VaanNila

Menu

Struts 1

Struts 2

Spring

Hibernate

Ant

Log4j

Spring > Spring SimpleFormController

Spring SimpleFormController

To handle forms in Spring you need to extend your controller class from SimpleFormController class. Here we will create a user registration form to understand how this works. The SimpleFormController is deprecated as of Spring 3.0 so if you are using Spring 3.0 or above use the annotate controllers instead.

```
01. package com.vaannila.web;
02.
03. import org.springframework.web.servlet.ModelAndView;
04. import org.springframework.web.servlet.mvc.SimpleFormController;
05.
06. import com.vaannila.domain.User;
07. import com.vaannila.service.UserService;
08.
09. @SuppressWarnings("deprecation")
10. public class UserController extends SimpleFormController {
11.
12.     private UserService userService;
13.
14.     public UserController() {
15.         setCommandClass(User.class);
16.         setCommandName("user");
17.     }
18.
19.     public void setUserService(UserService userService) {
20.         this.userService = userService;
21.     }
22.
23.     @Override
24.     protected ModelAndView onSubmit(Object command) throws Exception {
25.         User user = (User) command;
26.         userService.add(user);
27.         return new ModelAndView("userSuccess","user",user);
28.     }
29.
30. }
```

I am using Spring 3.0 so you see the `SuppressWarnings` annotation there. Here we extend the `UserController` from `SimpleFormController`, this makes the controller class capable of handling forms. Usually a form will be associated with a particular domain object, in our case it is the `User` class. In Spring this domain object is called command object by default. To refer the command object in the jsp page you need to set the command class using the `setCommandClass()` method in the constructor. Let say the User class has a name property, and to refer this in the jsp page you will use "`command.name`". You can also change this name by using the `setCommandName()` method. Here we set the name to user, so to access the user name in the jsp page we use "`user.name`".

You need to have a method to handle the form when the form is submitted, here the `onSubmit()` method is used for this purpose. The `onSubmit()` method has access to the command object, we first typecast the command object to `User` (our domain object) and then to register the user we call the `add()` method of the service class and finally return the `ModelandView` object.

All the forms field values will be submitted as Strings to the form controller. Spring has several pre registered property editors to convert the String values to common data types. Incase you have a custom data type you need to create custom property editors to handle them.

The User domain object has the following attributes.

```
01. package com.vaannila.domain;
02.
03. public class User {
04.
05.     private String name;
06.     private String password;
07.     private String gender;
08.     private String country;
09.     private String aboutYou;
10.     private String[] community;
11.     private Boolean mailingList;
12.
13.     public String getName() {
14.         return name;
15.     }
16.     public void setName(String name) {
17.         this.name = name;
18.     }
19.     public String getPassword() {
20.         return password;
21.     }
22.     public void setPassword(String password) {
23.         this.password = password;
24.     }
25.     public String getGender() {
26.         return gender;
27.     }
28.     public void setGender(String gender) {
29.         this.gender = gender;
```

Subscribe

RSS

Email

```
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
    public String getAboutYou() {
        return aboutYou;
    }
    public void setAboutYou(String aboutYou) {
        this.aboutYou = aboutYou;
    }
    public String[] getCommunity() {
        return community;
    }
    public void setCommunity(String[] community) {
        this.community = community;
    }
    public Boolean getMailingList() {
        return mailingList;
    }
    public void setMailingList(Boolean mailingList) {
        this.mailingList = mailingList;
    }


}
```

Our User Service interface.

```
package com.vaannila.service;

import com.vaannila.domain.User;

public interface UserService {

    public void add(User user);
}
```

Our User Service Implementation class.

```
package com.vaannila.service;

import com.vaannila.domain.User;

public class UserServiceImpl implements UserService {

    @Override
    public void add(User user) {
        //Persist the user object here.
        System.out.println("User added successfully");

    }

}
```

Let's now create the user registration form using the Spring form tags. To use the form tags you need to first import the Spring's form tag library.

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
<title>Registration Page</title>
</head>
<body>

<form:form method="POST" commandName="user">
<table>
    <tr>
        <td>User Name :</td>
        <td><form:input path="name" /></td>
    </tr>
    <tr>
        <td>Password :</td>
        <td><form:password path="password" /></td>
    </tr>
    <tr>
        <td>Gender :</td>
        <td>
            <form:radiobutton path="gender" value="M" label="M" />
            <form:radiobutton path="gender" value="F" label="F" />
        </td>
    </tr>
    <tr>
        <td>Country :</td>
        <td>
        <form:select path="country">
            <form:option value="0" label="Select" />
            <form:option value="1" label="India" />
            <form:option value="2" label="USA" />
            <form:option value="3" label="UK" />
        </form:select></td>
    </tr>
    <tr>
        <td>About you :</td>
        <td><form:textarea path="aboutYou" /></td>
    </tr>
    <tr>
        <td>Community :</td>
```

```
        <td>
            <form:checkbox path="community" value="Spring" label="Spring"
              />
            <form:checkbox path="community" value="Hibernate"
               abel="Hibernate" />
            <form:checkbox path="community" value="Struts" label="Struts"
              />
        </td>
    </tr>
    <tr>
        <td></td>
        <td>
        <form:checkbox path="mailingList" label="Would you like to join
          our mailinglist?" /></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit"></td>
    </tr>
</table>
</form:form>

</body>
</html>
```

Here the `path` attribute is used to bind the form fields to the domain object. Here we use the `HTTP POST` method to submit the form. Inorder to bind the form fields to the domain object successfully the command object should be set to the same name in the jsp page and the controller class. To set the command object name in the jsp page, use the `commandName` attribute of the form tag.

The web.xml file.

```
<?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_2_5.xsd"
 id="WebApp_ID" version="2.5">
  <display-name>SpringExample6</display-name>
  <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet. DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Next create the Spring Bean Configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="viewResolver"
    class="org.springframework.web.servlet.view.
       InternalResourceViewResolver"
    p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

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

    <bean name="/userRegistration.htm"
       class="com.vaannila.web.UserController" p:userService-
       ref="userService" p:formView="userForm" p:successView="userSuccess"
       />

</beans>
```

As you can see, we use "p" namespace here. The "p" namespace is simple and easy to use. Using "p" namespace the properties can be supplied using attributes, rather than elements.

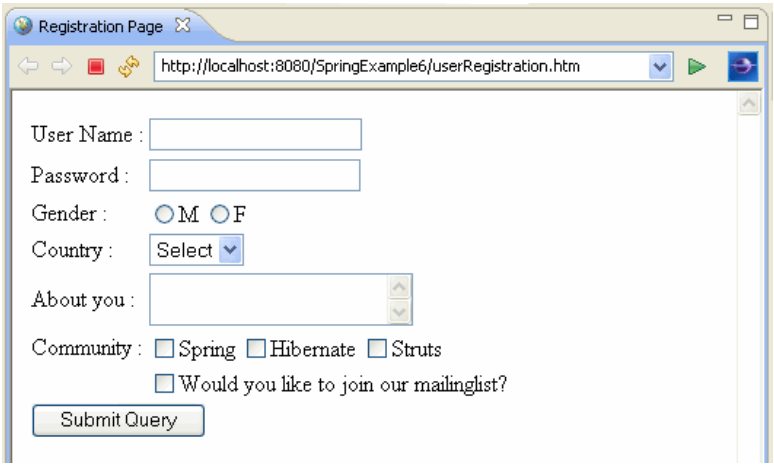
For injecting the simple types we use property name in the "p" namespace and for injecting references we add "`-ref`" suffix to it. For example we use `p:formView` for injecting the form view property and `p:userService-ref` for injecting the user service.

During the `HTTP GET` request the `formView` will be rendered. When the form is submitted (during the `HTTP POST` request) the onSubmit() method of the UserController class will be called, on successful execution of the method the `successView` will be rendered. Incase of any type conversion errors or validation errors the `formView` will be automatically displayed the user.

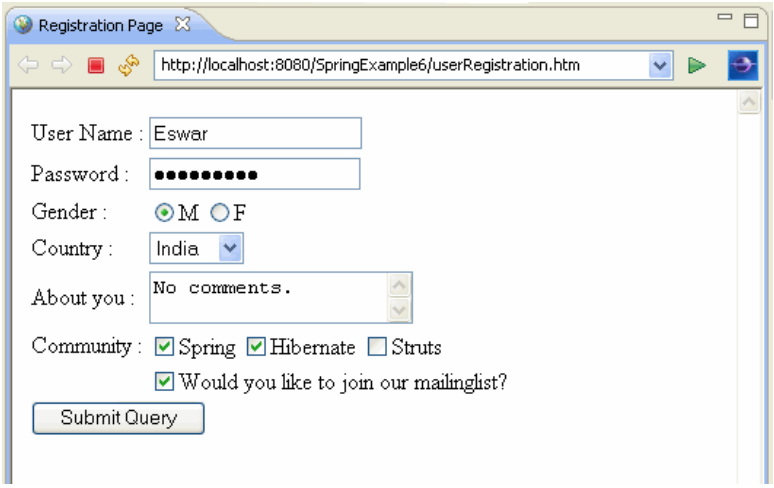
Run the example by executing the `redirect.jsp` file. The `redirect.jsp` file, redirect the request to "`userRegistration.htm`".

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<% response.sendRedirect("userRegistration.htm"); %>
```

The following user registration page will be displayed.



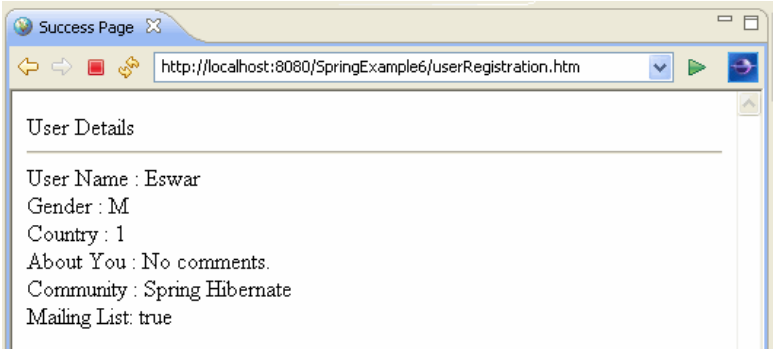
Fill the form and submit it.



The `onSubmit()` method of the `UserController` class will be called and the control will be transfered to the view `"userSuccess"`. We use `InternalResourceViewResolver` here, so the `userSuccess.jsp` page will be dispalyed. In the `userSuccess.jsp` page we dispaly all the user details using the jstl tags.

```
01. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02.     pageEncoding="ISO-8859-1"%>
03. <html>
04. <head>
05. <title>Success Page</title>
06. </head>
07. <body>
08. User Details
09. <hr>
10. User Name    : ${user.name} <br/>
11. Gender       : ${user.gender} <br/>
12. Country      : ${user.country} <br/>
13. About You    : ${user.aboutYou} <br/>
14. Community    : ${user.community[0]}  ${user.community[1]}
      ${user.community[2]}<br/>
15. Mailing List: ${user.mailingList}
16. </body>
17. </html>
```

The userSuccess.jsp page.



You can download and try the example here.

Source :**Download**

War :**Download**