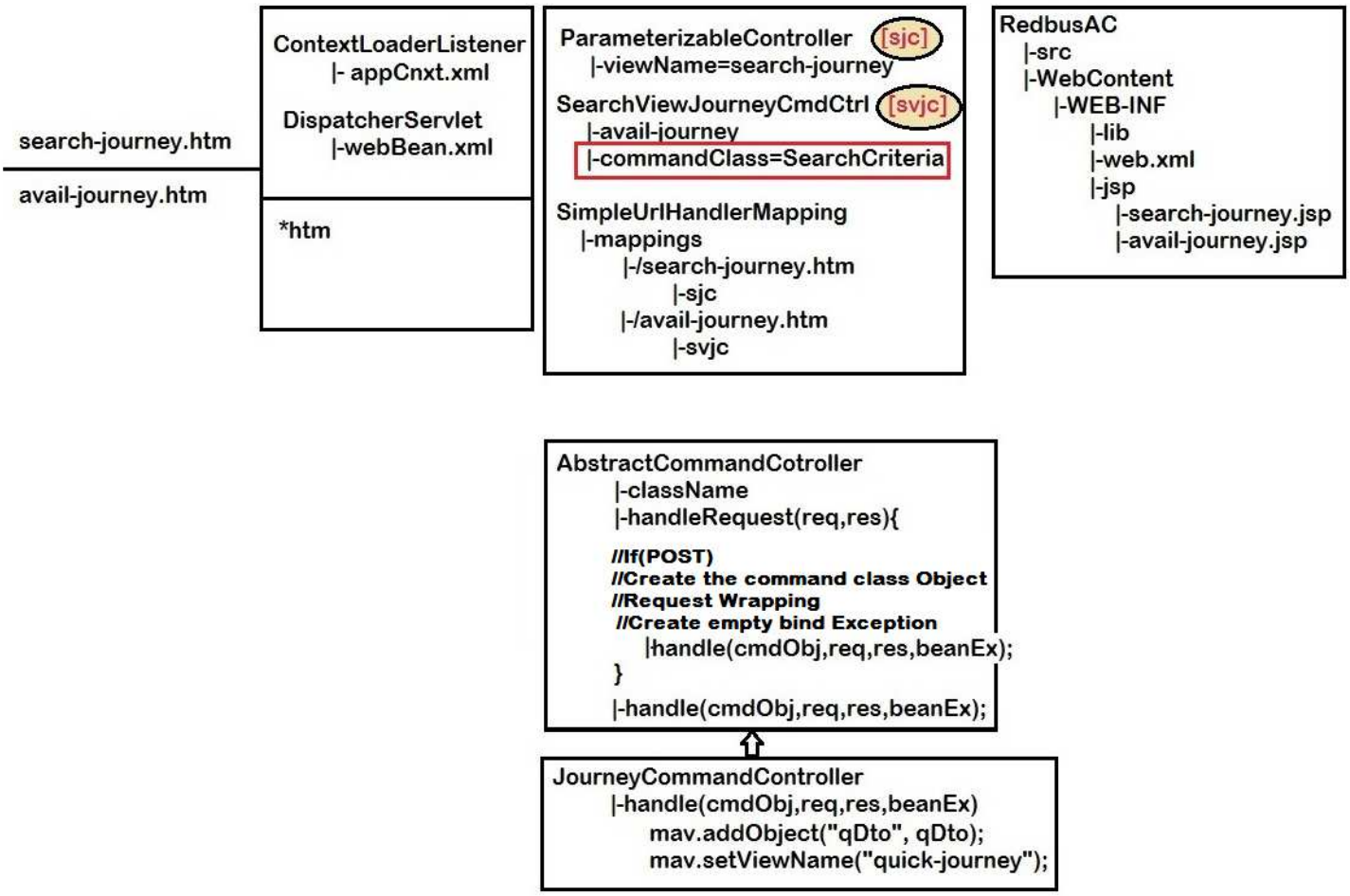


When we should go for AbstractCommandController

➔**This controller can be used at many places, it is specific to searching, as part of request, request data is there and request wrapping also there but FormHandling is not there then we can use this Controller.**

➔Here command represent a form and form represent input controller/data ,In spring command is a class which able to hold incoming data from the user.

➔All command attributes must same as input controller name.



How internally command class read by AbstractCommandController and How it will call my class by passing command Object ?

➔Internally DispatcherServlet call my class which extends from ACC and call the handle Request but my class doesn't contains any handleRequest.

➔It will goes to super class and find handleRequest is there or not.

➔Now ACC has handleRequest method once it found the method, he will read the incoming request i.e. post means user send some data then it will come to know he has to do request wrapping he will go the configure file and check any commandclass has been configured or not .if it is there then he will take the command class and create the object and perform the request wrapping.

➔Once request wrapping finished he will call my class handle method by passing req,resp,command object and binding Exception for father processing .

➔While working with AbstractCommandController we should do the Form Handling because this controller not ment for form Handling .

➔DispatcherServlet never perform request wrapping because it is not a command processing logic or pumbing logic.

search-journeys.jsp

```
<body style="font-family: consolas;font-size: 18px;">
     <form action="${pageContext.request.contextPath}/
                              do-search-journeys.htm" method="post">
          Source :<input type="text" name="source"/>
          Destination:<input type="text" name="dest"/>
          Journey Date:<input type="text" name="journeyDt"/>
                    <button type="submit">Search</button>
     </form>
</body>
```

avail-journeys.jsp

```
<body style="font-family: consolas;font-size: 18px">
     <table>
          <tr>
               <th>Source</th>
               <th>Destination</th>
               <th>Journey Date</th>
               <th>Bus Type</th>
               <th>Amount</th>
          </tr>
          <c:forEach items="${journeys}" var="journey">
               <tr>
                    <td>${journey.source}</td>
                    <td>${journey.dest}</td>
                    <td><fmt:formatDate value="${journey.journeyDt}" type="date"/></td>
                    <td>${journey.busType}</td>
                    <td>${journey.amount}</td>
               </tr>
          </c:forEach>
     </table>
</body>
```

```
<beans>

     <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
          <property name="mappings">
               <props>
                    <prop key="/search-journeys.htm">viewSearchJourneysController</prop>
                    <prop key="/do-search-journeys.htm">searchJourneyCommandController</prop>
               </props>
          </property>
     </bean>

     <bean id="viewSearchJourneysController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController">
          <property name="viewName" value="search-journeys" />
     </bean>

     <bean id="searchJourneyCommandController" class="com.rb.controller.SearchJourneyCommandController">
          <property name="commandClass" value="com.rb.command.JourneyCriteriaCommand" />
     </bean>

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

</beans>
```

```java
public class JourneyCriteriaCommand {
     protected String source;
     protected String dest;
     @DateTimeFormat(pattern = "dd/mm/yyyy")
     protected Date journeyDt;
     //Setters and Getters
}
```

```java
public class JourneyDto {
     protected String source;
     protected String dest;
     protected Date journeyDt;
     protected String busType;
     protected float amount;
     //Setters and Getters
}
```

```java
public class RedbusInitializer implements WebApplicationInitializer {
     @Override
     public void onStartup(ServletContext context) throws ServletException {
          XmlWebApplicationContext appContext = null;
          XmlWebApplicationContext webContext = null;
          ContextLoaderListener listener = null;
          DispatcherServlet dispatcher = null;

          appContext = new XmlWebApplicationContext();
          appContext.setConfigLocation("/WEB-INF/application-context.xml");
          listener = new ContextLoaderListener(appContext);
          context.addListener(listener);

          webContext = new XmlWebApplicationContext();
          webContext.setConfigLocation("/WEB-INF/webbeans.xml");
          dispatcher = new DispatcherServlet(webContext);
          ServletRegistration.Dynamic servlet = context.addServlet("dispatcher",
                    dispatcher);
          servlet.addMapping("*.htm");
     }
}
```

```java
public class SearchJourneyCommandController extends AbstractCommandController {
    @Override
    protected ModelAndView handle(HttpServletRequest request,
            HttpServletResponse response, Object command,
            BindException bindException) throws Exception {
        List<JourneyDto> journeys = null;
        JourneyDto journey = null;
        JourneyCriteriaCommand jcCommand = null;

        jcCommand = (JourneyCriteriaCommand) command;
        journeys = new ArrayList<JourneyDto>();
        journey = new JourneyDto();
        journey.setSource(jcCommand.getSource());
        journey.setDest(jcCommand.getDest());
        journey.setJourneyDt(jcCommand.getJourneyDt());
        journey.setBusType("A/C Sleeper");
        journey.setAmount(3938.34f);

        journeys.add(journey);

        journey = new JourneyDto();
        journey.setSource(jcCommand.getSource());
        journey.setDest(jcCommand.getDest());
        journey.setJourneyDt(jcCommand.getJourneyDt());
        journey.setBusType("A/C Semi Sleeper Volvo");
        journey.setAmount(938.34f);
        journeys.add(journey);

        ModelAndView mav = new ModelAndView();
        mav.addObject("journeys", journeys);
        mav.setViewName("avail-journeys");

        return mav;
    }

}
```

SimpleFormController

➔SimpleFormController has been deprecated in spring 3.0 .It is the more important controller which has been used to develop an application.

Whan to use SimpleFormController?

There are some characteristics available for a request then only we can use SFC.

1. If a request contains data
2. There is a request wrapping
3. Along with that Form Handling is there then only we can use SFC

Why it is called SimpleFormController ?

➔In general if we want to display static page we going to use parameterizableViewController or UrlFileViewController and for dynamic page we can use AbstractController.

➔Means we going to deals with multiple controller to perform the above tasks but SFC is the controller which help the programmer to render static page as well as dynamic page means it's very simple for the programmer to work.

➔In SFC there are two request considered

1.Initial phase request (GET)

2.Postback phase request (POST)

➔If we want form handling means we have to use spring tags in this controller.

Why we have to use Spring provided tags?

➔Upon requesting SFC will going to render the formview to the user.

➔After rendering user can fill that form and returns to SimpleFormController with same url patterns, But user may forget to fill some data or he has written some invalid data after validating we have to give return some page to the user will previous entered data along with error message for re-entering or re-correcting .

➔So using normal HTML or jsp page we can't perform such operation and logic because of that spring came up with their own tag library .

➔It internally going to converted into normal html tags only.

➔Whole dealing with form data rendering one more component is important i.e. command class object which help in rendering the data.

➔While writing the form tag we have to give the commandName for form data rendering by default command Name is command.

How the Spring Form looks like ?

➔Its same as HTML only just prefix form. We have to import a one tag we have to use spring tags.

➔There are almost all the tags are available in spring tag library.

```
<body>
    <form:form commandName="journey">
        Source      :<form:input path="source"/><br>
        Destination :<form:input path="destination"/><br>
        JourneyDate:<form:input path="journeyDate"/><br>
        <button type="submit">Add</button>
    </form:form>
</body>
```

Internals of SimpleForm controller

```
public class SimpleFormController implements Controller {
     String formView = null;
     String commandClass = null;
     String commandName = null;
     Validator validator = null;
     // Setters & Getters

     public ModelAndView handleRequest(request,response){
          Method method=req.getMethod();
          if (method==GET){
               //Means Initial phase Request
               //Create command class object by calling super class formBacking()
               //Read formview name
               //Add command class object to ModelAndView and return
               //to DispatcherServlet for rendering the view
          }
          else if (method==POST){
          //Means PostBack phase request
          //Creates empty command object
          //Wrap request data into command object
          //Check validator is exist or not
          if(validator!=null){
               //call support() method by passing command classType
               if(support()==true){
               //create Empty error object
               //call validate method
                    if(errors hasErrors()==true){
                         //bind error object to modelAndView with commandName.errors
                         //set the viewName FormView
                         //return controller to DispathcherServlet
                    }
               }
          }
          // call onsubmit() method
          // return modelAndView
     }
}
class AddFormController extends SFC {
     public  ModelAndView onSubmit(request,respponse,command,BindException){
          //Request processing logic
     }
}
```

Example#1 (Show the Static form page and add the criteria)

Command class

```
public class AddCriteria {
        protected String source;
        protected String destination;
        @DateTimeFormat(pattern="dd/mm/yyyy")
        protected  Date journeyDate;
        //setters and getters
}
```

success.jsp

```
<body>
<h1>Journey Added</h1><br>
Source:S{source}
</body>
</html>
```

add-journey.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
   <%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
            <title>Add Journey</title>
      </head>
      <body>
            <form:form commandName="journey">
                  Source        :<form:input path="source"/><br>
                  Destination   :<form:input path="destination"/><br>
                  JourneyDate:<form:input path="journeyDate"/><br>
                  <button type="submit">Add</button>
            </form:form>
      </body>
</html>
```

Application Initializer

```
public class RedBusInitializer implements WebApplicationInitializer{
      @Override
      public void onStartup(ServletContext context) throws ServletException {
            ContextLoaderListener contextLoaderListner=null;
            DispatcherServlet dispatcher=null;
            contextLoaderListner=new ContextLoaderListener();
            context.addListener(contextLoaderListner);
            dispatcher=new DispatcherServlet();
            ServletRegistration.Dynamic dy=context.addServlet("dispatcher", dispatcher);
            dy.addMapping("*.htm");
      }
}
```

SimpleFormController

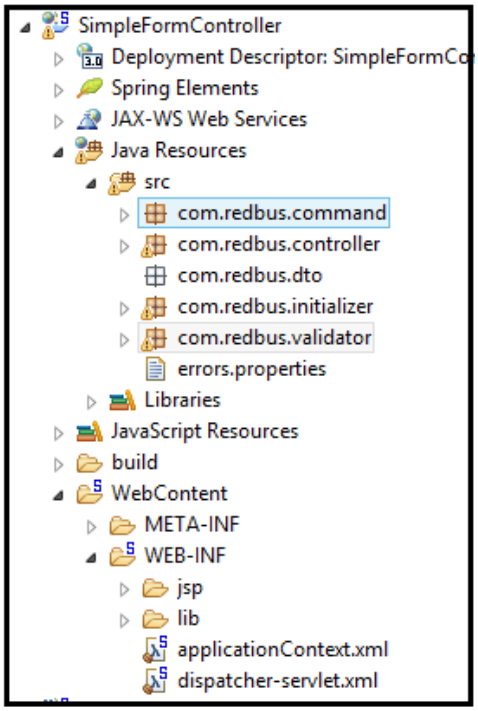
```
public class AddJourneySimpleFormController extends SimpleFormController{
	@Override
	protected ModelAndView onSubmit(HttpServletRequest request,
			HttpServletResponse response, Object command, BindException errors)
			throws Exception {
			ModelAndView mav=new ModelAndView();
			AddCriteria cri=(AddCriteria)command;
			mav.addObject("source", cri.getSource());
			mav.setViewName("success");
			return mav;
	}
}
```

dispatcher-servlet.xml

```
<?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 class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
		<property name="mappings">
			<props>
				<prop key="/add-journey.htm">qjcsfc</prop>
			</props>
		</property>
	</bean>
	<bean id="qjcsfc" class="com.redbus.controller.AddJourneySimpleFormController">
		<property name="formView" value="add-journey"/>
		<property name="commandClass" value="com.redbus.command.AddCriteria"/>
		<property name="commandName" value="journey"/>
		<property name="validator" ref="journeyValidator"/>
	</bean>
	<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="prefix" value="/WEB-INF/jsp/"/>
		<property name="suffix" value=".jsp"/>
	</bean>
	<bean id="journeyValidator" class="com.redbus.validator.JourneyValidator"/>

	<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
		<property name="basename" value="errors" />
	</bean>

</beans>
```

error.properties

```
source.blank=Source can't be blank
destination.blank=Destination can't be blank
journeyDate.invalid=Date should be future date
typeMismatch.journeyDate=Date can't be blank
```



Example#2(Show Dynamic page and edit option)

journeys.htm

ContextLoaderListener

DispatcherServlet
*.htm

errors.properties

```
source.blank=Source can't be blank
destination.blank=Destination can't be blank
journeyDate.invalid=Date can't be blank
amount.invalid=Amount can't be less than 1
typeMismatch.journeyDate=invalid date
typeMismatch.amount=invalid Amount
```

```
<beans>
    <bean id="messageSource" class="ResourceBundleMessageSource">
        <property name="basename" value="errors" />
    </bean>
    <bean id="journeyValidator" class="JourneyValidator"/>
    <bean id="ljc" class="ListJourneyController">
        ref="manageJourneyService"/>
    </bean>
    <bean id="editJourneyController" class="EditJourneyFormController">
        commandClass=JourneyCommand
        commandName=journey
        formView="edit-journey"
        manageJourneyService" ref="manageJourneyService"
        validator" ref="journeyValidator"
    </bean>
    <bean class="SimpleUrlHandlerMapping">
        "mappings">
            |-"/journeys.htm"=ljc
            |-"/edit-journey.htm"=editJourneyController
    </bean>
    <bean class="InternalResourceViewResolver">
        prefix"="/WEB-INF/jsp/"/>
        suffix"=".jsp"/>
    </bean>
</beans>
```

Journeys Details

http://localhost:8085/Redbus/journeys.htm

welcome to Redbus

Journey No	Source	Destination	Date	Amount
1001	BHUBANESWAR	ROURKELA	2017-04-21	460.0
1002	BHUBANESWAR	ROURKELA	2017-04-22	550.0
1003	BHUBANESWAR	ROURKELA	2017-04-23	350.0
1004	HYDERABAD	BANGALORE	2017-04-24	500.0

Edit Journey

Journey No :1004

Source: HYDERABAD

Destination: BANGALORE

Journey Date: 2017-04-24

Amount: 500.0

Submit

Initializer

```
public class RedBusInitializer implements WebApplicationInitializer{
    @Override
    public void onStartup(ServletContext context) throws ServletException {
        ContextLoaderListener contextLoaderListner=null;
        contextLoaderListner=new ContextLoaderListener();
        context.addListener(contextLoaderListner);

        DispatcherServlet dispatcher=null;
        dispatcher=new DispatcherServlet();
        ServletRegistration.Dynamic dy=context.addServlet("dispatcher", dispatcher);
        dy.addMapping("*.htm");
    }
}
```

edit-journey.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
    <body>
        <form:form commandName="journey" class="cl-form cl-card-4">
                Journey No :${journey.journeyNo}
                <form:hidden  path="journeyNo" />
                <label style="color: red;"><form:errors path="journeyNo"/></label><br>
                Source
                <form:input  path="source"/>
                <label style="color: red;"><form:errors path="source"/></label><br>

                Destination
                <form:input  path="destination"/>
                <label style="color: red;"><form:errors path="destination"/></label><br>
                Journey Date
                <form:input  path="journeyDate"/>
                <label style="color: red;"><form:errors path="journeyDate" /></label><br>
                Amount
                <form:input  path="amount"/>
                 <label style="color: red;"><form:errors path="amount"/></label><br>
                <input class="cl-btn cl-teal" type=submit Value=Submit >
        </form:form>
    </body>
</html>
```

journeys.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
  <%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c"%>
  <%@taglib uri="http://java.sun.com/jsp/jstl/fmt"  prefix="f"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Journeys Details</title>
    </head>
    <body>
        <marquee behavior="slide" direction="left" style="color: red; font-size: 30px "><h1>welcome to Redbus</h1></marquee>
        <table border="1">
            <tr>
                <th>Journey No</th>
                <th>Source</th>
                <th>Destination</th>
                <th>Date</th>
                <th>Amount</th>
            </tr>
            <c:forEach items="${journeys}" var="journey">
                <tr>
                    <td><a href="${pageContext.request.contextPath}/edit-journey.htm?jId=${journey.journeyNo}">${journey.journeyNo}</a></td>
                    <td>${journey.source}</td>
                    <td>${journey.destination}</td>
                    <td><f:formatDate pattern="dd/MM/YYYY" value="${journey.journeyDate}"></f:formatDate></td>
                    <td>${journey.amount}</td>
                </tr>
        </c:forEach>
        </table>
    </body>
</html>
```

```java
public class JourneyBo {
	protected int journeyNo;
	protected String source;
	protected String destination;
	protected Date journeyDate;
	protected int busNo;
	protected float amount;
	//Setters & Getters
}
```

```java
public class JourneyCommand {
	protected int journeyNo;
	protected String source;
	protected String destination;
	@DateTimeFormat(pattern="yyyy/mm/dd")
	protected  Date journeyDate;
	protected float amount ;
	//Setters & Getters
}
```

```java
public class JourneyDto {
	protected int journeyNo;
	protected String source;
	protected String destination;
	protected Date journeyDate;
	protected float amount;
	//Setters & Getters
}
```

```java
public class ListJourneyController extends AbstractController{
	private ManageJourneyService manageJourneyService;
	public void setManageJourneyService(ManageJourneyService manageJourneyService) {
		this.manageJourneyService = manageJourneyService;
	}
	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest request,
			HttpServletResponse response) throws Exception {
		List<JourneyDto> journeys=manageJourneyService.getJourneys();
		ModelAndView mav=new ModelAndView();
		mav.addObject("journeys", journeys);
		mav.setViewName("journeys");
		return  mav;
	}
}
```

```java
public class EditJourneyFormController extends ~~SimpleFormController~~ {
	private ManageJourneyService manageJourneyService;
	public ManageJourneyService getManageJourneyService() {
		return manageJourneyService;
	}

	public void setManageJourneyService(ManageJourneyService manageJourneyService) {
		this.manageJourneyService = manageJourneyService;
	}
	@Override
	protected Object formBackingObject(HttpServletRequest request)throws Exception {
		int journeyId=Integer.parseInt(request.getParameter("jId"));
		JourneyDto journeyDto=manageJourneyService.getJourney(journeyId);
		JourneyCommand command=new JourneyCommand();
		command.setJourneyNo(journeyDto.getJourneyNo());
		command.setSource(journeyDto.getSource());
		command.setDestination(journeyDto.getDestination());
		command.setJourneyDate(journeyDto.getJourneyDate());
		command.setAmount(journeyDto.getAmount());
		return command;
	}
	@Override
	protected ModelAndView onSubmit(HttpServletRequest request,
			HttpServletResponse response, Object obj, BindException errors)throws Exception {
		ModelAndView mav = new ModelAndView();
		JourneyCommand command=(JourneyCommand)obj;
		JourneyDto journeyDto=null;
		journeyDto=new JourneyDto();
		journeyDto.setJourneyNo(command.getJourneyNo());
		journeyDto.setSource(command.getSource());
		journeyDto.setDestination(command.getDestination());
		journeyDto.setJourneyDate(command.getJourneyDate());
		journeyDto.setAmount(command.getAmount());
		manageJourneyService.updateJourney(journeyDto);
		mav.setViewName("redirect:/journeys.htm");
		return mav;
	}
}
```

```
public class ManageJourneyService {
	private JourneyDao dao;

	public void setDao(JourneyDao dao) {
		this.dao = dao;
	}

	public List<JourneyDto> getJourneys(){
		//I take DTO and BO because all the attributes of Bo will not be there in DTO
		List<JourneyDto> journeyDtos=null;
		List<JourneyBo> journeyBos=null;

		journeyDtos=new ArrayList<JourneyDto>();
		journeyBos=dao.getJourneys();
		for (JourneyBo journeyBo : journeyBos) {
			JourneyDto jDto=new JourneyDto();
			jDto.setJourneyNo(journeyBo.getJourneyNo());
			jDto.setSource(journeyBo.getSource());
			jDto.setDestination(journeyBo.getDestination());
			jDto.setJourneyDate(journeyBo.getJourneyDate());
			jDto.setAmount(journeyBo.getAmount());
			journeyDtos.add(jDto);
		}
		return journeyDtos;
	}
	public JourneyDto getJourney(int journeyId){
		JourneyBo bo=null;
		JourneyDto dto=null;
		bo=dao.getJourney(journeyId);
		dto=new JourneyDto();
		dto.setJourneyNo(bo.getJourneyNo());
		dto.setSource(bo.getSource());
		dto.setDestination(bo.getDestination());
		dto.setJourneyDate(bo.getJourneyDate());
		dto.setAmount(bo.getAmount());
		return dto;
	}
	public void updateJourney(JourneyDto journeyDto){
		JourneyBo journeyBo=null;
		journeyBo=new JourneyBo();
		journeyBo.setJourneyNo(journeyDto.getJourneyNo());
		journeyBo.setSource(journeyDto.getSource());
		journeyBo.setDestination(journeyDto.getDestination());
		journeyBo.setJourneyDate(journeyDto.getJourneyDate());
		journeyBo.setAmount(journeyDto.getAmount());
		dao.updateJourney(journeyBo);
	}
}
```

```
public class JourneyDao {
	private JdbcTemplate jdbcTemplate;
	//private JourneyUpdate journeyUpdate=null;
	private final String SQL_GET_JOURNEYS="select * from JOURNEY_TBL";
	private final String SQL_GET_JOURNEY_BY_JOURNEY_ID="select * from JOURNEY_TBL where JOURNEY_NO=?";
	private final String SQL_UPDATE_JOURNEY="UPDATE JOURNEY_TBL SET SOURCE=?,
\			DESTINATION=?, JOURNEY_DT=?, AMOUNT=?  WHERE JOURNEY_NO=?";
	public JourneyDao(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
		//journeyUpdate=new JourneyUpdate(jdbcTemplate.getDataSource(), SQL_UPDATE_JOURNEY);
	}
	public List<JourneyBo> getJourneys(){
		return jdbcTemplate.query(SQL_GET_JOURNEYS,new JourneyRowMapper());
	}
	public JourneyBo getJourney(int journeyId){
		return jdbcTemplate.queryForObject(SQL_GET_JOURNEY_BY_JOURNEY_ID, new JourneyRowMapper(),new Object[]{journeyId});
	}
	public void updateJourney(JourneyBo journeyBo){
		//journeyUpdate.update(new Object[]{journeyBo.getSource(),journeyBo.getDestination(),
				journeyBo.getJourneyDate(),journeyBo.getAmount(),journeyBo.getJourneyNo()});
		jdbcTemplate.update(SQL_UPDATE_JOURNEY,journeyBo.getSource(),journeyBo.getDestination(),
					journeyBo.getJourneyDate(),journeyBo.getAmount(),journeyBo.getJourneyNo());
	}
	private final class JourneyRowMapper implements  RowMapper<JourneyBo>{
		@Override
		public JourneyBo mapRow(ResultSet rs, int row) throws SQLException {
			JourneyBo journeyBo=new JourneyBo();
			journeyBo.setJourneyNo(rs.getInt("JOURNEY_NO"));
			journeyBo.setSource(rs.getString("SOURCE"));
			journeyBo.setDestination(rs.getString("DESTINATION"));
			journeyBo.setJourneyDate(rs.getDate("JOURNEY_DT"));
			journeyBo.setAmount(rs.getFloat("AMOUNT"));
			journeyBo.setBusNo(rs.getInt("BUS_NO"));
			return journeyBo;
		}
	}
	/*public final class JourneyUpdate extends SqlUpdate{
		public JourneyUpdate(DataSource dataSource,String sqlQuery) {
			super(dataSource, sqlQuery);
			declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
			declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
			declareParameter(new SqlParameter(java.sql.Types.DATE));
			declareParameter(new SqlParameter(java.sql.Types.INTEGER));
			declareParameter(new SqlParameter(java.sql.Types.INTEGER));
			compile();
		}
	}	*/
}
```

applicationContext.cml

```
<?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">
	<import resource="config/persistancy-bean.xml"/>
	<import resource="config/service-bean.xml"/>
	<import resource="config/aop-bean.xml"/>
</beans>
```

persistency-beans.xml

```
<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="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
              <property name="driverClassName"  value="oracle.jdbc.driver.OracleDriver"/>
              <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
              <property name="username" value="spr_usr"/>
              <property name="password" value="welcome1"/>
       </bean>
       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
              <constructor-arg ref="dataSource"/>
       </bean>
       <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
              <property name="dataSource" ref="dataSource"/>
       </bean>
       <bean id="jDao" class="com.redbus.dao.JourneyDao">
              <constructor-arg ref="jdbcTemplate"/>
       </bean>
       </beans>
```

service-beans.xml

```
<?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="manageJourneyService" class="com.redbus.service.ManageJourneyService">
              <property name="dao" ref="jDao"/>
       </bean>
</beans>
```

aop-beans.xml

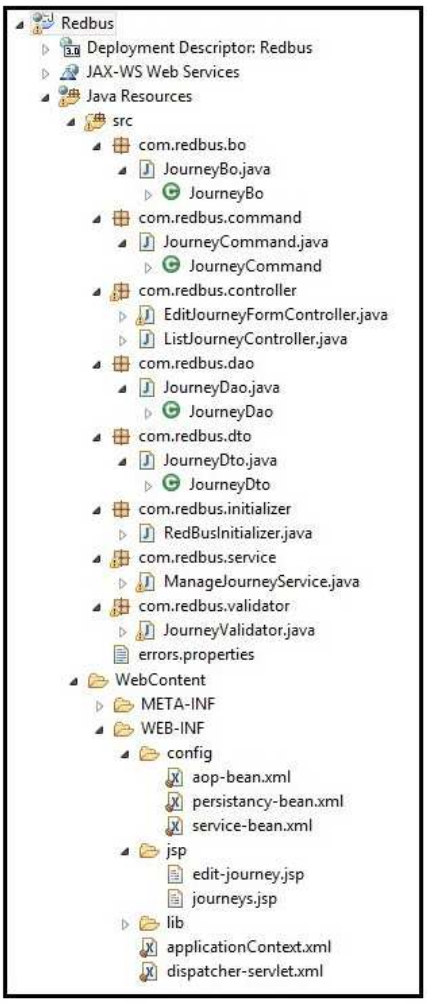
```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
              http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
              http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

              <tx:advice id="txAdvice" transaction-manager="transactionManager">
                     <tx:attributes>
                            <tx:method name="get*" read-only="true"/>
                     </tx:attributes>
              </tx:advice>
              <aop:config>
                     <aop:pointcut expression="within(com.redbus.service.*)" id="pc1"/>
                     <aop:advisor advice-ref="txAdvice" pointcut-ref="pc1"/>
              </aop:config>
</beans>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
              http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
              http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

              <tx:advice id="txAdvice" transaction-manager="transactionManager">
                     <tx:attributes>
                            <tx:method name="get*" read-only="true"/>
                     </tx:attributes>
              </tx:advice>
              <aop:config>
                     <aop:pointcut expression="within(com.redbus.service.*)" id="pc1"/>
                     <aop:advisor advice-ref="txAdvice" pointcut-ref="pc1"/>
              </aop:config>
</beans>
```

realspeed

Validator

```
public class JourneyValidator implements Validator {
	@Override
	public boolean supports(Class<?> classType) {
		if (classType.isAssignableFrom(JourneyCommand.class)) {
			return true;
		}
		return false;
	}
	@Override
	public void validate(Object command, Errors errors) {
		JourneyCommand journey=(JourneyCommand)command;
		if (journey.getSource()==null || "".equals(journey.getSource().trim())) {
			errors.rejectValue("source","source.blank");
		}
		if (journey.getDestination()==null || "".equals(journey.getDestination().trim())) {
			errors.rejectValue("destination", "destination.blank");
		}
		if (journey.getJourneyDate()==null) {
			errors.rejectValue("journeyDate", "journeyDate.invalid");
		}
		if (journey.getAmount()<=0) {
			errors.rejectValue("amount", "amount.invalid");
		}
	}
}
```

Structure



Importance of formBackingObject()

➔Generally when handleRequest() method is processing initial phase request it internally calls formBackingObjec() to create command class Object .
➔The Pre defined method always creates the empty command object .
➔If you want to create command object with dynamic data according to application requirement then we need to override this method in our controller class that extends form SimpleFormContorller.

Form Validation in SimpleFormController

➔Verifing the format and pattern of form data is called form validation.
➔Spring MVC allows to write form validation logic in separate class so that it can be linked to multiple controller classes having reusability.
➔To write a validator class we have to implementing our class from validator interface and override support(--) and validate(--) methods.
➔Write properties file having validation error messages.
➔Write ResourceBundleMessageSource and give the base name so that it can read that file from class path.
➔Configure validator class in dispatcher-servlet.xml and link that class with controller class using the property 'validator'.
➔Place the <form:errors path="*"> tag in jsp form submission page to display the form validation error messages.
➔The Simple controller can handle Three types of validations

1. **Form Validation**(like number/name can't be blank)
2. **typeMismatch** error (like date dd/mm/yyyy)
3. Application Logic errors like (not allowed so and so value to particular type)

➔Instead of hard coding error message directly in validator class and controller class it is recommended to place in properties file having our choice key and values.
➔But typeMismatch errors the key should be typeMismatch(properties) you have to take.

What is the Use of support(Class class) method and validator class of Validator class?

➔The support method holds the configured command class parameters and we can check whether the correct command class is configured or not using the Class class type calling AssignableForm(class) method.
➔If configure correctly it returns true otherwise it returns false ,so validate method will not execute.
➔Due to this we can avoid mis configuration in spring bean configuration file.

BindingResult(I)

➔When dispatcherServlet pass request as input to the simpleFormcontroller extract the data and perform the request wrapping and internal process, while conversation if there are some type conversion errors then it will not throw an ugly exception in the user.
➔Controller internally Bind that Exception Result to the Binding Result object like that if any internal Exception are there it will preserves .
➔After that it will call validator class to validate other attributes of command class by creating errors object.
➔If at all any errors are there we will bind them to errors object.
➔Once validation finished, controller checks errors.hasErrors() method if errors are there then it will bind that errors to commandName.errors and command object and return to DispatcherServlet.
➔DispatcherServlet call viewResolver to display the view form tag will conversation it will check first errors are there or not if at all there it will show those errors by presenting old values into test boxes.
➔The old values preserved by command object if at all any type conversion errors are there then such kind of values preserves by BindingResult Object.

BindException(C) implementation of (Binding Result and Errors)

➔While dealing with Simpleform controller we come across the class called BindException.
➔Actually BindException is class used for Handling the Exception across the application.

➔BindException class implements two interfaces i.e Errors and BindingResult.
➔Both interface plays different role while dealing with Bind Exception .Actually BindingResult used by Abstract class internally to Bind internal Exception .

ViewResolver

Spring has supplied few predefined view Resolver classes. They are :
➔InternalResourceViewResolver(sub class of UrlBasedViewReslover)
➔ResourceBundleViewResolver(configure view in property file)
➔XmlViewResolver(configure view in xml file)
➔BeanNameViewResolver(resolver spring beans as views)
➔UrlBasedViewReslover

InternalResourceViewResolver:
Useful to resolve jsp, of WEB-INF folder as views by default takes "JstlView" as view class.

views_en_US.properties
```
aboutus.(class)=org.springframework.web.servlet.view.JstlView
aboutus.url=/WEB-INF/jsp/about-us.jsp
```

views_hi_IN.properties
```
aboutus.(class)=org.springframework.web.servlet.view.JstlView
aboutus.url=/WEB-INF/jsp/about-in.jsp
```

about-in.jsp
```
<body style="font-family: consolas;font-size: 18px">
        I know you are in india.
</body>
```

about-us.jsp
```
<body style="font-family: consolas;font-size: 18px">
        I know you are in US.
</body>
```

views.xml
```
<?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="aboutus"
class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-INF/jsp/about-in.jsp" />
    </bean>

</beans>
```

```xml
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://java.sun.com/xml/ns/javaee"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
               http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
     id="WebApp_ID" version="3.0">
     <display-name>ViewResolverWeb</display-name>
     <servlet>
          <servlet-name>dispatcher</servlet-name>
          <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>
</web-app>
```

```xml
<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-3.1.xsd">

     <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
          <property name="mappings">
               <props>
                    <prop key="/about-us.htm">viewAboutUsController</prop>
               </props>
          </property>
     </bean>

     <bean id="viewAboutUsController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController">
          <property name="viewName" value="about-us" />
     </bean>

     <!-- <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
          <property name="basename" value="views" />
     </bean> -->

     <!-- <bean class="org.springframework.web.servlet.view.XmlViewResolver">
          <property name="location" value="/WEB-INF/views.xml"/>
     </bean> -->

     <!-- <bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

     <bean id="aboutus" class="org.springframework.web.servlet.view.JstlView">
          <property name="url" value="/WEB-INF/jsp/about-us.jsp" />
     </bean> -->

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

Reporting in PDF (iText Lib) and EXCEL (Poi Lib)

```
<body style="font-family: consolas;font-size: 18px;">
    <a href="${pageContext.request.contextPath}/day-journeys.htm">Today's Journeys</a>
</body>
```

```
<body style="font-family: consolas;font-size: 18px;">
    <table>
        <tr>
            <td><a href="${pageContext.request.contextPath}/day-journeys.htm?view=pdf">Pdf</a></td>
            <td> </td>
            <td><a href="${pageContext.request.contextPath}/day-journeys.htm?view=xls">Excel</a></td>
        </tr>
        <tr>
            <th>Source</th>
            <th>Destination</th>
            <th>Bus No</th>
        </tr>
        <c:forEach items="${journeys}" var="journey">
            <tr>
                <td>${journey.source}</td>
                <td>${journey.destination}</td>
                <td>${journey.busNo}</td>
            </tr>
        </c:forEach>
    </table>
</body>
```

```
xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.1.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">


    <mvc:view-controller path="/home.htm" view-name="home" />
    <bean name="/day-journeys.htm" class="com.rw.controller.ListJourneysController" />

    <bean id="journeys-xls" class="com.rw.view.JourneyExcelView" />
    <bean id="journeys-pdf" class="com.rw.view.JourneyPdfView" />

    <mvc:view-resolvers>
        <mvc:bean-name/>
        <mvc:jsp prefix="/WEB-INF/jsp/" suffix=".jsp" />
    </mvc:view-resolvers>
</beans>
```

```
public class JourneyDto {
    protected String source;
    protected String destination;
    protected int busNo;
    //Setters and Getters
}
```

```java
public class ReportingInitializer extends AbstractDispatcherServletInitializer {
	@Override
	protected WebApplicationContext createServletApplicationContext() {
		XmlWebApplicationContext servletApplicationContext = null;

		servletApplicationContext = new XmlWebApplicationContext();
		servletApplicationContext
				.setConfigLocation("/WEB-INF/dispatcher-servlet.xml");

		return servletApplicationContext;
	}
	@Override
	protected String[] getServletMappings() {
		return new String[] { "*.htm" };
	}
	@Override
	protected WebApplicationContext createRootApplicationContext() {
		return null;
	}
}
```

```java
public class ListJourneysController extends AbstractController {

	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest request,
			HttpServletResponse response) throws Exception {
		List<JourneyDto> journeys = null;
		ModelAndView mav = null;
		String page = "journeys";
		String view = null;

		journeys = new ArrayList<JourneyDto>();
		journeys.add(new JourneyDto("hyderabad", "banglore", 3335));
		journeys.add(new JourneyDto("banglore", "chennai", 3535));
		journeys.add(new JourneyDto("hyderabad", "pune", 6464));
		journeys.add(new JourneyDto("hyderabad", "goa", 9384));

		mav = new ModelAndView();
		mav.addObject("journeys", journeys);

		view = request.getParameter("view");
		if (view != null) {
			if (view.equals("pdf")) {
				page = "journeys-pdf";
			} else if (view.equals("xls")) {
				page = "journeys-xls";
			}
		}
		mav.setViewName(page);
		return mav;
	}
}
```

```java
public class JourneyExcelView extends AbstractExcelView {
	@Override
	protected void buildExcelDocument(Map<String, Object> model,
			HSSFWorkbook workBook, HttpServletRequest req,
			HttpServletResponse resp) throws Exception {
		List<JourneyDto> journeys = null;
		HSSFSheet sheet = null;

		journeys = (List<JourneyDto>) model.get("journeys");
		sheet = workBook.createSheet("journeys");

		for (int i = 0; i < journeys.size(); i++) {
			JourneyDto journey = journeys.get(i);
			HSSFRow row = sheet.createRow(i);

			HSSFCell sourceCell = row.createCell(1);
			sourceCell.setCellValue(journey.getSource());

			HSSFCell destCell = row.createCell(2);
			destCell.setCellValue(journey.getDestination());

			HSSFCell busNoCell = row.createCell(3);
			busNoCell.setCellValue(journey.getBusNo());
		}
	}
}
```

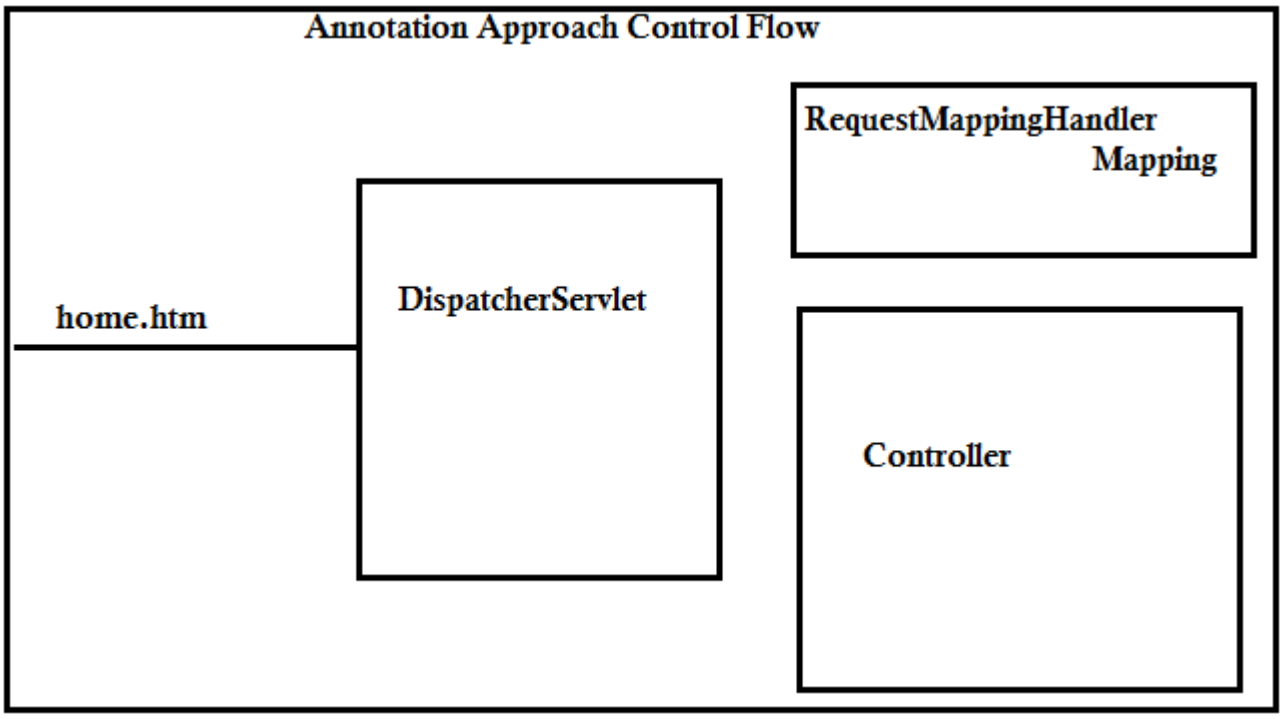
```java
public class JourneyPdfView extends AbstractPdfView {
	@Override
	protected void buildPdfDocument(Map<String, Object> model,
			Document pdfDocument, PdfWriter pdfWriter, HttpServletRequest req,
			HttpServletResponse resp) throws Exception {
		List<JourneyDto> journeys = null;

		journeys = (List<JourneyDto>) model.get("journeys");
		Table table = new Table(3);
		for (int i = 0; i < journeys.size(); i++) {
			JourneyDto dto = journeys.get(i);

			table.addCell(dto.getSource());
			table.addCell(dto.getDestination());
			table.addCell(String.valueOf(dto.getBusNo()));
		}
		pdfDocument.add(table);
	}
}
```

What Handler Interceptor why not Filter?

➔I have some common pre-processing and common post processing that i want to apply for each and every request that is coming to my application .
➔In case of servlet and jsp we need to write that in **Filter** .
➔In servlet and JSP filter is always applicable but in spring mvc always the request is received by DispatcherServlet .
➔Client will send the request to the DispatcherServlet , DispatcherServlet will forward the request to HandlerMapping , HandlerMapping will identify the controller and return the id of the controller to DispatcherServlet.
➔Then DispatcherServlet will forward the request to the controller .
➔Whenever the request is coming to DispatcherServlet ,the servlet container will indentifies for this target which is a Filter that has to get executed then the Filter doFilter() will be called.Then the Filter will complete pre-processing then forward the request to Dispatcher Servlet.
➔Dispatcher Servlet will handle the request. then forword the response to Filter then Filter will perform post processing the send the response to the client .
➔Whenever i send the request DipatcherServlet will received the request
Whenever DispatcherServlet received the request Then Dispatcherservlet will not check whether the request is valid or not It directly send the URL to Handler Mapping.
➔So that un-necessarily for a valid request or invalid request always the filter is executed.
➔So Filter is not a relevant solution when it comes to spring MVC .
➔So Handler Interceptor Came into picture.

Handler interceptor

➔We have one more additional component to HandlerMapping is a Interceptor .
➔So that whenever the DispatcherServlet received the request it forward the request to Handler Mapping , send the URl , Handler Mapping is trying to find is the URL is a relevant controller.
➔If the controller is identified, the name of the controller along with that the Interceptor that should execute for this controller both should be pass as the input for the DispatcherServlet.
➔Now Dispatcher Servlet create Handler Execution chain because we can add any number of Interceptor like security interceptor, Http Monitoring interceptor and logging interceptor .So multiple interceptor can apply for a request. So all the interceptor add to handler mapping .
➔Handler Mapping will identify the controller and the list of Interceptor that has to be executed in the order before the request will goes to the controller it create a handlerExicutionChain .



Example

booking-history.jsp

```
<body style="font-family: consolas;font-size: 18px">
     You are on Booking History
</body>
```

book-ticket.jsp

```
<body style="font-family: consolas;font-size: 18px">
          You are on Book Ticket
     </body>
```

dispatcher-servlet.xml

```
<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-4.1.xsd">

     <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
          <property name="mappings">
               <props>
                    <prop key="/book-ticket.htm">viewBookTicketController</prop>
               </props>
          </property>
          <property name="interceptors">
               <list>
                    <ref bean="timeIntervalHandlerInterceptor" />
               </list>
          </property>
     </bean>

     <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />

      <bean id="timeIntervalHandlerInterceptor"
     class="com.hi.interceptor.TimeIntervalHandlerInterceptor" />

     <bean id="viewBookTicketController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController">
          <property name="viewName" value="book-ticket" />
     </bean>

     <bean name="/booking-history.htm"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController">
          <property name="viewName" value="booking-history" />
     </bean>
     <bean
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
          <property name="prefix" value="/WEB-INF/jsp/" />
          <property name="suffix" value=".jsp" />
     </bean>
 </beans>
```

maintainance.jsp

```
<body style="font-family: consolas;font-size: 18px; color: red">
     Site under maintainance.
</body>
```

```
public class HIInitializer extends AbstractDispatcherServletInitializer {
     @Override
     protected WebApplicationContext createServletApplicationContext() {
          XmlWebApplicationContext servletApplicationContext = null;

          servletApplicationContext = new XmlWebApplicationContext();
          servletApplicationContext
                    .setConfigLocation("/WEB-INF/dispatcher-servlet.xml");

          return servletApplicationContext;
     }
     @Override
     protected String[] getServletMappings() {
          return new String[] { "*.htm" };
     }
     @Override
     protected WebApplicationContext createRootApplicationContext() {
          XmlWebApplicationContext rootApplicationContext = null;
          rootApplicationContext = new XmlWebApplicationContext();
          rootApplicationContext
                    .setConfigLocation("/WEB-INF/application-context.xml");

          return rootApplicationContext;
     }
}
```

```
public class TimeIntervalHandlerInterceptor extends HandlerInterceptorAdapter {
     @Override
     public boolean preHandle(HttpServletRequest request,
               HttpServletResponse response, Object handler) throws Exception {
          int hour = 0;
          Calendar calendar = null;
          calendar = Calendar.getInstance();
          hour = calendar.get(Calendar.HOUR_OF_DAY);
          if (hour >= 19 && hour <= 22) {
               response.sendRedirect("maintainance.jsp");
               return false;
          }
          return true;
     }
}
```

Annotation

➔Important Possible input parameter to a controller

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpWebRequest
5. HttpWebResponse
6. Command class
7. Model
8. ModelMap
9. Form Class @ModelAttribute
10. BindingResult
11. BindingException etc…

➔Important Possible return types to a controller

1. Void
2. String
3. ModelAndView
4. Model
5. ModelMap
6. Map
7. View etc…

@ModelAttribute:

➔Use @ModelAttribute base on parameter in handler method of controller class to specify the command class and its object to perform request wrapping.

➔@ModelAttribute is multipurpose annotation expose any data to to web view. Suppose I want to render a view page with master data that is coming from data base in this case I have to write a method which returns master data to render in view page and I have to annotate that method with @ModelAttribute.

Example#1

```
@Configuration
public class RootConfig {
}
```

```
@Configuration
public class WebConfig {
    @Bean(name="/home.htm")
    public Controller newViewHomeController(){
        ParameterizableViewController pvc=
            new ParameterizableViewController();
        pvc.setViewName("home");
        return pvc;
    }
    @Bean
    public ViewResolver newJspViewResolver(){
        InternalResourceViewResolver jspView=
            new InternalResourceViewResolver();
        jspView.setPrefix("/WEB-INF/jsp/");
        jspView.setSuffix(".jsp");
        return jspView;
    }
}
```

```
public class DefaultAnnotationInitializer implements WebApplicationInitializer{
    @Override
    public void onStartup(ServletContext context) throws ServletException {

        AnnotationConfigWebApplicationContext rootContext = null;
        AnnotationConfigWebApplicationContext webContext = null;

        rootContext = new AnnotationConfigWebApplicationContext();
        rootContext.register(RootConfig.class);
        context.addListener(new ContextLoaderListener(rootContext));

        webContext = new AnnotationConfigWebApplicationContext();
        webContext.register(WebConfig.class);

        DispatcherServlet dispatcher=new DispatcherServlet(webContext);

        ServletRegistration.Dynamic
            sconfig = context.addServlet("dispatcher",dispatcher);
        sconfig.setLoadOnStartup(2);
        sconfig.addMapping("*.htm");
    }
}
```

Example#2

```
@Configuration
public class RootConfig {
}
```

```
@Configuration
public class WebConfig {
	@Bean(name="/home.htm")
	public Controller newViewHomeController(){
		ParameterizableViewController pvc=
				new ParameterizableViewController();
		pvc.setViewName("home");
		return pvc;
	}
	@Bean
	public ViewResolver newJspViewResolver(){
		InternalResourceViewResolver jspView=
				new InternalResourceViewResolver();
		jspView.setPrefix("/WEB-INF/jsp/");
		jspView.setSuffix(".jsp");
		return jspView;
	}
}
```

```
public class DefaultAnnotationInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer{
	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootConfig.class};
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{WebConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[]{"*.htm"};
	}
}
```

```
<body>
	<h1 style="color: red">You are in Home Page</h1>
</body>
```

Example#3

```
@Configuration
public class RootConfig {
}
```

```
@Configuration
@EnableWebMvc
public class ServletConfig extends WebMvcConfigurerAdapter {
	@Override
	public void addViewControllers(ViewControllerRegistry registry) {
		registry.addViewController("/home.htm").setViewName("home");
	}

	@Override
	public void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp().prefix("/WEB-INF/jsp/").suffix(".jsp");
	}
}
```

```
public class AnnotationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{
	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootConfig.class};
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{ServletConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[]{"*.htm"};
	}
}
```

Example#4

```
@Configuration
public class RootConfig {

}
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.faw.controller")
public class WebConfig extends WebMvcConfigurerAdapter{

	@Override
	public void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp("/WEB-INF/jsp/", ".jsp");
	}
}
```

```
@Controller
public class HomeViewController {
	@RequestMapping("/home.htm")
	public String getViewName(){
		return "home";
	}
}
```

```
public class FAnnotationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootConfig.class};
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{WebConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[]{"*.htm"};
	}
}
```

Example#5

```
public class AnnotationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootConfig.class};
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{WebConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[]{"*.htm"};
	}
}
```

```
public class JourneyDto {
	protected int journeyId;
	protected String source;
	protected String destination;
	protected Date journeyDate;
	protected double amount;
	//parameterizable Constructor
	//Setters & Getters
}
```

```
public class TicketDto {
	protected int ticketNo;
	protected int journeyId;
	protected String passengerName;
	//parameterizable Constructor
	//Setters & Getters
}
```

```
@Configuration
public class RootConfig {
}
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.faw.controller")
public class WebConfig extends WebMvcConfigurerAdapter {
	@Override
	public void addViewControllers(ViewControllerRegistry registry) {
		registry.addViewController("/journey-dashboard.htm").setViewName("journey-dashboard");
	}

	@Override
	public void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp("/WEB-INF/jsp/", ".jsp");
	}
}
```

```
@Controller
public class ViewJourneyController {
	@RequestMapping("/journeys.htm")
	public String getJourneys(Model model){
		List<JourneyDto> journeys=new ArrayList<JourneyDto>();
		journeys.add(new JourneyDto(1,"Hyderabad", "bangalore", new Date(), 750));
		journeys.add(new JourneyDto(2,"Hyderabad", "Chennai", new Date(), 850));
		journeys.add(new JourneyDto(3,"Hyderabad", "Pune", new Date(), 800));
		journeys.add(new JourneyDto(4,"Hyderabad", "Mumbai", new Date(), 950));

		model.addAttribute("journeys", journeys);
		return "journeys";
	}
}
```

```
@Controller
public class ViewTicketController {
	@RequestMapping("/ticket-info.htm")
	public String getTicketInfo(@RequestParam("jId")int journeyId,ModelMap modelMap){
		List<TicketDto> tickets=new ArrayList<TicketDto>();
		tickets.add(new TicketDto(101,1,"Dhananjaya"));
		tickets.add(new TicketDto(102,2,"Lizarani"));
		tickets.add(new TicketDto(103,3,"Madhusmita"));
		tickets.add(new TicketDto(104,4,"Sasmita"));

		for (TicketDto ticketDto : tickets) {
			if (journeyId==ticketDto.getJourneyId()) {
				modelMap.addAttribute("ticket", ticketDto);
			}
		}
		return "ticket-info";
	}
}
```

journey-dashboard.htm

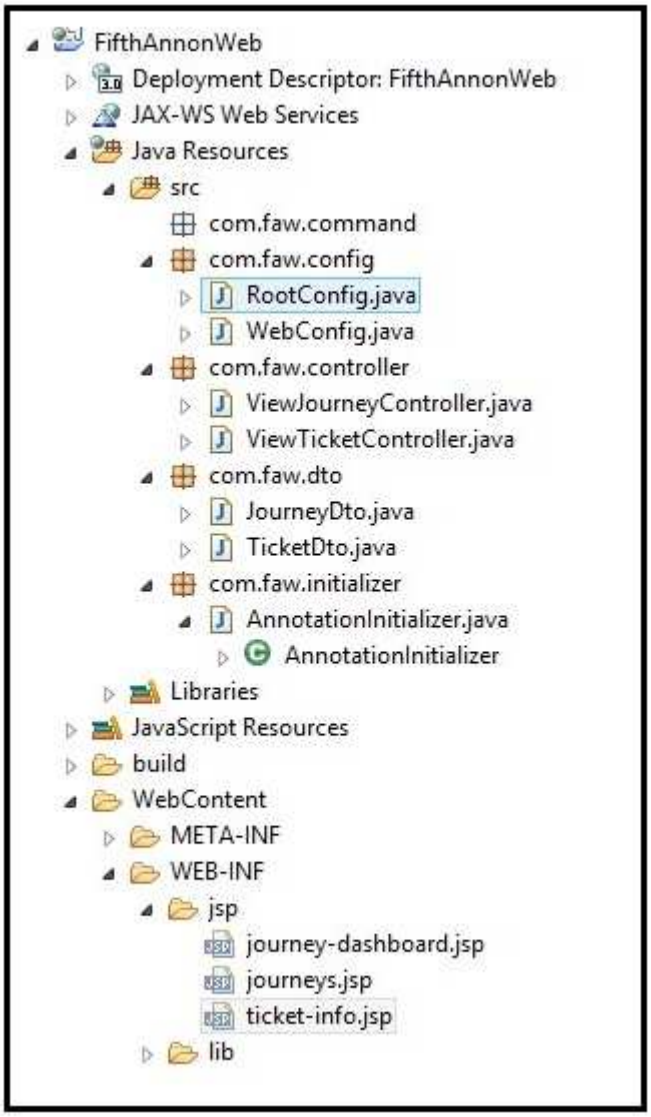
```
<body>
		<a href="${pageContext.request.contextPath}/journeys.htm">
		Journey Information
	</a>
</body>
```

journeys.htm

```
<body>
	<h1>Journey Info</h1>
	<table border="1">
	  <tr>
	    <td>Source------Destination</td>
	  </tr>
	  <c:forEach items="${journeys}" var="journey" >
	    <tr>
	      <td> <a href="${pageContext.request.contextPath}/ticket-info.htm?jId=${journey.journeyId}">
	                ${journey.source}------ ${journey.destination}</a></td>
	    </tr>
	    </c:forEach>
	</table>
</body>
```

ticket-info.htm

```
<body>
	<h1>Ticket Information</h1>
	<table border="1">
		<tr>
			<td>TicketNo</td>
			<td>Journey Id</td>
			<td>PassengerName</td>
		</tr>
			<tr>
				<td>${ticket.ticketNo}</td>
				<td>${ticket.journeyId}</td>
				<td>${ticket.passengerName}</td>
			</tr>
	</table>
</body>
```



Example#6

```
public class AnnotationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootConfig.class};
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{WebConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[]{"*.htm"};
	}
}
```

```
public class Criteria {
	protected String medicineName;
	protected String composition;
	protected int unit;
	// Default Constructor (mandatory)
	// Parameterizable Constructor
	//Setters and Getters
}
```

```
@Configuration
public class RootConfig {
}
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.saw.controller")
public class WebConfig extends WebMvcConfigurerAdapter {

	@Override
	public void addViewControllers(ViewControllerRegistry registry) {
		registry.addViewController("/search-medicine.htm")
				.setViewName("search-medicine");
	}

	@Override
	public void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp("/WEB-INF/jsp/", ".jsp");
	}
}
```

```java
@Controller
public class ViewMedicineController{
	@RequestMapping("/medicine-info.htm")
	public String search(@ModelAttribute Criteria medicineCriteria,ModelMap modelMap){
		MedicineDto dto=new MedicineDto(
					medicineCriteria.getMedicineName(),
					medicineCriteria.getComposition(),
					new Random().nextInt(10));
		modelMap.addAttribute("medicineInfo", dto);
		return "medicine-info";
	}
}
```

search-medicine.jsp

```jsp
<form action="${pageContext.request.contextPath}/medicine-info.htm" method="post">
	<table>
	<tr>
		<td>Medicine Name</td>
		<td>
			<input type="text" name="medicineName">
		</td>
	</tr>
	<tr>
		<td>Composition</td>
		<td>
			<input type="text" name="composition">
		</td>
	</tr>
		<tr>
			<td colspan="3">
				<input type="submit" value="Search">
			</td>
		</tr>
	</table>
</form>
```

medicine-info.jsp

```jsp
<body>
	<table>
		<tr>
			<td>Medicine Name</td>
			<td>${medicineInfo.medicineName }</td>
		</tr>
		<tr>
			<td>Composition</td>
			<td>${medicineInfo.composition }</td>
		</tr>
		<tr>
			<td>Unit</td>
			<td>${medicineInfo.unit}</td>
		</tr>
	</table>
</body>
```

End to End Application including Log4J (Configuration and Annotation Mixed)

add-product.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
		<title>Add Product</title>
	</head>
	<body style="font-family: consolas;font-size: 20px;">
		<form:form modelAttribute="productForm">
			<span style="color: red;">
				<form:errors path="*"/>
			</span>
			<br/>
			<table>
				<tr>
					<td>Name</td>
					<td><form:input path="name"/></td>
				</tr>
				<tr>
					<td>Type :</td>
					<td><form:input path="type"/></td>
				</tr>
				<tr>
					<td>Manufacturer:</td>
					<td>
						<form:select path="manufacturer">
							<form:option value="0"> </form:option>
							<c:forEach  items="${manufacturers}" var="manuf">
								<form:option value="${manuf.manufacturerId}">${manuf.name}</form:option>
							</c:forEach>
						</form:select>
					</td>
				</tr>
				<tr>
					<td>Price:</td>
					<td><form:input path="price"/></td>
				</tr>
				<tr>
					<td colspan="2">
						<input type="submit" value="add"/>
					</td>
				</tr>
			</table>
		</form:form>
	</body>
</html>
```

add-product-success.jsp

```
<body>
	<p style="font-family: consolas;font-size: 20px">
		Your product with id ${id} has been added
	</p>
</body>
```

errors.properties

```
id.invalid=Id should be non-zero positive integer
name.blank=Name cannot be blank
manufacturer.blank=Manufacturer cannot be blank
price.invalid=Price should be greater than zero
typeMismatch.id=Id should be integer
typeMismatch.price=Price should be valid Value
```

log4j.properties

```
log4j.rootLogger=TRACE,Console,FileOut
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.conversionPattern=%20c %d - %20c%c - %m - %M%n
log4j.appender.FileOut=org.apache.log4j.FileAppender
log4j.appender.FileOut.File=d:/pm.out
log4j.appender.FileOut.layout=org.apache.log4j.PatternLayout
log4j.appender.FileOut.layout.conversionPattern=%20c %d - %20c%c - %m - %M%n
```

ServletConfig

```
@Configuration
@ComponentScan({ "com.pm.controller", "com.pm.validator" })
public class ServletConfig extends WebMvcConfigurationSupport {

	@Bean(name = "messageSource")
	public MessageSource createMessageSource() {
		ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
		messageSource.setBasename("errors");
		return messageSource;
	}

	@Override
	protected Validator getValidator() {
		return new LocalValidatorFactoryBean();
	}

	@Override
	protected void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp("/WEB-INF/jsp/", ".jsp");
	}
}
```

```
@Configuration
@ImportResource("classpath:com/pm/common/application-context.xml")
@ComponentScan({"com.pm.dao", "com.pm.service"})
public class RootConfig {

}
```

aop-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tx="http://www.springframework.org/schema/tx"
	xmlns:aop="http://www.springframework.org/schema/aop"
	xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

	<tx:advice id="txAdvice" transaction-manager="transactionManager">
		<tx:attributes>
			<tx:method name="save*" read-only="false" />
			<tx:method name="*" read-only="true" />
		</tx:attributes>
	</tx:advice>

	<aop:config>
		<aop:pointcut expression="within(com.pm.service.*)" id="txPointcut" />
		<aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />
	</aop:config>
</beans>
```

application-context.xml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<?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">

	<import resource="classpath:com/pm/common/persistence-beans.xml" />
	<import resource="classpath:com/pm/common/aop-beans.xml" />

</beans>
```

persistence-beans.xml

```
<?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="dataSource"
		class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
		<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
		<property name="username" value="pms_user" />
		<property name="password" value="welcome1" />
	</bean>

	<bean id="sessionFactory"
		class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
		<property name="dataSource" ref="dataSource" />
		<property name="hibernateProperties">
			<props>
				<prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
				<prop key="hibernate.show_sql">true</prop>
			</props>
		</property>
		<property name="mappingResources">
			<list>
				<value>com/pm/entities/Product.hbm.xml</value>
				<value>com/pm/entities/Manufacturer.hbm.xml</value>
				<value>com/pm/entities/Specification.hbm.xml</value>
			</list>
		</property>
	</bean>

	<bean id="hibernateTemplate" class="org.springframework.orm.hibernate4.HibernateTemplate">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>

	<bean id="transactionManager"
		class="org.springframework.orm.hibernate4.HibernateTransactionManager">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>
</beans>
```

realspeed

```java
public class AddProductForm {
	private String name;
	private String type;
	private int manufacturer;
	private float price;
	//Setters and Getters
}
```

```java
public class ManufacturerDto {
	private int manufacturerId;
	private String name;
	//Setters and Getters
}
```

```xml
<hibernate-mapping package="com.pm.entities">
	<class name="Manufacturer" table="MANUFACTURER">
		<id name="manufacturerId" column="MANUFACTURER_ID">
			<generator class="increment" />
		</id>
		<property name="name" column="NAME" />
		<property name="brand" column="BRAND_NM" />
		<property name="estDate" column="EST_DT" />
		<set name="products">
			<key column="MANUFACTURER_ID" />
			<one-to-many class="Product" />
		</set>
	</class>
</hibernate-mapping>
```

```java
public class Manufacturer {
	private int manufacturerId;
	private String name;
	private String brand;
	private Date estDate;
	private Set<Product> products;
	//Setters and getters
}
```

```xml
<hibernate-mapping package="com.pm.entities">
	<class name="Product" table="PRODUCTS">
		<id name="productId" column="PRODUCT_ID">
			<generator class="increment" />
		</id>
		<property name="name" column="PRODUCT_NM" />
		<property name="type" column="TYPE" />
		<property name="price" column="PRICE" />
		<property name="offerType" column="OFFER_TYPE" />
		<many-to-one name="manufacturer" column="MANUFACTURER_ID" />
		<set name="specifications">
			<key column="PRODUCT_ID"/>
			<one-to-many class="Specification"/>
		</set>
	</class>
</hibernate-mapping>
```

```java
public class Product {
	private int productId;
	private String name;
	private String type;
	private float price;
	private String offerType;
	private Manufacturer manufacturer;
	private Set<Specification> specifications;
	//Setters and Getters
}
```

```xml
<hibernate-mapping package="com.pm.entities">
	<class name="Specification" table="SPECIFICATIONS">
		<id name="specificationId" column="SPECIFICATION_ID">
			<generator class="increment" />
		</id>
		<property name="specificationName" column="SPECIFICATION_NM" />
		<property name="description" column="DESCR" />
		<many-to-one name="product" column="PRODUCT_ID" />
	</class>
</hibernate-mapping>
```

```java
public class Specification {
	private int specificationId;
	private String specificationName;
	private String description;
	private Product product;
	//Setters and Getters
}
```

```java
public class ProductManagementInitializer extends
		AbstractAnnotationConfigDispatcherServletInitializer {

	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class<?>[] { RootConfig.class };
	}

	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class<?>[] { ServletConfig.class };
	}

	@Override
	protected String[] getServletMappings() {
		return new String[] { "*.htm" };
	}

}
```

```java
@Component
public class AddProductFormValidator implements Validator {

	@Override
	public boolean supports(Class<?> commandClass) {
		return commandClass.isAssignableFrom(AddProductForm.class);
	}

	@Override
	public void validate(Object command, Errors errors) {
		AddProductForm productForm = null;

		productForm = (AddProductForm) command;
		if (productForm.getName() == null
				|| productForm.getName().trim().length() == 0) {
			errors.rejectValue("name", "name.blank");
		}

		if (productForm.getManufacturer() <= 0) {
			errors.rejectValue("name", "manufacturer.blank");
		}

		if (errors.hasFieldErrors("price") == false) {
			if (productForm.getPrice() <= 0) {
				errors.rejectValue("price", "price.invalid");
			}
		}
	}
}
```

```
@Controller
@RequestMapping("/add-product.htm")
public class AddProductFormController {
	private static Logger logger = Logger
			.getLogger(AddProductFormController.class);

	@Autowired
	private ProductService productService;

	@InitBinder
	public void initBinder(WebDataBinder binder) {
		logger.debug("entered into initBinder()");
		binder.addValidators(new AddProductFormValidator());
	}

	@ModelAttribute("manufacturers")
	public List<ManufacturerDto> populateManufacturers(
			HttpServletRequest request) {
		List<ManufacturerDto> manufacturers = null;

		logger.info("calling productService to get all manufacturers");
		manufacturers = productService.getAllManufacturers();
		logger.debug("retrieved " + manufacturers.size() + " manufacturers");
		return manufacturers;
	}

	@RequestMapping(method = RequestMethod.GET)
	public String setupProductForm(Model model) {
		AddProductForm form = null;

		logger.info("entered into setupProductForm()");
		form = new AddProductForm();
		model.addAttribute("productForm", form);
		return "add-product";
	}

	@RequestMapping(method = RequestMethod.POST)
	public String addProduct(Model model,
			@Valid @ModelAttribute("productForm") AddProductForm form,
			Errors errors) {
		if (errors.hasErrors()) {
			logger.debug("errors found while submitting the form : "
					+ errors.getErrorCount());
			return "add-product";
		}
		logger.info("saving product with product name : " + form.getName());
		int productId = productService.saveProduct(form);
		logger.debug("saved product with product id : " + productId);
		model.addAttribute("id", productId);
		return "add-product-success";
	}
}
```

```java
@Service
public class ProductService {
	private static Logger logger = Logger.getLogger(ProductService.class);

	@Autowired
	private ProductDao productDao;

	public List<ManufacturerDto> getAllManufacturers() {
		List<ManufacturerDto> manufacturerDtos = null;
		List<Manufacturer> manufacturers = null;

		logger.info("entered into getAllManufacturers");
		manufacturers = productDao.getAllManufacturers();
		logger.debug("productDao returned : " + manufacturers.size()
				+ " manufacturers");

		manufacturerDtos = new ArrayList<ManufacturerDto>();
		for (Manufacturer manufacturer : manufacturers) {
			manufacturerDtos.add(new ManufacturerDto(manufacturer
					.getManufacturerId(), manufacturer.getName()));
		}

		return manufacturerDtos;
	}

	public int saveProduct(AddProductForm productForm) {
		Product product = null;
		Manufacturer manufacturer = null;

		logger.info("entered into saveProduct()");
		product = new Product();
		product.setName(productForm.getName());
		product.setType(productForm.getType());
		product.setPrice(productForm.getPrice());
		logger.debug("retrieving manufacturer from productDao for the manufacturerId : "
				+ productForm.getManufacturer());

		manufacturer = productDao
				.getManufacturer(productForm.getManufacturer());
		logger.debug("retrieved manufacturer : " + manufacturer.getName()
				+ " for manufacturer id : " + productForm.getManufacturer()
				+ " from productDao");

		product.setManufacturer(manufacturer);
		logger.info("calling productDao to save product");
		int productId = productDao.saveProduct(product);
		logger.debug("productDao saved product with productId : " + productId);

		return productId;
	}
}
```

```java
@Repository
public class ProductDao {
	private static Logger logger = Logger.getLogger(ProductDao.class);

	@Autowired
	private HibernateTemplate hibernateTemplate;

	public List<Manufacturer> getAllManufacturers() {
		logger.info("entered into getAllManufacturers");
		return (List<Manufacturer>) hibernateTemplate.find("from Manufacturer",
				null);
	}

	public Manufacturer getManufacturer(int manufacturerId) {
		logger.debug("entered into getManufacturer(" + manufacturerId + ")");
		return hibernateTemplate.load(Manufacturer.class, manufacturerId);
	}

	public int saveProduct(Product product) {
		logger.info("entered into saveProduct()");
		return (int) hibernateTemplate.save(product);
	}
}
```

Spring ORM

➔Spring ORM framework allows you to integrate with hibernate , Java persistence API(JPA) ,Java Data Object (JDO) and iBATIS for resource management and data access object (DAO) implementations and other startegies.

Benefits of Using ORM

➔**Easier Testing** :Spring IOC approach allows you to swap easily the implementations and configuration location of HibernateSessionFactory instances . so that you can point your configuration to various environment without modifying the source code.
➔**Common data access Exception**: Spring instead of exposing ORM specific checked exception to the top level tier's of the application, It will wrap the technology specific exception in to a common runtime DataAccessException Hirarchy.
➔**Integrated Transaction Management:** Instead of dealing with ORM technology related transactional related code ,It allows You to declaratively manage the transactionality using AOP Declarative Transaction management tags <tx:advice or Annotation driven @Transactional annotation

Integrated With Hibernate

➔In order to use Spring ORM with hibernate ,You need to declare your Hibernate Language Object using declarative mapping.hbm file or annotate your classes with hibernation annotations , In order to perform the Operations using these classes You need to declare the Hibernate SessionFactory and then inject into Hibernate Template it allows us to perform the Database operations using Template Approach.
➔To create Hibernate Template object we need Hibernate Session Factory object.

➔We can use Factory Beans like "LocalSessionFactoryBean" or "AnnotationSessionFactoryBean" classes as beans to create sessionFactory Object by supplying Hibernate Configuration Properties of Hibernate Configuration File or Mapping Annotations based Domain class and also DataSource Object.

➔Then the Hibernate Template will injected to DAO to perform the persistency operation.

ORM Configuration Annotation and Configuration(Annotation Class)

```
<?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="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"  value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="spr_usr"/>
        <property name="password" value="welcome1"/>
    </bean>

    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="annotatedClasses">
            <list>
                <value>org.hsi.entities.Journey</value>
                <value>org.hsi.entities.Bus</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
                <prop key="show_sql">true</prop>
            </props>
        </property>
    </bean>

    <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="jDao" class="com.redbus.dao.JourneyDao">
        <constructor-arg ref="hibernateTemplate"/>
    </bean>

    <aop:config>
            <aop:pointcut expression="execution(* com.hsi.service.ManageJourneyService.*(..))"
                                                                    id="manageJourneyService"/>
            <aop:advisor advice-ref="journeyTxAdvice" pointcut-ref="manageJourneyService"/>
        </aop:config>
        <tx:advice id="journeyTxAdvice" transaction-manager="transactionManager">
            <tx:attributes>
                <tx:method name="insert" read-only="false" propagation="REQUIRED"/>
            </tx:attributes>
        </tx:advice>
</beans>
```

ORM Configuration Annotation and Configuration (MappingResource)

```
<?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="dataSource"
		class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
		<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
		<property name="username" value="pms_user" />
		<property name="password" value="welcome1" />
	</bean>

	<bean id="sessionFactory"
		class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
		<property name="dataSource" ref="dataSource" />
		<property name="hibernateProperties">
			<props>
				<prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
				<prop key="hibernate.show_sql">true</prop>
			</props>
		</property>
		<property name="mappingResources">
			<list>
				<value>com/pm/entities/Product.hbm.xml</value>
				<value>com/pm/entities/Manufacturer.hbm.xml</value>
				<value>com/pm/entities/Specification.hbm.xml</value>
			</list>
		</property>
	</bean>

	<bean id="hibernateTemplate" class="org.springframework.orm.hibernate4.HibernateTemplate">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>

	<bean id="transactionManager"
		class="org.springframework.orm.hibernate4.HibernateTransactionManager">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>
</beans>
```

ORM Configuration Approach

```
@Configuration
@PropertySource("classpath:db.properties")
public class PersistenceConfig {
	@Autowired
	protected Environment env;

	@Bean
	public DataSource newDataSource() {
		DriverManagerDataSource dataSource = new DriverManagerDataSource();

		dataSource.setDriverClassName(env.getProperty("db.driverClassname"));
		dataSource.setUrl(env.getProperty("db.url"));
		dataSource.setUsername(env.getProperty("db.username"));
		dataSource.setPassword(env.getProperty("db.password"));

		return dataSource;
	}

	@Bean(autowire = Autowire.BY_TYPE)
	public LocalSessionFactoryBean newSessionFactory() {
		Properties hibernateProperties = null;
		LocalSessionFactoryBean sessionFactoryBean = null;

		sessionFactoryBean = new LocalSessionFactoryBean();
		sessionFactoryBean.setAnnotatedClasses(new Class<?>[] { Journey.class,
				Bus.class });
		hibernateProperties = new Properties();
		hibernateProperties.put("hibernate.dialect",
				"org.hibernate.dialect.Oracle10gDialect");
		hibernateProperties.put("show_sql", "true");

		sessionFactoryBean.setHibernateProperties(hibernateProperties);
		return sessionFactoryBean;
	}

	@Bean(autowire = Autowire.BY_TYPE)
	public HibernateTemplate newHibernateTemplate() {
		HibernateTemplate hibernateTemplate = null;

		hibernateTemplate = new HibernateTemplate();
		return hibernateTemplate;
	}

	@Bean(name = "transactionManager", autowire = Autowire.BY_TYPE)
	public HibernateTransactionManager newHibernateTransactionManager() {
		HibernateTransactionManager transactionManager = null;

		transactionManager = new HibernateTransactionManager();
		return transactionManager;
	}

}
```

Spring With Hibernate Integration Annotation Approach Example

add-journey.jsp

```
<body style="font-family: consolas; font-size: 18px">
     <h2>Add Journey</h2>
     <form:form modelAttribute="journey">
          <span style="color: red;">
               <form:errors path="*" />
          </span>
          Source:<form:input path="source" />
          Destination:<form:input path="destination"/>
          JourneyDate:<form:input path="journeyDate"/>
          Bus No:<form:input path="busNo"/>
          Amount:<form:input path="amount"/>
          <input type="submit" value="add" />
     </form:form>
</body>
```

journey-details.jsp

```
<body style="font-family: consolas;font-size: 18px">
     Journey with Id : ${journeyId} has been added
</body>
```

db.properties

```
db.driverClassname=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:xe
db.username=spring_usr
db.password=welcome1
```

errors.properties

```
source.blank=Source cannot be blank
destination.blank=Destination cannot be blank
amount.invalid=Amount should be non-zero positive integer
```

```
public class RedbusInitializer extends
          AbstractAnnotationConfigDispatcherServletInitializer {
     @Override
     protected Class<?>[] getRootConfigClasses() {
          return new Class<?>[] { RootConfig.class };
     }
     @Override
     protected Class<?>[] getServletConfigClasses() {
          return new Class<?>[] { WebConfig.class };
     }
     @Override
     protected String[] getServletMappings() {
          return new String[] { "*.htm" };
     }
}
```

```
public class JourneyForm {
     protected String source;
     protected String destination;
     @DateTimeFormat(pattern = "dd/mm/yyyy")
     protected Date journeyDate;
     protected int busNo;
     protected float amount;
     //Settters & Getters
}
```

```
public class JourneyBo {
     protected int journeyId;
     protected String source;
     protected String destination;
     protected Date journeyDate;
     protected int busNo;
     protected float amount;
     //Setters & Getters
}
```

```java
@Configuration
@Import(PersistenceConfig.class)
@EnableTransactionManagement(proxyTargetClass = true)
@ComponentScan(basePackages = { "com.rb.dao", "com.rb.service" })
public class RootConfig {

}
```

```java
@Configuration
@PropertySource("classpath:db.properties")
public class PersistenceConfig {
	@Autowired
	protected Environment env;

	@Bean
	public DataSource newDataSource() {
		DriverManagerDataSource dataSource = new DriverManagerDataSource();

		dataSource.setDriverClassName(env.getProperty("db.driverClassname"));
		dataSource.setUrl(env.getProperty("db.url"));
		dataSource.setUsername(env.getProperty("db.username"));
		dataSource.setPassword(env.getProperty("db.password"));

		return dataSource;
	}

	@Bean(autowire = Autowire.BY_TYPE)
	public LocalSessionFactoryBean newSessionFactory() {
		Properties hibernateProperties = null;
		LocalSessionFactoryBean sessionFactoryBean = null;

		sessionFactoryBean = new LocalSessionFactoryBean();
		sessionFactoryBean.setAnnotatedClasses(new Class<?>[] { Journey.class,
				Bus.class });
		hibernateProperties = new Properties();
		hibernateProperties.put("hibernate.dialect",
				"org.hibernate.dialect.Oracle10gDialect");
		hibernateProperties.put("show_sql", "true");

		sessionFactoryBean.setHibernateProperties(hibernateProperties);
		return sessionFactoryBean;
	}

	@Bean(autowire = Autowire.BY_TYPE)
	public HibernateTemplate newHibernateTemplate() {
		HibernateTemplate hibernateTemplate = null;

		hibernateTemplate = new HibernateTemplate();
		return hibernateTemplate;
	}

	@Bean(name = "transactionManager", autowire = Autowire.BY_TYPE)
	public HibernateTransactionManager newHibernateTransactionManager() {
		HibernateTransactionManager transactionManager = null;

		transactionManager = new HibernateTransactionManager();
		return transactionManager;
	}

}
```

```java
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.rb.controller", "com.rb.validator" })
public class WebConfig extends WebMvcConfigurerAdapter {

	@Bean(name = "messageSource")
	public MessageSource newMessageSource() {
		ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
		messageSource.setBasename("errors");
		return messageSource;
	}

	@Override
	public void configureViewResolvers(ViewResolverRegistry registry) {
		registry.jsp("/WEB-INF/jsp/", ".jsp");
	}

}
```

```java
@Controller
@RequestMapping("/add-journey.htm")
public class AddJourneyFormController {
	@Autowired
	private JourneyFormValidator journeyFormValidator;

	@Autowired
	private ManageJourneyService manageJourneyService;

	@InitBinder
	public void initBinder(WebDataBinder binder) {
		binder.addValidators(journeyFormValidator);
	}
	@RequestMapping(method = RequestMethod.GET)
	public String showJourneyForm(Model model) {
		JourneyForm journeyForm = null;

		journeyForm = new JourneyForm();
		model.addAttribute("journey", journeyForm);
		return "add-journey";
	}
	@RequestMapping(method = RequestMethod.POST)
	public String addJourney(Model model,
			@Valid @ModelAttribute("journey") JourneyForm journeyForm,
			BindingResult errors) {
		if (errors.hasErrors()) {
			// display back source page
			return "add-journey";
		}
		// no errors
		int journeyId = manageJourneyService.addJourney(journeyForm);
		model.addAttribute("journeyId", journeyId);
		return "journey-details";
	}
}
```

```java
@Entity
@Table(name = "BUS")
public class Bus {
	@Id
	@GeneratedValue(generator = "hib_increment")
	@GenericGenerator(name = "hib_increment", strategy = "increment")
	@Column(name = "BUS_NO")
	protected int busNo;
	@Column(name = "SERVICE_NO")
	protected String serviceNo;
	protected String type;
	protected int capacity;

	@OneToMany(mappedBy = "bus")
	protected Set<Journey> journeys;
	//Setters & Getters
}
```

```java
@Entity
@Table(name = "JOURNEY")
public class Journey {
	@Id
	@GeneratedValue(generator = "hib_increment")
	@GenericGenerator(name = "hib_increment", strategy = "increment")
	@Column(name = "JOURNEY_NO")
	protected int journeyNo;
	protected String source;
	protected String destination;
	@Column(name = "JOURNEY_DT")
	protected Date journeyDate;
	protected float amount;

	@ManyToOne
	@JoinColumn(name = "BUS_NO")
	protected Bus bus;
	//Setters & Getters
}
```

```java
@Component
public class JourneyFormValidator implements Validator {

	@Override
	public boolean supports(Class<?> classType) {
		return classType.isAssignableFrom(JourneyForm.class);
	}

	@Override
	public void validate(Object object, Errors errors) {
		JourneyForm form = null;

		form = (JourneyForm) object;
		if (form.getSource() == null || form.getSource().trim().length() <= 0) {
			errors.rejectValue("source", "source.blank");
		}
		if (form.getDestination() == null
				|| form.getDestination().trim().length() <= 0) {
			errors.rejectValue("destination", "destination.blank");
		}

		if (form.getAmount() <= 0) {
			errors.rejectValue("amount", "amount.invalid");
		}
	}
}
```

```java
@Service
public class ManageJourneyService {
	@Autowired
	protected JourneyDao journeyDao;

	@Transactional(readOnly = false)
	public int addJourney(JourneyForm journeyForm) {
		JourneyBo journeyBo = null;
		int journeyId = 0;

		journeyBo = new JourneyBo();
		journeyBo.setSource(journeyForm.getSource());
		journeyBo.setDestination(journeyForm.getDestination());
		journeyBo.setJourneyDate(journeyForm.getJourneyDate());
		journeyBo.setBusNo(journeyForm.getBusNo());
		journeyBo.setAmount(journeyForm.getAmount());

		journeyId = journeyDao.saveJourney(journeyBo);

		return journeyId;
	}
}
```

```java
@Repository
public class JourneyDao {
	@Autowired
	protected HibernateTemplate hibernateTemplate;

	public int saveJourney(JourneyBo journeyBo) {
		Journey journey = null;
		int journeyId = 0;
		Bus bus = null;

		bus = hibernateTemplate.load(Bus.class, journeyBo.getBusNo());
		journey = new Journey();
		journey.setSource(journeyBo.getSource());
		journey.setDestination(journeyBo.getDestination());
		journey.setAmount(journeyBo.getAmount());
		journey.setJourneyDate(journeyBo.getJourneyDate());
		journey.setBus(bus);

		journeyId = (int) hibernateTemplate.save(journey);
		return journeyId;
	}
}
```

InitFormBinder

➔While working with form binder we can override initBinder(--,--) method to register properties editor that are required to use while binding given form data.

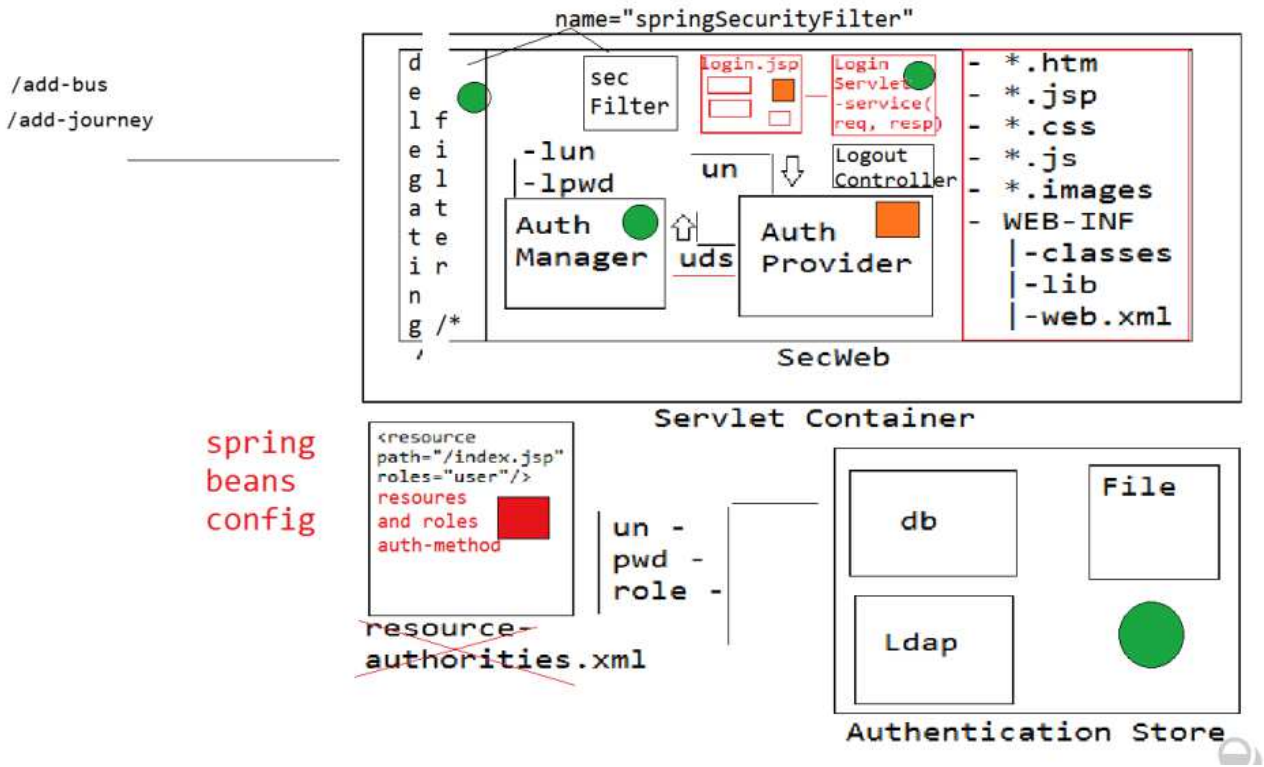
➔If we are using annotation approach we can add validator through @InitBinder and initBinder(--) method.

Spring Security

➔To perform dependency injection on servlet filter component it can't be possible because its class must be maintained by IOC container as spring bean having configuration in spring bean configuration file.
➔Due to this it can't take request from client as filter is not configured as web component in web.xml file.
➔To overcome this problem we configure a special proxy filter in web.xml file to take the request from the client to pass the request to filter component that is there in IOC container as spring bean.
➔The special filter is "org.springframework.web.filter.DelegatingProxyFilter".
➔But we need to match logical name of DelegatingProxyFilter with bean name of filter component.

Security (authentication + authorization)

➔Checking the identity of the user is called authentication.
➔Checking the access permission of the user is called authorization.
➔Provide the authorization /access permission not based on user name ,provide them based on roles.



How to Secure our application?

By default Security is not enable .We need to implement the security for an application for securing of our resource of our application.
First we need :
Authentication Store :
➔Authentication store may be LDAP Server ,DB or file where we need to store user information like username ,password and the role of the user .
➔We should able to access the credential within the authentication store and should able to validate in the authentication store.
➔So we need Some component For accessing the data from the authentication store, thats why authentication **Provider** will be there.
➔For every security mechanism it could be j2ee,it could be spring or it could be our custom security .Everyone has to implement the component in this way .This could be the final architecture of the security .
➔So Authentication Provider will access the authentication information to the Authentication Store and passes the data to **Authentication Manager.**
➔Authentication Provider will access the authentication information to the Authentication Store and passes the data to **Authentication Manager** User details like username, password and role of user pass to Authentication Manager .
➔Authentication manager will have 2-task one is

Authentication
Authorization

➔Both the aspect will manage by Authentication Manager Whenever the user will try to log in to the application, Whenever the user will try to submit the request to the login.jsp by entering the username and password the request comes to LoginServlet.java .
➔Filter will check user is login or not? If not logged in again the request is comes to logging again filter will come again it will forward.
➔So filter should not be applied in login.jsp and LoginServlet.java.Without login user should be access login.jsp and LoginServlet.java
➔Now the user fist enter the user name and password in the login.jsp and enter the submit .
➔Then The request is comes to LoginServlet.java. The LoginServlet takes the username and password and goto Authentication Manager, asking whether the current user apply name of username and password is valid or invalid?
➔If it is invalid throws an exception and if it is valid takes the user details of the user representing the role of the user will return to the loginServlet .
➔Then Login Servlet has to maintain the authentication information of the current user who is accessing the system in the session level .
➔In the LoginServlet it has to create one session .
➔The current user who is accessing the system and it should be placed to the logged in user details object in to the session and forward the user to the home page.
➔Whenever the user is trying to access any of the resource of our application request will comes to whom **Authentication Filter.**

What does the Authentication Filter will do?

➔Authentication filter will not check the incoming request takes the request and goes to the session .Checks session is there or not there ? User is logged in user or not logged in user it will check first? Rather It will check what is the resource the user is trying to access how by going to **resource-autherity.xml** check the resource is the user is trying access has authority or anyone can access or not .
➔That means ammoniums user can access or only authenticated user only will be permit to access first it will check .
➔If ammoniums user also can access it will not redirect to login page .it just forward the request to the corresponding resource .

➔If the resource the user is requesting has to be authorise user can access then it goes to the session check current user is logged in or not by check to the session If not redirect the user to the login.jsp page and asking to logged in before accessing the resource .

➔If the user is already logged in the resource authority and user authority is matching or not .
If matching forward the request to the resource if not throws an Exception saying unauthorised access of the resources. All these thing doing by the Authentication Filter .

➔Security is something is required all the application that's why the underline J2EE is provided the Security mechanism .
➔If you use j2ee Security all the component we are not to write J2ee provide some commonest like Authentication Filter,login and LoginServlet ,Autentication Manager ,Authentication Provider and Authentication Store .

Who will Read the data From Authentication Store?

Authentication Provider
➔Authentication Provider will not validate he will take the user information and pass to other class within our application.

Why Authenticate Provider will not validate the username and password ?

➔Validation logic must be rewritten in other class when we switch from one Authentication Store to other Authentication Store.
➔Validating the credential are be separate from fetching the credential from Authentication Store .
Authentication provider will access the details and passes the input to the Authentication Manager.

➔Authentication Provider will fetch the username related record from the authentication store with which we need to compare the provider username and password is matching with the Authenticate username password .
➔That mean the Authenticate provider we need to pass the login username .Then he is going to pass User Details Object .User Details Object means Username, password, Role of the user will pass the input to Authentication Manager.

What does my Authenticate manager will do?

➔The authenticate manager will compare the user details with login user.
➔If it is valid then it take the role of the user once it is identify role of user then it is going to the configuration file resource-authority.xml which resource can access by which authorization role user has to configure by the programmer .
➔What are the resources are there in our system, and there resources are access by which role user will configure by us .
➔Which is not automatic which is manually configure because i am the owner i am deciding which page is accessing by whom.

In non Spring environment we can implement security .
JavaEE Web security
➔Here Authentication Manager and Authentication Provider work will be taken care by servletContainer based on the configuration in web.xml to reduce border on programmer .
➔**But there are some limitations**

1. Web.xml entries are not portable across the multiple servers.
2. Some web server don't support this model.
3. Going to commercial Application server only for security model is very cost effective.
4. Some server don't support LDAP server as authentication store.

To overcome this problem use spring security comes into picture

1. It run in all the servers and works irrespective they support security or not.
2. Support all the model and authentication Stores
3. It is also declarative security model.
4. Can be applied for normal web application and Spring MVC web applications.

Note

➔We can configure spring security definitions in spring bean configuration file having security name space , based on the configuration internally one ServletFilter will be generated as spring bean in IOC Container having bean name as "springSecurityFilterChain" so to take request from client add to pass to request the above generated filter we need to configure "DelegatingFilterProxy" in web.xml with logical name "spring SecurityFilterChain" .

We need Spring security jars of

Spring-security-3.0.5 RELEASE.jar contains

1. spring-security-config-3.0.5 RELEASE.jar
2. spring-security-core-3.0.5 RELEASE.jar
3. spring-security-web-3.0.5 RELESE.jar

To use Web security we need

Authentication Store	Spring has provided its own Authentication Store if You want to store data in other Authentication Store you need to write your own Authentication Provider by implementing your class from userDetailsService
Authentication Provider	Spring has provider
Authentication Manager	Spring has Provided
Login Servlet	Spring has provided if you want your own login page you can write but you need to security-beans.xml
Filter	Spring has provide DelegatingFilterProxy we need to configure as " spring SecurityFilterChain"
Role Configuration	You need to configure that in spring bean configuration file. No need to write in web.xml

Basic Authentication

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mvc="http://www.springframework.org/schema/mvc"
	xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<mvc:view-controller path="/home.htm" view-name="home" />
	<mvc:view-controller path="/logout-success.htm" view-name="logout"/>

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

home.jsp

```
<body style="font-family: consolas; font-size: 18px;">
	You are using spring security <a href="${pageContext.request.contextPath}/logout.htm">Logout</a>
</body>
```

logout.jsp

```
<body style="font-family: consolas; font-size: 18px;">
	You are logged out
</body>
```

security-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.2.xsd
            http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <security:http auto-config="true">
        <security:logout logout-url="/logout.htm" logout-success-url="/logout-success.htm"/>
        <security:intercept-url pattern="/home.htm" access="ROLE_AGENT"/>
    </security:http>

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="john" password="welcome1" authorities="ROLE_AGENT"/>
                <security:user name="sriman" password="welcome1" authorities="ROLE_ADMIN"/>
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>

</beans>
```

web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://java.sun.com/xml/ns/javaee"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
       id="WebApp_ID" version="3.0">
       <display-name>BasicSecu</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/application-context.xml;/WEB-INF/config/security-beans.xml</param-value>
    </context-param>

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

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <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>

</web-app>
```

Database Authentication

home.jsp

```
<body style="font-family: consolas; font-size: 18px;">
        ${pageContext.request.userPrincipal.name}
        You are using spring security with database <a href="${pageContext.request.contextPath}/logout.htm">Logout</a>

</body>
```

login.jsp

```
<body>
            <h2>Login</h2>
            <span style="color: red;">
                <%if(request.getParameter("error") != null){
                    out.print("un/pwd not valid");
                }
                %>
            </span>
            <form action="${pageContext.request.contextPath}/j_spring_security_check" method="post">
                Username: <input type="text" name="j_username"/>
                Password: <input type="password" name="j_password"/>
                <input type="submit" value="login"/>
            </form>
        </body>
```

logout.jsp

```
<body style="font-family: consolas; font-size: 18px;">
        You are logged out
</body
```

application-contect.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">

        <import resource="config/persistence-beans.xml" />
        <context:component-scan base-package="com.das.dao;com.das.security.service" />
</beans>
```

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
                http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">

        <mvc:view-controller path="/login.htm" view-name="login"/>
        <mvc:view-controller path="/home.htm" view-name="home"/>
        <mvc:view-controller path="/logout-success.htm" view-name="logout"/>

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

web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0">
    <display-name>DBAuthSec</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/application-context.xml;/WEB-INF/config/security-beans.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <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>

</web-app>
```

persistence-beans.xml

```
<?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="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="username" value="spring_usr" />
        <property name="password" value="welcome1" />
    </bean>
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <constructor-arg ref="dataSource" />
    </bean>

</beans>
```

security-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:security="http://www.springframework.org/schema/security"
	xsi:schemaLocation="http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.2.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<security:http auto-config="true">
		<security:form-login login-page="/login.htm"
			login-processing-url="/j_spring_security_check" username-parameter="j_username"
			password-parameter="j_password" default-target-url="/home.htm"
			always-use-default-target="true" authentication-failure-handler-ref="authenticationFailureHandler" />

		<security:logout logout-url="/logout.htm"
			logout-success-url="/logout-success.htm" />
		<security:intercept-url pattern="/home.htm"
			access="ROLE_AGENT" />
	</security:http>

	<security:authentication-manager>
		<security:authentication-provider
			user-service-ref="userDetailsServiceImpl" />
	</security:authentication-manager>

	<bean id="authenticationFailureHandler"
		class="org.springframework.security.web.authentication.ExceptionMappingAuthenticationFailureHandler">
		<property name="exceptionMappings">
			<props>
				<prop
					key="org.springframework.security.authentication.BadCredentialsException">
					/login.htm?error=badCredentails
				</prop>
			</props>
		</property>
	</bean>
</beans>
```

```
public class UserBo {
	protected int userId;
	protected String username;
	protected String password;
	protected String role;
	//Setters and Getters
}
```

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
	@Autowired
	private UserDao userDao;

	@Override
	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
		UserBo userBo = null;
		UserDetails userDetails = null;

		userBo = userDao.findUserByName(username);
		userDetails = new UserDetailsImpl(userBo.getUsername(), userBo.getPassword(), userBo.getRole());

		return userDetails;
	}

}
```

```
public class UserDetailsImpl implements UserDetails {
	protected String username;
	protected String password;
	protected String role;
	protected Collection<SimpleGrantedAuthority> authorities;
	public UserDetailsImpl(String username, String password, String role) {
		this.username = username;
		this.password = password;
		this.role = role;
		authorities = new ArrayList<SimpleGrantedAuthority>();
		authorities.add(new SimpleGrantedAuthority(role));

	}
	@Override
	public Collection<? extends GrantedAuthority> getAuthorities() {
		return authorities;
	}
	@Override
	public String getPassword() {
		return password;
	}
	@Override
	public String getUsername() {
		return username;
	}
	@Override
	public boolean isAccountNonExpired() {
		return true;
	}
	@Override
	public boolean isAccountNonLocked() {
		return true;
	}
	@Override
	public boolean isCredentialsNonExpired() {
		return true;
	}
	@Override
	public boolean isEnabled() {
		return true;
	}

}
```

```java
@Repository
public class UserDao {
	private final String SQL_GET_USER_BY_NM = "SELECT * FROM USERS U INNER JOIN
				USER_ROLES UR ON U.USER_ROLE_ID = UR.USER_ROLE_ID WHERE USER_NM = ?";

	@Autowired
	private JdbcTemplate jdbcTemplate;

	public UserBo findUserByName(String userName) {
		return jdbcTemplate.queryForObject(SQL_GET_USER_BY_NM, new RowMapper<UserBo>() {
			@Override
			public UserBo mapRow(ResultSet rs, int row) throws SQLException {
				UserBo bo = new UserBo();
				bo.setUserId(rs.getInt("USER_ID"));
				bo.setUsername(rs.getString("USER_NM"));
				bo.setPassword(rs.getString("PASSWORD"));
				bo.setRole(rs.getString("ROLE_NM"));

				return bo;
			}
		}, new Object[] { userName });
	}
}
```