**Spring Annotation Based Controller Tutorial**

If you are using annotations based controller then their is no need to extend any class or implement any interface. The only thing you need to do to make your simple java class to become a Spring controller is to add the @Controller annotation to it.

The following listing shows the UserController class.

01.package com.tutorials4u.web;

02.

03.import org.springframework.beans.factory.annotation.Autowired;

04.import org.springframework.stereotype.Controller;

05.import org.springframework.ui.ModelMap;

06.import org.springframework.web.bind.annotation.ModelAttribute;

07.import org.springframework.web.bind.annotation.RequestMapping;

08.import org.springframework.web.bind.annotation.RequestMethod;

09.import org.springframework.web.bind.annotation.SessionAttributes;

10.

11.import com.tutorials4u.domain.User;

12.import com.tutorials4u.service.UserService;

13.

14.@Controller

15.@RequestMapping("/userRegistration.htm")

[16.@SessionAttributes("user")](mailto:16.@SessionAttributes(%22user%22)) //MODEL

17.public class UserController {

18.

19.    private UserService userService;

20.

21.    @Autowired

22.    public void setUserService(UserService userService) {

23.        this.userService = userService;

24.    }

25.

26.    @RequestMapping(method = RequestMethod.GET)

27.    public String showUserForm(ModelMap model)  {

28.        User user = new User();

29.        model.addAttribute(user);

30.        return "userForm";

31.    }

32.

33.    @RequestMapping(method = RequestMethod.POST)

34.    public String onSubmit(@ModelAttribute("user") User user) {

35.        userService.add(user);

36.        return "redirect:userSuccess.htm";

37.    }

38.

39.}

The **@Controller** annotation is used to mark any java class as a controller class.

The **@RequestMapping** annotation is used to map the web request "/userRegistration.htm" to the UserController class.

The **@SessionAttributes** annotation is used to store the model object in the session. In our case the model object is user.

Using the **@Autowired** annotation the container can wire your beans automatically. By default autowire is done by type. Here any class that is compatible with UserService class will be chosen.

In the controller class you need to specify two methods, one for handling the HTTP GET request and the other for handling the HTTP POST request. These methods can have arbitrary names. When the form is first rendered the showUserForm() method will be invoked and when the form is submitted for processing the onSubmit() method will be invoked.

The **@RequestMapping** annotation is used to indicate the type of HTTP request.

The command object associated with the form is initialized in the showUserForm() method. The showUserForm() method has access to the ModelMap, in the showUserForm() method, create an instance of the command object and add it to the ModelMap. To add attribute to the ModelMap you can use the addAttribute() method. This method will automatically generate the attribute names for each element. In our case the user object will by default get the name as "user", you can also override the default value.

After the form is filled and submitted the onSubmit() method will be called. You can access the command object using the **@ModelAttribute** annotation. Here the command object name is user, the one that we set in the ModelMap. After getting the command object, call the service method to register the user and redirect the control to the "userSuccess.jsp" page.

Yes, here we redirect to the "userSuccess.htm" instead of just returning "userSuccess" this is necessary because if we simply return "userSuccess" the userSuccess.jsp page will be displayed, but when you refresh the page the form will be resubmitted, this is something that we don't want.

The "userSuccess.htm" request is mapped to the UserSuccessController class, here the redirect() method will be called and the control will be transferd to the userSuccess.jsp page.

The following code shows the UserSuccessController class.

01.package com.tutorials4u.web;

02.

03.import org.springframework.stereotype.Controller;

04.import org.springframework.web.bind.annotation.RequestMapping;

05.

06.@Controller

07.public class UserSuccessController {

08.

09.    @RequestMapping("/userSuccess.htm")

10.    public String redirect()

11.    {

12.        return "userSuccess";

13.    }

14.}

To enable Spring to auto-detect the controllers you need to specify the base package of the controllers using the <context:component-scan> element in the Spring configuration file. DefaultAnnotationHandlerMapping and AnnotationMethodHandlerAdapter are preregistered in the web application context by default. But you need to explicitly register them, incase you have registered any other handler mappings or handler adapters explicitly.

01.<?xml version="1.0" encoding="UTF-8"?>

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

03.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

04.    xmlns:context="http://www.springframework.org/schema/context"

05.    xsi:schemaLocation="http://www.springframework.org/schema/beans

06.        http://www.springframework.org/schema/beans/spring-beans.xsd

07.        http://www.springframework.org/schema/context

08.        http://www.springframework.org/schema/context/spring-context.xsd">

09.

10.

11.    <bean id="viewResolver"

12.        class="org.springframework.web.servlet.view. InternalResourceViewResolver"

13.        p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

14.

15.

16.    <bean id="userService" class="com.tutorials4u.service.UserServiceImpl" />

17.

18.    <context:component-scan base-package="com.tutorials4u.web" />

19.

20.    <bean class="org.springframework.web.servlet.mvc.annotation. DefaultAnnotationHandlerMapping" />

21.

22.    <bean class="org.springframework.web.servlet.mvc.annotation. AnnotationMethodHandlerAdapter" />

23.

24.</beans>

**Spring Form Tags Tutorial**

In this example you will see how to populate the form with dynamic values. Here is the user registration form.

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

02.<html>

03.<head>

04.<title>Registration Page</title>

05.

06.</head>

07.<body>

08.

09.<form:form method="POST" commandName="user">

10.<table>

11.    <tr>

12.        <td>User Name :</td>

13.

14.        <td><form:input path="name" /></td>

15.    </tr>

16.    <tr>

17.        <td>Password :</td>

18.        <td><form:password path="password" /></td>

19.

20.    </tr>

21.    <tr>

22.        <td>Gender :</td>

23.        <td>

24.            <form:radiobutton path="gender" value="M" label="M" />

25.            <form:radiobutton path="gender" value="F" label="F" />

26.

27.        </td>

28.    </tr>

29.    <tr>

30.        <td>Country :</td>

31.        <td>

32.

33.            <form:select path="country">

34.            <form:option value="0" label="Select" />

35.            <form:options items="${countryList}" itemValue="countryId" itemLabel="countryName" />

36.            </form:select>

37.        </td>

38.    </tr>

39.

40.    <tr>

41.        <td>About you :</td>

42.        <td><form:textarea path="aboutYou" /></td>

43.    </tr>

44.    <tr>

45.

46.        <td>Community :</td>

47.        <td><form:checkboxes path="communityList" items="${communityList}" itemValue="key" itemLabel="value" /></td>

48.    </tr>

49.    <tr>

50.        <td></td>

51.

52.        <td>

53.        <form:checkbox path="mailingList" label="Would you like to join our mailinglist?" />

54.        </td>

55.    </tr>

56.    <tr>

57.        <td colspan="2"><input type="submit" value="Register"></td>

58.

59.    </tr>

60.</table>

61.</form:form>

62.

63.</body>

64.</html>

Here we populate the countryList and the communityList from the back-end. The items attribute holds the collection. The itemValue and itemLabel attributes holds the key and value respectively. ItemLabel is the one that will be displayed to the user and itemValue is the one that will be passed when that particular item is selected.



Here we have three domain objects User, Country and Community. The User object is the one that is associated with the form.

01.package com.tutorials4u.domain;

02.

03.import java.util.List;

04.

05.@SuppressWarnings("unchecked")

06.public class User {

07.

08.    private String name;

09.    private String password;

10.    private String gender;

11.    private String country;

12.    private List countryList;

13.    private String aboutYou;

14.    private String[] community;

15.    private List communityList;

16.    private Boolean mailingList;

17.

18.    public String getName() {

19.        return name;

20.    }

21.    public void setName(String name) {

22.        this.name = name;

23.    }

24.    public String getPassword() {

25.        return password;

26.    }

27.    public void setPassword(String password) {

28.        this.password = password;

29.    }

30.    public String getGender() {

31.        return gender;

32.    }

33.    public void setGender(String gender) {

34.        this.gender = gender;

35.    }

36.    public String getCountry() {

37.        return country;

38.    }

39.    public void setCountry(String country) {

40.        this.country = country;

41.    }

42.    public List getCountryList() {

43.        return countryList;

44.    }

45.    public void setCountryList(List countryList) {

46.        this.countryList = countryList;

47.    }

48.    public String getAboutYou() {

49.        return aboutYou;

50.    }

51.    public void setAboutYou(String aboutYou) {

52.        this.aboutYou = aboutYou;

53.    }

54.    public String[] getCommunity() {

55.        return community;

56.    }

57.    public void setCommunity(String[] community) {

58.        this.community = community;

59.    }

60.    public List getCommunityList() {

61.        return communityList;

62.    }

63.    public void setCommunityList(List communityList) {

64.        this.communityList = communityList;

65.    }

66.    public Boolean getMailingList() {

67.        return mailingList;

68.    }

69.    public void setMailingList(Boolean mailingList) {

70.        this.mailingList = mailingList;

71.    }

72.

73.

74.}

The User object has a countryList and communityList to hold the list of countries and communities respectively.

The countryList contains a list of Country objects.

01.package com.tutorials4u.domain;

02.

03.public class Country {

04.

05.    private int countryId;

06.    private String countryName;

07.

08.    public Country(int countryId, String countryName)

09.    {

10.        this.countryId=countryId;

11.        this.countryName=countryName;

12.    }

13.

14.    public int getCountryId() {

15.        return countryId;

16.    }

17.    public void setCountryId(int countryId) {

18.        this.countryId = countryId;

19.    }

20.    public String getCountryName() {

21.        return countryName;

22.    }

23.    public void setCountryName(String countryName) {

24.        this.countryName = countryName;

25.    }

26.

27.}

The countryId is used to refer the country in the back-end and the countryName to dispaly the country in the front-end.

Similarly the communityList contains a list of community objects.

01.package com.tutorials4u.domain;

02.

03.public class Community {

04.

05.    private String key;

06.    private String value;

07.

08.    public Community(String key, String value)

09.    {

10.        this.key = key;

11.        this.value = value;

12.    }

13.

14.    public String getKey() {

15.        return key;

16.    }

17.    public void setKey(String key) {

18.        this.key = key;

19.    }

20.    public String getValue() {

21.        return value;

22.    }

23.    public void setValue(String value) {

24.        this.value = value;

25.    }

26.

27.}

Here the value is the one that will be dispalyed in the front-end and the key is the one that will be used in the back-end.

In the controller class you need to override the referenceData() method. In this method you can set all the default values that should be loaded when the form is displayed to the user. This method will be called automatically.

01.package com.tutorials4u.web;

02.

03.import java.util.HashMap;

04.import java.util.Map;

05.

06.import javax.servlet.http.HttpServletRequest;

07.

08.import org.springframework.web.servlet.ModelAndView;

09.import org.springframework.web.servlet.mvc.SimpleFormController;

10.

11.import com.tutorials4u.domain.User;

12.import com.tutorials4u.service.UserService;

13.

14.@SuppressWarnings("deprecation")

15.public class UserController extends SimpleFormController {

16.

17.    private UserService userService;

18.

19.    public UserController() {

20.        setCommandClass(User.class);

21.        setCommandName("user");

22.    }

23.

24.    public void setUserService(UserService userService) {

25.        this.userService = userService;

26.    }

27.

28.    @SuppressWarnings("unchecked")

29.    @Override

30.    protected Map referenceData(HttpServletRequest request) throws Exception {

31.        Map referenceData = new HashMap();

32.        referenceData.put("countryList", userService.getAllCountries());

33.        referenceData.put("communityList", userService.getAllCommunities());

34.        return referenceData;

35.    }

36.

37.    @Override

38.    protected ModelAndView onSubmit(Object command) throws Exception {

39.        User user = (User) command;

40.        userService.add(user);

41.        return new ModelAndView("userSuccess","user",user);

42.    }

43.

44.}

In the referenceData() method we first create a HashMap and add the countryList and the communityList to it. This method will be called before the form is rendered so the list will be populated before that.

When you run the example you will see the user registration form. On submitting the form the userSuccess.jsp page will be displayed. In the userSuccess.jsp page we use jstl tags to display the details.

01.<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

02.<html>

03.<head>

04.<title>Success Page</title>

05.</head>

06.<body>

07.

08.User Details

09.<hr>

10.User Name   : <c:out value="${user.name}"></c:out> <br/>

11.Gender      : <c:out value="${user.gender}"></c:out> <br/>

12.Country     : <c:out value="${user.country}"></c:out> <br/>

13.

14.About You   : <c:out value="${user.aboutYou}"></c:out> <br/>

15.Community   :

16.<c:forEach var="community" items="${user.communityList}" >

17.    <c:out value="${community}"></c:out>

18.</c:forEach> <br />

19.

20.Mailing List: <c:out value="${user.mailingList} "></c:out>

21.</body>

22.</html>

**Spring Annotation Based Controller Tutorial**

In this example you will see how to populate a form using Spring annotations. The annotated user controller class is shown below.

01.package com.tutorials4u.web;

02.

03.import java.util.List;

04.

05.import org.springframework.beans.factory.annotation.Autowired;

06.import org.springframework.stereotype.Controller;

07.import org.springframework.ui.ModelMap;

08.import org.springframework.web.bind.annotation.ModelAttribute;

09.import org.springframework.web.bind.annotation.RequestMapping;

10.import org.springframework.web.bind.annotation.RequestMethod;

11.import org.springframework.web.bind.annotation.SessionAttributes;

12.

13.import com.tutorials4u.domain.Community;

14.import com.tutorials4u.domain.Country;

15.import com.tutorials4u.domain.User;

16.import com.tutorials4u.service.UserService;

17.

18.@Controller

19.@RequestMapping("/userRegistration.htm")

20.@SessionAttributes("user")

21.public class UserController {

22.

23.    private UserService userService;

24.

25.    @Autowired

26.    public void setUserService(UserService userService) {

27.        this.userService = userService;

28.    }

29.

30.    @ModelAttribute("countryList")

31.    public List<Country> populateCountryList() {

32.        return userService.getAllCountries();

33.    }

34.

35.    @ModelAttribute("communityList")

36.    public List<Community> populateCommunityList() {

37.        return userService.getAllCommunities();

38.    }

39.

40.    @RequestMapping(method = RequestMethod.GET)

41.    public String showUserForm(ModelMap model) {

42.        User user = new User();

43.        model.addAttribute("user", user);

44.        return "userForm";

45.    }

46.

47.    @RequestMapping(method = RequestMethod.POST)

48.    public String onSubmit(@ModelAttribute("user") User user) {

49.        userService.add(user);

50.        return "redirect:userSuccess.htm";

51.    }

52.

53.}

The populateCountryList() and populateCommunityList() methods are used to populate the country and community list respectively. The @ModelAttribute annotation when used at the method level is used to indicate that the method contain reference data used by the model, so it should be called before the form is loaded. This is similar to overriding the referenceData() method when extending the SimpleFormController.

You can also do this in the showUserForm() method like this.

1.@RequestMapping(method = RequestMethod.GET)

2.public String showUserForm(ModelMap model) {

3.    User user = new User();

4.    model.addAttribute(userService.getAllCountries());

5.    model.addAttribute(userService.getAllCommunities());

6.    model.addAttribute("user", user);

7.    return "userForm";

8.}

Since we use ModelMap here by default the names of the list will be countryList and communityList.