JSP Action tags/Standard Actions

Action Tags

- Action tags translate into to certain java code. They are used so that the JSP file doesn't have too much embedded java code.
- Actions may affect the current out stream and use, modify and/or create objects.
- Action elements follow the syntax of an XML element.
- JSP tags are case-sensitive!
- Action can be either
  - Standard action
    - These action tags translate into to certain predefined java code
  - Custom action
    - These are user-defined actions

HCL

Standard Action Tags

- The JSP specification includes some actions that are *standard* and must be implemented by all conforming JSP containers;
  - **`<jsp:useBean>`**
  - **`<jsp:setProperty>`**
  - **`<jsp:getProperty>`**
  - **`<jsp:param>`**
  - **`<jsp:forward>`**
  - **`<jsp:include>`**

<jsp:useBean>

- This tag is used to instantiate or create a reference to a Java Class (Bean) inside a JSP page.
- The basic semantic tries to find an existing object using **`id`** and **`scope`**.
- If the object is not found it will attempt to create the object using the other attributes.
- If the bean has *instantiated* (rather than located) , and if it has body tags or elements (between **`<jsp:useBean>`** and **`</jsp:useBean>`**), executes the body tags.

Java Beans

- A Java Bean is a reusable software component that can be visually manipulated in builder tools

A java bean is simply a java class that meet the following criteria:

1. The class must be public.
2. The class must have a no-arguments constructor.
3. The class must provide set and get methods to access variables.
4. Java bean is **`Serializable`**

Bean -Simple properties

- Simple property is a bean property with a single value whose changes are independent of changes in any other property.
- To add simple properties to a bean, add appropriate get*XXX* and set*XXX* methods (or is*XXX* and set*XXX* methods for a boolean property).
  - **`public void setSize(int sz) {this.sz=sz; }`**
  - **`public int getSize() {return this.sz; }`**
- Note that for boolean to specify accessor isXXX() can also be used
  - **`public boolean isRetired() { return this.r; }`**

HCL

Bean-indexed property

- An indexed property is an array of properties
- Individual values
  - **`PropertyElement getPropertyName(int index)`**
  - **`public void setPropertyName(int index,PropertyElement element)`**
- Entire array
  - **`public PropertyElement[] getPropertyName()`**
  - **`public void setPropertyName(PropertyElement element[])`**

Bean - Example

```
public class Employee implements java.io.Serializable{
private String empno;
private String name;
public Employee(){}
public void setEmpno(String empno){ this.empno=empno;}
public void setName(String name){    this.name=name;
 }
public String getEmpno(){
return empno;}
public String getName(){return name;
}}
```

HCL

<jsp:useBean> Syntax

```
<jsp:useBean id="beanInstanceName"
   scope="page|request|session|application"
{
   class="package.class" [ type="package.class" ]|
    beanName="{package.class | '${' Expression '}' |
    <%= expression %>}" type="package.class" |
   type="package.class"
}
{ /> | > other elements
</jsp:useBean> }
```

Some things that are clear from the above syntax are:

1. At least one of type and class must be present
2. it is not valid to provide both class and beanName

Can you arrive a few more like these?

id

- `<jsp:useBean id="beanInstanceName" ....`
  - The variable name used to identify the instance in the specified scope.
  - To refer to an instance already been created in the scope id and scope value must match.
  - Is case sensitive and should conform to the current scripting language variable-naming conventions
  - The id attribute of a useBean tag is mandatory and can be an arbitrary string, but must be unique within any one page
  - This variable name can be usede in expressions or scriptlets in the JSP page.

HCL

scope

- **<jsp:useBean id="beanInstanceName"**
  scope="page|request|session|application" **...**
  - The scope in which the bean exists and the variable named in **id** is available. Default is **page**.
  - **scope** value can be
    - **page**→ the bean is available to the entire page
    - **request**→ bean is added as an attribute to request object and is accessible through **request.getAttribute**. If the request is forwarded (using **<jsp:forward>** or **<jsp:include>** tag) to another JSP, this object is available.
    - **session** → bean is added as an attribute to **HttpSession** object and is accessible through **session.getAttribute.** If page directive has **session** attribute as **false,** fatal translation error occurs..
    - **application** → bean is added as an attribute to **ServletContext** object and is accessible through **application.getAttribute**

HCL

class

```
<jsp:useBean id="beanInstanceName"
    scope="page|request|session|application"
{       class="package.class" ……
```

- The fully qualified name of the class that defines the implementation of the object
- Instantiates a bean from a class, using the new keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor.
- Name is case sensitive
- If the **`class`** and **`beanName`** attributes are not specified, the object must be present in the given scope.

HCL

beanName

```
<jsp:useBean id="beanInstanceName"
   scope="page|request|session|application”{
   class="package.class" [ type="package.class" ]|
    beanName=..
```

- Bean is instantiated from
  - a class, (this is same as when class attribute is used)
  - a serialized template, that is it can point to a file that contains a serialized bean (the extension .ser is used to indicate a serialized object).
  - an expression that evaluates to a class or serialized template.

  .

type

```
jsp:useBean id="beanInstanceName"
   scope="page|request|session|application"
{  class="package.class" [ type="package.class" ]|
     beanName="{package.class | '${' Expression '}' |
    <%= expression %>}" type="package.class" |
   type="package.class"
```

- The type is the scripting variable defined.
- If type is used with class or beanName, it is required to be either the class/ type of beanName itself, a superclass of the class/ type of beanName, or an interface implemented by the class/ type of beanName .
- Otherwise a **`ClassCastException`** occurs at request time when the assignment of the object referenced to the scripting variable is attempted.
- If unspecified, the value is the same as the value of the class attribute.

HCL

3 placeholder for `type`

- **`type="package.class"`**
  - If the class and beanName attributes are not specified the object must be present in the given scope. If the object that exists in the scope it is returned and assigned to a variable of type as specified in **`package.class`**.
  - No bean is instantiated
- **`class="package.class" type="package.class"`**
  - Instantiates a bean from the class named in class and assigns the bean the data type you specify in type.
- **`beanName="{package.class | <%= expression %>}" type="package.class"`**
  - Instantiates a bean from the class or expression or assigns the bean the data type you specify in type

Example: Using <jsp:useBean>

- **<jsp:useBean id="list" class="java.util.Vector" scope="session"/>**

  Translates into:

```
java.util.Vector list=(java.util.Vector)
session.getAttribute("list");
if(list==null)
list= new java.util.Vector();
session.setAttribute("list",list);
```

- **<jsp:useBean id="list" type="java.util.Vector" scope="session"/>**

  Translates into:

```
java.util.Vector v=(java.util.Vector)
session.getAttribute("list");
```

  So if the bean "list" does not exist in the session an exception will be thrown at runtime!

HCL

- **<jsp:useBean id="list" class="java.util.Vector" type="java.lang.Object" scope="session" />**

  Same as the as 1 but returns an `Object` class.

  Translates into:

```
java.lang.Object list=session.getAttribute("list");
if(list==null)
list= new java.util.Vector();
session.setAttribute("list",list);
```

beanName attribute example

```
<%String s="c.Cust";%>
<jsp:useBean id="a" beanName="<%=s%>" type="c.Cust"/>
```

where c.Cust is a java bean which implements **`Serializable`** interface.

Note that if while **`beanName`** allows result of an expression to be assigned to itself, class attribute does not allow this.

```
<jsp:useBean id="a" beanName="c/Cust.ser"
   type="c.Cust"/>
```

where c/Cust.ser is the name of the file that contains serialized customer object.

<jsp:setProperty>

<jsp:setProperty name="beanName" prop_expr />

- This tag is used along with **<jsp:useBean>** tag, to set values of the bean property.
- **prop_expr**

  property="property" value="value"

  property="property"

  property="property" param ="param"

  property="*"
- **property**: name of the bean property. It can also be '*'. If so, if request parameters have the same name as the bean property, then the value of the request parameter is automatically set to the bean property.
- **value**: Value to be set to the property.
- **param:** The name of the request parameter whose value the bean property will be set to.
- **name**: is the name of the bean instance which has been created using the **<jsp:useBean>** tag.

<jsp:getProperty>

- Places the value of a Bean instance property, converted to a String, into the implicit out object, from which you can display the value as output.

- `<jsp:getProperty name="name" property="propName" />`

Example: jsp:setProperty, jsp:getProperty

Student.java

sr.Student
-name -reg
+String getName() +String getReg() +void setReg(String reg) +void setName(String nm)

stud.setName(request.
getParameter("nm")

stud.setName(request.
getParameter("name")

```
<jsp:useBean id="stud" class="sr.Student">
<jsp:setProperty name="stud" property="name"/>
<jsp:setProperty name="stud" property="name" param="nm" />
<jsp:setProperty name="stud" property="reg" value="1234"/>
</jsp:useBean>
..
<jsp:getProperty name="stud" property="reg"/>
```

→Displays value of reg

HCL

More on <jsp:useBean >

```
<jsp:useBean id="s" class="sr.Student" scope="session">
<jsp:getProperty property="name" name="s"/>
</jsp:useBean>
```

- If the stud bean is not created in the session then, it is created and so **<jsp:getProperty property="name" name="s"/>** is executed.
- If the stud bean is already there in the session then, it is not created and so **<jsp:getProperty property="name" name="s"/>** is not executed.
- To get the value of name property for already created bean in a session (or any scope), **<jsp:getProperty>** must be outside the **<jsp:useBean>.**

```
<jsp:useBean id="s" class="sr.Student" scope="session"/>
<jsp:getProperty property="name" name="s"/>
```

Form parameters and bean properties

stud.html

```
…
<FORM ACTION="stud.jsp"
METHOD="post">
<INPUT TYPE="text" NAME="name">
<INPUT TYPE="text" NAME="reg">
…
```

```
<jsp:useBean id="stud" class="sr.Student">

<jsp:setProperty name="stud" property="*"/>
</jsp:useBean>
```

On submitting the form the *name* and *reg* property of the *stud* bean is automatically set to the values entered by the user.

If there is no matching request parameter for a bean property then it remains unset. You should always make sure that your bean's default constructor sets appropriate default values.

HCL

Valid Value Assignments

When the bean attributes are assigned from form parameters, the Web container will automatically attempt to convert the String into the type of the bean parameter. This conversion will work for the following types:

- **`byte or Byte`** As indicated in **`Byte.valueOf(String)`**
- **`boolean or Boolean`** As indicated in **`Boolean.valueOf(String)`**
- **`char or Character`** As indicated in **`String.charAt(0)`**
- **`double or Double`** As indicated in **`Double.valueOf(String)`**
- **`int or Integer`** As indicated in **`Integer.valueOf(String)`**
- **`float or Float`** As indicated in **`Float.valueOf(String)`**
- **`long or Long`** As indicated in **`Long.valueOf(String)`**
- **`short of Short`** As indicated in **`Short.valueOf(String)`**
- **`Object`** As if new **`String(string-literal)`**

HCL

<jsp:forward>

- Allows the JSP to transfer the control to another JSP, servlet or static HTML page.
- The jsp that forwards the request should not use the output stream that is used to communicate to the client prior to forwarding the request.
- This is similar to **`RequestDisplatcher's forward()`** method
- **`<jsp:forward page="URL"/>`**

<jsp:include>

- Allows static or dynamic resource to be included in the current JSP page at the request processing time.
- Similar to **`RequestDispatcher's include()`** method.
- **`<jsp:include page="URL"/>`**
- **`<jsp:include page="URL" flush="true"/>`** results in the buffer being flushed to the browser when action is executed.

<jsp:param>

- Used to provide other tags with additional information in the form of the name-value pairs.
- Used in conjunction with the **<jsp:include>** and **<jsp:forward>**
- **<jsp:param name="p1" value="pv">**

```
<jsp:include page="x.jsp">
        <jsp:param name="p1" value="pv1"/>
        <jsp:param name="p2" value="pv2"/>
</jsp:include>
```

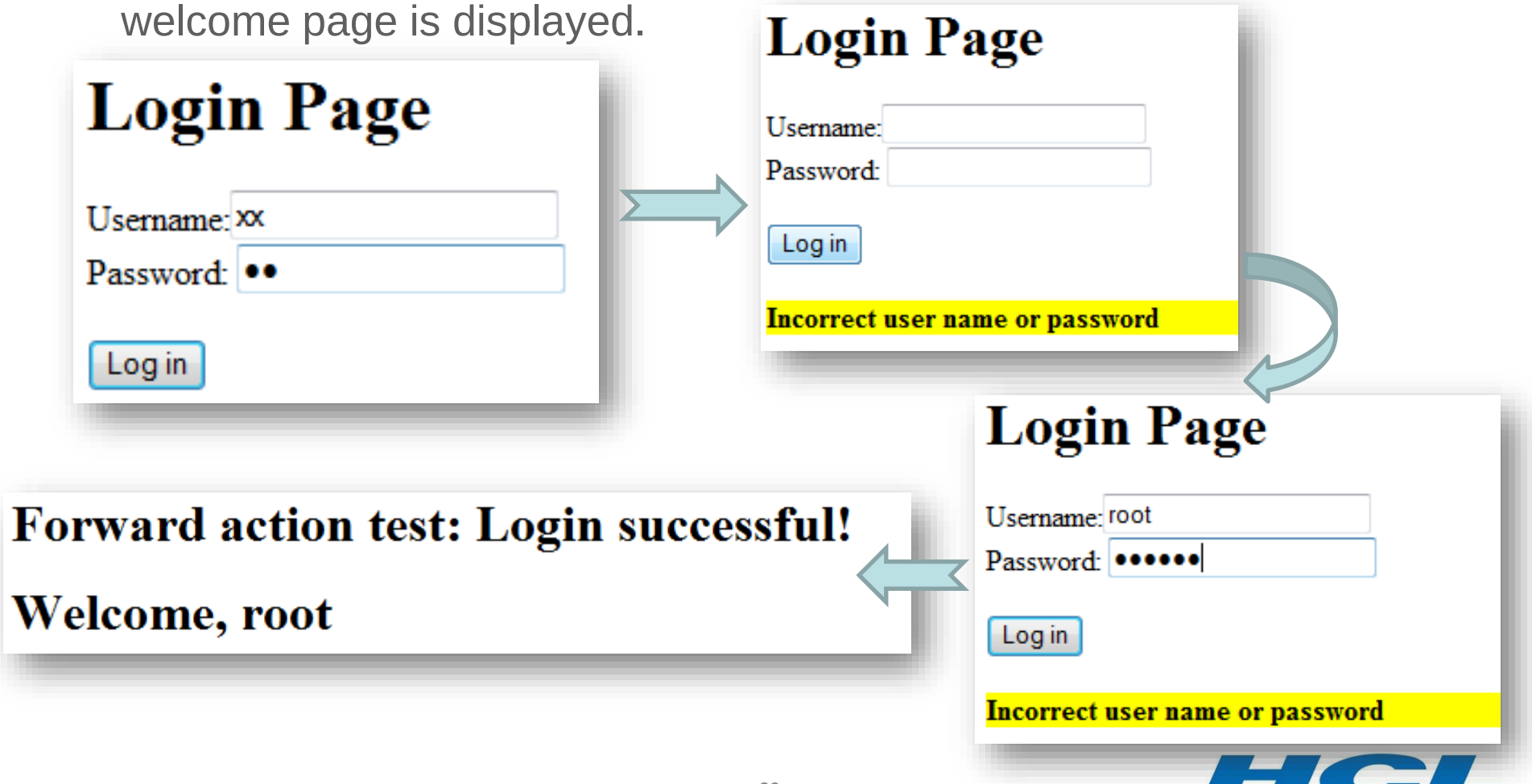
- The argument can be extracted by using **request.getParameter("p1")** in a servlet/jsp.

<jsp:include> and <%@ include>

<jsp:include>	<%@ include>
<jsp: include page="filename" />	<%@ include file="filename" %>
Done at the request processing time	Done at the translation time.(That is, the contents of the included file will be included in the JSP source at the point where the tag is defined and therefore processed by the JSP page compilation procedure.
Both static and dynamic content can be included.	The included file may contain both static content and other JSP tags.

Example: forward and include

- User enters login and password which is validated. If invalid values are provided login page is displayed with error message, otherwise welcome page is displayed.



index.jsp

```
<html>
<head>
<title>Forward and include action test page</title>
</head><body>
<h1>Login Page</h1>
<form method="post" action="forward.jsp">
<p>Username:<input type="text" name="userName">
<br> Password:<input type="password" name="password">
</p>
<p><input type="submit" value="Log in"></p>
</form>
<%if(request.getParameter("error")!=null){ %>
<h4 style="background-color : yellow">
<%=request.getParameter("error")%></h4>
<%} %>
</body>
</html>
```

forward.jsp

```
<html><head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%if ((request.getParameter("userName").equals
("root")) && (request.getParameter("password").
  equals("flower"))) {%>
<jsp:include page="forward2.jsp"/>
<% }  else { %>
<jsp:forward page="index.jsp">
<jsp:param value="Incorrect user name or password"
name="error"/>
</jsp:forward>
<%} %>
</body></html>
```

forward2.jsp

```
<html>
<head><title>Forward action test: Login
successful!</title></head>
<body>
<h1>Forward action test: Login successful!<h1>
<p>Welcome, <%=request.getParameter("userName") %>
</body>
</html>
```

<jsp:plugin>, <jsp:params>, <jsp:fallback>

- This tag results in the execution of a specified applet or JavaBean in the client browser, preceded by a download of Java plugin software if necessary.
- **<jsp:param>** tags between the **<jsp:params>** are used to specify parameters to the applet or JavaBean.
- **<jsp:fallback>** is used to delimit alternative text to execute if the plugin cannot run.
- Example:

```
<jsp:plugin type=applet code="MyApplet.class"
codebase="/html" >
    <jsp:params><jsp:param name="uname" value="Meena" />
    </jsp:params>
    <jsp:fallback>Unable to start the plugin.
    </jsp:fallback>
</jsp:plugin>
```