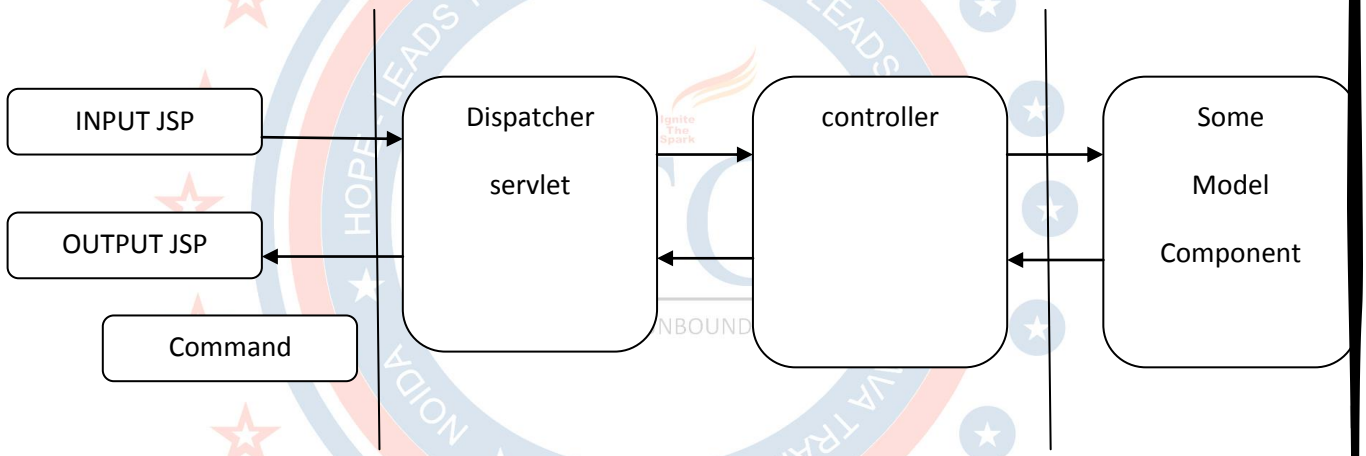
Spring MVC:

- Spring MVC is web Framework like struts1, struts2 and JSF.
- Spring MVC is used to develop web applications easily and quickly with less maintenance.
- Spring MVC is covers web Layer which includes Presentation layer and Controller layer.
- Spring MVC is implemented based on:
  - MVC Architecture.
  - Front controller Design Pattern.
  - Servlets and JSP.

Framework	FrontController
Struts 1	Action Servlet
Struts 2	FilterDispatcher
JSF	FacesServlet
Spring MVC	DispatcherServlet

Spring MVC Basic Architecture



- You should configure DispatcherServlet in web.xml as follows:

```
<Servlet>
        <Servlet-name>jtcindia</Servlet-name>
        <Servlet-class>org.springframework.web.servlet.DispatcherServlet</Servlet-class>
        <load-on-startup>1</load-on-startup>
</Servlet>
<Servlet-mapping>
        <Servlet-name>jtcindia</Servlet-name>
        <url-pattern>*jtc</url-pattern>
</Servlet-mapping>
```

- At web container start up,
  - DispatcherServlet will be loaded, instansiated and initialized by calling init() method.
  - Init() of DispatcherServlet will try to identify the spring configuration Document with the following naming convensions.

Servlet name-servlet.xml
Ex:
Jtcindia-servlet.xml

- DispatcherServlet creates the Application context (XmlWebApplicationContext) object by reading all the beans specified in the spring configuration Document.
- If spring configuration Document is not found with the given naming convecsions then ApplicationContext object will not be created by the DispatcherServlet.

Ex:

```
Public class DispactcherServlet extends HttpServlet{
ApplicationContext ctx=null;
Aublic void init(ServletConfig cfg){

//1.try to get the spring Configuration Document with the default naming convensions
String xml=”jtcindia”+”Servlet.xml

//if found then creates the ApplicationContext object
Ctx=new Xml webApplicationContext(xml);
}
….
}
```

After Deplying and starting spring MVC based web Application, Following tasks will happen at web container start up

1. Web container loads, creates and initializes DispatcherServlet by calling init () method.
2. DispatcherServlet’s init () performs the following:
    - Identifies spring Configuration Document file.
    - Spring Container instance will be created by reading all the beans from identified spring configuration Document.
    - Initializes DispatcherServlet Servlet With Spring Container instances.

Following tasks will happen at web container shutdown time

1. Web container calls destroy () method of DispatcherServlet.
2. Destroy() method of DispatcherServlet destroys the Spring container instance

- You can also create the ApplicationContext object with the following configuration in web.xml

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/jtcindia.xml</param-value>
</context-param>
<listener>
```

<listener-class> org.springframework.web.context.contextLoaderListener</listener-class>
</listener>
Ex:

```
Class contextLoaderListener implements servletContextListener{
Public void contextInitialized (servletContextEvent){
        servletContext sc=e.getServletContext();
        string xml=sc.getlnitParameter("contextConfigLocation");
        if(xml!=null){
        ctx=new xmlWebApplicationContext(xml);
        //Stotes the ApplicationContext in servletContext.
        }
}
….
}
```

First spring MVC Example with myEclipse:

1. Create the web project as follows;
   a. Select File > New > web Project
   b. Provide project name: Jtc64 and click on Finish button.
2. Add Spring capabilities as follows:
   a. Select the project and Right click
   b. Select myEclipse > Add spring capabilities
   c. Select the following:
      I. Spring version spring 3.o
      II. Spring 3.0 AOP Libraries
      III. Spring 3.0 core Libraries
      IV. Spring 3.0 persistence core Libraries
      V. Spring 3.0 J2EE Libraries
      VI. Spring 3.0 web Libraries
   d. Click on Next.
   e. Provide spring Configuration Document file name as jtcindia Servlet.xml
   f. Click on Finish button.
3. Move the spring Configuration Document (jtcindia-servlet.xml) from src folder to web Root/WEB-INF folder.
4. Add DispatcherServlet Configuration in web. Xml.
5. Design the JSPs required for your application requirement and places them in web Root folder.
6. Write the required controller classes, command classes etc and place them in the required packages.
7. Configure the required beans in the spring configuration Document.
8. Deploy and RUN.

Jtc64: Files required

Index.jsp	Show.jsp
courseController.java	courseService.java
Web.xml	Jtcindia-servlet.xml

Index.jsp

```
%@taglib prefix="c" uri="http:// java.sun.com.jsp
/jstl/core"%
<html><body>
<br><h1>java Training Center<br/>
<a href="<c:url value="course.jtc"/>">show JTC
Courses</a> </h1>
</body></html>
```

Show.jsp

```
<%@ taglib prefix="jtc" url=http://java.sun.com/
jsp/jstl/core%>
<h2>course information:</h2>
<ul><jtc:forEach var="cou"items="${AL}">
<li><h1>${cou}</h1>
</jtc:forEach> </ul>
```

Course Controller.java

```
Package com.jtcindia.spring mvc;
Import java.util.ArrayList;
Import javax.servlet.http.*;
Import org.springframework.beans factory.
Annotation. Autowired;
Import
org.springframework.web.servlet.ModelAndView;
Import org.springframework.web.servlet.mvc.
AbstractController;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class courseController extends
AbstractController
{
@Autowired
Private CourseService cs;
Protected ModelAndView
handleRequestlnternal(HttpServletRequest request,
HttpServletResponse response) throws Exception {
ArrayList<string> al=cs.getCources();
return new modelAndView("show","AL","al);
}
```

CourseService.java

```
Package com.jtcindia.spring.mvc;
Import java.utilArrayList;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class CourseService {
Public ArrayList<string> getCourses(){
ArrayList<string> al=new ArrayList<string>();
Al.add("JAVA");
Al.add("JDBC");
Al.add("servlets");
Al.add("jsp");
Al.add("struts");
Al.add("JSF");
Return al;
}
```

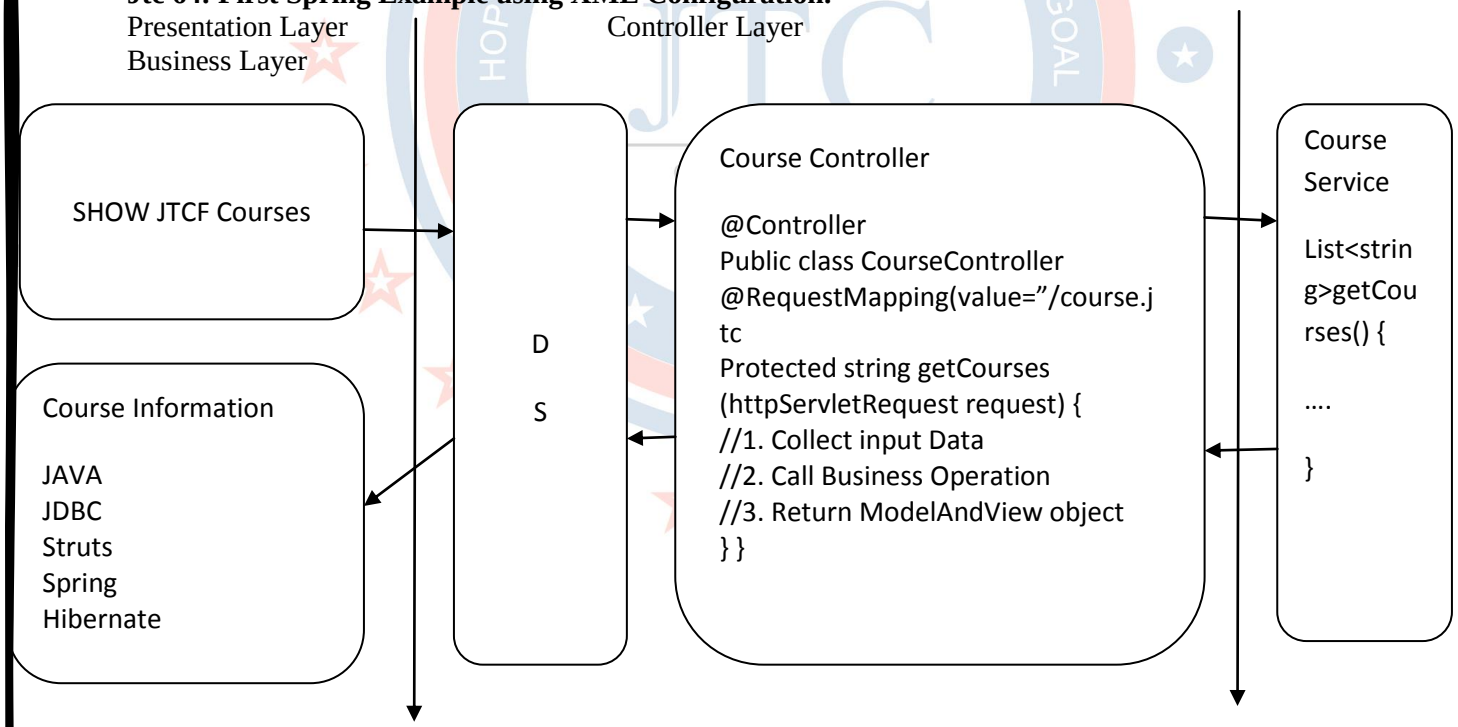
Web.xml

```
<web-app….>
<servet>
<Servlet-name>jtcindia</Servlet-name>
<Servlet-class>org.springframework.web.servlet.
DispatcherServlet</Servlet-class>
<load-on-startup>1</load-on-startup>
</Servlet>
<Servlet-mapping>
<Servlet-name>jtcindia</Servlet-name>
<url-pattern>*.jtc</url-pattern>
</Servlet-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Jtcindia-servlet.xml

```
<beans xmlns=http://www.springframework.org/schema/beans
Xmlns:xsi=http://www.w.3.org/2001/XMLSchema-instance
Xmlns:p=http://www.springframework.org/schema/p
Xmlns:context=http://www.springframework.org/schema/context
Xsi;schemaLocation=”http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring.beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd”>
<context:annotation-config/>
<bean
        Class=”org.springframework.web.servlet.view.lnternalResourceViewResolver”>
        <property name=”viewClass”
                Value=”org.springframework.web.servlet.view.internalResourceView”/>
         <property name=”prefix” value”/”/>
         <property name=”suffix” value=”jsp”/>
</bean>
        <bean name=”/course.jtc” class=”com.jtcindia.spring.muc.Coursecontroller”/>
        <bean id=”cs” class=”com.jtcindia.spring.mvc.courseService”/>
</beans>
```

Jtc 64: First Spring Example using XML Configuration.



Jtc65: Files required

Index.jsp	Show.jsp
courseController.java	courseService.java
Web.xml	Jtcindia-servlet.xml

CourseController.java

```
Package com.jtcindia.spring.mvc;
Import java.util.*;
Import javax.servlet.http.*;
Import org.springframework.beans.factory.
annotation.Autowired;
Import org.springframework.stereotype.controller;
Import org.sprinframework.web.bind.annotation.
requestMapping;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
```

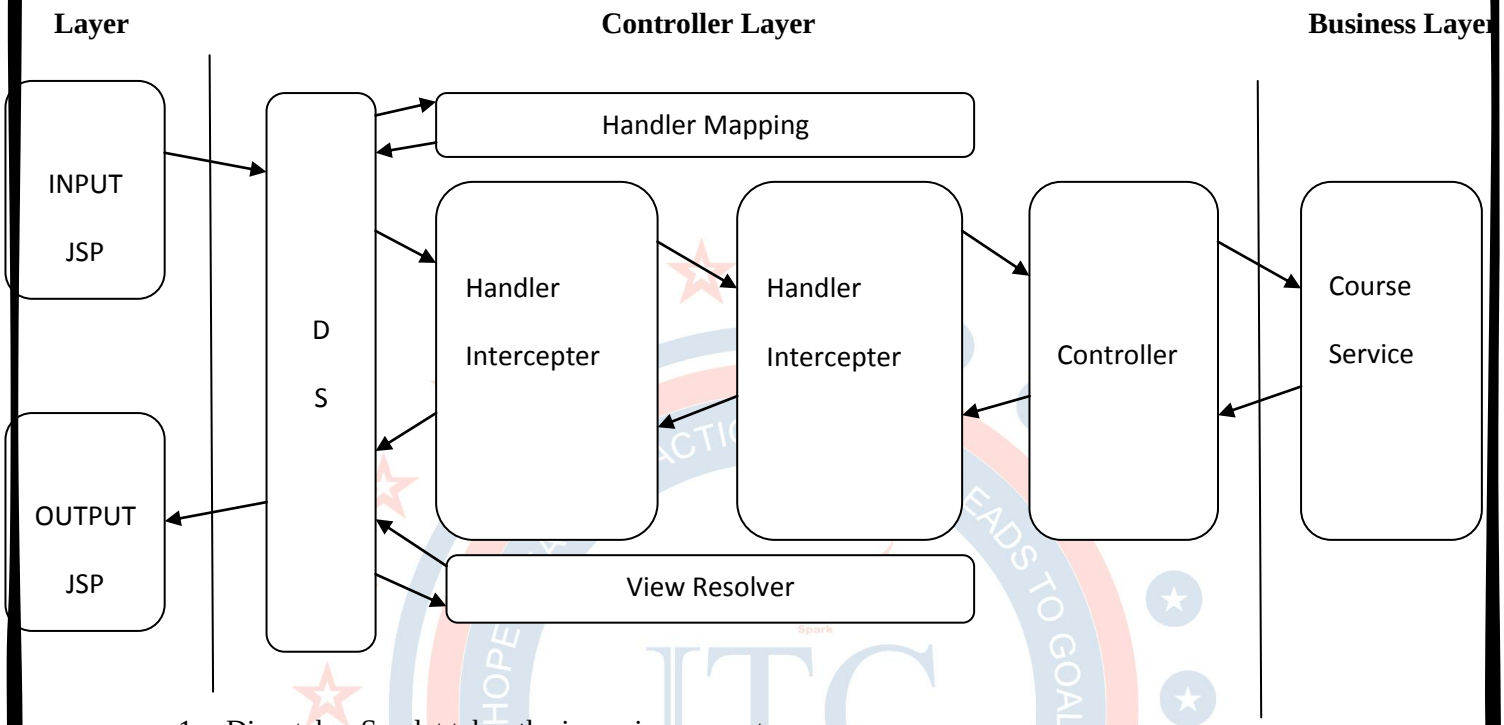
```
@Controller
Public class CourseController {
@Autowired
Private CourseService cs;

@RequestMapping(value="/course.jtc")
Protected string getCourses(HttpServletRequest
Request) throws Exception {
ArrayList<String> al= cs.getCourses();
Request.setAttribute("AL",al);
Return "show";
}
}
```

Jtcindia-servlet.xml

```
<beans xmlns=http://www.springframework.org/schema/beans
Xmlns:xsi=http://www.w.3.org/2001/XMLSchema-instance
Xmlns:p=http://www.springframework.org/schema/p
Xmlns:context=http://www.springframework.org/schema/context
Xsi;schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring.beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:annotation-config/>
<bean
        Class="org.springframework.web.servlet.view.lnternalResourceViewResolver">
        <property name="viewClass"
                Value="org.springframework.web.servlet.view.internalResourceView"/>
         <property name="prefix" value"/"/>
         <property name="suffix" value="jsp"/>
</bean>
        <bean id="cs" class="com.jtcindia.spring.mvc.courseService"/>
</beans>
```

Spring MVC Baisc Flow



1. Dispatcher Servlet takes the incoming request.
2. Dispatcher Servlet contacts the Handle Mapping with incoming request URL (/course.jtc)
3. Handler Mapping identifies and returns the Controller (CourseController) specified for the request URI (/course.jtc);
4. Dispatcher servlet identifies and invokes one or more Handler interceptors reqistered with spring container if any then DispactcherServlet invokes the controller
   handleRequestInternal() method of CourseController -with XML
   getCourse() method of CourseController -with Annotation
5. After finishing contropller method exection,all the registered Handler interceptors will be called in the reverse order one by one.
   finally DispatcherServlet gets ModelAndView class object (which contains view logical name and model data).
   Finally DispatcherServlet gets the view logical name.
6. Dispatcher Servlet contacts the viewResolver with the view logical name (show).
7. DispatcherServlet gets the view (/show.jsp) from viewResolver.
8. DispatcherServlet forwards the identified view (/show.jsp) to the client.

Jtc 66: Login Example using spring MVC with XML Configuration.
Jtc 66: Files required

Index.jsp	login.jsp
home.java	LoginController.java
User.java	UserValidator.java
Messages.properties	Web.xml

Jtcindia-servlet.xml	

Index.jsp

```
%@taglib prefix=”c” uri=”http:// java.sun.com.jsp
/jstl/core”%
<html><body>
<br><h1>java Training Center<br/>
<a href=”<c:url value=”course.jtc”/>”>show JTC
Courses</a>
</h1></body></html>
```

login.jsp

```
<%@ taglib prefix=”from” uri=http://www.spring
framework.org/tags/form%>
<html><body><center>
<h1>User Account Login</h1>
<form:form method=”post” commandName=”user”>
<table>
<tr><td>Usernate</td>
<td><from:input path=”username”/></td>
<td><font color=red size=5>
<form:errors path=”username”></font></td></tr>
<tr><td>password</td>
<td><form:password</td>
<td><form:password path=”password”/></td>
<td><font color=red size=5>
<form:errors path=”password”/></font></td></tr>
</table>
<br> <input type=”submit” value=”Account Login”>
</form:form>
</center></body></html>
```

home.java

```
<%@ taglib prefix=”c”
uri=http://java.sun.com/jsp/jstl/core%>
<html><body>
<h1> Hi ${user.username}; your Login Successful</h1>
<h1> this is you Home page</h1>
</body></html>
```

LoginController.java

```
Package com.jtcindia.spring.mvc;
Import org.springframework.validation.BindException;
Import org.springframework.web.servlet.mvc.simple form
Controller;
Import org.springframework.web.servlet.ModelAndView;
Import javax.servlet.servletException;
Import javax.servlet.http.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class LoginController ectends
simpleFormController {
Public modelAndView onSubmit(Object command,
BindException errors) throws servletException {
System.out.println(“in onSubmit()”);
User user=(user) command;
String un=user.getUsername();
String pw=user.getPassword();
String  view=””;
If(un.equals(uw)){
View=getSuccessView();
}else{
View=getFormView();
}
Return new modelAndView (view,”user”,user);
}
Public object formBacking object (HttpServletRequest
request) throws servletException {
System.out.println(“in formBackingObject()”);
User user=new user();
User.setUsername(“Som Prakash”);
Return user;
}  }
```

User.java

```
Package com.jtcindia.spring.mvc;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class user {
Private string username;
Private string password;
// setters and getters
}
```

UserValidator.java

```
Package com.jtcindia.spring.mva;
Import org.springframework.validation.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class userValidator implements validator {
Public Boolean supports(class clazz) {
Return user.class.equals(clazz);
}
Public void validate(object command, Errors errors) {
User user=(user) command;
If(user.getUsername()== null||
User.getUsername().length()==0) {
Errors.rejectValue("username",
"error.username.required", null,"Username required.");
}
If(user.getPassword() == null ||
User.getPassword().length()==0) {
Errors.rejectValue("password",
"error.password.required",null,"password required.");
}
}
```

Messages.properties

```
Error.username.required=username is mandatory
Error.password.required= password is mandatory
```

Jtcindia-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans….>
<context:annotation-config/>
<bean
class="org.springframework.web.servlet.view.internal
ResourceViewResolver">
<property name="viewClass"
value="org.springframework.web.servloet.view.internal
ResourceView"/>
<property name="prefix" value="/"/>
<property name="suffix" value="jsp"/>
</bean>

<bean name="/login.jtc"
Class="com.jtcindia.spring.mvc.loginController">
<property name='sessionForm" value="true"/>
<property name="commandName" value="user"/>
<property name="commandclass"
Value="com.jtcindia.spring.mvc.user"/>
<property name="validator">
<bean class="com.jtcindia.spring.mvc.userValidator"/>
</property>
<property name="formView" value="login"/>
<property name="successView" value="home"/>
</bean>
<bean id="messageSource" class="org.springframework.
context.support.ResourceBundleMessageSource">
<property name="basename" value="messages"/>
</bean>
</beans>
```

Spring MVC Request processing Flow (with XML);

When you request for resource (JSP or View), following tasks will happen:

1. Dispatcher Servlet takes the incoming request.
2. Dispatcher Servlet contacts the Handle Mapping with incoming request URL (/course.jtc)
3. Handler Mapping returns the Controller (LoginController) specified for the request URI (/course.jtc);
4. DispatcherServlet gets the formView property value (login)
   string fv=lc.getFormView();
5. DispatcherServlet contacts the view Resolver with the view logical name (login).
6. DispatcherServlet gets the view (/login.jsp) from view Resolver.
7. DispatcherServlet Checks whether identified view (/login.jsp) contains <form:form> tag or not.
8. If identified View (/login.jsp) does not contain <form:form> tag then that view will be forwarded to client.
9. If identified view (/login.jsp) contains <form:form> tag then DS invokes the controller method object obj=lc.formBackingObject(request);
10. DispatcherServlet gets the command object returned by formBacking object() method
11. DispatcherServlet gets data from command object by calling getter methods on command object and populates that into the correnponding fields of the jsp form.
12. DispatcherServlet Forwards the identified view (/login.jsp) to the client.

When you send the request by submitting form, following tasks will happen:

1. DispatcherServlet takes the incoming request.
2. DispatcherServlet contacts the Handler mapping with incoming request URI (/login.jtc)
3. Handler Mapping returns the controller (LoginController) specified for the request URI (/login.jtc)
4. DispatcherServlet popuplates client submitted data.
5. DispatcherServlet popublates client submitted data into command object by calling setter methods.
6. DispatcherServlet creates the Errors object.
7. DispatcherServlet invokes the validate () methods by passing command object and errors object.
   uv.supports(user.class);
   uv.validate(command,errors);
8. DispatcherServlet checks whether errors object contains any errors.
9. If errors found the DispatcherServlet gets the form View property value (login) and Gees to STEP 12.
10. If no errors found the DispatcherServlet invokes the controller method
    a. modelAndView mav=cl.onSubmit (command);
11. DispatcherServlet get the model And View class object (which contains view logical name and model data).
12. DispatcherServlet contacts the viewResolver with the view logical name.(login from STEP 9 or home from STEP 11)
13. DispatcherServlet gets the view (/home.jsp or /login.jsp)
14. DispatcherServlet forwards the identified view (/home.jsp or /login.jsp) to the client.

Jtc 67: Login Example using spring MVC with XML Configuration.
Jtc 67: Files required

Index.jsp	login.jsp
home.java	LoginController.java
User.java	UserValidator.java
Messages.properties	Web.xml
Jtcindia-servlet.xml	

Login.jsp

```
<%@ taglib prefix="form" uri=http://www.springframework.org/tags/form%>

<html><body><center>
<h1>User Account Login</h1>
<form:form action="verifyUser.jtc"method="post"
commandName="user">
//Same as Jtc67
</center></body></html>
```

LoginController.java

```
Package com.jtcindia.spring.mvc;
Import java.util.map;
Import org.springframework.beans.factory.
annotation.Autowired;
Import org.springframework.stereotype.controller;
Import
org.springframework.validation.BindingResoult;
Import
org.springframework.web.bind.annotation.*;
Import javax.servlet.servletException;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
@Controller
@sessionAttributes
Public class Login Controller {
@Autowired
Private userValidator userValidator;

@RequestMapping(value="/verifyUser",method=
requestMethod.POST)
Public string verifyUser@ModelAttribute("user")
User user, Binding Result result) {
System.out.println("verifyUser()");
userValidator.validate(user, result);
if(result.hasErrors()) {
system.out.println(result.getErrorCount());
return "login";
}
// User user=(user) command;
String un=user.getusername();
String pw=user.getPassword();
System.out.println(un);
System.out.println(pw);
If (!un.equals(pw)) {
Return "login";
}
Return "home";
}

@RequestMapping("/login")
Public string showLoginForm(Map model) throws
servletException {
system.out.println("showLoginForm");
User user=new User();
User.setUsername("Som Prakash");
Model.put("user",user);
Return "login";
}
}
```

Jtcindia-servlet.xml

```
<beans xmlns=http://www.springframework.org/schema/beans
```

```
Xmlns:xsi=http://www.w.3.org/2001/XMLSchema-instance
Xmlns:p=http://www.springframework.org/schema/p
Xmlns:context=http://www.springframework.org/schema/context
Xsi;schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring.beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:annotation-config/>
<bean
        Class="org.springframework.web.servlet.view.lnternalResourceViewResolver">
        <property name="viewClass"
                Value="org.springframework.web.servlet.view.internalResourceView"/>
         <property name="prefix" value"/"/>
         <property name="suffix" value="jsp"/>
</bean>
        <bean name="/course.jtc" class="com.jtcindia.spring.muc.Coursecontroller"/>
        <bean id="cs" class="com.jtcindia.spring.mvc.courseService"/>
</beans>
```

Spring MVC Request processing Flow (with Annotation)

When you request for resource (jsp or view), following tasks will happen:

1. Dispatcher Servlet takes the incoming request.
2. Dispatcher Servlet contacts the Handle Mapping with incoming request URL (/course.jtc)
3. Handler Mapping returns the Controller (LoginController) and corresponding method of the controller with the specified for the request URI (/course.jtc);
4. Dispatcher servlet creates the Map object.
   a. Map model=new HashMap();
5. DispatcherServlet invokes the controller method
   a. String vn=lc.showLoginForm (model);
6. Dispatcher Servlet gets the view Logical Name (login) from controller method.
7. Dispatcher Servlet contacts the viewResolver with the view logical name (login).
8. DispatcherServlet gets the view (/login.jsp) from View Resolver.
9. DispatcherServlet checks whither identified view (/login.jsp) contains <form :form> tag or not.
10. If identified view (/login.jsp) does not contain <form:form> tag then that view will be forwarded to client.
11. If identified view (/login.jsp) contains <form:form> tag then DS Collects the model Attribute (which is the Command object) from the Map.
    User user=(user) model.get("user");
12. DispatcherServlet gets data from command object by calling getter methods on command object and populates that into the correnponding fields of the jsp form.
13. Dispatcher Servlet forwards the identified view (/login.jsp) to the Client.

When you send the request by submitting form, following tasks will happen:

1. DispatcherServlet takes the incoming request.
2. DispatcherServlet contacts the Handler mapping with incoming request URI (/verifyUser.jtc)
3. Handler mapping returns the controller (loginController) aand corresponding method of the controller with the specified for the request URI (/verifyUser.jtc)
4. DispatchderSergvlet Collects the model Attribute (Which is the command object) from the Map.
   a. user user=(user) model.get("user");
5. Dispatcher Servlet collects client submitted data.
6. Dispatcher Servlet popuplates client submitted data into command object by calling setter methods.
7. Dispatcher Servlet creates the BindingResult object.
8. DispatcherServlet invokes the Controller method (verifyUser()) by passing command and BindingResult objects.
   a. string vn=Ic.verifyUser (command, results);
9. Dispatcher Servlet gets the string object which is the view logical name.
10. Dispatcher Servlet contacts the view Resolver with the view logical name (login or home).
11. Dispatcher Servlet gets the view (/home.jsp or /login.jsp)
12. Dispatcher Servlet forwards the identified view (/home.jsp or /login.jsp) to the client.

Jtc 68: Registration Example using spring MVC with XML Configuration.
Jtc 68: Files required

Index.jsp	login.jsp
home.java	RegisterController.java
Student.java	StudentValidator.java
Messages.properties	Web.xml
Jtcindia-servlet.xml	

Index.jsp

```
<%@ taglib prefix="c" uri=http://java.sun.com/jsp
/jstl/core"%>
<html><body>
<br><h1> java Training center<br/>
<a href="<c:url value="register.jtc"/>">New
Student Registration</a></h1>
</body></html>
```

Home.jsp

```
<%@ taglib prefix="c" uri=http://java.sun.com/jsp
/jstl/core"%>
<html><body>
<h1>Hi${student.sname}; Your Registration
Successful</h1>
<h1>this is your Home page</h1>
</body></html>
```

```
Public modelAndView on Submit (object
Command )throws servletException {
Student stu=(student) command;
System.out.println(stu.getSid());
System.out.println(stu.getSname());
System.out.println(stu.getEmail());
System.out.println(stu.getphone());
System.out.println(stu.getGender ());
System.out.println(stu.getQualification());
String tim[]=stu.getTimings();
For(int i=0;i<tim.length;++) {
System.out.println(tim[i]);
}
System.out.println(stu.getRemarks());
String view=getSuccessView();
Return new modelAndView(view,"student",stu);
}
Protected object
```

RegisterController.java

```
Package com.jtcindia.spring.mvc;
Import
org.sprigframework.web.servlet.mvc.simpleFolrm
controller;
Import org.springframework.web.servlet.Model
and View;
Import javax.servlet.servletException;
Import javax.servlet.http.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class RegisterController extends
Simple form controller {
```

```
formBackingObject(HttpServletRequest request)
throws servletException {
student stu=new student();
stu.setSid("JTC-99");
return stu;
}
}
```

Register.jsp

```
<%@ taglib prefix="form" uri=http://www.springframework.org/tags/form%>
<html><body><center>
<form:form method="post" commandname="student">
<table>
<tr><td align="center" colspan="3"> Student Registration Form</td></tr>
<tr><td><p > Student id:</td>
<td><form:input path="sid"/></td>
<td><form:errors path="sid"/></td></tr>
<tr><td>Name:</td>
<td><form:input path="sname"/></td>
<td><fornt color=red size=4><form:errors path="sname"/></font></td></tr>
<tr><td>Email id:</td>
<td><form:input path="email"/></td>
<td><font color=red size=4><form:errors path="email"/></font></td></tr>
<tr><td>phone No:</td>
<td><form:input path="phone"/></td>
<td><font color=red size=4><form:errors path="phone"/></font></td></tr>
<tr><td>Suitable Timings</td>
<td><form:checkbox path ="timings value="07.30 AM- 09.30AM/> 07.30AM-09.30AM <br>
<form:checkbox path ="timings" value=10.30A.M 02.30 PM/> 10.30 AM-02.30PM <br>
<form:checkbox
Path="timings" value="04.00AM 06.00PM/> 04.00AM-06.00PM <br>
<form;checkbox path="timings" value="06.30AM 08.30PM/> 06.30 AM-08.30 PM <br>
<form:checkbox
Path="timings" value="weekends"/> weekends(only Advance) <br></td>
<td> <font color=red size=4><form:errors path="timings"/></font></td></tr>
```

```
<tr><td>Gender</td>
<td><form:radiobutton path=”gender” value=”Male”/>Male <br/>
<form :radiobutton path=”value=”Female”/>Female<br/></td>
<td><font color=red size=4><form:errors path=”gender”/></font></td></tr>
<tr><form:select path=”;qualification”>
<form:option value=”…..select option……”/>
<form:option value=”M.Sc”/>
<form:option value=”B.Sc”/>
<form:option value=”M.C.A”/>
<form:option value=”B.C.A”/>
<form:option value=”M.Tech”/>
<form:option value=”B.Tech”/>
</form:select> </td><td><font color=red size=4><form:errors path=”qualification”/></font></td></tr>
<tr><td>Remarks</td>
<td><form:textarea path=”remarks” rows=”5” cols=”40”/></td>
<td><font color=red size=4><form:errors path=’remarks”/>< /font></td></tr>
<tr><td align=”center” colspan=”3”>
<input type=”submit” value=”Register now”/></td></tr>
</table>
</form:form>
</center></body></html>
```

StudentValidator.java

```
Package com.jtcindia.spring.mvc;
Import org.springframework.validation.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class studentValidator implements validator{
Public Boolean supports (class clazz) {
Seturn student.class.equals(clazz);
}
Public void validate(object obj, Errors errors) {
Student stu=(Student) obj:
If(stu.getSname()==null||stu.getSname().length()==0) {
Errors.rejectValue(“sname”,”errors.sname.required”,new object[]{},”name is Required.”);
}
If (stu.getEmail()==null||stu.getEmail().length()==0) {
Errors.rejectValue(“email”,”errors.email.required”,new object[]{},”Email is Required.”);
}
Else if(!((stu.getEmail().contains(“@”))&&(stu.getEmail().endsWith(“.com”)||stu.getEmail().endswith
(“.co.in”)|| stu.getEmail,().endsWith(“.in”))){
Errors.rejectValue(“email”,”errors.email.invalid”,new object[] {},”phone is Required.”);
}
```

```
Else if(stu.getPhone().length()!=10) {
Errors.rejectValue("phone","errors.phone.invalid",new object[] {},"phone contains 10 Required.");
}
Else if (stu.getPhone().length()==10) {
Try{
Integer.parselnt(stu.getPhone());
}catch(Exception e){
Errors.rejectValue("phone","errors.phone.invalid",new object[] {},"phone contains only digits.");
} }
If(stu.getTimings().length<1){
Errors.rejectValue("timings","errors.timings",new object[] {},"select suitable timings.");
}
If(stu.getGender()==null||stu.getGender().length()==0){
Errors.rejectValue("gender","errors.gender",new object[] {},"Gender is Required.");
}
If(stu.getQualification()==null||stu.getQualification().length()==0 ||
stu.getQualification().equals("…..select option……")) {
Errors.rejectValue("qualification","errors.qualification",new object[] {},"select qualification.");
} } }
```

Student.java

```
Package com.jtcindia.spring.mvc;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class student {
Private string sname;
Private string phone;
Private string email;
Private string timings;
Private string qualification;
Private string gender;
Private string gender;
//Setters and Getters
}
```

Student.java

```
Errors.sname.required=student Name is mandatory
Errors.email.required=Email is mandatory
Errors.email.invalid=Email is lnvalid
Errors.email.required=phone is mandatory
Errors.phone.invalid=phone is invalid
```

Jtcindia-servlet.xml

```
<beans….>
<context:annotation-config/>
<bean
class="org.springframework.web.servlet.wiew.
lnternalREsourceViewResolver">
<property name="viewClass"
calue="org.springframework.web.servlet.view.
internalResourceView"/>
<property name="prefix" value="/"/>
<property name="suffix" value=".jsp"/>
</bean>
<bean name="/register.jtc"class="com.jtcindia.
spring.mvc.RegisterController">
<property name="sessionForm" value="true/>
<property name="commandName"
value="student"/>
<property name="commandClass" value="com.
Jtcindia.spring.mvc.student"/>
<property name="validator">
<bean
class="com.jtcindia.spring.mvc.studentValidator"/>
</property>
```

	<property name=”formView” value=”register”/> <property name=”successView” value=”home”/> </bean> <bean id=”messgeSource” class=”org.spring framework.context.support.ResourceBundle MessageSource”> <property name=”besename” value=”messages”/> </bean> </beans>

Jtc 69: Registration Example using spring MVC with Annotations.
Jtc 69: Files required

Index.jsp	register.jsp
home.java	RegisterController.java
Student.java	StudentValidator.java
Messages.properties	Web.xml
Jtcindia-servlet.xml	

Index.jsp

```
<%@ taglib prefix=”form” uri=http://www.springframework.org/tags  /form”%>
<html><body><center>  <h1> java Training center</h1>
<form:form action=”registerStudent.jtc” method=”post” commandName=”student”>
   // Same as Jtc 68
</center></body></html>
```

RegistoerController.java

```
Package com.jtcindia.spring.mvc;
Import java.util.Map;
Import org.springframework.beans.factory.annotation.
Autowired;
Import org.springframework.stereotype.controller;
Import org.springframework.validation.BindingResult;
Import org.springframework.web.bind.annotation.*;
Import javax.servlet.servletException;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
@Controller
@SessionAttributes
Public class RegisterCfontroller {
@Autowired
Private StudentValidator studentValidator;
@RequestMapping(value="/registerStudent",method=
RequestMethod.POST)
Public string registerStudent(@ModelAttribute("stuent")
Studentstu, BindingResult result) throws ServletException {
System.out.println("registerStudent");
Student Validator.validate(stu, result)l
If(result.hasErrors()){
System.out.println(result.getErrorCount());
Return"register";
}
System.out.println(stu.getSid());
System.out.println(stu.getSname());
System.out.pirntln(stu.getEmail());
System.out.pirntln(stu.getphone());
System.out.pirntln(stu.getGender());
System.out.pirntln(stu.getQualification());
String.out.println(tim[i]);
}
System.out.println(stu.getRemarks());
Return "home";
}
@RequestMapping (value="/register")
Protected string showRegisterForm(Map model)
Throws
Servlet Exception {
System.out.println("showRegisterForm");
Student stu=new student();
Stu.setSid(JTC-99");
Model.put("student" ,stu);
Return"register";
}
}
```

Jtcindia.servlet.xml

```
<beans…..>
<context:annotation-config/>
<context:component-scan base-package="com.
Jtcindia.spring.mvc"/>

<bean class="org.springframework.web.servlet.
View.internalResourceViewResolver">
<property name="viewClass" value="org.
Springframework.web.servlet.view.internalResource
View"/>
<property name="prefiex" value="/"/>
<property name="suffix" value=".jsp"/>
</bean>
<bean id="studentValidator" class="com.jtcindia.
Spring.mvc.stuydentValidator"/>
<bean id="messageSource" class="org.
Springframework.context.support.ResourceBundle
MessageSource">
<property name="basename" value="messages"/>
</bean>
</beans>
```

XML	Annotation
JSP Form does not contain action attribute	JSP Form must contain action attribute
Controller must be configured in spring Configuration Document.	Controller should not be configured in spring Configuration Document. Make sure that <context:component-scan> is present. @Controller annotation is present in the controller class.
Validate () method will be called by Framework Servlet	Validate () method should be called by you in the controller.