



Apache Struts: Handling Request Parameters with Form Beans

Struts 1.2 Version

Core Servlets & JSP book: www.coreservlets.com
More Servlets & JSP book: www.moreservlets.com
Servlet/JSP/Struts/JSF Training: courses.coreservlets.com
Slides © Marty Hall, <http://www.coreservlets.com>, books © Sun Microsystems Press



For live Struts training, please see
JSP/servlet/Struts/JSF training courses at
<http://courses.coreservlets.com/>.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

Slides © Marty Hall, <http://www.coreservlets.com>, books © Sun Microsystems Press

Agenda

- **Three new ideas**
 - Automatically created bean representing request data
 - Using bean:write to output bean properties
 - Using bean:message to output constant strings
- **Defining form beans**
- **Declaring form beans**
- **Outputting properties of form beans**
 - bean:write
 - JSP 2.0 expression language
- **Defining and outputting regular beans**
- **Using properties files**
 - To reuse fixed strings
 - To support internationalization

5

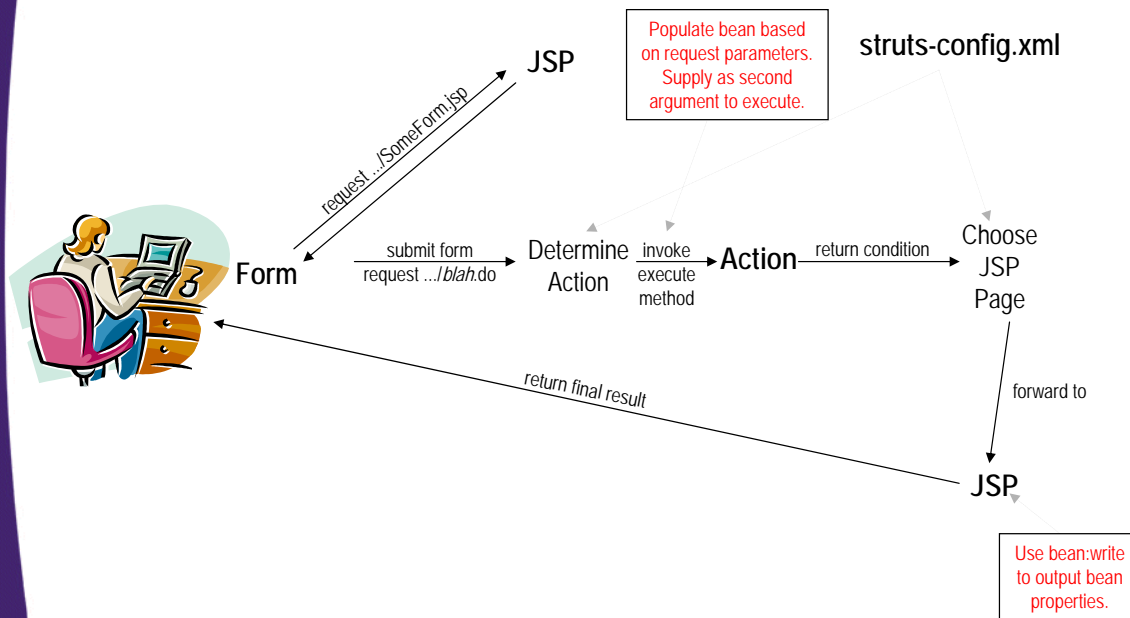
Apache Struts:Beans

www.coreservlets.com



Using Beans

Struts Flow of Control



7

Apache Struts:Beans

www.coreservlets.com

Struts Flow of Control

- **The user requests a form**
 - For now, we use normal HTML to build the form
 - Later we will use the Struts `html:form` tag
- **The form is submitted to a URL of the form *blah.do*.**
 - That address is mapped by `struts-config.xml` to an Action class
- **The execute method of the Action object is invoked**
 - One of the arguments to execute is a form bean that is automatically created and whose properties are automatically populated based on incoming request parameters of the same name
 - The Action object then invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope.
 - The Action uses `mapping.findForward` to return a condition, and the conditions are mapped by `struts-config.xml` to various JSP pages.
- **Struts forwards request to the appropriate JSP page**
 - The page can use `bean:write` or the JSP 2.0 EL to output bean properties
 - The page can use `bean:message` to output fixed strings

8

Apache Struts:Beans

www.coreservlets.com

New Capabilities

- **Very important idea**
 - Struts lets you define a bean that represents the incoming request data. Struts will create and populate the bean for you, and pass it to the Action as the second argument to the execute method.
 - Bean must extend ActionForm
 - Bean must be declared in struts-config.xml with form-beans
- **Moderately important idea**
 - You can use the Struts bean:write tag to output bean properties in JSP pages.
- **Somewhat important idea**
 - You can use the Struts bean:message tag to output constant strings that came from a properties file.
 - File declared in struts-config.xml with message-resources

The Six Basic Steps in Using Struts: Updates for Bean Use

1. **Modify struts-config.xml.**
Use WEB-INF/struts-config.xml to:
 - Map incoming .do addresses to Action classes
 - Map return conditions to JSP pages
 - **Declare any form beans that are being used.**
 - Restart server after modifying struts-config.xml.
2. **Define a form bean.**
 - This bean extends ActionForm and represents the data submitted by the user. It is automatically populated when the input form is submitted. More precisely:
 1. The reset method is called (useful for session-scoped beans)
 2. For each incoming request parameter, the corresponding setter method is called
 3. The validate method is called (possibly preventing the Action)

The Six Basic Steps in Using Struts: Updates for Bean Use

3. Create results beans.

- These are normal beans of the sort used in MVC when implemented directly with RequestDispatcher. That is, they represent the results of the business logic and data access code. These beans are stored in request, session, or application scope with the `setAttribute` method of `HttpServletRequest`, `HttpSession`, or `ServletContext`, just as in normal non-Struts applications.

4. Define an Action class to handle requests.

- Rather than calling `request.getParameter` explicitly as in the previous example, **the `execute` method casts the `ActionForm` argument to the specific form bean class**, then uses getter methods to access the properties of the object.

The Six Basic Steps in Using Struts: Updates for Bean Use

5. Create form that invokes *blah.do*.

- This form can use the **`bean:message`** tag to output standard messages and text labels that are defined in the properties file that is declared with `message-resources` in `struts-config.xml`.
 - Later, we will also use the `html:form` tag to guarantee that the textfield names correspond to the bean property names, and to make it easy to fill in the form based on values in the app

6. Display results in JSP.

- The JSP page uses the **`bean:write`** tag to output properties of the form and result beans.
- It may also use the `bean:message` tag to output standard messages and text labels that are defined in a properties file (resource bundle).

Example 1: Form and Results Beans

- **URL**
 - `http://hostname/struts-beans/register1.do`
- **Action Class**
 - `BeanRegisterAction`
 - Instead of reading form data explicitly with `request.getParameter`, the `execute` method uses a bean that is automatically filled in from the request data.
 - As in the previous example, this method returns "success", "bad-address", or "bad-password"
- **Results pages**
 - `/WEB-INF/results/confirm-registration1.jsp`
 - `/WEB-INF/results/bad-address1.jsp`
 - `/WEB-INF/results/bad-password1.jsp`

New Features of This Example

- **The use of a bean to represent the incoming form data.**
 - This bean extends the `ActionForm` class, is declared in `struts-config.xml` with the `form-bean` tag, and is referenced in `struts-config.xml` with `name` and `scope` attributes in the `action` element.
- **The use of a regular bean to represent custom results.**
 - As with beans used with regular MVC, this bean need not extend any particular class and requires no special `struts-config.xml` declarations.
- **The use of the Struts `bean:write` tags to output bean properties in the JSP page that displays the final results.**
 - This is basically a more powerful and concise alternative to the standard `jsp:getProperty` tag. Before we use `bean:write`, we have to import the "bean" tag library as follows.

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
      prefix="bean" %>
```

Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
 - As before, we use the action element (to designate that BeanRegisterAction should handle requests for register1.do).
- **Map return conditions to JSP pages**
 - As before, we use multiple forward elements, one for each possible return value of the execute method
- **Declare any form beans that are being used.**
 - Use **form-bean** (within form-beans) with these two attributes:
 - **name**: a name that will match the name attribute of the action element.
 - **type**: the fully qualified classname of the bean.
 - Note that the form-beans section goes *before* action-mappings in struts-config.xml, not inside action-mappings
 - Here is an example:

```
<form-beans>
  <form-bean name="userFormBean"
    type="coreservlets.UserFormBean"/>
</form-beans>
```

Step 1 (Modify struts-config.xml), Continued

- **Update action declaration**
 - After declaring the bean in the form-beans section, you need to add two new attributes to the action element: name and scope
 - **name**: a bean name matching the form-bean declaration.
 - **scope**: request or session. Surprisingly, session is the default, but always explicitly list the scope anyhow. We want request here.
 - Here is an example.

```
<action path="/register1"
  type="coreservlets.BeanRegisterAction"
  name="userFormBean"
  scope="request">
```

Step 1 (Modify struts-config.xml) -- Final Code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC ... >
<struts-config>
  <form-beans>
    <form-bean name="userFormBean"
               type="coreservlets.UserFormBean"/>
  </form-beans>
  <action-mappings>
    <action path="/register1"
            type="coreservlets.BeanRegisterAction"
            name="userFormBean"
            scope="request">
      <forward name="bad-address"
                path="/WEB-INF/results/bad-address1.jsp"/>
      <forward name="bad-password"
                path="/WEB-INF/results/bad-password1.jsp"/>
      <forward name="success"
                path="/WEB-INF/results/confirm-registration1.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

17

www.coreservlets.com

Step 2 (Define a Form Bean)

- A form bean is a Java object that will be automatically filled in based on the incoming form parameters, then passed to the execute method. Requirements:
 - It must extend **ActionForm**.
 - The argument to execute is of type ActionForm. Cast the value to your real type, and then each bean property has the value of the request parameter with the matching name.
 - It must have a **zero argument constructor**.
 - The system will automatically call this default constructor.
 - It must have **settable bean properties that correspond to the request parameter names**.
 - That is, it must have a `setBlah` method corresponding to each incoming request parameter named *blah*. The properties should be of type String (i.e., each `setBlah` method should take a String as an argument).
 - It must have **gettable bean properties for each property that you want to output in the JSP page**.
 - That is, it must have a `getBlah` method corresponding to each bean property that you want to display in JSP without using Java syntax.

18

Apache Struts:Beans

www.coreservlets.com

Step 2 (Define a Form Bean) -- Code Example

```
package coreservlets;
import org.apache.struts.action.*;

public class UserFormBean extends ActionForm {
    private String email = "Missing address";
    private String password = "Missing password";

    public String getEmail() { return(email); }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() { return(password); }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

19

Apache Struts:Beans

www.coreservlets.com

Step 3 (Create Results Beans)

- **These are normal beans of the type used in regular MVC (i.e., implemented with RequestDispatcher)**
 - The form bean represents the *input* data: the data that came from the HTML form. In most applications, the more important type of data is the *result* data: the data created by the business logic to represent the results of the computation or database lookup.
 - Results beans need to have getter and setter methods like normal JavaBeans, but need not extend any particular class or be declared in struts-config.xml.
 - They are stored in request, session, or application scope with the setAttribute method of HttpServletRequest, HttpSession, or ServletContext, respectively.

20

Apache Struts:Beans

www.coreservlets.com

Step 3, Bean Code Example

```
package coreservlets;

public class SuggestionBean {
    private String email;
    private String password;

    public SuggestionBean(String email, String password) {
        this.email = email;
        this.password = password;
    }

    public String getEmail() {
        return(email);
    }

    public String getPassword() {
        return(password);
    }
}
```

Step 3, Business Logic Code (To Build SuggestionBean)

```
package coreservlets;

public class SuggestionUtils {
    private static String[] suggestedAddresses =
        { "president@whitehouse.gov",
          "gates@microsoft.com",
          "palmisano@ibm.com",
          "ellison@oracle.com" };
    private static String chars =
        "abcdefghijklmnopqrstuvwxyz0123456789#@$%^&*?!";

    public static SuggestionBean getSuggestionBean() {
        String address = randomString(suggestedAddresses);
        String password = randomString(chars, 8);
        return(new SuggestionBean(address, password));
    }
    ...
}
```

Step 4 (Define an Action Class to Handle Requests)

- This example is similar to the previous one except that we do not call `request.getParameter` explicitly.
 - Instead, we extract the request parameters from the already populated form bean.
 - Specifically, we take the `ActionForm` argument supplied to `execute`, cast it to `UserFormBean` (our concrete class that extends `ActionForm`), and then call getter methods on that bean.
 - Also, we create a `SuggestionBean` and store it in request scope for later display in JSP.
 - This bean is the result of our business logic, and does not correspond to the incoming request parameters

Step 4 (Define an Action Class to Handle Requests) -- Code

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             ... request, ... response)
    throws Exception {
    UserFormBean userBean = (UserFormBean)form;
    String email = userBean.getEmail();
    String password = userBean.getPassword();
    if ((email == null) ||
        (email.trim().length() < 3) ||
        (email.indexOf("@") == -1)) {
        request.setAttribute("suggestionBean",
                             SuggestionUtils.getSuggestionBean());
        return(mapping.findForward("bad-address"));
    } else if ((password == null) ||
               (password.trim().length() < 6)) {
        request.setAttribute("suggestionBean",
                             SuggestionUtils.getSuggestionBean());
        return(mapping.findForward("bad-password"));
    } else {
        return(mapping.findForward("success"));
    }
}
```

Step 5 (Create Form that Invokes *blah.do*)

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>New Account Registration</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>New Account Registration</H1>
<FORM ACTION="register1.do"
      METHOD="POST">
  Email address: <INPUT TYPE="TEXT" NAME="email"><BR>
  Password: <INPUT TYPE="PASSWORD" NAME="password"><BR>
  <INPUT TYPE="SUBMIT" VALUE="Sign Me Up!">
</FORM>
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Alternatives for Beans

- **Use JSP scripting elements.**
 - This approach is out of the question; it is precisely what Struts is designed to avoid.
- **Use `jsp:useBean` and `jsp:getProperty`.**
 - This approach is possible, but these tags are a bit clumsy and verbose.
- **Use the JSTL `c:out` tag.**
 - This approach is not a bad idea, but it is hardly worth the bother to use JSTL just for this situation. So, unless you are using JSTL elsewhere in your application anyhow, don't bother with `c:out`.

Step 6 (Display Results in JSP) Alternatives for Beans

- **Use the JSP 2.0 expression language.**
 - This is perhaps the best option if the server supports JSP 2.0. In these examples, we will assume that the application needs to run on multiple servers, some of which support only JSP 1.2.
- **Use the Struts bean:write tag.**
 - This is by far the most common approach when using Struts. Note that, unlike c:out and the JSP 2.0 expression language, bean:write automatically filters special HTML characters, replacing < with < and > with >. You can disable this behavior by specifying

```
<bean:write name="beanName"
             property="beanProperty"
             filter="false">
```
 - So, in this example we use bean:write. Before we do so, however, we have to import the "bean" tag library as follows.

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
```

Step 6 (Display Results in JSP) First Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Email Address</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Email Address</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
      prefix="bean" %>
The address
"<bean:write name="userFormBean" property="email"/>"
is not of the form username@hostname (e.g.,
<bean:write name="suggestionBean" property="email"/>).
<P>
Please <A HREF="register1.jsp">try again</A>.
</CENTER>
</BODY></HTML>
```


Step 6 (Display Results in JSP) Second Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Password</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Password</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>
The password
"<bean:write name="userFormBean" property="password"/>"
is too short; it must contain at least six characters.
Here is a possible password:
<bean:write name="suggestionBean" property="password"/>.
<P>
Please <A HREF="register1.jsp">try again</A>.
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Third Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Success</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>You have registered successfully.</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>
<UL>
  <LI>Email Address:
    <bean:write name="userFormBean" property="email"/>
  <LI>Password:
    <bean:write name="userFormBean" property="password"/>
</UL>
Congratulations
</CENTER>
</BODY></HTML>
```

Example 1: Results (Initial Form)

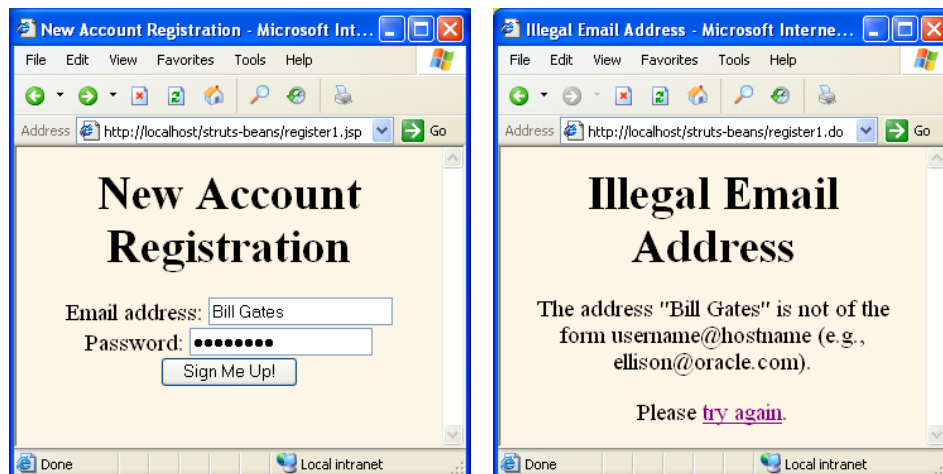


31

Apache Struts:Beans

www.coreservlets.com

Example 1: Results (Illegal Address)

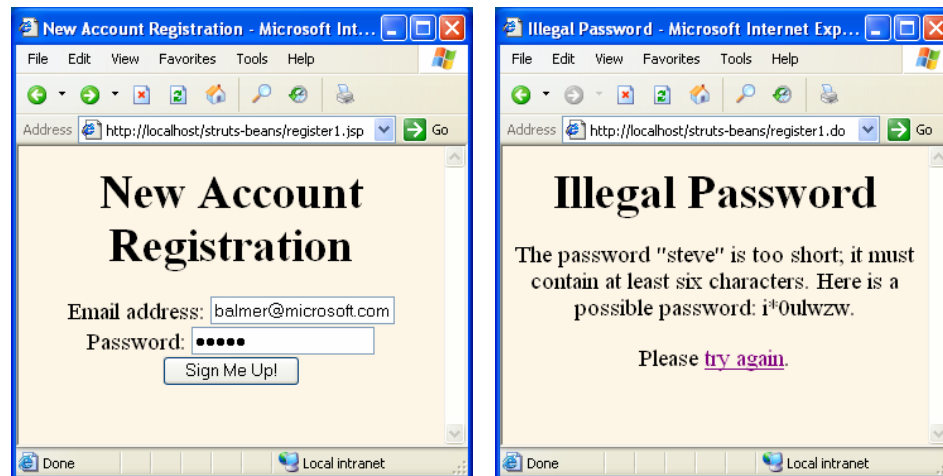


32

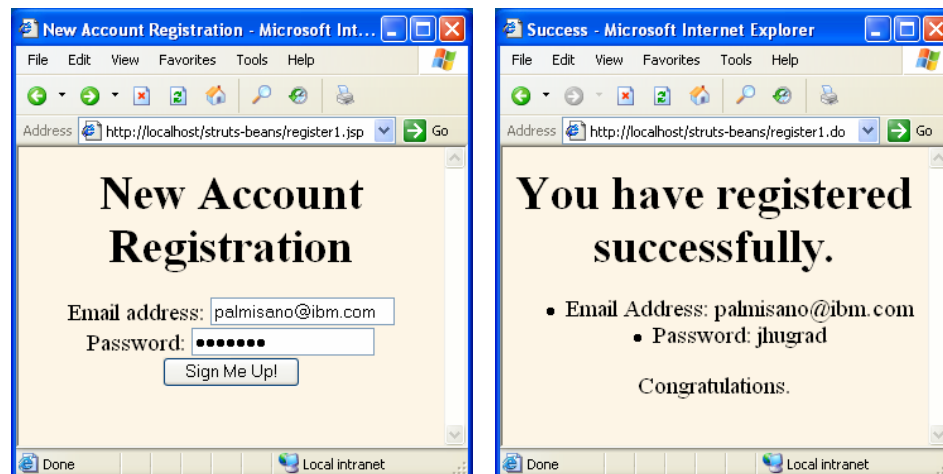
Apache Struts:Beans

www.coreservlets.com

Example 1: Results (Illegal Password)



Example 1: Results (Legal Address and Password)



Using the JSP 2.0 Expression Language with Struts

- **Pros**
 - The JSP 2.0 EL is shorter, clearer, and more powerful
- **Cons**
 - The bean:write tag filters HTML chars
 - There is no EL equivalent of bean:message
 - The EL is available only in JSP 2.0 compliant servers
 - E.g., Tomcat 5 or WebLogic 9, not Tomcat 4 or WebLogic 8.x
 - JSP 2.0 compliance list: <http://theserverside.com/reviews/matrix.tss>
- **To use the EL, use servlet 2.4 version of web.xml:**
 - You also must remove the taglib entries

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
  version="2.4">
...
</web-app>
```

35

Apache Struts:Beans

www.coreservlets.com

Struts Example: Confirmation Page (Classic Style)

```
...
Congratulations. You are now signed up for the
Single Provider of Alert Memos network!
<%@ taglib uri="http://struts.apache.org/tags-bean"
  prefix="bean" %>
<UL>
  <LI>First name:
    <bean:write name="contactFormBean" property="firstName"/>
  <LI>Last name:
    <bean:write name="contactFormBean" property="lastName"/>
  <LI>Email address:
    <bean:write name="contactFormBean" property="email"/>
  <LI>Fax number:
    <bean:write name="contactFormBean" property="faxNumber"/>
</UL>
...
```

36

Apache Struts:Beans

www.coreservlets.com

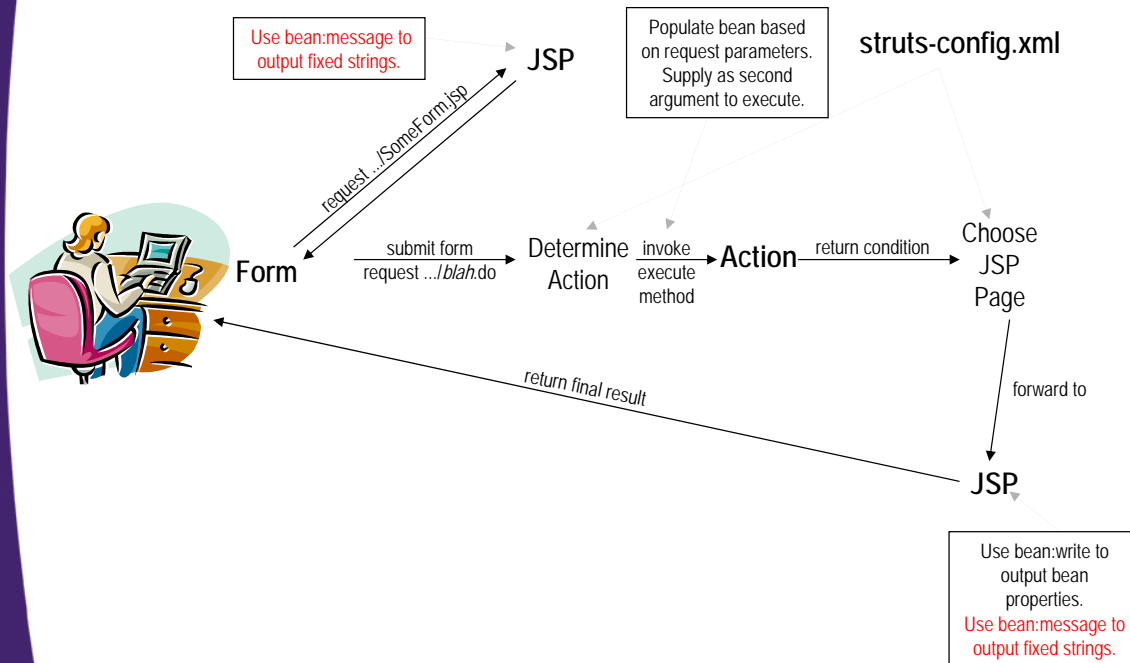
Struts Example: Confirmation Page (JSP 2.0 Style)

```
...
Congratulations. You are now signed up for the
Single Provider of Alert Memos network!
<UL>
  <LI>First name: ${contactFormBean.firstName}
  <LI>Last name: ${contactFormBean.lastName}
  <LI>Email address: ${contactFormBean.email}
  <LI>Fax number: ${contactFormBean.faxNumber}
</UL>
...
```



Using Properties Files (Resource Bundles)

Struts Flow of Control



39

Apache Struts:Beans

www.coreservlets.com

New Steps

- 1. Create a properties file in WEB-INF/classes**
 - E.g., WEB-INF/classes/MessageResources.properties
- 2. Define strings in the properties file**
 - some.key1=first message
 - some.key2=second message
 - some.key3=some parameterized message: {0}
- 3. Load the properties file in struts-config.xml**
 - `<message-resources parameter="MessageResources"/>`
- 4. Output the messages in JSP pages**
 - Load the tag library
 - `<% @ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>`
 - Output the messages using bean:message
 - First message is `<bean:message key="some.key1"/>`
 - Second: `<bean:message key="some.key2"/>`
 - Third: `<bean:message key="some.key3" arg0="replacement"/>`

40

Apache Struts:Beans

www.coreservlets.com

Advantages of Properties Files

- **Centralized updates**
 - If a message is used in several places, it can be updated with a single change.
 - This is consistent with the Struts philosophy of making as many changes as possible in config files, not in Java or JSP code.
- **I18N**
 - If you use messages pervasively in your JSP pages, you can internationalize your application by having multiple properties files corresponding to the locale, as with standard I18N in Java
 - MessageResources.properties
 - MessageResources_jp.properties
 - MessageResources_es.properties
 - MessageResources_es_mx.properties

41

Apache Struts:Beans

www.coreservlets.com

Example 2: Beans Plus Messages

- **URL**
 - `http://hostname/struts-beans/register2.do`
- **Action Class**
 - BeanRegisterAction
 - Exact same class as previous example, no changes whatsoever
 - Since execute method is unchanged, it still returns "success", "bad-address", or "bad-password"
- **Results pages**
 - The execute method has same return values, but we map them to different results pages
 - `/WEB-INF/results/confirm-registration2.jsp`
 - `/WEB-INF/results/bad-address2.jsp`
 - `/WEB-INF/results/bad-password2.jsp`

42

Apache Struts:Beans

www.coreservlets.com

Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
 - As before, we use the action element (to designate that BeanRegisterAction should handle requests for register2.do).
- **Map return conditions to JSP pages**
 - As before, we use multiple forward elements, one for each possible return value of the execute method
- **Declare any form beans that are being used.**
 - As before, we use a form-bean entry with name and type attributes
- **Declare a properties file**
 - The message-resources element is used to refer to a properties file:
`<message-resources parameter="MessageResources" null="false"/>`
 - The parameter attribute refers to the location of the properties file
 - Relative to WEB-INF/classes and with the .properties file extension implied.
 - E.g., "MessageResources" refers to WEB-INF/classes/MessageResources.properties, and "foo.bar.baz" refers to WEB-INF/classes/foo/bar/baz.properties.
 - The null attribute determines whether missing messages should be flagged. If the value is true, references to nonexistent messages result in empty strings. If the value is false, references to nonexistent messages result in warning messages like ???keyName???

43

Apache Struts:Beans

www.coreservlets.com

Step 1 (Modify struts-config.xml) -- Final Code

```
...
<struts-config>
  <form-beans>
    <form-bean name="userFormBean"
               type="coreservlets.UserFormBean"/>
  </form-beans>
  <action-mappings>
    ...
    <action path="/register2"
            type="coreservlets.BeanRegisterAction"
            name="userFormBean"
            scope="request">
      <forward name="bad-address"
                path="/WEB-INF/results/bad-address2.jsp"/>
      <forward name="bad-password"
                path="/WEB-INF/results/bad-password2.jsp"/>
      <forward name="success"
                path="/WEB-INF/results/confirm-registration2.jsp"/>
    </action>
  </action-mappings>
  <message-resources parameter="MessageResources"
                    null="false"/>
</struts-config>
```

44

Apache Struts:Beans

www.coreservlets.com

Step 1 (Properties File)

- **WEB-INF/classes/MessageResources.properties**

```
form.title=Registration
form.emailPrompt=Email Address
form.passwordPrompt=Password
form.buttonLabel=Sign Me Up
form.errorHeading=Error
form.emailError=Illegal email address
form.emailSuggestion=Possible legal address
form.passwordError=Password is too short
form.passwordSuggestion=Possible legal password
form.tryAgain=Please <A HREF="register2.jsp">try again</A>.
form.confirmationTitle=Successful Registration
form.congrats=Congratulations
```

Steps 2, 3, and 4

- **Define a form bean.**
 - This example uses the same UserFormBean as the previous example.
- **Create results beans.**
 - This example uses the same SuggestionBean and SuggestionUtil classes as the previous example.
- **Define an Action class to handle requests.**
 - This example uses the same BeanRegisterAction class as the previous example.
 - However, the struts-config.xml file associates the Action with a different incoming URL (register2.do instead of register1.do), and associates the return values with different JSP pages.

Step 5 (Create Form That Invokes *blah.do*)

- Instead of directly listing headings, prompts, and textual messages, they are taken from properties file
 - That way, if the messages change (or if you have multiple versions in different languages), the messages can be updated without modifying the actual JSP pages.
 - Also, some of the prompts are used in more than one page, so extracting the prompts from the properties file limits changes to one location, even though the prompts are used in multiple locations.
 - In a later example, we will internationalize the application by making a Spanish-language version of the properties file (but changing nothing else in the entire app!)

Step 5 (Create Form That Invokes *blah.do*)

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>
    <bean:message key="form.title"/>
</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1><bean:message key="form.title"/></H1>
<FORM ACTION="register2.do" METHOD="POST">
    <bean:message key="form.emailPrompt"/>:
    <INPUT TYPE="TEXT" NAME="email"><BR>
    <bean:message key="form.passwordPrompt"/>:
    <INPUT TYPE="PASSWORD" NAME="password"><BR>
    <INPUT TYPE="SUBMIT"
        VALUE="<bean:message key="form.buttonLabel"/>">
</FORM>
</CENTER>
</BODY></HTML>
```


Step 6 (Display Results in JSP) First Possible Page

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>

<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>
    <bean:message key="form.errorHeading"/>
</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1><bean:message key="form.errorHeading"/></H1>
<UL>
    <LI><bean:message key="form.emailError"/>:
        <bean:write name="userFormBean" property="email"/>
    <LI><bean:message key="form.emailSuggestion"/>:
        <bean:write name="suggestionBean" property="email"/>
</UL>
<bean:message key="form.tryAgain"/>
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Second Possible Page

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>

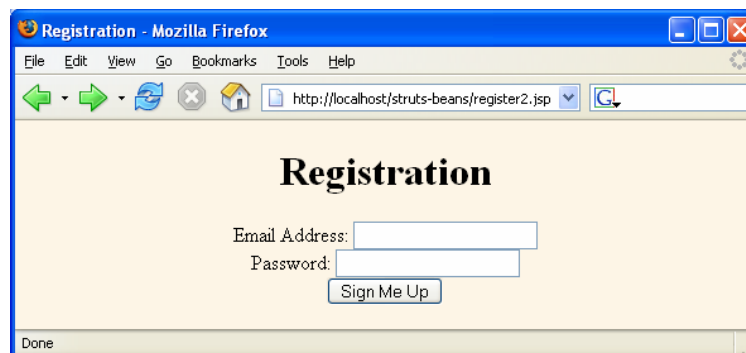
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>
    <bean:message key="form.errorHeading"/>
</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1><bean:message key="form.errorHeading"/></H1>
<UL>
    <LI><bean:message key="form.passwordError"/>:
        <bean:write name="userFormBean" property="password"/>
    <LI><bean:message key="form.passwordSuggestion"/>:
        <bean:write name="suggestionBean" property="password"/>
</UL>
<bean:message key="form.tryAgain"/>
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Third Possible Page

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
    prefix="bean" %>

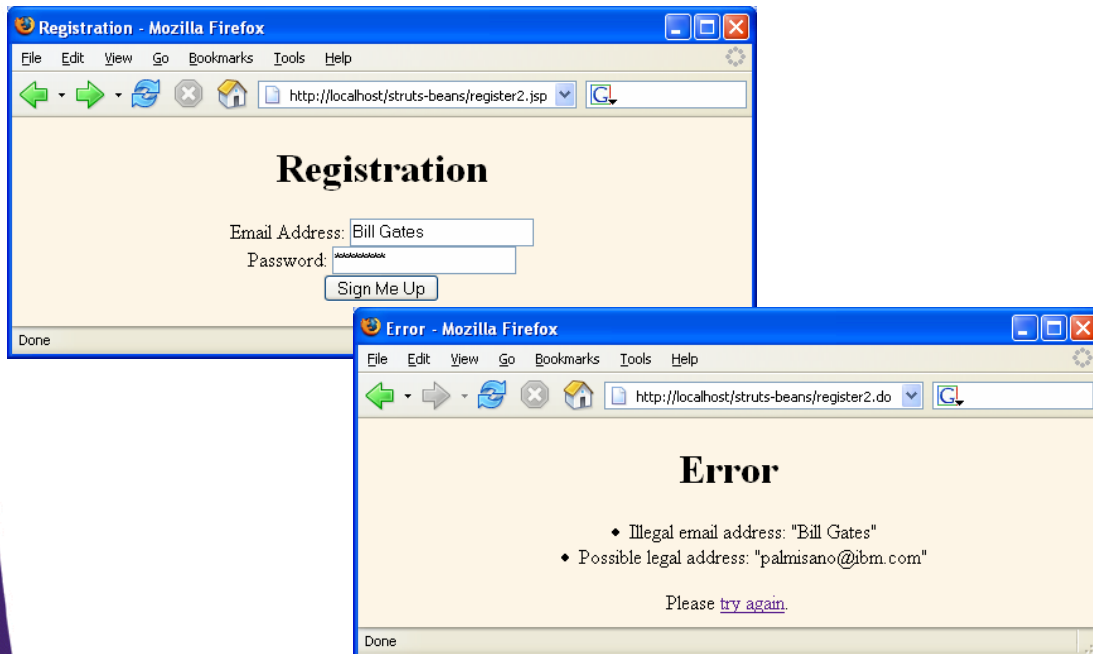
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>
    <bean:message key="form.confirmationTitle"/>
</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1><bean:message key="form.confirmationTitle"/></H1>
<UL>
    <LI><bean:message key="form.emailPrompt"/>:
        <bean:write name="userFormBean" property="email"/>
    <LI><bean:message key="form.passwordPrompt"/>:
        <bean:write name="userFormBean" property="password"/>
</UL>
<bean:message key="form.congrats"/>.
</CENTER>
</BODY></HTML>
```

Example 2: Results (Initial Form)

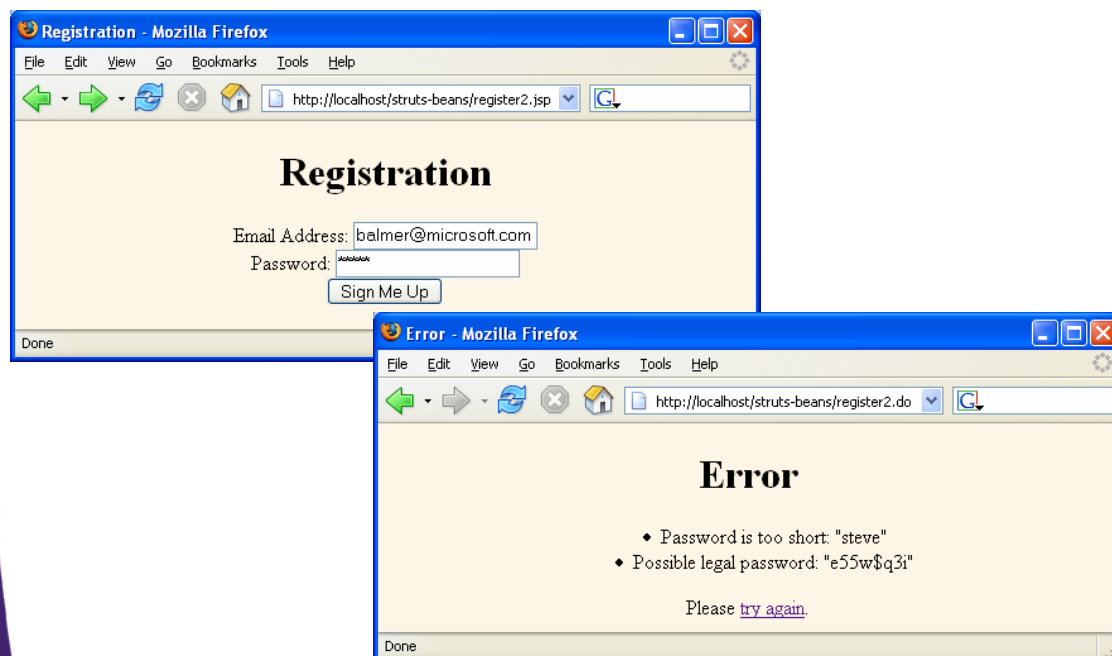


The screenshot shows a Mozilla Firefox browser window titled "Registration - Mozilla Firefox". The address bar displays "http://localhost/struts-beans/register2.jsp". The page content features a heading "Registration" in a large, bold, serif font. Below the heading, there are two labels: "Email Address:" and "Password:", each followed by a text input field. A "Sign Me Up" button is positioned below the password field. The status bar at the bottom of the browser window shows the word "Done".

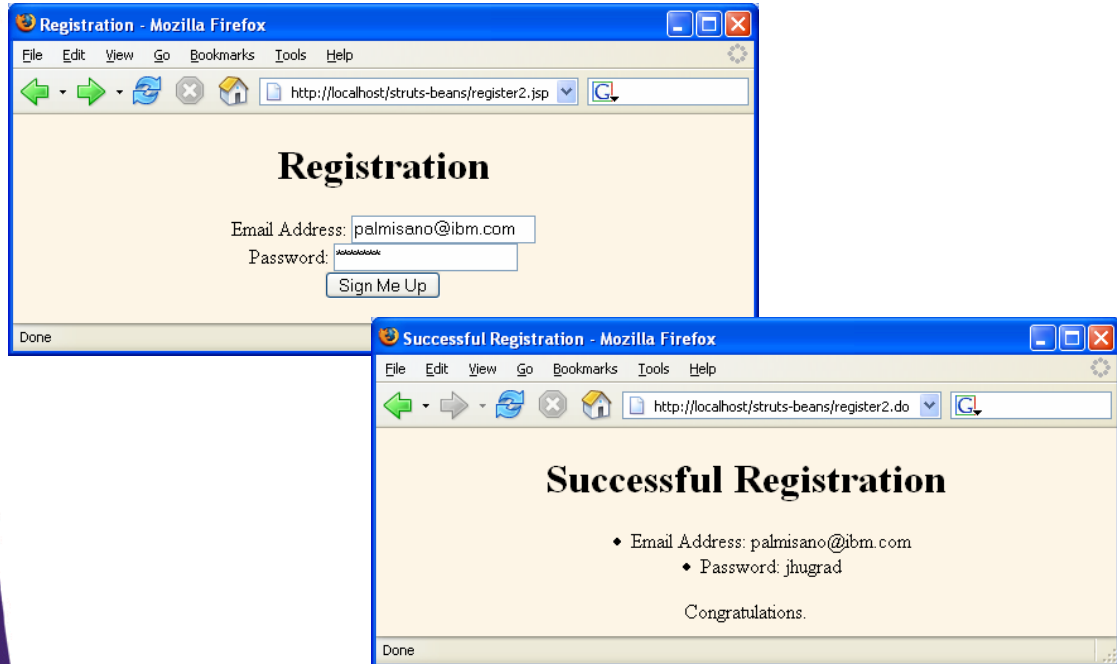
Example 2: Results (Illegal Address)



Example 2: Results (Illegal Password)



Example 2: Results (Legal Address and Password)



Internationalization (I18N)

Loading Alternate Properties Files

- **Default properties file**
 - When you say
`<message-resources parameter="someName" ...>`
WEB-INF/classes/someName.properties is loaded and treated as the default file
- **More specific properties file**
 - The system automatically looks for additional, specialized files corresponding to your Locale
 - someName_es.properties, someName_es_mx.properties, etc.
 - Entries from more specific file override entries from default file
 - Locale is automatically determined by browser language settings
 - Locale can also be set explicitly (e.g., based on incoming checkbox value) in an Action with setLocale

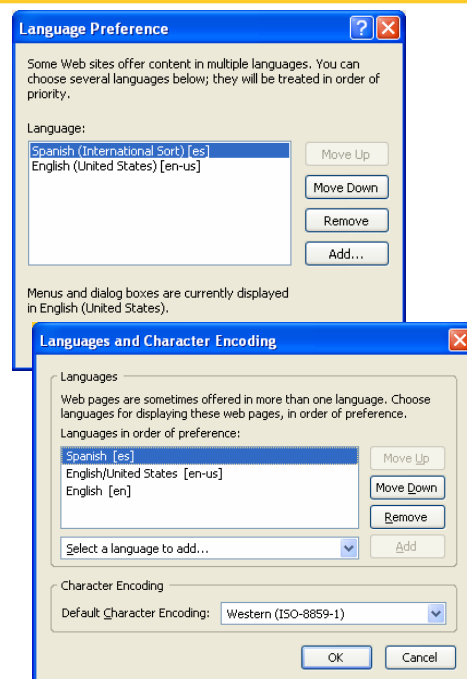
57

Apache Struts:Beans

www.coreservlets.com

Setting Language Preferences in Browsers

- **Internet Explorer**
 - Tools, Internet Options, Languages
 - Click Add, select language, OK
 - Move to top of list using "Move Up"
- **Firefox**
 - Tools, Options, Languages
 - Click Add, select language, Add
 - Move to top of list using "Move Up"



58

Apache Struts:Beans

www.coreservlets.com

Internationalizing the Registration Example for English or Spanish

- **Need MessageResources_**es**.properties**

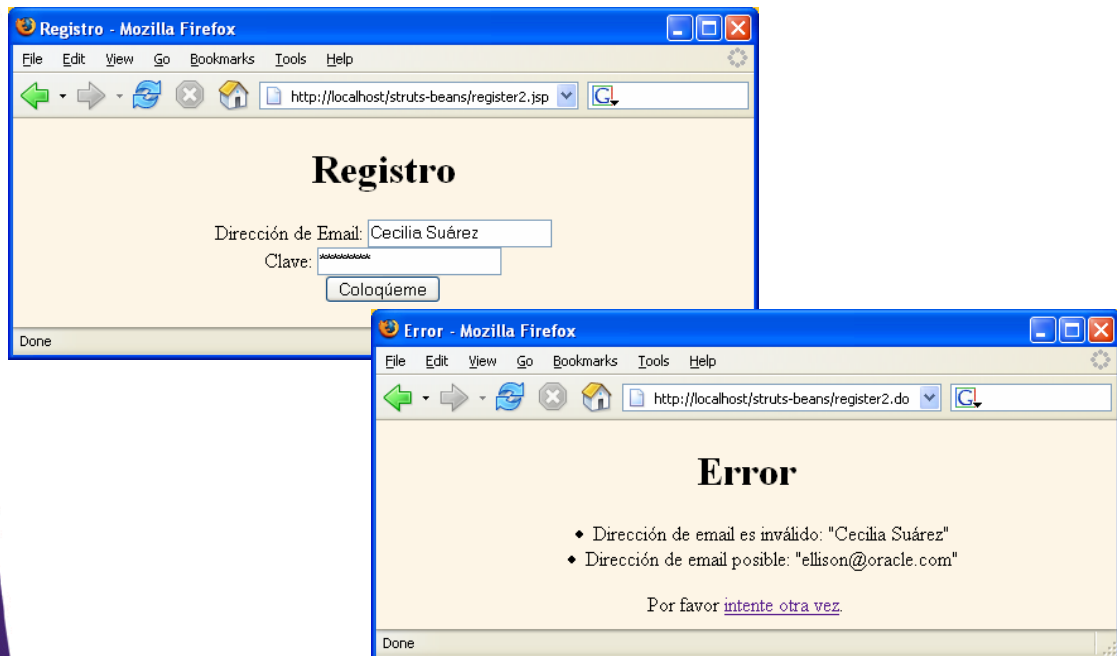
- This is *all* that is needed.
 - No changes to any config file or code!

```
form.title=Registro
form.emailPrompt=Dirección de Email
form.passwordPrompt=Clave
form.buttonLabel=Coloqueme
form.errorHeading=Error
form.emailError=Dirección de email es inválido
form.emailSuggestion=Dirección de email posible
form.passwordError=Clave es demasiado corto
form.passwordSuggestion=Clave posible
form.tryAgain=Por favor <A HREF="register2.jsp">intente otra
vez</A>.
form.confirmationTitle=Registro Acertado
form.congrats=Felicitaciones
```

Example 3: Results (Initial Form)



Example 3: Results (Illegal Address)

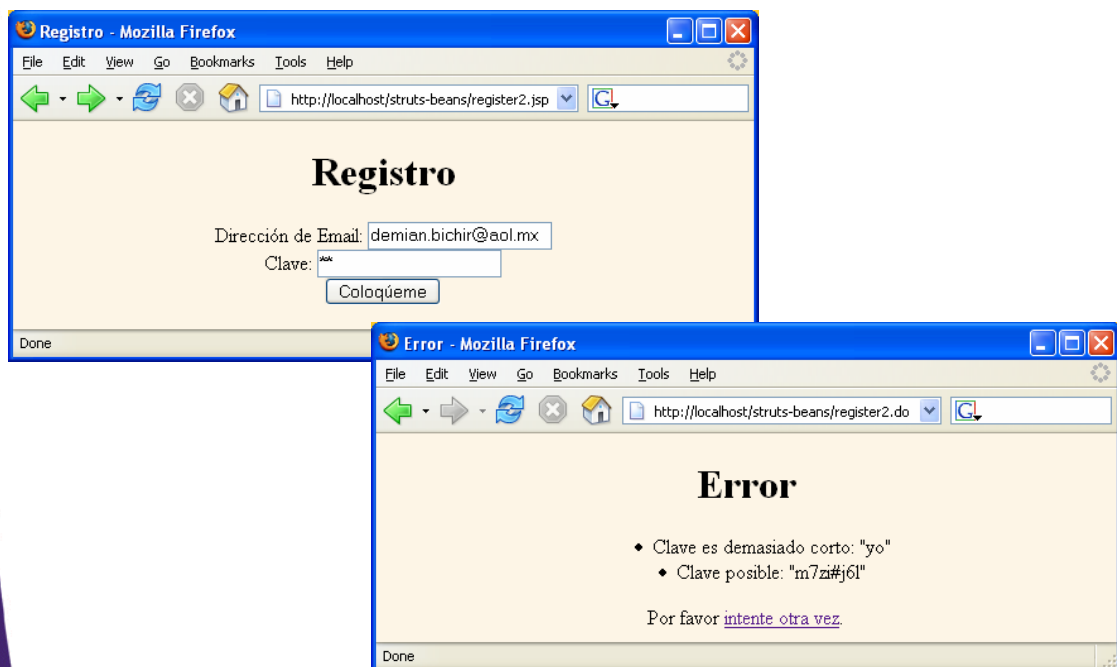


61

Apache Struts:Beans

www.coreservlets.com

Example 3: Results (Illegal Password)

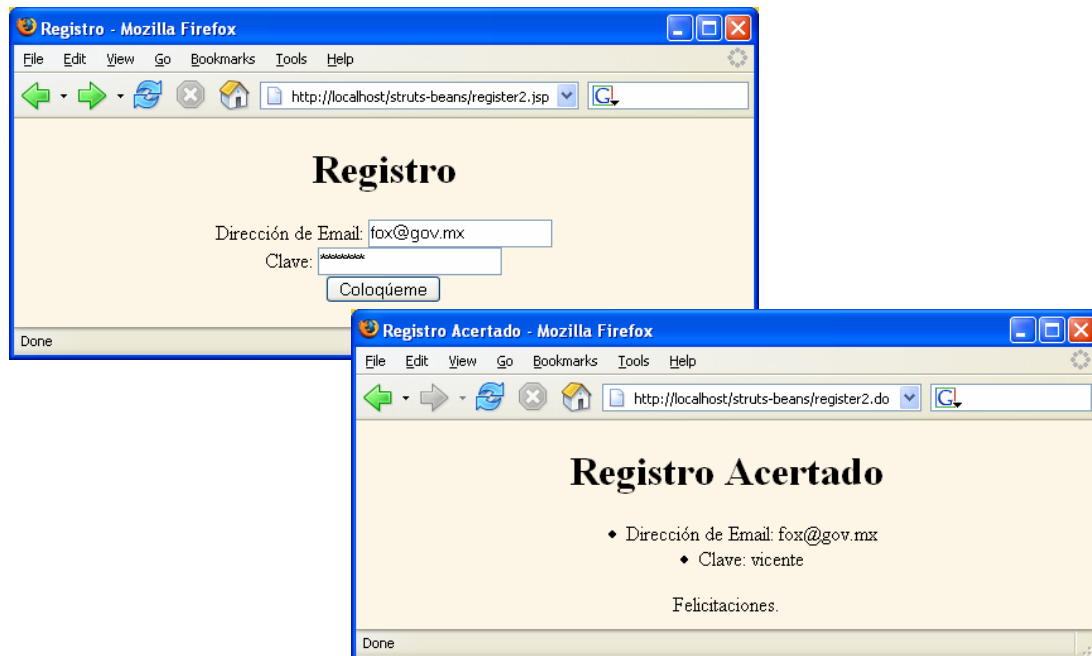


62

Apache Struts:Beans

www.coreservlets.com

Example 3: Results (Legal Address and Password)



63

Apache Struts:Beans

www.coreservlets.com

Summary

- **Struts will automatically create and populate a form bean representing incoming request data**
 - Form beans extend ActionForm and are declared with form-bean in struts-config.xml
 - Access a form bean by casting the ActionForm argument of execute to the concrete class
 - Properties that match request params are automatically filled in
 - Results beans extend no particular classes
- **Output bean properties with bean:write**
 - Both form and results beans can be output with bean:write (or the JSP 2.0 expression language)
- **Output constant strings with bean:message**
 - Load properties files with message-resources
 - I18N via specialized properties files

64

Apache Struts:Beans

www.coreservlets.com



Questions?

Core Servlets & JSP book: www.coreservlets.com
More Servlets & JSP book: www.moreservlets.com
Servlet, JSP, Struts, JSF, and Java Training Courses:
courses.coreservlets.com

Slides © Marty Hall, <http://www.coreservlets.com>, books © Sun Microsystems Press