**Spring MVC by Naveen**

**Spring MVC (Model View Controller):**

MVC is a framework which support up-to model part i.e. as we know first Presentation Layer (view) comes then Controller and finally Model( which contains Business classes , DAO classes and some services classes) .

Here in Model we can go through simple POJO classes for all the classes (business, DAO and services classes). For controller also we can use simple POJO classes.

So the question is who will execute these POJO classes. So the answer is IOC container will execute these classes. It means these POJO classes will run on top of IOC container.

In the Spring MVC they have given fallowing features.

1. **Form backup support (FBS):**

It is useful for presentation (view) part. By using this FBS we can hold data into bean classes whenever we try to submit form data. We can apply validation for these beans and if we find any errors, we can populate the validation error with the bean into same form.

Let’s say we have some form field (Id, Name, Email and Mobile). Suppose if we entered invalid mobile number and in this case we want do server side validation then we need to submit form dataafter submission then these data will be stored into bean and will execute validation. After completion validation the validation error need to print into the same form like “Mobile Number is not valid”. Along with displaying this message it needs to delete the existing mobile number value and previous valid field data need to maintain as such in the textbox. So these are nothing but form backup support.

But to achieve this form backup support for presentation part we will have to use spring tags libraries. In struts also we have form backup support and we have to use struts tag libraries. So by using spring tag we have to create presentation (view) part.

1. **Controller Support (Multi-action controllers and multi-form controller**):

**Here** they are supporting Multiple-action and multiple form support

By using this multi-action and multi-form controller what we can do?

**Multi-action controllers**

Let’s say we have multiple input forms like

1. Student registration form
2. Student search form
3. Student update form

Now if all the above three operations we want to execute from one single controller class, then we can write all three operation into one single controller like (registration () method for student registration, search () method for Student search, and update () method for Student update). So this case is nothing but Multi- Action controller because each form has their own method (action) the controller class.

Diagram for multiple form with single controller.

**Multi-form controller**

Let’s say we have Adhaar registration which has multiple forms and one single controller. Suppose if we have Adhaar form1 which consists of field (First Name, Last Name and Age). Adhaar form2 consists of education detail like (High School, Intermediate, and Graduations etc.). Adhaar form3 consists of address detail (Address Line 1, Address Line 2 etc.). Now here we have multi form and we have submitted all from data from one method only.

In case of **servlet** we have to store two form data (Adhaar form1 and Adhaar form2) into session object and finally in the third from we will call one controller method to submit all form data into server for validation. In this controller method we will read from 1 and 2 data from session and from3 data from request object and finally we need to submit it. So this case is nothing but multi-form controller because here several multi-forms have only one method-action in the controller class.

In struts we have Validator action form support, by using Validator action form we can store multiple form’s data into one single bean and finally we can process it. So the same type of action classes they have given here in spring also. By using action classes we can handle validation as well as multiple from support

1. **Validation Support:**
2. **Internationalization (I18N) support**
3. **Interceptors support**
4. **View Resolver**
5. **Exception Handling**
6. **Modularization support.**

**In spring all the Model part (business and service classes) can be executed by IOC container. Spring MVC IOC container name is WebApplicationContext.**

**Note:** For designing MVC framework the recommended architecture models are JSP model Architecture. By fallowing JSP model architecture only the spring vendors designed the Spring MVC framework. **The other frameworks like Struts and JSF also have developed by using JSP Model Architecture.**

JSP Model Architecture:

There are four (4) JSP model architecture. As per JSP model architecture multiple inputs requests we need to handle by only one single controller and this controller need to executes the common required operation (like validation) for this multiple input pages . So this common required operation we can write into controller class instead of writing in each pages for their own validation.

Another operation is to read data from JSP we need to write request.getParameter (“id”) for each input and we need to store these data into some fields according to their type. Like String name = request.getParameter (“name”); so here we have another operation to read data so we can write one common operation readData () into controller. And thus after reading data using readData () we can hold the data into beans objects and from these beans objects we can collect required data.

So all the common operation has to be handled by one single controller and after executing this common operation we need to execute the specific operation for executing specific operation need to execute some helper controller classes. For example.

Let’s say we have three from (Student registration, Student update, Login) so all the common operation has to execute from one single controller class after that the specific operation has to be executed from helper controller classes like for Student registration one StudentRegistration controller , for Student update one SutdentUpdate controller and for Login one LoginController. So instead of reading jsp form data from helper controller classes we will read these common operations from one single controller class and only specific code need to write into helper controller class.

So finally we can say that in three forms we have reduced three common operation from helper classes in the same way if have thousand forms for application, so we can reduce thousand time for reading and validation form data. We can get bean objects from validator controller and finally we can send it to for required model operation

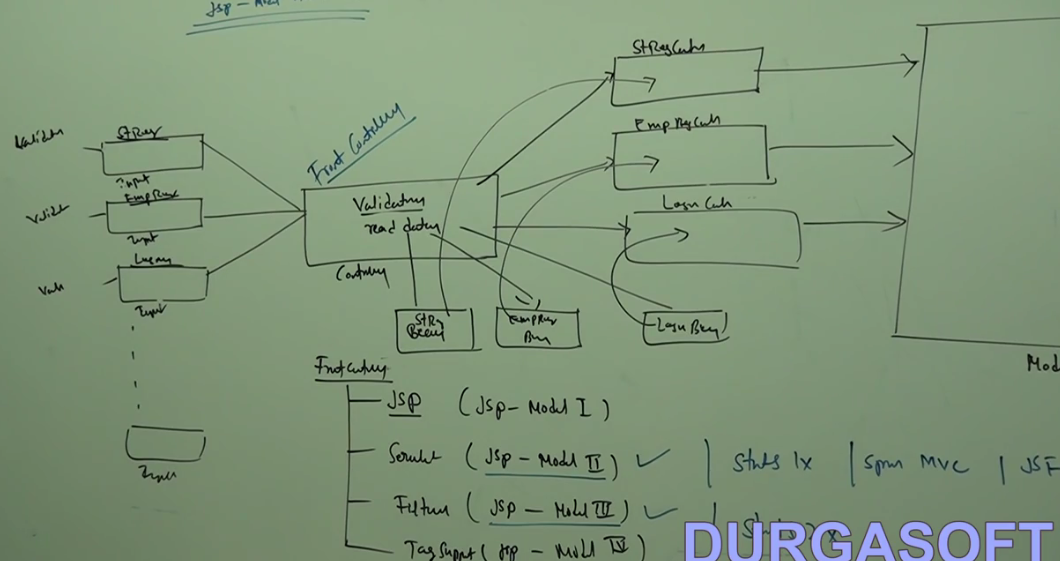
Diagram:

So as per JSP model architecture we have to write all common operation into one single controller. We can call this controller “Front Controller”.

1. So if the front controller is JSP then it is call JSP-Model 1 architecture.
2. So if the front controller is Servlet then it is call JSP-Model 2 architecture.
3. So if the front controller is Filter then it is call JSP-Model 3 architecture.
4. So if the front controller is Tag- support then it is call JSP-Model 4 architecture.

Note: 1- Out of these four jsp models the recommended models are Servlet and Filter i.e. JSP Model 2 and 3. Because if we go with Model 1 and 4 then in both the cases we have to use JSP page as controller and JSP page is not much secure that Servlet and Filter.

2- By using JSP Model 2 and 3 we have several frameworks. E.g. Struts 1x is based on JSP Model 2, Struts 2x is based on JSP Model 3. And by using JSP Model 2 only spring people develop Spring MVC framework

****

So finally we can say that by using JSP-Model-2 architecture this spring MVC they designed. Since they used JSP-Model-2 so under controller part we have to use servlet class i.e. front controller should be a servlet class. JSF also is designed by using JSP-Model-2.

So only one single controller called as Front Controller that we need to configure for multiple input pages and that single controller they designed and gave it to us. In this controller they have given common required operation (like **validation, Internationalization, controller support, expectation handling support, tiles designing, Template support**) for all the ERP applications. And if we have any user specific operations they gave some helper controller by using which we can write the user specific operation.

**Q**: So the question is which front controller class we need to use for Spring MVC?

**A:** In Spring MVC FC name is **DispatcherServlet**. In Strut 1x the FC name is **ActionServlet** and for Struts 2x the FC name **FilterDispatcher**. In JSF FC name is **FacesServlet.**

**Front Controller configuration:** To configure front controller we have to use front controller design pattern and as per FC design pattern multiple input/action we need to map to one single controller. Now the question arises how to map multiple action to one single controller (DispatcherServlet).

Let’s say we have multiple inputs (like 1- Registration Page, 2-Search Page, 3-Login Page). Now if all three form we need to Map to one single controller (DS). To map forms request to controller we required from action URL pattern. Let’s say the action URL patter of all three forms are 1- / req, 2- /search, 3-/login respectively. Now all the URL patter will be mapped inside web.xml like given below.

|  |  |  |
| --- | --- | --- |
| /reg  Registration Page | **Web.xml- using complete character sequence**  <web-app>  <servlet>  <servlet-name>DS</servlet-name>  <servlet-class>  DispatcherServlet  </servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>DS</servlet-name>  <url-pattern>/reg</url-patter>  <url-pattern>/search</url-patter>  <url-pattern>/login</url-patter>  </servlet-mapping>  </web-app> | **Web.xml(for two MVC -spring and struts)**  <web-app>  <servlet>  <servlet-name>**SpringFC**</servlet-name>  **<servlet-class>DispatcherServlet</servlet-class>**  </servlet>  <servlet>  <servlet-name>**StrutsFC**</servlet-name>  **<servlet-class>ActionServlet</servlet-class>**  </servlet>  <servlet-mapping>  <servlet-name> **SpringFC** </servlet-name>  <url-pattern>**/\***</url-patter>  </servlet-mapping>  <servlet-mapping>  <servlet-name> **StrutsFC**</servlet-name>  <url-pattern>**/\***</url-patter>  </servlet-mapping>  </web-app>  **Note**: Will get ambiguity here by using same url pattern (/\*). |
| /search  Search Page |
| /login  Login Page  /update  StudentUpdate  /delete  StudentDelete |

1. Here in the above diagram we have three input form with their action url pattern which will be configured in web.xml
2. In web.xml we have configured the front controller (DispatcherServlet) developed by spring.
3. In web.xml we have configured all the form action URL which being mapped to one single controller class DispatcherServlet. So if request comes from any of three forms we are trying to request only one single controller DispatcherServlet.
4. Similarly if we have 1000 input form then under <servlet-mapping> we will have to map 1000 url patter for 1000 form respectively in the web.xml file. Apart from this we have to fallow some naming conventions i.e. every url-pattern should be unique. It should not repeat again. So we can say that this will increase number tags in web.xml file. So instead of using 1000 url-pattern in the web.xml by using one single expression tag also we can map all 1000 form’s request to one single controller DS.
5. We can use three expressions under <url-pattern> tag to map number of form’s request. These express is given by Tomcat-Apache development team. Three allowed pattern by tomcat container are
6. **Complete character sequence**(Full Name of URL pattern) : As we have written in the first Web.xml
7. **/\*** (In this case any type of form’s request we can pass to DispatcherServlet). /\* is not recommended if our application has to go through multiple MVC framework. This is recommended for only one single MVC framework. If we use this pattern in multiple MVC frameworks then we will get ambiguity i.e. it creates the ambiguity that which MVC framework controller class, the request is to send.
8. **\*.ext** (in this case it will pass only that form’s request whose extentation will match with the URL pattern extentation. This URL pattern is recommended if our application has to go through multiple MVC frameworks (like Spring MVC Framework and Struts MVC framework).

**Note:** Here in the 3rd column of the above example we have defined /\* for both spring as well as Struts MVC framework to send the request to its controller classes. Now if two pattern (/\*) are same then the question arises that which request will go to DispatcherServlet (Spring MVC) and which request will go to ActionServlet (Struts MVC) ? Here, there is chance to get Ambiguity that which form’s request need to go to DS and which form’s request need to go to AS.

**Ans:** So in this case it is not possible to identify that which forms’s request will go to which controller class. So to avoid this ambiguity, (**\*.ext)** is recommended to find out which form’s request is for which controller class. Here as an extentation (.ext) we can use anything like (\*.ApplicationName or \*. Icici or \*.Anyname). Generally for struts we use (**\*.do or \*.action**).

So here in the above example to avoid ambiguity, to recognize spring form we will use (.sp) and for Struts we will use (\*.st) . So the above example can be written now.

|  |  |  |
| --- | --- | --- |
| /reg**.sp**  Registration Page | **Web.xml(for two MVC -spring and struts)**  <web-app>  <servlet>  <servlet-name>**SpringFC**</servlet-name>  **<servlet-class>DispatcherServlet</servlet-class>**  </servlet>  <servlet>  <servlet-name>**StrutsFC**</servlet-name>  **<servlet-class>ActionServlet</servlet-class>**  </servlet>  <servlet-mapping>  <servlet-name> **SpringFC** </servlet-name>  <url-pattern>**\*.sp**</url-patter>  </servlet-mapping>  <servlet-mapping>  <servlet-name> **StrutsFC**</servlet-name>  <url-pattern>**\*st**</url-patter>  </servlet-mapping>  </web-app>  **Note**: Will not get ambiguity here by using same url pattern (**\*.ext**). | **So in the spring Web.xml will be:-**  <web-app>  <servlet>  <servlet-name>DS</servlet-name>  <servlet-class>  **Org.springframework.web.servlet.DispatcherServlet**  </servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>DS</servlet-name>  <url-pattern>\*.htm </url-patter>  </servlet-mapping>  </web-app> |
| /search**.sp**  Search Page |
| /login**.sp**  Login Page  /update**.st**  StudentUpdate  /delete.**st**  StudentDelete |

Now here in the second column example: if any form’s request comes with extentation (**.sp**) so according to <servlet-name> the request will go to **DispatcherServlet** controller class.

In the same way if form’s request comes with (.st) extentation then it will Struts controller class **ActionServlet**. So in multi MVC framework application (\*.ext) is recommended.

In struts there is recommendation to use (\*.do) for struts-1 and (\*.action) for struts-2. But there is no any recommendation for spring we can use any extentation in spring framework. But generally we use (.htm) in case of spring as shown in the above example in third column.

**Example of Hello and its explanation:**  Steps

1. Now create one web.xml and configure spring front controller (DispatcherServlet), servlet name and URL pattern.
2. Create one input page either using simple html tag or using spring form’s tag. Form will have one input filed Name and one submit button and one from action (action =”./hello.htm”). This htm request first will to DispatcherServlet. DS is a predefined servlet and my request operation is to print hello.
3. So to print hello one specific user defined controller class (HelloController.java) needs to create. In case of servlet we need to implement our user defined servlet from servlet interface or Generic servlet or HttpServlet. But in spring we already have one predefined DS. This DS need to map the same form request(action =”./hello.htm”) to our user defined controller class (HelloController.java)
4. To implement user defined controller (HelloController.java) we have to use fallowing controller classes.
5. Controller (I): We can implement this interface from our controller class (HelloController.java). OR
6. AbstractController(C): our controller class (HelloController.java) can extend this class. OR
7. AbstractCommandController(C) OR
8. SimpleFormController( C) OR
9. AbstractWizardFormController( C) OR
10. MultiActionController (C) OR
11. Or we can implement our controller class from Stereo type annotation using @Controller.
12. While implementing or extending above controller class to create our own user defined controller class then we have to override one method handleRequest whose return type will be ModelAndView: handleRequest method is an abstract method of Controller (I) interface.

**Public ModelAndView handleRequest (HttpServletRequest req, HttpServletResponse res) { }**

1. Here we are trying to Map input page request to DispatcherServlet in web.xml and from DispatcherServlet to user-defined controller class (HelloController.java). So to map input request to dispatcher servlet we have create one xml called Web.xml. Similarly to map request from DS to user-defined controller class (HelloController.java) we will create one more xml whose name will put like given below: **servletName-spring.xml** (like - DS-spring.xml).
2. **So** in this xml file using <bean> tag we will configure input page url request (/hello.htm) and user-defined controller class (HelloController). By reading this xml file DispatcherServlet will understands that for URL request (/hello.htm) HelloController.java class has to execute. And In HelloController class the method **handleRequest** will be execute.

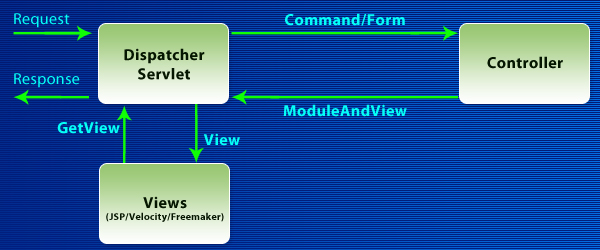
**Example Pending: So final project structure of HelloWorld example in spring**

1. Index.jsp
2. Success.jsp
3. Web.xml
4. Spring.xml
5. HelloWorld.java

**Spring MVC Controllers – Controllers hierarchy in Spring MVC**

In this we will understand the controller’s hierarchy in Spring MVC Module. The Spring MVC module provides a lot of flexibility to easily develop MVC based web applications. It provides many controllers that can be used to achieve different jobs.

Spring MVC module is based on the MVC design pattern. The main components involved are DispatcherServlet, Controller and Views. In Spring MVC DispatcherServlet plays very important role. It handles the user request and delegates it with Controller. Following diagram shows the very simplified architecture:



In this Spring MVC, DispatcherServlet works as the controller and it delegates the request to the Controller. A developer extends the abstract controller provided by the framework and writes the business logic there. The actual business related processing is done in the Controller.

* In other word: each of input form’s request (/hello.htm) must be sent to front controller DS first.
* Now the FC DS will delegate the form’s request to HandlerMapping of spring framework.
* The HandlerMapping will read spring configuration file [<servlet-name>-spring.xml] and will understand the mapping and based on input request and controller class against input URL request it will suggest the user-defined controller class name to FC DS.
* Now after identifying the user-defined controller name FC DS delegates the request and response object to user-defined controller class.
* In user-defined controller class there we have one handleRequest method which will process the request and will return the values in form of ModelAndView object to the FC DS. The handleRequest method is similar to service () method of servlet.
* In this object ModelAndView we combines/put collection map object and **viewName** of the success or failure page. So finally ModelAndView object will return the ViewName and Map object to FC DS.
* Now the FC DS will send this viewName to viewResolver and viewResolver will return the location along with extentation (.jsp/.velocity/.htm) of the view page to FC DS.
* Now finally FC DS will append the viewName with extentation (like success.jsp) and send the output as view page to the client. Here Success is the viewName and .jsp is the extentation.

Note: So here in spring MVC the three main components are (handlerMapping, user-defined Controller and ViewResolver).First handlerMapping will execute then controller class and finally viewResolver will be executed. By using handlerMapping we can find user-defined controller dynamically. The main default name is BeanNameUrlHandlerMapping class. We can handlerMapping and viewResolver as helper classes for controller class.

Handler Mapping Class:

1. **BeanNameUrlHandlerMapping**: This is default handler mapping class which read the spring configuration file and identify the user-defined controller class against action url name came from FC DS. We don’t need to configure this class in spring configuration file as we have done in case of below handler mapping classes.

|  |
| --- |
| **Syntax** : <bean name = “/urlpatter.ext” class=”Controller class name” />  <bean name = “/**hello.htm**” class=” **HelloController**” />  AS we can see that the there is direct mapping of action url to its controller class. So when the handler mapping will read spring.xml file then it will suggest the same controller class (HelloController) to FC DS to which the action url (/**hello.htm)** is mapped. In this case if handler mapping finds such configuration (direct mapping of action url with controller class) in spring.xml then by default it uses BeanNameUrlHandlerMapping class and we don’t need to configure BeanNameUrlHandlerMapping class in spring.xml file. Generally we used to map id with controller class but here they have used bean name instead of id. |

1. **ControllerClassNameUrlHanderMapping**: This handler mapping class identifies the user-defined controller class name by matching requested URL mapping name with user-defined controller class name. Here in this case we have to fallow naming conventions and request URL mapping name and user-defined controller class name should be same.

|  |
| --- |
| **<bean class=” ControllerClassNameUrlHanderMapping”/>**  **<bean class=”HelloController” />**  Here we have configured **ControllerClassNameUrlHanderMapping** class and user defined controller class **HelloController.** As soon as the handler mapping will this the above line in spring.xml then it will understand that the controller name has to find-out with request url mapping name. If hander mapping finds any controller class similar to requested url mapping came from FD DS then handler mapping will suggest the same controller class to FC DS. Here there is no any direct mapping of requested url patter with controller class. |

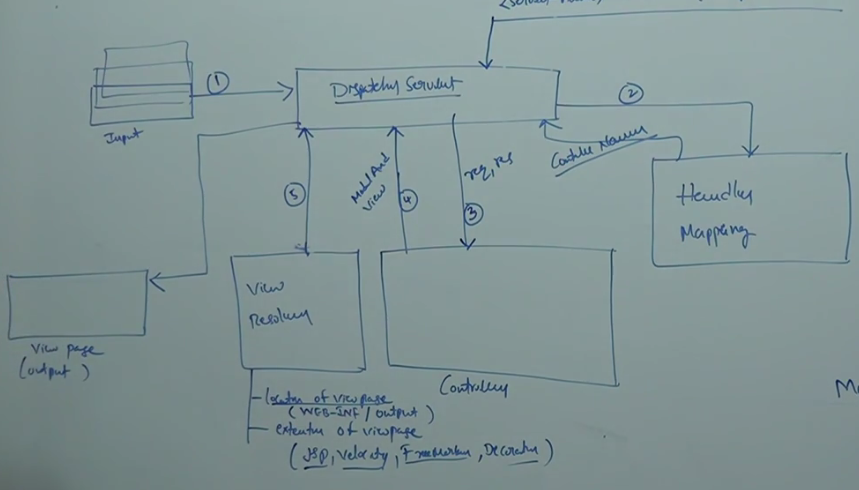
1. **SimpleUrlHandlerMapping:** This handler mapping class is used to map user-defined controller class bean id defined in spring configuration file. Like given below.

|  |  |
| --- | --- |
| **<bean class=”SimpleUrlHandlerMapping”>**  **<property name =”mappings”>**  **<props>**  **<prop key =”/hello.htm”> r <prop>**  **</props>**  **</property>**  **<bean id =”r” class=”HelloController” />** | **-**Where mapping is the property name (setter method) given by **SimpleUrlHandlerMapping class.**  **-** Here the request pattern**(/ hello.htm)** is mapped with r using <prop>  Finally the same r is mapped with controller class using bean id |
| **SimpleUrlHandlerMapping is recommended.** | |

**In spring application we can have multiple spring configuration file so should know how to configure spring configuration file in spring:**

**There are three ways to configure spring configuration file in spring.**

1. **<servlet-name>-spring.xml**
2. **Using init param and providing exact location of spring configuration file in web.xml**
3. **Using context param and providing exact location of spring configuration file in web.xml**



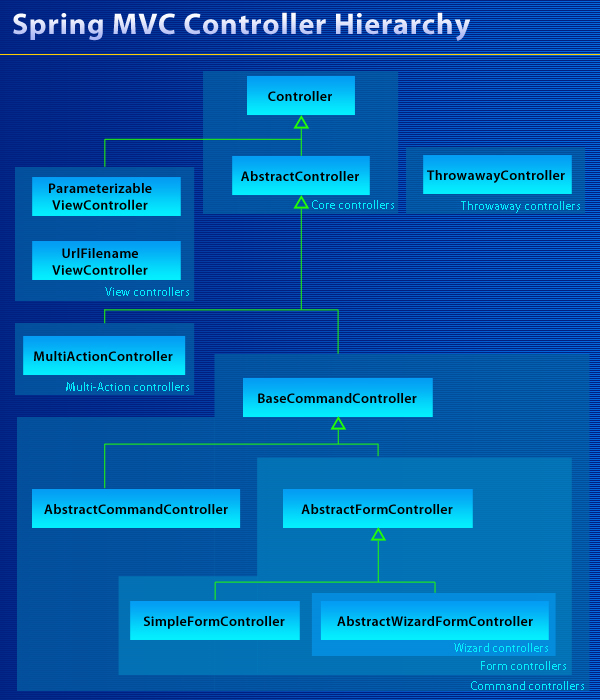
**HelloWorld Example In spring MVC by implementing Controller Interface :** Web.xml

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <web-appid=*"WebApp\_ID"*version=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>SpringMVC</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <!-- Default handlerMapping -->  <servlet>  <servlet-name>HelloWorld</servlet-name>  <servlet-class>org.springframeowrk.web.servlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>HelloWorld</servlet-name>  <url-pattern>\*.htm</url-pattern>  </servlet-mapping>  </web-app> |

Now we will create spring configuration file which will have name <servlet-name>-servlet.xml. Like HelloWorld-servlet.xml. In this xml file we will have configuration about default handler mapping class, user-defined class, InternalViewResolver for finding out the location and extentation of view page.

|  |  |
| --- | --- |
| **HelloWorld-servlet.xml**  <!DOCTYPE beans PUBLIC"-//SPRING//DTD BEAN 2.0//EN""http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <!-- Default Handler Mapping -->  <! -- Even no need to configure the below class in xml. Without configuring, we can map action url mapping with controller class-->  <beanclass=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*/>  <!-- user-defined controller class must be configured with ***name*** indicating action url -->  <bean ***name***=*"/helol.htm"* class=*"spring.controller.HelloController"*/>  <! -- View resolver configuration .Full qualified class name of view resolver. In this class we have two setter method setSuffix and setPrefix. By using we can pass the required location and by using suffix we can pass required page extentation-->  <beanclass=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  <!-- where "/" is for default location and ".jsp" is for page extentation -->  </bean>  </beans>  **Note: Here we have put the name of spring configuration file according to naming conventions <servlet-name>-servlet.xml, we cannot modify this name otherwise dispatcher servlet will not recognize this file and file not found exception will get.** | |
| **Index.jsp**  <h1>Hello World</h1>  <form action=*"./hello.htm"*>  Name:<input type=*"text"* name=*"name"*/>  <input type=*"submit"* value=*"sayHello"*/>  </form> | |
| **HelloWorld.java (user-defined controller class)**  **Package** spring.controller;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.Controller;  **public class** HelloController **implements** Controller {  @Override  **public** ModelAndView handleRequest(HttpServletRequest req , HttpServletResponse res) **throws** Exception{    String name=req.getParameter("name");  Map map=**new** HashMap();  map.put ("msg", "Hello ...."+name);  ModelAndView mav = **new** ModelAndView("success",map);  **Return** mav;  }  }  **Note:** Here we have created HelloController class by implementing Controller interface and must have to override one method handleRequest whose return type is ModelAndView. Here we have read name using (req.getParameter) and put the value into map object. Finally the map object (map) and success page we need to add to ModelAndView object (mav) and this **mav object we need to return to DispatcherServlet.** Finally based on the view name (success) DS will contact to viewResolver and will ask the InternalViewResolver for location and extentation for this view name (success). Finally based on location and extentation DS will the return the view page to the user. | |
| **Finally we need a success page (success.jsp)**  Here in success.jsp we just need to read key of map “msg” using expression as given below  ${msg} | |
|  | **Finally we get output as given below** |

**Controller’s hierarchy in Spring MVC:**



**Based on the above structure we have fallowing seven type of controller in the Spring MVC.**

|  |  |
| --- | --- |
| 1. **Core Controller** | 1. **Controller (I)** : contains one abstract method (AM) handleRequest() 2. **AbstractController (C):** contains AM handleRequestInternal |
| 1. **View Controller** | 1. **ParameterizableViewController(C)** 2. **UrlFilenameViewController (C)** |
| 1. **MultiActionController** | 1. MultiActionController**(C)** |
| 1. **ThrowawayController** | 1. ThrowawayController**(C)** |
| 1. **WizardControllers** | 1. AbstractWizardFormController**(C)** |
| 1. **FormController** | 1. AbstractFormController**(C)** 2. AbstractWizardFormController**(C)** 3. SimpleFormController**(C)** |
| 1. **CommandController** | 1. **All FormController class** 2. AbstractCommandController**(C)** 3. BaseCommandController**(C)** |

**Configuring spring configuration file using SimpleUrlHandlerMapping**

|  |
| --- |
| **HelloWorld-servlet.xml**  <!DOCTYPE beans PUBLIC"-//SPRING//DTD BEAN 2.0//EN""http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <!-- SimpleUrlHandlerMapping -->  <! – We need to configure SimpleUrlHandlerMapping inside bean-->  **<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*/>**  <!— the below mapping is properties type -- >  **<property name =”mappings”>**  **<props>**  **<prop key =”/hello.htm”>***sh* **<prop>**  **</props>**  **</property>**  <!-- user-defined controller class must be configured with ***id*** which is mapping controller class with action url -->  <bean ***id***=*"sh”* class=*"spring.controller.HelloController"*/>  <! -- View resolver configuration .Full qualified class name of view resolver. In this class we have two setter method setSuffix and setPrefix. By using we can pass the required location and by using suffix we can pass required page extentation-->  <beanclass=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  <!-- where "/" is for default location and ".jsp" is for page extentation -->  </bean>  </beans> |

**Note**: The remaining file (web.xml, index.jsp, success.jsp) will be same.

**Configuring spring configuration file using ControllerClassNameHanderMapping**

|  |
| --- |
| **HelloWorld-servlet.xml**  <!DOCTYPE beans PUBLIC"-//SPRING//DTD BEAN 2.0//EN""http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <!-- SimpleUrlHandlerMapping -->  <! – We need to configure SimpleUrlHandlerMapping inside bean-->  **<bean class=*"org.springframework.web.servlet.handler.ControllerClassNameHandlerMapping"*/>**  <! -- User-defined controller no need to have any name or id attributes. Here by fallowing naming convention handler mapping will suggest the controller class name.  The action url name and controller class must be similar e.g. action url name is hello and controller class name is also Hello -->  <bean class=*"spring.controller.HelloController"*/>  <! -- View resolver configuration .Full qualified class name of view resolver. In this class we have two setter method setSuffix and setPrefix. By using we can pass the required location and by using suffix we can pass required page extentation-->  <beanclass=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  <!-- where "/" is for default location and ".jsp" is for page extentation -->  </bean>  </beans> |

**Note**: The remaining file (web.xml, index.jsp, success.jsp) will be same.

**Customizing spring configuration file name using Init-param or context-param**

As we know that while configuring spring configuration file then we have to fallow one naming convention <servlet-name>-servlet.xml to put the name of file as well as we cannot change file location. If we try to change this naming convention the DS present inside web.xml, will not recognize the spring configuration file.

But we can change the naming convention and file location by using Init-parameter or context-parameter inside web.xml. So the corresponding web.xml will be given as.

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <web-appid=*"WebApp\_ID"*version=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>SpringMVC</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <servlet>  <servlet-name>HelloWorld</servlet-name>  <servlet-class>org.springframeowrk.web.servlet.DispatcherServlet</servlet-class>  **<!—Init Param for spring configuration file location -- >**  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/spring.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>HelloWorld</servlet-name>  <url-pattern>\*.htm</url-pattern>  </servlet-mapping>  </web-app> |
| **HERE**: IN THE ABOVE FILE by using init param we have changed the spring configuration file naming convention. The file name is spring.xml instead of HelloWorld-servlet.xml |

**Note**: The remaining file (spring.xml, index.jsp, success.jsp) will be same.

**Customizing spring configuration file name using context-param**

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <web-appid=*"WebApp\_ID"*version=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>SpringMVC</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <servlet>  <servlet-name>HelloWorld</servlet-name>  <servlet-class>org.springframeowrk.web.servlet.DispatcherServlet</servlet-class>  **<!—** context **Param for spring configuration file location -- >**  **<listener>** <listener-class>[org.springframework.web.context.ContextLoaderListener](https://stackoverflow.com/questions/6210757/java-lang-classnotfoundexception-org-springframework-web-context-contextloaderl)<listener-class> **</listener>**  <context-param>  <param-name>**contextConfigLocation**</param-name>  <param-value>**classpath**:spring.xml</param-value>  </context -param>  </servlet>  <servlet-mapping>  <servlet-name>HelloWorld</servlet-name>  <url-pattern>\*.htm</url-pattern>  </servlet-mapping>  </web-app> |
| Here in case of context-param to load spring.xml file form class path first we need to load listener class then using **contextConfigLocation** param name will load the spring.xml |

**HelloController class creation by extending AbstractController class:**

|  |
| --- |
| **HelloWorld.java (user-defined controller class)**  **Package** spring.controller;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.Controller;  **public class** HelloController **extends** **AbstractController** {  @Override  **public** ModelAndView **handleRequestInternal**(HttpServletRequest req , HttpServletResponse res) **throws** Exception{    String name=req.getParameter("name");  Map map=**new** HashMap();  map.put ("msg", "Hello ...."+name);  ModelAndView mav = **new** ModelAndView("success",map);  **Return** mav;  }  } |

1. Here instead of implementing controller interface we have extended an abstract class AbstractController class.
2. Here instead of overriding handleRequest method of controller interface we have to override handleRequestInternal method whose return type will be ModelAndView.
3. **Note**: Since Controller is the parent of AbstractController class so handleRequest method can be inherited in AbstractController class also. So we can use the same method handleRequest instead of handleRequestInternal.
4. Remaining file like web.xml , success.jsp , index.jsp and spring.xml will be same.

**HelloController class creation by extending ParameterizableViewController**

|  |
| --- |
| **HelloWorld.java (user-defined controller class)**  **Package** spring.controller;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.Controller;  **public class** HelloController **extends** **ParameterizableViewController** {  @Override  **public** ModelAndView **handleRequestInternal**(HttpServletRequest req , HttpServletResponse res) **throws** Exception{    String name=req.getParameter("name");  Map map=**new** HashMap();  map.put ("msg", "Hello ...."+name);  ModelAndView mav = **new** ModelAndView(getViewName(),map);  **Return** mav;  }  } |

1. Here instead of implementing controller interface we have extended an abstract class **ParameterizableViewController** class.
2. Here instead of overriding handleRequest method of controller interface we have to override handleRequestInternal method whose return type will be ModelAndView.
3. **Note**: Since Controller is the parent of AbstractController class so handleRequest method can be inherited in AbstractController class also. So we can use the same method handleRequest instead of handleRequestInternal.
4. ParameterizableViewController is the child class of Controller interface and AbstractController class. So it can have both the method handleRequest and handleRequestInternal.
5. **Note**: Here the only difference between AbstractController and ParameterizableViewControllerclass is that inModelAndView object no needs to append any static success name. We can get that success name from getter method **getViewName** () and this success name is passed from spring configuration xml file as shown below in the xml file.
6. Remaining file like web.xml, success.jsp, index.jsp and spring.xml will be same.

|  |
| --- |
| **HelloWorld-servlet.xml**  <!DOCTYPE beans PUBLIC"-//SPRING//DTD BEAN 2.0//EN""http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <!-- SimpleUrlHandlerMapping -->  <! – We need to configure SimpleUrlHandlerMapping inside bean-->  **<bean class=*"org.springframework.web.servlet.handler.ControllerClassNameHandlerMapping"*/>**  <bean class=*"spring.controller.HelloController"*>  <property name=*"viewName"* value=*"success"*/>  <!- viewName is the setter method of ParameterizableViewController class which is setting the  success name -- >  </bean>  <beanclass=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>  </beans> |

1. Note: in value we must need to pass because the setter method viewName internally having dependency-check. So setter-method values we must need to pass from the xml file.
2. So here we get dynamic style to get view page name.

**HelloController class creation by extending UrlFilenameViewController**

1. This class is just doing forward operation.
2. In case simple hyper link request if we want to map hyper link request to any form then we can use UrlFilenameViewController class.
3. In this class spring developer have implemented method already so we will not override any method we just need to use this class. In this class they have implemented handleRequestInternal method whose job is to read url action by request parameter and the same url action name it will append to the ModelAndView object without any extentation (like .htm).
4. In other if we submit the URL action hello.htm then it will append hello in ModelAndView object and same name it delegate to DS and further the viewResolver will try to redirect hello.jsp page.

**Steps to redirect jsp page using UrlFilenameViewController**

1. Create one index.jsp having two hyperlinks (FirstPage and SecondPage).
2. Create two jsp page FirstPage.jsp and SecondPage.jsp
3. Create one web.xml with servlet-name and DispatcherServlet class configuration.
4. Create one spring configuration file <servlet-name>-servlet.xml. In this xml file configure the action url name with **UrlFilenameViewController** class inside <bean> tag

|  |  |
| --- | --- |
| **Index.jsp**  <h1>Hello World</h1>  <pre>  <a href =”./first.htm”> First Page </a>  <a href =”./second.htm”> Second Page</a>  </pre>  Note: Here instead of first.htm we can write first.jsp also but when it redirect to first.jsp page the whole name (first.js) will be displayed on the browser url that is why for security purpose we used to write some different extentation in action url. | |
| First.jsp  <h1>First Page</h1>  <pre>  Hi Welcome to first page.  </pre> | Second.jsp  <h1>Second Page</h1>  <pre>  Hi Welcome to second page.  </pre> |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <web-appid=*"WebApp\_ID"*version=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>  <display-name>SpringMVC</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <!-- Default handlerMapping -->  <servlet>  <servlet-name>Forward</servlet-name>  <servlet-class>org.springframeowrk.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name> Forward</servlet-name>  <url-pattern>\*.htm</url-pattern>  </servlet-mapping>  </web-app> | |

As in the web.xml file the servlet name is Forward then now we will create one spring configuration file with name [Forward-spring.xml]. This xml file will have configuration about action URL of hyperlinks and **UrlFilenameViewController** inside <bean> tag. And this file will have configuration about the location and extentation of view page.

|  |
| --- |
| **Forward -servlet.xml**  <!DOCTYPE beans PUBLIC"-//SPRING//DTD BEAN 2.0//EN""http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <!-- *UrlFilenameViewController* -->  <! – We need to configure hyperlinks url name with *UrlFilenameViewController* class inside bean tag-->  <bean name=”/first.htm” class=*"org.springframework.web.servlet.mvc.UrlFilenameViewController"/*>  <bean name=”/second.htm”class=*"org.springframework.web.servlet.mvc.UrlFilenameViewController"/*>  <beanclass=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>  </beans> |

Note: when container will read this file then it will return first as view name to DS and dispatcher servlet will delegate this name to InternalViewResolver and InternalViewResolver will append .jsp with first and return first.jsp to DS and finally DS will return this page to user. Same this happen for second also.

Note: **AbstractController is an abstract class which is implemented by implementing Controller interface. Apart from this every controller class extends AbstractController class and can overrides its abstract method handleRequest ().In other words every controller class is the child of AbstractController class.**

**Example Pending:**

**ThrowawayController:**

**MultiActionController**:

1. It is also a child of AbstractController class.
2. In case of MultiActionController we should not override handleRequest () & handleRequestInternal () method. Only user-defined method we should write over here and method signature should be similar to handleRequest method only. So return type will be ModelAndView.
3. The user-defined method name should be same as action url of the form.
4. This MultiActionController class is useful to handle multiple form action operations.

**Use case:**

Example: Let’s say we have two action form

1. Employee Register form : Having action url (./empreg.htm)
2. 2- Employee Update form: Having action url (./empudate.htm)

Now for these two forms if we want to handle operation from only one single controller then we can write our own user-defined controller by extending MultiActionController class with handleRequest method signature whose return type would be ModelAndView.

1. Create one user-defined controller class EmpCurdController by extending MultiActionController.
2. Create two user-defined methods with handleRequest method signature whose name should be same as form’s action URL.
3. Public ModelAndView empsave (req, res) method for employee register.
4. Public ModelAndView empupdate (req, res) method for employee update.

**Q:** Since here we are not override handleRequest and handleRequestInternal method and here we are writing user-defined method then the question arises that who will execute these methods.

**A:** Whenever we submit the request from the browser the request will go to DispatcherServlet and DS will call handleRequest method internally inside user-defined controller class **EmpCurdController** further the handleRequest method will call handleRequestInternal method. This handleRequestInternal method will read our form’s action name. So in case of employee register form request it will call empsave method as employee register form’s action name is empsave and in case of Employee update form request it will call empupdate method as the employee update form’s action name is empupdate.

**Note: If we don’t want to fallow this default naming convention i.e. if we don’t want to write method name as form’s action name then we need to use MethodNameResolver. By using MethodNameResolver we can change method name.**

**Example:**

|  |  |
| --- | --- |
|  | 1. Create emp\_reg.jsp and emp\_update.jsp. 2. Create web.xml and configure DS and url-pattern. 3. Create **MultiActionController-servlet.xml** and configure user-defined controller using *SimpleUrlHandlerMapping class.* 4. Now create user-defined controller class and inside this class create user-defined method. The method name will be exactly same as defined action URL pattern in emp\_reg.jsp and emp\_update.jsp. |
| <h1>Employee Registration</h1>  <form action=*"./empreg.htm"*>  <pre>  ID: <input type=*"text"* name=*"id"*/>  Name:<input type=*"text"* name=*"name"*/>  Email:<input type=*"text"* name=*"email"*/>  Address: <textarea rows=*"5"* cols=*"15"*  name=*"address"*></textarea>  <input type=*"submit"* name=*"save"*/>  </pre>  </form> | <h1>Employee Update</h1>  <form action=*"./empupdate.htm"*>  <pre>  ID:<input type=*"text"* name=*"id"*/>  Name:<input type=*"text"* name=*"name"*/>  Email:<input type=*"text"*  name=*"email"*/>  Address: <input type=*"text"*  name=*"address"*/>  <input type=*"submit"* name=*"update"*/>  </pre>  </form> |

|  |
| --- |
| **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"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>  <display-name>MultiActionController</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  </welcome-file-list>  <servlet>  <servlet-name>MultiActionController</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>MultiActionController</servlet-name>  <url-pattern>\*.htm</url-pattern>  </servlet-mapping>  </web-app>  **Note: This is just simple web.xml with DispatcherServlet and action url configuration.** |
| **MultiActionController-servlet.xml**  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <bean  class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>  <property name=*"mapping"*>  <props>  <prop key=*"/empreg.htm"*>Empcrud</prop>  <prop key=*"/empupdate.htm"*>Empcrud</prop>  </props>  </property>  </bean>  <bean id=*"*Empcrud*"* class=*"emp.controller.EmpCurdMAController"*/>  <bean class=*"org.springframework.web.servlet.view.InternalViewResolver"*>  <property name=*"prefix"* value=*"/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>  </beans>  **Note:** |

|  |
| --- |
| **package** emp.controller;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.multiaction.MultiActionController;  **public** **class** EmpCurdMAController **extends** MultiActionController{  // Employee registration operation  **public** ModelAndView empreg(HttpServletRequest req, HttpServletResponse res) **throws** Exception {  ///String ID = req.getParameter("id");  String name = req.getParameter("name");  String email = req.getParameter("email");  String adderss = req.getParameter("address");  // Loading the oracle driver  Class.*forName*("oracle.jdbc.OracleDriver");  // Get connection object  Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");  // creating auto id  ResultSet rs = con.createStatement().executeQuery("select max(id) from emp");  **int** maxid=0;  **if**(rs.next()){  maxid=rs.getInt(1);  maxid++;  }  PreparedStatement ps=con.prepareStatement ("insert into emp values (?,?,?,?)");  ps.setInt(1, maxid);  ps.setString(2, name);  ps.setString(3, email);  ps.setString(4, adderss);  **int** i = ps.executeUpdate();  con.close();  ModelAndView mav= **null**;  **if**(i!=0){  mav = **new** ModelAndView("success");  }**else**{  mav=**new** ModelAndView("fail");  }  **return** mav;  }  // Employee update operation  **public** ModelAndView empupdate(HttpServletRequest req, HttpServletResponse res) **throws** Exception {  **int** ID = Integer.*parseInt*(req.getParameter("id"));  String name = req.getParameter("name");  String email = req.getParameter("email");  String adderss = req.getParameter("address");  Class.*forName*("oracle.jdbc.OracleDriver");  // Get connection object  Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");  PreparedStatement ps = con.prepareStatement ("update emp set name =?, email =? , address = ? where id =?");    ps.setString(1, name);  ps.setString(2, email);  ps.setString(3, adderss);  ps.setInt(4, ID);  **int** i = ps.executeUpdate();  con.close();  ModelAndView mav= **null**;  **if**(i!=0){  mav = **new** ModelAndView("success");  }**else**{  mav=**new** ModelAndView("fail");  }  **return** mav;  }  } |