Jakarta Struts: Handling Request Parameters with Form Beans

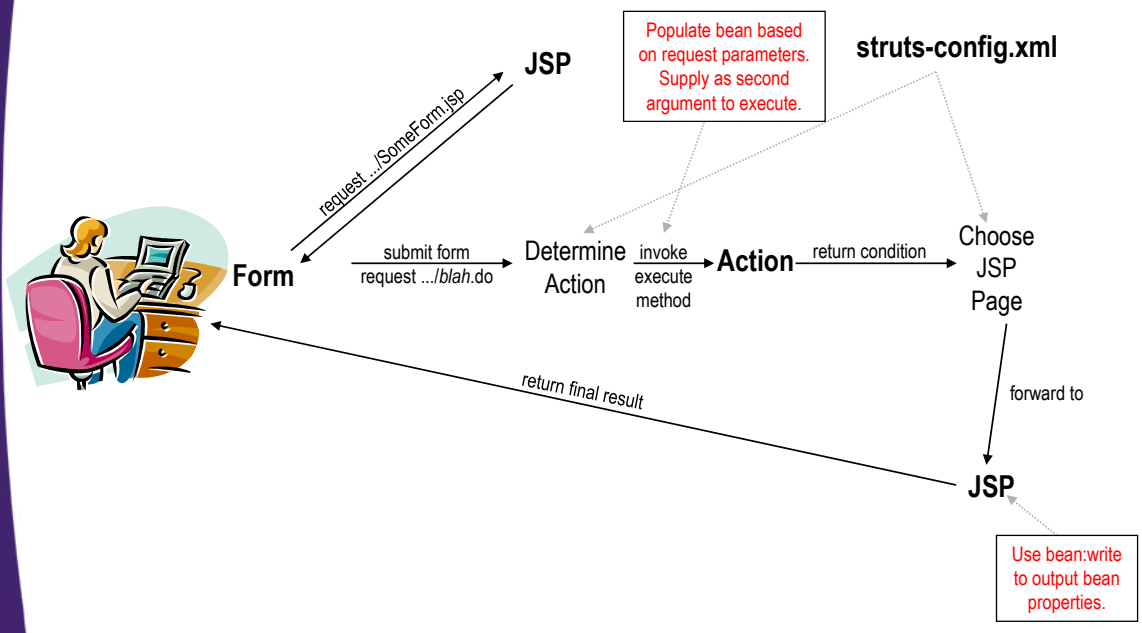
Struts 1.*x* Version

© 2010 Marty Hall

Agenda

- **Two new ideas**
  - Automatically creating bean to represent request data
  - Using bean:write to output bean properties
- **Defining form beans**
- **Declaring form beans**
- **Outputting properties of form beans**
  - bean:write
  - JSP 2.0 expression language
- **Defining and outputting regular beans**

Struts Flow of Control



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 also use bean:message to output fixed strings

New Capabilities

- **Defining a form bean**
  - 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
- **Outputting bean properties**
  - You can use the Struts bean:write tag to output bean properties in JSP pages.

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.**
   - For now, we will use static HTML
     - Later, we will 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
     - Later, we will also use bean:message to output fixed strings from a properties file
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>
```

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 set*Blah* method corresponding to each incoming request parameter named *blah*. The properties should be of type String (i.e., each set*Blah* 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 get*Blah* method corresponding to each bean property that you want to display in JSP without using Java syntax.

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;
  }
}
```

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.

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 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

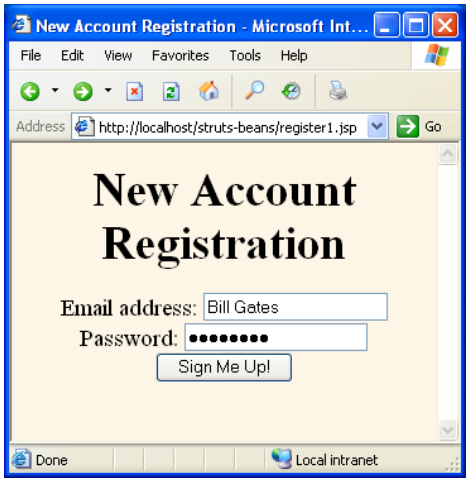
```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Password</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Password</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
The password
"<bean:write name="userFormBean" property="password"/>"
is too short; it must 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

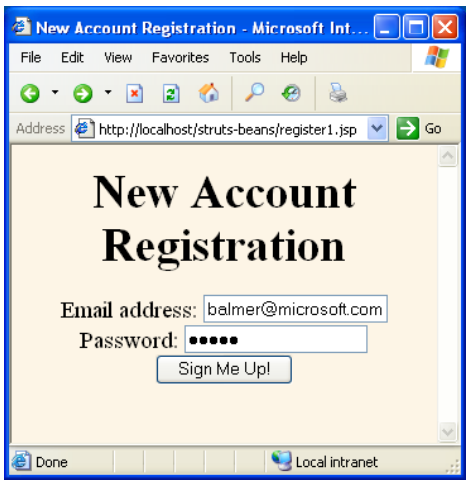
```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Success</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>You have registered successfully.</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
<UL>
  <LI>Email Address:
      <bean:write name="userFormBean" property="email"/>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


Example 1: Results (Initial Form)

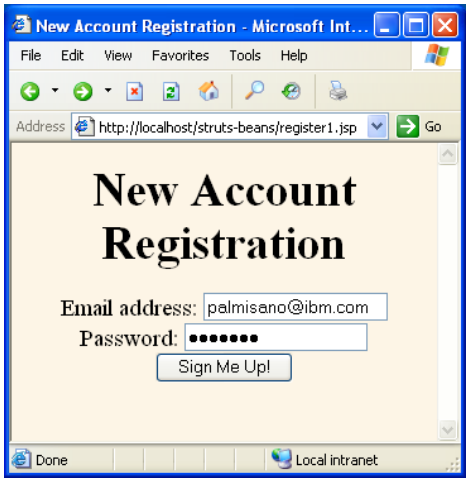
Example 1: Results (Illegal Address)

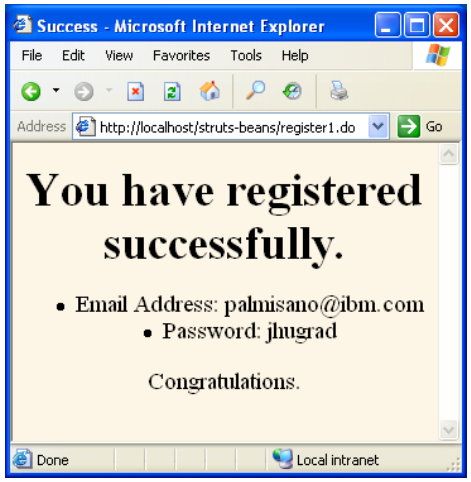


Example 1: Results (Illegal Password)



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>
```

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>
…
```


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>
…
```

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
    - You can also use the JSP 2.0 expression language if you do not need other Struts capabilities and if you update the web.xml declaration

Questions?