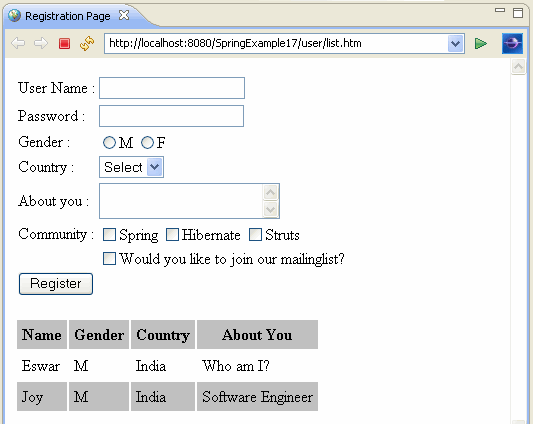
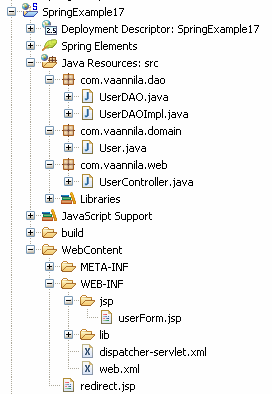
**Spring Hibernate Integration**

In this tutorial you will see how to integrate spring and hibernate. I assume you are comfortable with both spring and hibernate. At the end of this example you will learn to create a form, through which you can add a user and list all the existing users as shown below.



Here is the directory structure of the example.



You need to have the following lib files in the WEB-INF/lib directory.

01.antlr-2.7.6

02.antlr-runtime-3.0

03.commons-collections-3.1

04.commons-dbcp

05.commons-logging-1.0.4

06.commons-pool

07.dom4j-1.6.1

08.ejb3-persistence

09.hibernate3

10.hibernate-annotations

11.hibernate-commons-annotations

12.hsqldb

13.javassist-3.4.GA

14.jstl

15.jta-1.1

16.org.springframework.asm-3.0.0.M3

17.org.springframework.beans-3.0.0.M3

18.org.springframework.context-3.0.0.M3

19.org.springframework.context.support-3.0.0.M3

20.org.springframework.core-3.0.0.M3

21.org.springframework.expression-3.0.0.M3

22.org.springframework.jdbc-3.0.0.M3

23.org.springframework.orm-3.0.0.M3

24.org.springframework.transaction-3.0.0.M3

25.org.springframework.web-3.0.0.M3

26.org.springframework.web.servlet-3.0.0.M3

27.slf4j-api-1.5.6

28.slf4j-simple-1.5.6

29.standard

The most important part of the example is the spring bean configuration. Here is the spring bean configuration file.

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"

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

05.xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

06.

07.<bean id="viewResolver" class="org.springframework.web.servlet.view.

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

09.

10.

11.<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

12.<property name="driverClassName" value="org.hsqldb.jdbcDriver"/>

13.<property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>

14.<property name="username" value="sa"/>

15.<property name="password" value=""/>

16.</bean>

17.

18.

19.<bean id="mySessionFactory" class="org.springframework.orm.hibernate3.

20.annotation.AnnotationSessionFactoryBean">

21.<property name="dataSource" ref="myDataSource" />

22.<property name="annotatedClasses">

23.<list>

24.<value>com.tutorials4u.domain.User</value>

25.

26.</list>

27.</property>

28.<property name="hibernateProperties">

29.<props>

30.<prop key="hibernate.dialect"> org.hibernate.dialect.HSQLDialect</prop>

31.

32.<prop key="hibernate.show\_sql">true</prop>

33.<prop key="hibernate.hbm2ddl.auto">create</prop>

34.</props>

35.</property>

36.</bean>

37.

38.

39.<bean id="myUserDAO" class="com.tutorials4u.dao.UserDAOImpl">

40.<property name="sessionFactory" ref="mySessionFactory"/>

41.</bean>

42.

43.<bean name="/user/\*.htm" class="com.tutorials4u.web.UserController" >

44.<property name="userDAO" ref="myUserDAO" />

45.</bean>

46.

47.

48.</beans>

Here we use the Jakarta Commons DBCP BasicDataSource to setup a JDBC DataSource. I am using hsqldb database here, if you are using mysql then you need to change this configuration.

Hibernate SessionFactory can be configured in the spring bean configuration file itself as shown above, you need not have a seperate hibernate configuration file( hibernate.cfg.xml ) to do this. I am using Hibernate annotations in this example, so I am listing all the annotated classes using the annotatedClasses property. All the hibernate related configurations can be done using the hibernateProperties.

We use a seperate DAO class to interact with the database. Using setter injection we inject the Hibernate SessionFactory.

We have a MultiActionController class to handle the web requests, so we use wildcard charcter in the bean name ("/user/\*.htm").

Here is the User class with the hibernate annotations, if you want to add any database related constraints, then you need to do it here.

01.package com.tutorials4u.domain;

02.

03.import javax.persistence.Column;

04.import javax.persistence.Entity;

05.import javax.persistence.GeneratedValue;

06.import javax.persistence.Id;

07.import javax.persistence.Table;

08.

09.@Entity

10.@Table(name="USER")

11.public class User {

12.

13.private Long id;

14.private String name;

15.private String password;

16.private String gender;

17.private String country;

18.private String aboutYou;

19.private String[] community;

20.private Boolean mailingList;

21.

22.@Id

23.@GeneratedValue

24.@Column(name="USER\_ID")

25.public Long getId() {

26.return id;

27.}

28.public void setId(Long id) {

29.this.id = id;

30.}

31.

32.@Column(name="USER\_NAME")

33.public String getName() {

34.return name;

35.}

36.public void setName(String name) {

37.this.name = name;

38.}

39.

40.@Column(name="USER\_PASSWORD")

41.public String getPassword() {

42.return password;

43.}

44.public void setPassword(String password) {

45.this.password = password;

46.}

47.

48.@Column(name="USER\_GENDER")

49.public String getGender() {

50.return gender;

51.}

52.public void setGender(String gender) {

53.this.gender = gender;

54.}

55.

56.@Column(name="USER\_COUNTRY")

57.public String getCountry() {

58.return country;

59.}

60.public void setCountry(String country) {

61.this.country = country;

62.}

63.

64.@Column(name="USER\_ABOUT\_YOU")

65.public String getAboutYou() {

66.return aboutYou;

67.}

68.public void setAboutYou(String aboutYou) {

69.this.aboutYou = aboutYou;

70.}

71.

72.@Column(name="USER\_COMMUNITY")

73.public String[] getCommunity() {

74.return community;

75.}

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

77.this.community = community;

78.}

79.

80.@Column(name="USER\_MAILING\_LIST")

81.public Boolean getMailingList() {

82.return mailingList;

83.}

84.public void setMailingList(Boolean mailingList) {

85.this.mailingList = mailingList;

86.}

87.

88.}

Our DAO class implements the UserDAO interface, here we have just two methods one to save the user details and other to list all the users.

01.package com.tutorials4u.dao;

02.

03.import java.util.List;

04.import com.tutorials4u.domain.User;

05.

06.public interface UserDAO {

07.

08.public void saveUser(User user) ;

09.public List<User> listUser() ;

10.}

In the DAO class we use Hibernate Template to access the database. To create a Hibernate Template instance, you need a Session Factory, for this purpose we injected the sessionFactory property in the Spring bean configuration file.

01.package com.tutorials4u.dao;

02.

03.import java.util.List;

04.import org.hibernate.SessionFactory;

05.import org.springframework.orm.hibernate3.HibernateTemplate;

06.import com.tutorials4u.domain.User;

07.

08.public class UserDAOImpl implements UserDAO {

09.

10.private HibernateTemplate hibernateTemplate;

11.

12.public void setSessionFactory(SessionFactory sessionFactory)

13.{

14.this.hibernateTemplate = new HibernateTemplate(sessionFactory);

15.}

16.

17.@Override

18.public void saveUser(User user) {

19.hibernateTemplate.saveOrUpdate(user);

20.}

21.

22.@Override

23.@SuppressWarnings("unchecked")

24.public List<User> listUser() {

25.return hibernateTemplate.find("from User");

26.}

27.

28.}

Hibernate Template is thread safe and reusable. You need not manually open and close Session, Hibernate Template will do that for you.

The UserController class extends MultiActionController class. The UserDAOImpl instance is injected using setter injection. In the add method we call the saveUser()method and redirect the control to the "list.htm" url. This will invoke the list() method. In the list method you add two things to the modelMap, the user list to display the list of users and an instance of the user object to bind the form fields in the userForm.jsp page.

01.package com.tutorials4u.web;

02.

03.import javax.servlet.http.HttpServletRequest;

04.import javax.servlet.http.HttpServletResponse;

05.

06.import org.springframework.ui.ModelMap;

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

08.import org.springframework.web.servlet.mvc.multiaction. MultiActionController;

09.

10.import com.tutorials4u.dao.UserDAO;

11.import com.tutorials4u.domain.User;

12.

13.public class UserController extends MultiActionController {

14.

15.private UserDAO userDAO;

16.

17.public void setUserDAO(UserDAO userDAO) {

18.this.userDAO = userDAO;

19.}

20.

21.public ModelAndView add(HttpServletRequest request,

22.HttpServletResponse response, User user) throws Exception {

23.userDAO.saveUser(user);

24.return new ModelAndView("redirect:list.htm");

25.}

26.

27.public ModelAndView list(HttpServletRequest request,

28.HttpServletResponse response) throws Exception {

29.ModelMap modelMap = new ModelMap();

30.modelMap.addAttribute("userList", userDAO.listUser());

31.modelMap.addAttribute("user", new User());

32.return new ModelAndView("userForm", modelMap);

33.}

34.}

In the jsp page we use Spring Form tags to display the form fields and jstl tags to display the list of users.

01.<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

02.pageEncoding="ISO-8859-1"%>

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

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

05.

06.<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>

07.<html>

08.<head>

09.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

10.<style type="text/css">

11..even {

12.background-color: silver;

13.}

14.</style>

15.<title>Registration Page</title>

16.

17.</head>

18.<body>

19.

20.<form:form action="add.htm" commandName="user">

21.<table>

22.<tr>

23.<td>User Name :</td>

24.

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

26.</tr>

27.<tr>

28.<td>Password :</td>

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

30.

31.</tr>

32.<tr>

33.<td>Gender :</td>

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

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

36.

37.</tr>

38.<tr>

39.<td>Country :</td>

40.<td><form:select path="country">

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

42.

43.<form:option value="India" label="India" />

44.<form:option value="USA" label="USA" />

45.<form:option value="UK" label="UK" />

46.</form:select></td>

47.</tr>

48.<tr>

49.

50.<td>About you :</td>

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

52.</tr>

53.<tr>

54.<td>Community :</td>

55.

56.<td><form:checkbox path="community" value="Spring" label="Spring" />

57.<form:checkbox path="community" value="Hibernate" label="Hibernate" />

58.<form:checkbox path="community" value="Struts" label="Struts" /></td>

59.</tr>

60.<tr>

61.

62.<td></td>

63.<td><form:checkbox path="mailingList"

64.label="Would you like to join our mailinglist?" /></td>

65.</tr>

66.<tr>

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

68.

69.</tr>

70.</table>

71.</form:form>

72.<c:if test="${fn:length(userList) > 0}">

73.<table cellpadding="5">

74.<tr class="even">

75.

76.<th>Name</th>

77.<th>Gender</th>

78.<th>Country</th>

79.<th>About You</th>

80.

81.</tr>

82.<c:forEach items="${userList}" var="user" varStatus="status">

83.<tr class="<c:if test="${status.count % 2 == 0}">even</c:if>">

84.<td>${user.name}</td>

85.

86.<td>${user.gender}</td>

87.<td>${user.country}</td>

88.<td>${user.aboutYou}</td>

89.</tr>

90.

91.</c:forEach>

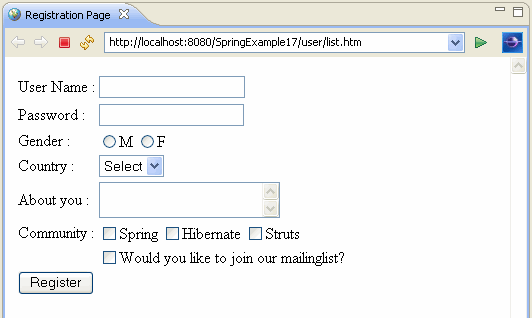
92.</table>

93.</c:if>

94.</body>

95.</html>

Now you can execute the example by running the redirect.jsp page. You will see the following page.



Aftering entering few records you will see the user's list dispalyed below.

