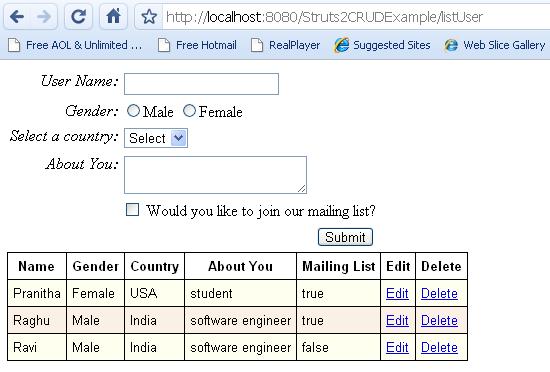
**Struts 2 CRUD Tutorial**

Learn how to perform Create, Read, Update and Delete (CRUD) operations using Struts 2.

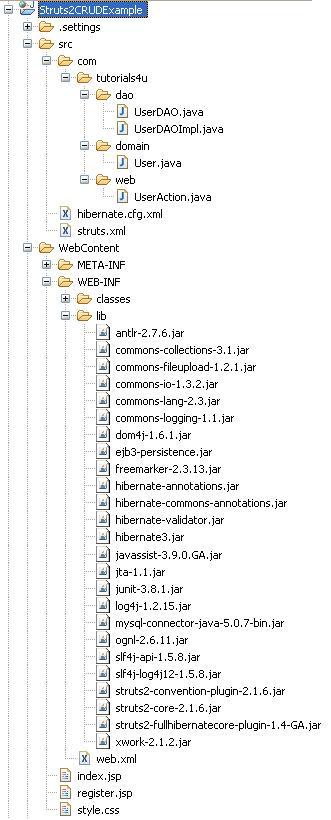
In this example you will see how to perform Create, Read, Update and Delete (CRUD) operations. I will be explaining only the points that is not covered in the previous examples.

Let's get started, the screen shot of the example is shown below.



You will have options to edit and delete in addition to the one we saw before. ( [Struts 2 Hibernate Integration](http://www.tutorials4u.net/struts2-tutorial/struts2_hibernate_integration.html) )

The directory structure of the example.



Let's see the flow from the back-end.

The UserDAOImpl has four methods to perform the various CRUD operations.

01.package com.tutorials4u.dao;

02.

03.import java.util.List;

04.

05.import org.hibernate.Session;

06.import org.hibernate.Transaction;

07.

08.import com.googlecode.s2hibernate.struts2.plugin.annotations.SessionTarget;

09.import com.googlecode.s2hibernate.struts2.plugin.annotations.TransactionTarget;

10.import com.tutorials4u.domain.User;

11.

12.public class UserDAOImpl implements UserDAO {

13.

14.    @SessionTarget

15.    Session session;

16.

17.    @TransactionTarget

18.    Transaction transaction;

19.

20.    /\*\*

21.     \* Used to save or update a user.

22.     \*/

23.    @Override

24.    public void saveOrUpdateUser(User user) {

25.        try {

26.            session.saveOrUpdate(user);

27.        } catch (Exception e) {

28.            transaction.rollback();

29.            e.printStackTrace();

30.        }

31.    }

32.

33.    /\*\*

34.     \* Used to delete a user.

35.     \*/

36.    @Override

37.    public void deleteUser(Long userId) {

38.        try {

39.            User user = (User) session.get(User.class, userId);

40.            session.delete(user);

41.        } catch (Exception e) {

42.            transaction.rollback();

43.            e.printStackTrace();

44.        }

45.    }

46.

47.    /\*\*

48.     \* Used to list all the users.

49.     \*/

50.    @SuppressWarnings("unchecked")

51.    @Override

52.    public List<User> listUser() {

53.        List<User> courses = null;

54.        try {

55.            courses = session.createQuery("from User").list();

56.        } catch (Exception e) {

57.            e.printStackTrace();

58.        }

59.        return courses;

60.    }

61.

62.    /\*\*

63.     \* Used to list a single user by Id.

64.     \*/

65.    @Override

66.    public User listUserById(Long userId) {

67.        User user = null;

68.        try {

69.            user = (User) session.get(User.class, userId);

70.        } catch (Exception e) {

71.            e.printStackTrace();

72.        }

73.        return user;

74.    }

75.

76.}

The org.hibernate.Session and org.hibernate.Transaction objects are injected using the Full Hibernate Plug-in 1.4 GA.

The User domain object with the JPA annotations to create the USER table.

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 gender;

16.    private String country;

17.    private String aboutYou;

18.    private Boolean mailingList;

19.

20.    @Id

21.    @GeneratedValue

22.    @Column(name="USER\_ID")

23.    public Long getId() {

24.        return id;

25.    }

26.    public void setId(Long id) {

27.        this.id = id;

28.    }

29.

30.    @Column(name="USER\_NAME")

31.    public String getName() {

32.        return name;

33.    }

34.    public void setName(String name) {

35.        this.name = name;

36.    }

37.

38.    @Column(name="USER\_GENDER")

39.    public String getGender() {

40.        return gender;

41.    }

42.    public void setGender(String gender) {

43.        this.gender = gender;

44.    }

45.

46.    @Column(name="USER\_COUNTRY")

47.    public String getCountry() {

48.        return country;

49.    }

50.    public void setCountry(String country) {

51.        this.country = country;

52.    }

53.

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

55.    public String getAboutYou() {

56.        return aboutYou;

57.    }

58.    public void setAboutYou(String aboutYou) {

59.        this.aboutYou = aboutYou;

60.    }

61.

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

63.    public Boolean getMailingList() {

64.        return mailingList;

65.    }

66.    public void setMailingList(Boolean mailingList) {

67.        this.mailingList = mailingList;

68.    }

69.

70.}

Our UserAction implements ModelDriven interface, so the domain object User will be exposed as a model object. Use ActionContext.getContext().get(ServletActionContext. HTTP\_REQUEST) method to access the HttpServeletRequest object in action.

01.package com.tutorials4u.web;

02.

03.import java.util.ArrayList;

04.import java.util.List;

05.

06.import javax.servlet.http.HttpServletRequest;

07.

08.import org.apache.struts2.ServletActionContext;

09.

10.import com.opensymphony.xwork2.ActionContext;

11.import com.opensymphony.xwork2.ActionSupport;

12.import com.opensymphony.xwork2.ModelDriven;

13.import com.tutorials4u.dao.UserDAO;

14.import com.tutorials4u.dao.UserDAOImpl;

15.import com.tutorials4u.domain.User;

16.

17.public class UserAction extends ActionSupport implements ModelDriven<User> {

18.

19.    private static final long serialVersionUID = -6659925652584240539L;

20.

21.    private User user = new User();

22.    private List<User> userList = new ArrayList<User>();

23.    private UserDAO userDAO = new UserDAOImpl();

24.

25.    @Override

26.    public User getModel() {

27.        return user;

28.    }

29.

30.    /\*\*

31.     \* To save or update user.

32.     \* @return String

33.     \*/

34.    public String saveOrUpdate()

35.    {

36.        userDAO.saveOrUpdateUser(user);

37.        return SUCCESS;

38.    }

39.

40.    /\*\*

41.     \* To list all users.

42.     \* @return String

43.     \*/

44.    public String list()

45.    {

46.        userList = userDAO.listUser();

47.        return SUCCESS;

48.    }

49.

50.    /\*\*

51.     \* To delete a user.

52.     \* @return String

53.     \*/

54.    public String delete()

55.    {

56.        HttpServletRequest request = (HttpServletRequest) ActionContext.getContext().get( ServletActionContext.HTTP\_REQUEST);

57.        userDAO.deleteUser(Long.parseLong( request.getParameter("id")));

58.        return SUCCESS;

59.    }

60.

61.    /\*\*

62.     \* To list a single user by Id.

63.     \* @return String

64.     \*/

65.    public String edit()

66.    {

67.        HttpServletRequest request = (HttpServletRequest) ActionContext.getContext().get( ServletActionContext.HTTP\_REQUEST);

68.        user = userDAO.listUserById(Long.parseLong( request.getParameter("id")));

69.        return SUCCESS;

70.    }

71.

72.    public User getUser() {

73.        return user;

74.    }

75.

76.    public void setUser(User user) {

77.        this.user = user;

78.    }

79.

80.    public List<User> getUserList() {

81.        return userList;

82.    }

83.

84.    public void setUserList(List<User> userList) {

85.        this.userList = userList;

86.    }

87.

88.}

In the struts configuration file we have four different actions corresponding to the different CRUD operations. During the save, update and delete operations, we need to update the list of users displayed below so we redirect the result to the listUser acion.

01.<!DOCTYPE struts PUBLIC

02."-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

03."http://struts.apache.org/dtds/struts-2.0.dtd">

04.

05.<struts>

06.    <package name="default" extends="hibernate-default">

07.        <action name="saveOrUpdateUser" method="saveOrUpdate" class="com.tutorials4u.web.UserAction">

08.            <result name="success" type="redirect">listUser</result>

09.        </action>

10.        <action name="listUser" method="list" class="com.tutorials4u.web.UserAction">

11.            <result name="success">/register.jsp</result>

12.        </action>

13.        <action name="editUser" method="edit" class="com.tutorials4u.web.UserAction">

14.            <result name="success">/register.jsp</result>

15.        </action>

16.        <action name="deleteUser" method="delete" class="com.tutorials4u.web.UserAction">

17.            <result name="success" type="redirect">listUser</result>

18.        </action>

19.    </package>

20.

21.</struts>

The hibernate.cfg.xml file contains the following configuration.

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

02.<!DOCTYPE hibernate-configuration PUBLIC

03.    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

04.    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

05.<hibernate-configuration>

06.    <session-factory>

07.        <property name="hibernate.connection.driver\_class">

08.            com.mysql.jdbc.Driver

09.        </property>

10.        <property name="hibernate.connection.url">

11.            jdbc:mysql://localhost:3306/tutorials4u

12.        </property>

13.        <property name="hibernate.connection.username">root</property>

14.        <property name="connection.password">root</property>

15.        <property name="connection.pool\_size">1</property>

16.        <property name="hibernate.dialect">

17.            org.hibernate.dialect.MySQLDialect

18.        </property>

19.        <property name="show\_sql">true</property>

20.        <property name="hbm2ddl.auto">create</property>

21.        <mapping class="com.tutorials4u.domain.User" />

22.    </session-factory>

23.

24.</hibernate-configuration>

The deployment descriptor contains the following configuration.

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

02.<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\_2\_5.xsd" id="WebApp\_ID" version="2.5">

03.  <display-name>Struts2CRUDExample</display-name>

04.  <filter>

05.        <filter-name>struts2</filter-name>

06.        <filter-class>

07.            org.apache.struts2.dispatcher.ng.filter. StrutsPrepareAndExecuteFilter </filter-class>

08.    </filter>

09.    <filter-mapping>

10.        <filter-name>struts2</filter-name>

11.        <url-pattern>/\*</url-pattern>

12.    </filter-mapping>

13.

14.    <welcome-file-list>

15.        <welcome-file>index.jsp</welcome-file>

16.    </welcome-file-list>

17.

18.</web-app>

In the register.jsp page we use the Struts 2 tags to display the form elements.

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

02.    pageEncoding="ISO-8859-1"%>

03.<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

04.<%@taglib uri="/struts-tags" prefix="s"%>

05.<html>

06.<head>

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

08.<title>Registration Page</title>

09.

10.<s:head />

11.<style type="text/css">

12.@import url(style.css);

13.</style>

14.</head>

15.<body>

16.<s:form action="saveOrUpdateUser">

17.    <s:push value="user">

18.        <s:hidden name="id" />

19.        <s:textfield name="name" label="User Name" />

20.        <s:radio name="gender" label="Gender"

21.            list="{'Male','Female'}" />

22.        <s:select name="country" list="{'India','USA','UK'}"

23.            headerKey="" headerValue="Select"

24.            label="Select a country" />

25.        <s:textarea name="aboutYou" label="About You" />

26.        <s:checkbox name="mailingList"

27.            label="Would you like to join our mailing list?" />

28.        <s:submit />

29.    </s:push>

30.</s:form>

31.

32.<s:if test="userList.size() > 0">

33.    <div class="content">

34.    <table class="userTable" cellpadding="5px">

35.        <tr class="even">

36.            <th>Name</th>

37.            <th>Gender</th>

38.            <th>Country</th>

39.            <th>About You</th>

40.            <th>Mailing List</th>

41.            <th>Edit</th>

42.            <th>Delete</th>

43.        </tr>

44.        <s:iterator value="userList" status="userStatus">

45.            <tr

46.                class="<s:if test="#userStatus.odd == true ">odd</s:if> <s:else>even</s:else>">

47.                <td><s:property value="name" /></td>

48.                <td><s:property value="gender" /></td>

49.                <td><s:property value="country" /></td>

50.                <td><s:property value="aboutYou" /></td>

51.                <td><s:property value="mailingList" /></td>

52.                <td>

53.                <s:url id="editURL" action="editUser">

54.                    <s:param name="id" value="%{id}"></s:param>

55.                </s:url>

56.                <s:a href="%{editURL}">Edit</s:a>

57.                </td>

58.                <td>

59.                <s:url id="deleteURL" action="deleteUser">

60.                    <s:param name="id" value="%{id}"></s:param>

61.                </s:url>

62.                <s:a href="%{deleteURL}">Delete</s:a>

63.                </td>

64.            </tr>

65.        </s:iterator>

66.    </table>

67.    </div>

68.

69.</s:if>

70.</body>

71.</html>

The **push** tag is used to move the object to the top of the ValueStack. During add operation we will refer to the model object User exposed by the ModelDriven inteface, in this case the push tag is not necessary. But during update operation we refer to the JavaBean property user that is returned by the listUserById() method, now the push tag will be useful, it pushes the User object to the top of the ValueStack so we need not use the second-level OGNL expression language like user.name to access the domain object properties.

1.public String edit()

2.{

3.    HttpServletRequest request = (HttpServletRequest) ActionContext.getContext().get( ServletActionContext.HTTP\_REQUEST);

4.    user = userDAO.listUserById(Long.parseLong(request. getParameter("id")));

5.    return SUCCESS;

6.}

The **url** tag is used to create a new URL. Along with the URL we append the id value, this can be done using the param tab. In the OGNL expression language we use "%{}" as the escape sequence to refer a value on the ActionContext.