# Apache Struts: Prepopulating and Redisplaying Input Forms

## 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**




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

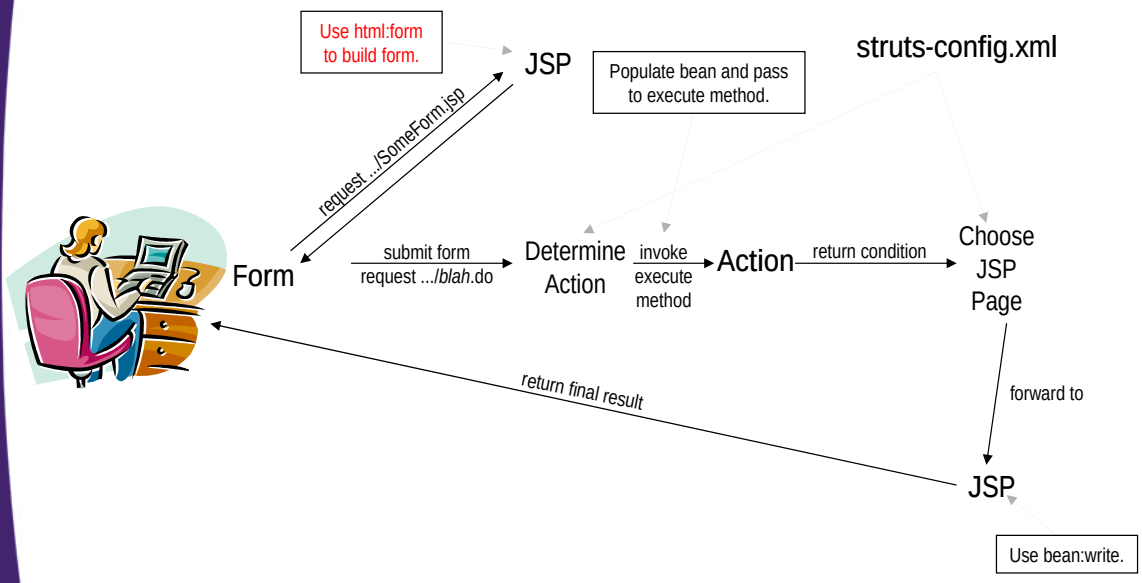
Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

# Agenda

- **Using the Struts html: tags to build HTML forms that have three important characteristics:**
  - The textfield names and the bean properties are guaranteed to stay in synch
  - The textfield values can be prepopulated based on the values in a bean.
    - That is, the initial values of the form elements can be taken from a Java object.
  - The forms can be redisplayed when they are submitted with incomplete or incorrect values.
    - Specifically, when they are redisplayed, they can maintain the values that the end user already entered.



# Struts Flow of Control



# Struts Flow of Control

- **The user requests a form**
  - This form is built with html:form, html:text, and similar elements
    - Keeps input field names in synch with bean property names
    - Lets initial values of textfields and other input elements come from the app
- **The form is submitted to a URL of the form *blah*.do.**
  - That address is mapped by struts-config.xml to an Action object,
- **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 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



# The Six Basic Steps in Using Struts: Updates

1. **Modify struts-config.xml. Use WEB-INF/struts-config.xml to:**
   - Map incoming .do addresses to Action objects
   - Map return conditions to JSP pages
     - If a mapping appears in two places, it goes in global-forwards
   - Declare any form beans that are being used.
   - Restart server after modifying struts-config.xml.
2. **Define a form bean.**
   - Still extends ActionForm and has a bean property for each incoming request parameter
   - In addition to being used in the execute method of the Action and in the final JSP pages, the bean will also be used in the initial input form to give names and values to the various input elements.

# The Six Basic Steps in Using Struts: Updates

3. **Create results beans.**
   - These are normal beans of the sort used in MVC when implemented directly with RequestDispatcher, and are created and used in the same way as described in the previous section.
4. **Define an Action class to handle requests.**
   - As in the previous section, rather than calling request.getParameter explicitly, 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

5. **Create form that invokes blah.do.**
   - Rather than using the standard HTML FORM and INPUT tags, we now use html:form and html:text (and related tags).
   - The html:form tag associates a bean with the form, and html:text automatically uses bean property names for each textfield NAME and bean property values for each textfield VALUE.
   - In addition, as in the previous section, this form can still use the bean:message tag to output standard messages and text labels.
6. **Display results in JSP.**
   - As before, the JSP page uses bean:write to output properties of the form and result beans.
   - It may also use bean:message to output standard messages and text labels that are defined in a properties file.

# New Techniques

- **Using html:form to declare form in initial JSP page.**
  - The action should *exactly* match path attribute of action element in struts-config.xml
- **Using the Struts html:form element instead of the standard HTML FORM element yields four results:**
  - A bean is associated with the form.
    - A bean of the type specified by form-bean is automatically used
  - The Web application prefix is prepended automatically.
    - You say <html:form action="/actions/..."> to get <FORM ACTION="**/webAppPrefix**/actions/..." ...>.
  - The .do suffix is appended automatically.
    - You say <html:form action="/actions/blah"> to get <FORM ACTION="/webAppPrefix/actions/blah**.do**" ...>.
  - POST, not GET, is the default METHOD.
    - You say <html:form action="/actions/blah"> to get <FORM ACTION="/webAppPrefix/actions/blah.do" **METHOD="POST"**>.



# New Techniques

- **Using html:text and similar elements to declare the input fields of the form.**
  - The NAME of each input field is taken from the bean property name, and the VALUE is taken from the bean property value. For example, using
    **<html:text property="firstName"/>**
    is equivalent to first declaring a bean of the appropriate type, then doing
    **<INPUT TYPE="TEXT" NAME="firstName"**
    **VALUE="<%= theBean.getFirstName() %>">.**
  - Not only does this provide initial values for your form fields, but it also makes it easier for you to be sure that the field names match the bean property names. Since, in the execute method of the Action object, the form bean is filled in by matching up request parameter names with bean property names, it is critical that the names stay in synch.

# Prepopulating Forms

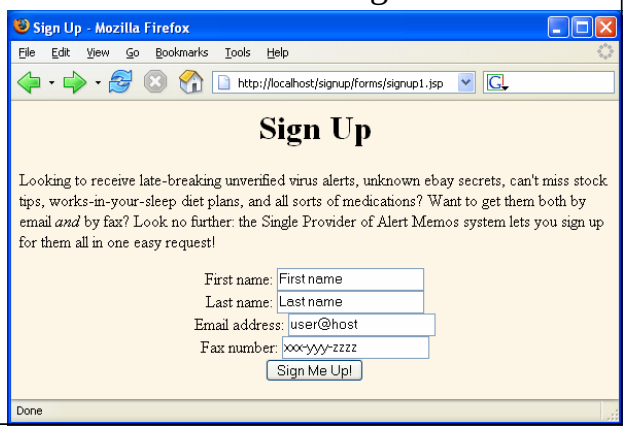


# Prepopulating Forms

- **In many cases, you want the initial form to be based on data defined in your application.**
  - If the application data changes, you want the initial values of the form fields to change automatically.
- **To implement this behavior:**
  - **Use html: form for main form.**
    - In the input form, you should use html:form, and should specify action="/path/blah", not action="*/webAppPrefix*/path/blah.*do*". Also, POST is the default method for html:form.
  - **Use html:text for textfields.**
    - In your input form, you should use <html:text property="propertyName"/> to declare input textfields. Each textfield NAME will be taken from the bean property name, and each textfield VALUE will come from the bean property value.
  - **Use html:xxxx for other input elements.**
    - In your input form, use html:button, html:checkbox, html:textarea, etc., to declare submit buttons, checkboxes, text areas, etc.

# Example 1: Signing up for Alerts/Notifications

- **URL**
  - http://*hostname*/signup/actions/signup1.do
- **Action Class**
  - SignupAction1
    - Uses a ContactFormBean that is automatically filled in
    - The execute method returns "success" or "missing-value"
- **Results pages**
  - /WEB-INF/results/confirmation.jsp
  - /WEB-INF/results/missing-value.jsp






# Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
  - As before, we use the action element (to designate that SignupAction1 should handle requests for signup1.do).
- **Map return conditions to JSP pages**
  - As before, we use multiple forward elements, one for each possible return value of the execute method
  - However, since two *different* Actions will eventually use the *same* condition to indicate the *same* JSP page, the repeated mapping will go in global-forwards.

```
<global-forwards>
  <forward name="success"
           path="/WEB-INF/results/confirmation.jsp"/>
</global-forwards>
```

- **Declare any form beans that are being used.**
  - As before, we use name and type attributes within form-bean
  - As before, we also add name (the bean name as given in form-bean) and scope (request) attributes to the action declaration.

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

```
<struts-config>
  <form-beans>
    <form-bean name="contactFormBean"
               type="coreservlets.ContactFormBean"/>
  </form-beans>
  <global-forwards>
    <forward name="success"
             path="/WEB-INF/results/confirmation.jsp"/>
  </global-forwards>
  <action-mappings>
    <action path="/actions/signup1"
            type="coreservlets.SignupAction1"
            name="contactFormBean"
            scope="request">
      <forward name="missing-value"
               path="/WEB-INF/results/missing-value.jsp"/>
    </action>
    ...
  </action-mappings>
</struts-config>
```



# Step 2 (Define a Form Bean)

- **Same behavior in the Action**
  - The form bean extends ActionForm, is automatically filled in with the incoming form parameters, and is passed to the execute method of the Action
- **Also used in the form**
  - In addition to being used in the Action, the html:form tag indicates that a new instance of the form bean will be created and used to fill in the fields of the input form
    - The NAME of the input field comes from the bean property name and the VALUE comes from the bean property value.
    - For example, the following slide shows a form bean corresponding to contact information for a person that will be added to an email/fax list in our application. It contains properties for first name, last name, email address, etc.

# Step 2 (Define a Form Bean) – ContactFormBean Basic Properties

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

public class ContactFormBean extends ActionForm {
  private String firstName = "First name";
  private String lastName = "Last name";
  private String email = "user@host";
  private String faxNumber = "xxx-yyy-zzzz";

  public String getFirstName() {
    return(firstName);
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }
  ... // lastName, emailAddress
```



# Step 2 (ContactFormBean), Cont. – Code for Checking Missing Values

```
  ...
private String[] defaultValues =
  { firstName, lastName, email, faxNumber };

public boolean isMissing(String value) {
  if ((value == null) || (value.trim().equals(""))){
    return(true);
  } else {
    for(int i=0; i<defaultValues.length; i++) {
      if (value.equals(defaultValues[i])) {
        return(true);
      }
    }
    return(false);
  }
}
```

# Step 3 (Create Results Beans)

- **The form bean represents the input data**
  - I.e., the data that came from the HTML form.
- **The results beans represent the output data**
  - I.e., the data created by the business logic to represent the results of the computation or database lookup.
- **The next slide shows a bean (MessageBean) that will be used to store simple error messages.**



# Step 3 (Create Results Beans) – Code for MessageBean

```
package coreservlets;

public class MessageBean {
  private String message = "";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

# Step 4 (Define an Action Class to Handle Requests)

- **In this example, we check each of the required input parameters (first name, last name, email address, and fax number) to see if it is missing.**
  - If it is missing, we use mapping.findForward to return a "missing-value" condition, which is mapped by struts-config.xml to an error page that contains a message saying which parameter was missing.
  - If all parameters are present, we return "success", which results in an order-confirmation page.
- **This example is similar to the previous one.**
  - Again, we do not call request.getParameter explicitly, but instead extract the request parameters from the already populated form bean. Specifically, we take the ActionForm argument supplied to execute, cast it to ContactFormBean (our concrete class that extends ActionForm), and then call getter methods on that bean.
  - Also, we pass the error message to the JSP page by creating a MessageBean and storing it in request scope.
    - It is also common to store results beans in a field of the form bean



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

```
public class SignupAction1 extends Action {
  public ActionForward
                 execute(ActionMapping mapping,
                         ActionForm form,
                         HttpServletRequest request,
                         HttpServletResponse response)
      throws Exception {
    ContactFormBean userBean = (ContactFormBean)form;
    String firstName = userBean.getFirstName();
    String lastName = userBean.getLastName();
    String email = userBean.getEmail();
    String faxNumber = userBean.getFaxNumber();
    if (userBean.isMissing(firstName)) {
      makeWarning(request, "first name");
      return(mapping.findForward("missing-value"));
    }
    ...
    } else {
      return(mapping.findForward("success"));
    }
  }
```

# Step 4 (Define an Action Class to Handle Requests) – Code Cont.

```
  protected void makeWarning(HttpServletRequest request,
                             String message) {
    MessageBean messageBean = new MessageBean();
    messageBean.setMessage(message);
    request.setAttribute("messageBean", messageBean);
  }
}
```



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

- **This form is very different from the ones in the previous examples.**
  - Instead of using the standard HTML FORM tag, we import and use the html:form tag. The Web application prefix, the .do suffix, and the POST method are all generated automatically. So, we use

```
<%@ taglib uri="http://struts.apache.org/tags-html"
           prefix="html" %>
<html:form action="/actions/signup1">
  ...
</html:form>
```

  - to get something equivalent to:

```
<FORM ACTION="/signup/actions/signup1.do"
      METHOD="POST">
  ...
</FORM>
```

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

- **Bean corresponding to the action is used**
  - Since the action of html:form matches the path attribute of the action declaration in struts-config.xml, a bean is automatically created when the input page is accessed.
    - The type of bean created is determined by looking at the name that goes with the action, and finding the form-bean with the same name.
    - In this case, the bean used is a ContactFormBean
- **Bean names/properties are used for form fields**
  - After declaring the form and associating it with a bean, we use html:text to build input elements whose NAME and VALUE are taken from the form bean property names and values.



# Step 5 (Create Form that Invokes *blah*.do), html:text Details

- **The following**

  First name: **<html:text property="firstName"/>**

**results in something similar to:**

```
<jsp:useBean id="contactBean" scope="request"
             class="coreservlets.ContactFormBean"/>
First name:
<INPUT TYPE="TEXT" NAME="firstName"
       VALUE="${contactBean.firstName}">
```

  - (And even worse if JSP 2.0 were not available)

# Step 5 (Create Form that Invokes *blah*.do) – Final Code

```
… the
Single Provider of Alert Memos
system lets you sign up for them all in one easy
request!
<P>
<CENTER>
<%@ taglib uri="http://struts.apache.org/tags-html"
           prefix="html" %>
<html:form action="/actions/signup1">
  First name: <html:text property="firstName"/><BR>
  Last name: <html:text property="lastName"/><BR>
  Email address: <html:text property="email"/><BR>
  Fax number: <html:text property="faxNumber"/><BR>
  <html:submit value="Sign Me Up!"/>
</html:form>
</CENTER>
</BODY></HTML>
```



# Step 6 (Display Results in JSP)

- **In this example, there are two possible JSP pages:**
  - One for missing input
  - One for success.
- **We use the MessageBean to customize the error messages in the page that is used for missing input.**
  - That way, there does not need to be a separate page for each type of error.
- **As in the previous example, bean:write is used to output bean properties**
  - without having to resort to explicit Java code

# Step 6 (Display Results in JSP) -- First Possible Page

```
<!DOCTYPE ...>
<HTML>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
<HEAD><TITLE>Missing
<bean:write name="messageBean" property="message"/>
</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H2>Missing
<bean:write name="messageBean" property="message"/>!
</H2>
Please <A HREF="../forms/signup1.jsp">try again</A>.
</CENTER>
</BODY></HTML>
```

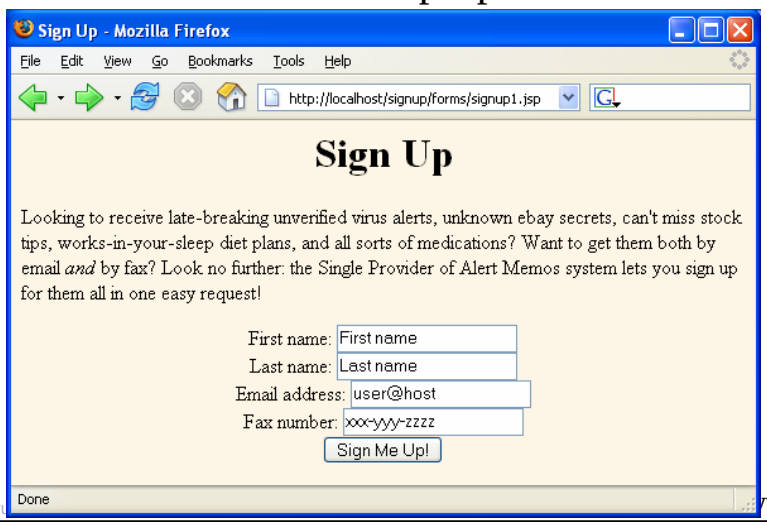


# Step 6 (Display Results in JSP) -- Second Possible Page

```
…<H1>Confirmation</H1>
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>
To be removed from the network, send email
<A HREF="mailto:blackhole@spam-network.com">here</A>.
</CENTER>
</BODY></HTML>
```

# Example 1: Results

- **First, the HTML form is invoked with the URL http://localhost/signup/forms/signup1.jsp.**
  - Notice that the textfields are prepopulated with the values of the ContactFormBean properties.



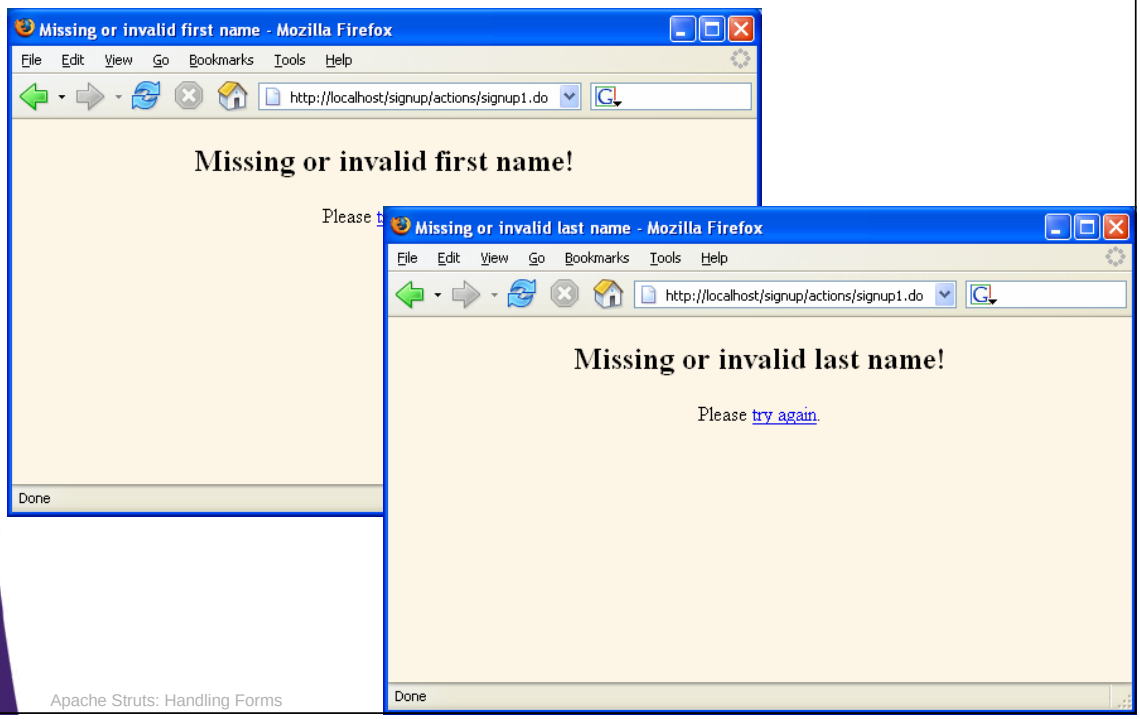




# Example 1: Results

- **Usage of html:form**
  - The action in the html:form element is /actions/signup1
  - Resultant URL is http://localhost/signup/actions/signup1.do.
- **Action processing**
  - This address is mapped by struts-config.xml to the SignupAction1 class, whose execute method is invoked.
  - This method examines the first name, last name, email address, and fax number to see if any are missing.
    - If so, it calls makeWarning to store an error message in a request-scoped bean and returns mapping.findForward with a value of "missing-value".
    - Otherwise it returns "success"
  - The return values are mapped by struts-config.xml to /WEB-INF/results/missing-value.jsp or /WEB-INF/results/confirmation.jsp
  - However, since the MessageBean stores the specific error message, the error page is different in each of the error cases.

# Example 1: Results for Missing Data






# Example 1: Results for Complete Data

# Design Strategy: URLs. Option 1: No Subdirectories

- Action path gives a name only:
```
<action path="/signup1"...>
```
- Form placed in top-level Web application directory:
  - http://*hostname*/signup/signup1.jsp
- Form specifies top-level name:
```
<html:form action="/signup1" ...>
```



# Design Strategy: URLs. Option 2: Subdirectories

- Action path gives a pseudo-directory name:
```
<action path="/actions/signup1"...>
```
- Form placed in Web application subdirectory:
  - http://*hostname*/signup/forms/signup1.jsp
- Form specifies path with pseudo-directory:
```
<html:form action="/actions/signup1.do"
  ...>
```

# Design Strategy: URLs

- **Pros of using directories**
  - For actions
    - Having all action URLs start with a similar path (/actions in this case) makes it easier to distinguish action URLs from other URLs.
    - Having all action URLs start with a similar path makes it much easier to use web.xml to apply filters or Web application security settings to actions.
  - For forms
    - Having all forms in the same physical directory simplifies organization and editing of the forms.
    - Having all forms start with a similar path (/forms in this case) makes it much easier to use web.xml to apply filters or Web application security settings to forms.
- **Cons of using directories**
  - More confusing to beginners
    - Where is the "actions" directory? (There is none)



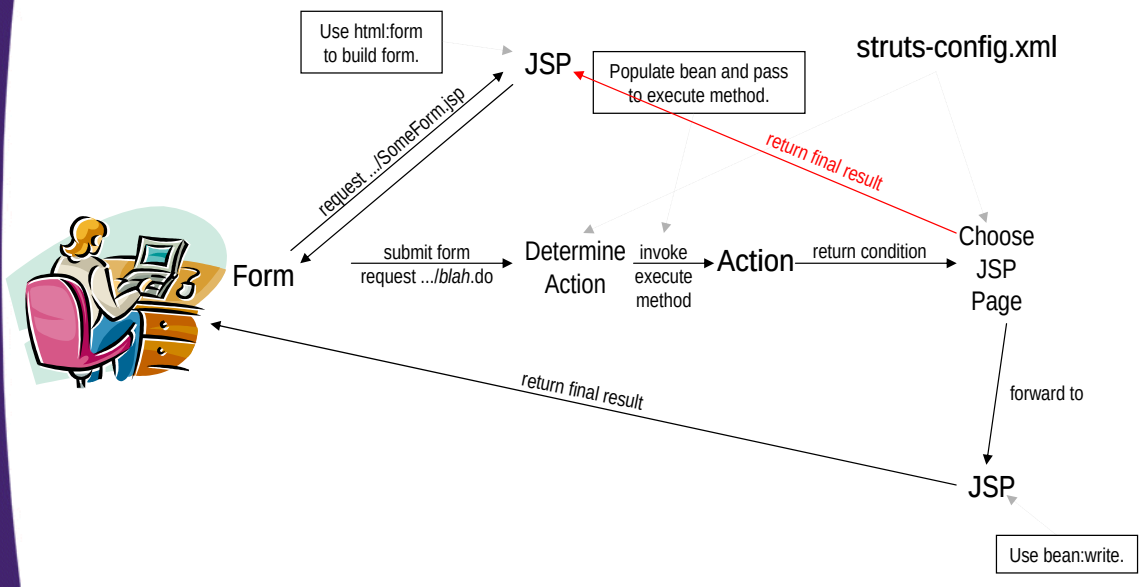
# Redisplaying Forms

# Redisplaying Forms

- **In the previous example, we show how to display a form whose initial values come from a newly-created bean**
- **In this example, we show how to redisplay a form and base its values on bean whose properties have already been filled in**
  - This second technique lets you submit a form, check if all the required data is supplied, and redisplay the form if any of the data is missing.
  - Most importantly, when you redisplay the form, you can maintain the previous values that the end user entered.
  - Implementing this behavior involves two tasks
    1. Mapping an error condition back to the input page
    2. Storing and displaying an error message



# Struts Flow of Control



# Redisplaying Forms: First Task

- **Mapping error condition back to input form**
  - In struts-config.xml, the forward entry corresponding to missing data should supply the address of the input form for the path, rather than supplying the address of a new JSP page as in the previous examples.

```
<forward name="missing-value"
         path="/forms/signup2.jsp"/>
```

  - It is also common to use redirect="true" so that the system uses response.sendRedirect instead of RequestDispatcher.forward to transfer to the input form.
    - Using a redirect is slightly slower since it involves a roundtrip network connection, but it exposes the URL of the input form to the users, which is more familiar since they use that URL when they originally access the form. This approach is illustrated in the section on manual validation.

```
<forward name="missing-value"
         path="/forms/signup2.jsp"
         redirect="true"/>
```



# Redisplaying Forms: Second Task

- **Creating and displaying error messages**
  - When redisplaying your input form, you should display an error message that tells the user which data they failed to supply.
  - The Action can store this message in a separate bean, or, more simply, in a special property of the form bean specifically designed for storing error messages.
  - By making the default value of this bean property be an empty string, you can avoid logic that requires you to distinguish the initial display of the form from a redisplay.

# Second Task, ContactFormBean Code for Storing Error Messages

```
package coreservlets;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class ContactFormBean extends ActionForm {
  private String firstName = "First name";
  private String lastName = "Last name";

  ....

  private String warning = "";

  public String getWarning() {
    return(warning);
  }

  public void setWarning(String baseWarning) {
    this.warning =
      "<H2><FONT COLOR=RED>Missing or illegal " +
      baseWarning + "!</FONT></H2>";
  }

}
```



# Redisplaying Forms: Second Task, Continued

- **Here is an example of how to output the error message.**
  - Notice that we use filter="false" because the error message contains HTML tags.

```
<bean:write name="contactFormBean"
            property="warning"
            filter="false"/>
```

  - Also note that this statement must come *after* the html:form tag, because on initial request, the bean does not exist until html:form is encountered

# Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
  - As before, we use the action element (to designate that SignupAction2 should handle requests for signup2.do).
- **Map return conditions to JSP pages**
  - As before, we use multiple forward elements, one for each possible return value of the execute method. The "success" value is still mapped to /WEB-INF/results/confirmation.jsp.
  - For the "missing-value" entry, we specify the location of the original input form, rather than the location of a new JSP page.

```
<forward name="missing-value"
         path="/forms/signup2.jsp"/>
```

- **Declare any form beans that are being used.**
  - As before, we use name and type attributes within form-bean
  - As before, we also add name (the bean name as given in form-bean) and scope (request) attributes to the action declaration.



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

```
<struts-config>
  <form-beans>
    <form-bean name="contactFormBean"
               type="coreservlets.ContactFormBean"/>
  </form-beans>
  <global-forwards>
    <forward name="success"
             path="/WEB-INF/results/confirmation.jsp"/>
  </global-forwards>
  <action-mappings>
    ...
    <action path="/actions/signup2"
            type="coreservlets.SignupAction2"
            name="contactFormBean"
            scope="request">
      <forward name="missing-value"
               path="/forms/signup2.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

# Step 2 (Define a Form Bean)

- **This example uses the same ContactFormBean as the previous example.**
  - However, in this example we make use of the warning property, which does not correspond to a request parameter but rather is used to send missing-entry-warnings from the Action back to the input form.



# Step 2 (Define a Form Bean)

```
package coreservlets;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class ContactFormBean extends ActionForm {
  private String firstName = "First name";
  private String lastName = "Last name";

  ....

  private String warning = "";

  public String getWarning() {
    return(warning);
  }

  public void setWarning(String baseWarning) {
    this.warning =
      "<H2><FONT COLOR=RED>Missing or illegal " +
      baseWarning + "!</FONT></H2>";
  }

}
```

# Step 3 (Create Results Beans)

- **No new results in confirmation page**
  - Since the confirmation page merely displays the data entered by the user (which is taken from the form bean), the confirmation page uses no results beans.
- **Original form shows warning messages**
  - Instead of a separate warning-message-bean, the warning message is stored inside the form bean itself
    - This is convenient since the form already has access to the form bean, since the html:form tag creates it (for request-scoped beans) or verifies it already exists (for session-scoped beans)



# Step 4 (Define an Action Class to Handle Requests)

- **This example is very similar to the previous one.**
  - However, instead of storing missing-value warning messages in a separate bean that will be displayed in a separate JSP page, the warnings are stored in form-bean, which is used when the original input page is redisplayed.
- **Changes to code**
  - **Same execute method.** We keep the *same* execute method, so we subclass the old Action and inherit execute
  - **New showWarning method.** We replace the showWarning method with a version that stores the warnings in the form bean (intended for the original input page) rather than in the MessageBean (intended for a custom error page).

# Step 4 (Create an Action Object to Handle Requests) -- Code

```
package coreservlets;
import javax.servlet.http.*;

public class SignupAction2 extends SignupAction1 {
  protected void makeWarning(HttpServletRequest request,
                             String message) {
    HttpSession session = request.getSession();
    ContactFormBean contactFormBean =
      (ContactFormBean)session.getAttribute
                                    ("contactFormBean");
    contactFormBean.setWarning(message);
  }
}
```



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

- **This form is very similar to the one in the previous example.**
  - Again, we use the html:form and html:text elements to build an HTML form whose form field values are derived from bean properties.
  - We output a warning message that reminds the user which field they omitted. The default warning message is an empty string, so rather than testing to see if the form is being initially displayed or redisplayed, we always output the error message (with filter="false" because the error message can contain HTML tags).
    - If the user access the form directly, the warning message is empty
    - If the system forwards to the form, the warning message is inserted first

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

```
…
<CENTER>
<%@ taglib uri="http://struts.apache.org/tags-html"
           prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
<html:form action="/actions/signup2">
  <bean:write name="contactFormBean" property="warning"
              filter="false"/>
  First name: <html:text property="firstName"/><BR>
  Last name: <html:text property="lastName"/><BR>
  Email address: <html:text property="email"/><BR>
  Fax number: <html:text property="faxNumber"/><BR>
  <html:submit value="Sign Me Up!"/>
</html:form>
</CENTER>
```

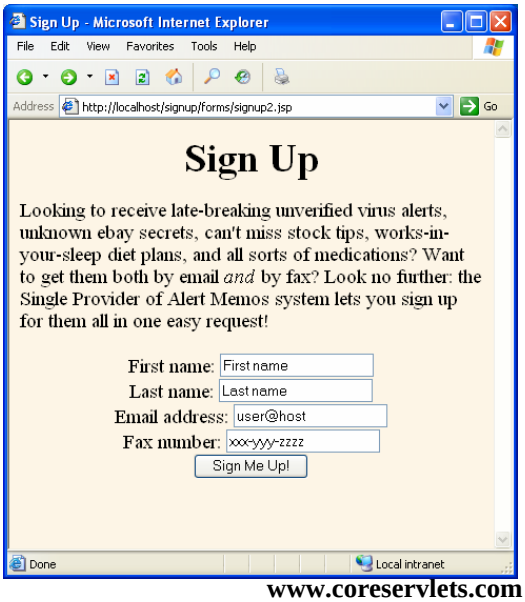


# Step 6 (Display Results in JSP)

- **In this example, there are two possible return values from the execute method of the Action object: "missing-value" and "success".**
  - Since the "missing-value" is mapped by struts-config.xml back to the original input page, there is only one real results page.
  - Since the results in this example are exactly the same as in the previous example, we use the *same* JSP page.
    - Rather than repeating the "success" entry for both actions, we put it in global-forwards

# Example 2: Results

- **First, the HTML form is invoked with the URL http://localhost/signup/forms/signup2.jsp.**
  - As before, the textfields are prepopulated with values from the ContactFormBean properties



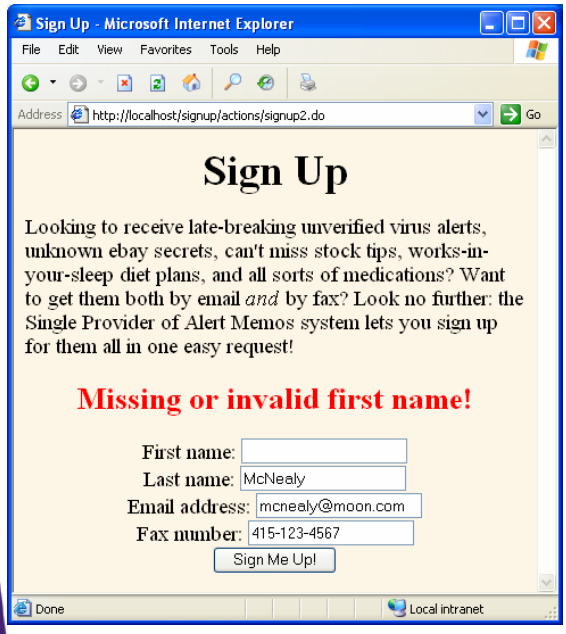


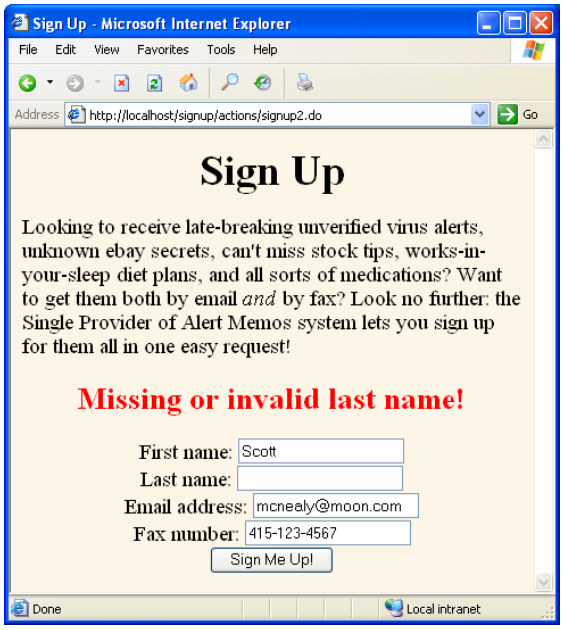


# Example 2: Results

- **Usage of html:form**
  - The "warning" property is output (but default value is empty string)
  - The action in the html:form element is /actions/signup2
  - Resultant URL is http://localhost/signup/actions/signup2.do.
- **Action processing**
  - This address is mapped by struts-config.xml to the SignupAction2 class, whose execute method is invoked.
  - The inherited execute method from SignupAction1 examines the input values to see if any are missing. If so, it calls makeWarning and returns mapping.findForward with a value of "missing-value". Otherwise it returns "success".
  - The "success" value is mapped by struts-config.xml to /WEB-INF/results/confirmation.jsp as before
  - The "missing-value" value is mapped by struts-config.xml to /forms/signup2.jsp, the *original* input page
    - However, since the makeWarning method stores the error message in the form bean, the input form can display it

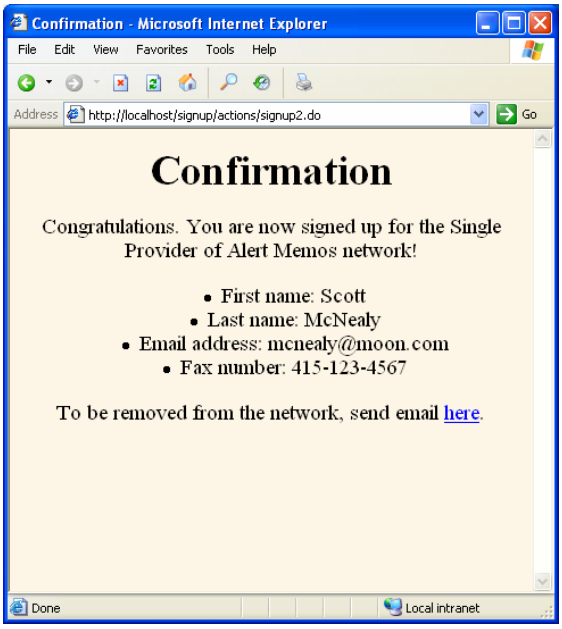
# Example 2: Results






# Example 2: Results



# Other Capabilities of html: Library

- **html:rewrite**
  - Simplifies handling of relative URLs
  - You use URLs beginning with slashes, relative to Web-app home. System tacks Web app name on front.

```
<IMG SRC="<html:rewrite page='/images/pic.jpg'/>"…>
```

  - See examples in section on Tiles
- **html:base**
  - Creates BASE definition so that relative URLs are with respect to real location of JSP page, not URL of Action.
  - Useful in previous example, where input form could be displayed with two different URLs
  - Does not work when JSP pages are in WEB-INF
- **html:javascript**
  - Inserts JavaScript code for client-side validation
  - See examples in section on automatic validation



# Summary

- **Use html:form, html:text, and similar elements to build original form**
  - Guarantees that parameter names match bean properties
  - Lets you derive textfield values from bean
    - Prepopulating: bean default values
    - Redisplay: previously submitted values
- **To redisplay a form, simply map return condition back to input form**
  - Plus store error message in extra bean property
  - The association of form with bean automatically handles the redisplay of all previously entered values
- **Preview: validation**
  - Better way to store and display error messages
  - Better reuse of missing/invalid checks

# 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**