Web Component Development Using Java

Session: 9

Standard Actions and JavaBeans

Objectives

- Explain the concept of standard actions in JSP
- Explain how to use the <jsp:include> element
- Explain how to use the <jsp:forward> element
- Explain how to use the <jsp:param> element
- Explain how to use the <jsp:plugin> element
- Explain how to use the <jsp:fallback> element
- Explain how to use the <jsp:text> element
- Explain the concept of JavaBeans
- Explain how to declare and access JavaBeans components in JSP
- Explain accessing JavaBean properties from scripting elements
- Explain how to access non-string data type properties from scripting elements
- Explain how to access indexed properties from scripting elements

Introduction

- Standard actions:
  - Are XML like tags.
  - Take the form of an XML tag with a name prefixed `jsp`.
  - Are used for:

Forwarding requests and performing includes in pages.

Embedding the appropriate HTML on pages.

Interacting between pages and JavaBeans.

Providing additional functionality to tag libraries.

Use of Standard Actions 1-2

- JSP standard actions are performed when a browser requests for a JSP page.
- The properties of standard actions are as follows:

It use <jsp> prefix.

The attributes are case sensitive.

Values in the attributes must be enclosed in double quotes.

Standard actions can be either an empty or a container tag.

- **Syntax:**

```
<jsp:action_name
 attribute="value"
attribute="value"/>
```

Or

```
<jsp:action_name
attribute="value"
attribute="value">
   . . .
</jsp:action_name>
```

Use of Standard Actions 2-2

- Various standard action tags available in JSP are as follows:

<jsp:include> 1-3

- The `<jsp:include>` element offers choice to include either a static or dynamic file in the current requested JSP page.
- For static file, the content is included in the calling JSP file.
- For dynamic file, it acts on a request and sends back a result that is included in the JSP page.
- When the user is not sure whether the file is static or dynamic, use `<jsp:include>` element, as it handles both types of files.

- **Syntax**:

```
<jsp:include page="weburl" flush="true" />
```

where,

- `page` specifies the relative Web address of the page to be included in the current page.
- `flush` attribute is used to flush out the data stored in the buffer. The default value is false.

<jsp:include> 2-3

- The code snippet demonstrates the use of `<jsp:include>` action to display current date and time on the JSP page.

```
<!- index.jsp -->
<html>
    <head>
    <meta http-equiv="Content-
Type" content="text/html;
charset=UTF-8">

<title> Dynamic Content
Inclusion </title>
</head>

<body>
<h4><font color="BLUE">
Displaying Current Date and Time
</font></h4>

 <b>Today is: </b> <jsp:include
page="printdate.jsp"/>

 <p><i> The Date and Time are
displayed as a result of
evaluation of another JSP page.
</i></p>
</body>

</html>
```

```
<!-- printdate.jsp-->

<%@page
contentType="text/html"
pageEncoding="UTF-8"
import="java.util.*"%>
<html>
  . . .
 <body>

 <%
 Date today = new Date();

out.print(today.toString(
));
 %>


</body>
</html>
```

<jsp:include> 3-3

❖ **Output:**

<jsp:forward> 1-3

- It is used to redirect the request object containing the client request from one JSP to another target page.
- The target page can be an HTML file, another JSP file, or a Servlet.
- In the source JSP file, the code after the `<jsp:forward>` element is not processed.
- To pass parameter names and values to the target file, user can use a `<jsp:param>` clause with `<jsp:forward>` element.
- **Syntax:**

```
<jsp:forward page="url"/>
```

where,

- `page` specifies the relative Web address of the target page.

<jsp:forward> 2-3

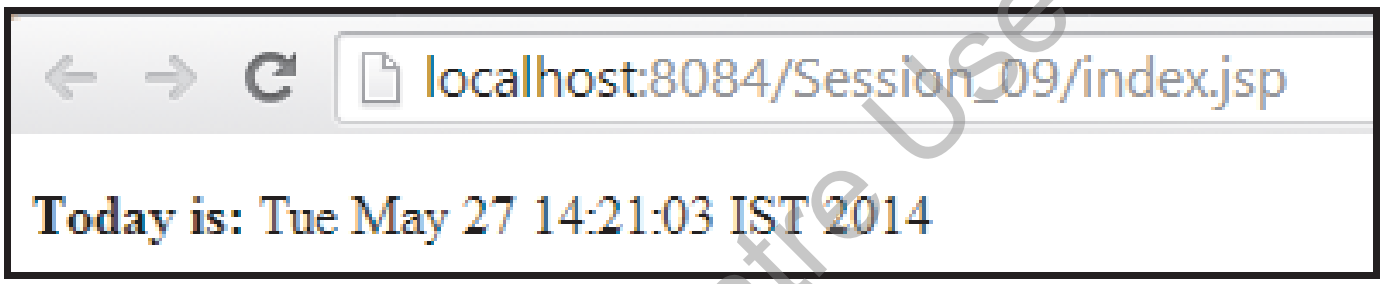
- The code snippet demonstrates the use of `<jsp:forward>` action to display current date and time on the next JSP page.

```
<!-- index.jsp -->
<%@page
contentType="text/html"
pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>
. . .
<body>
 <h4><font color="BLUE">
Displaying Current Date and
Time </font></h4>

 <!-- Forwards request to the
other Web resource -->
 <jsp:forward
page="printdate.jsp"/>

 <p><i> The request is
forwarded to the next page to
display Date and Time.
</i></p>

</body>
</html>
```

```
<!-- printdate.jsp -->
<%@page
contentType="text/html"
pageEncoding="UTF-8"
import="java.util.*"%>
<html>
. . .
<body>
 <%! Date today = new
Date();%>

 <b> Today is: </b>

<%
out.print(today.toString()
);
%>


</body>
</html>
```

<jsp:forward> 3-3

❖ **Output:**



<jsp:param> 1-3

- It allows the user to pass one or more name and value pairs as parameters to an included or forwarded resource such as a JSP page, Servlet, or other resource that can process the parameter.
- It allows to send more than one parameters to the included or forwarded resource, user can use more than one `<jsp:param>` clause.
- **Syntax**:

```
<jsp:param name="parameterName" value="{parameterValue
| <%= expression %>}" />
```

where,

- `name` attribute specifies the parameter name and that takes a case-sensitive literal string.
- `value` attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at request time.

<jsp:param> 2-3

- The code snippet demonstrates forwarding request to the page along with the parameter values.

```
<!-- product.jsp -->
<html>
 .. .
<body>
        <jsp:include page="order.jsp" flush="true">
        <jsp:param name="currency_type" value="Dollar"/>
        <jsp:param name="amount" value="$110"/>
        </jsp:include>
</body>
</html>
```

```
<!-- order.jsp -->
<body>
<% String currency =
  request.getParameter("currency_type");
  String amount = request.getParameter("amount");
%>
 <b><u> Param Values </u></b> <br/><br/>
  <% out.println("Currency: " + currency); %>
<br/><br/>
  <% out.println("Amount: " + amount); %>
  </body>
```

<jsp:param> 3-3

❖ **Output:**

<jsp:plugin> 1-3

- It plays or displays an object, using a Java plugin that is available in the browser or downloaded from a specified URL.
- The plug-in object can be an applet or a JavaBean component that can be displayed on a JSP page.
- The `<jsp:plugin>` element is replaced by either an `<object>` or `<embed>` element.
- The `<jsp:plugin>` element provides attributes that are used for formatting Java object. These attributes are similar to HTML tag attributes.
- To pass values to the Java objects, the `<jsp:param>` or `<jsp:params>` actions can be used.

<jsp:plugin> 2-3

❖ **Syntax**:

```
<jsp:plugin type="bean|applet" code="classFileName"
codebase="classFileDirectoryName" { align="alignment"
| archive="archiveList" | height="height" |
hspace="hspace" | jreversion="jreversion" |
name="componentName" | vspace="vspace" | width="width"
| nspluginurl="url" | iepluginurl="url" |
<jsp:params>
   { <jsp:param name="paramName" value= paramValue" />
}+
</jsp:params> }
  { <jsp:fallback> arbitrary-text </jsp:fallback> } >
</jsp:plugin>
```

where,

- Attributes specified between { } are providing configuration data for presenting the component on the Web page
- <jsp:params> element provides parameters to the Java object
- <jsp:fallback> element specifies the content to be used if the plug-in is not existing

<jsp:plugin> 3-3

- The code snippet demonstrates the <jsp:plugin> element to display an applet on the JSP page.

```
<jsp:plugin type=applet code="game.class""
codebase="/html">

<jsp:params>
  <jsp:param name="image"
                  value="image/shape.mol" />
</jsp:params>


<jsp:fallback>
 <p> Unable to start the plugin. </p>
</jsp:fallback>


</jsp:plugin>
```

<jsp:fallback>

- A text message that conveys the user that a plug-in could not start is known as fallback.
- If the plug-in starts, but the applet or bean does not, the plug-in usually displays a popup window explaining the error to the user.
- The `<jsp:fallback>` action specifies an alternative message to the browser, if the browser fails to start the plug-in.
- **Syntax**:

```
<jsp:fallback> text message </jsp:fallback>
```

<jsp:text>

- It encloses contents that are displayed within the body of a JSP page or a JSP document.
- It does not have any sub-element and can appear anywhere in the JSP page where content has to be displayed in the output.
- The `<jsp:text>` element can contain expressions which are evaluated and their result is displayed in the output.
- The code snippet demonstrates the use of expression language with `<jsp:text>` element.

```
<html>

<body>
   <jsp:text> This is the product page.
   </jsp:text>
   . . .
   <jsp:text>Rectangle Perimeter is: ${2* 10 + 2*
   20} </jsp:text>

</body>
</html>
```

JavaBeans Actions

- Are reusable components that can be developed in Java.
- Define the interactivity of Java objects.
- Allow creation of Java object that can be embedded in multiple applications, servlets, and JSP.
- Requirements for JavaBeans are as follows:

They must be a public class.

They must have a public constructor with no arguments.

They must have no public instance variables.

They must have getter and setter methods to read and write the bean properties.

They must implement `java.io.Serializable` interface.

Components of JavaBeans 1-3

- JavaBeans are Java classes that include:

Properties

- It represents bean attributes that can be used to modify the appearance or behavior of the JavaBean.

Methods

- It allows implementing a bean and can be called from other components or from a scripting environment, such as JSP.

Events

- It allows communication between JavaBeans.

Components of JavaBeans 2-3

- The code snippet demonstrates the JavaBean component.

```
import java.io.Serializable;

public class Person implements Serializable
{
  Private String firstName;

  // Constructor
      public Person() { }

  // Sets name
     public void setFirstName(String name)
     {
        firstName=name;
     }
  // Retrieves name
     public string getFirstName()
     {
        return firstName;
     }
  // End of Person class
```

Components of JavaBeans 3-3

- The getter and setter methods are public methods defined in a JavaBean class, `Person`.
- These methods are also referred as accessor methods and allow retrieving and setting variable values of JavaBean properties.

Get method

It allows retrieving a variable value

Set method

It allows setting or changing the value of a property

- **Syntax**:

```
public returnType
getPropertyName()
```

where,

- `returnType` is the data type returned by the get method.

- **Syntax**:

```
public void
setPropertyname(datatype
parameter)
```

where,

- `parameter` is the property that will be set.

Declaring JavaBeans in JSP 1-3

- The `<jsp:useBean>` element is used to locate or instantiate a JavaBean component.
- Steps followed by `<jsp:useBean>` to locate or instantiate the bean are as follows:

Attempts to locate a bean within the scope.

Defines an object reference variable with the name.

Stores a reference to it in the variable, if it retrieves the bean.

Instantiates it from the specified class, if it cannot retrieve the bean.

- **Syntax**:

```
<jsp:useBean id="BeanName"
             class="BeanClass"
   scope = "page|session|application|request"/>
```

where,

- `id` uniquely creates a name for referring the bean.
- `class` specifies the fully qualified class name that defines bean.
- `scope` specifies the scope within which the bean is available. The default scope is `page` if not specified.

Declaring JavaBeans in JSP 2-3

Consider that the **Person** JavaBean class is created in the Web application under **com.bean** package. To access the properties of the **Person** class within the JSP page, the user need to instantiate it.

- The **jsp:useBean** action is used to include the bean in the current JSP page.
- The code snippet demonstrates the instantiation of **Person** JavaBean in the JSP page.

```
<html>
. . .
 <body>
    <jsp:useBean id="personID"

          class="com.bean.Person" scope="request"/>

 </body>
</html>
```

- The code checks if any bean class instance with reference `personID` exists in the request scope
  - If Yes, it accesses the bean reference.
  - If No, it creates a new instance and sets it with request scope.

Declaring JavaBeans in JSP 3-3

- Following is the Java equivalent code for the code snippet shown on the previous slide:

```
Person personID = (Person)
request.getAttribute("personID");

if (personID == null)
{
     personID = new Person();
     request.setAttribute("personID", personID);
}
```

Setting Value in JavaBeans 1-3

- The `<jsp:setProperty>` element sets the value of the properties in a bean using the bean's setter methods.
- The user must declare the JavaBean with `<jsp:useBean>` before setting a property value with `<jsp:setProperty>`.
- `<jsp:setProperty>` can be used to set property values by passing:
  - All the values accepted from the user
  - A specific value to set a specific property of the JavaBean
- **Syntax**:

```
<jsp:setProperty name = "BeanAlias" property =
"PropertyName" value = "Value" param =
"Parameter" />
```

where,

- `name` specifies the id of the bean used in `jsp:useBean` action.
- `property` specifies the property name of the bean.
- `value` specifies the explicit value to set for the property.
- `param` specifies the value sent in the request parameter to be assigned to a property.

Setting Value in JavaBeans 2-3

- The code snippet demonstrates how to set the value for the **firstName** property of **Person** class in JSP.

```
<html>
  . . .
   <body>
    <jsp:useBean id="personID"
class="com.bean.Person" scope="request"/>

    <!-- Sets the value for the firstname
property-->

 <jsp:setProperty name="personID"
property="firstName"
value="John"/>

  </body>
</html>
```

where,

- `jsp:useBean` action is used to include the `Person` class in the current JSP page.
- `jsp:setProperty` action is used to set the value, `John` to the `firstName` property for the bean instance, **personID.**

Setting Value in JavaBeans 3-3

- The user can also set the value for the **`firstName`** by retrieving it from the request parameter sent through HTML form field. Thus, the action tag will be:
  - `<jsp:setProperty name="personID" property="firstName"/>.`
- The value of the request parameter **`firstName`** is retrieved and set in the bean property firstName.
- If the supplied request parameter name is not same as bean property name, then user can use the `param` attribute.
- For example, if the HTML form field name to accept users' first name is set as name, then `jsp:setProperty` action can be written as:
  - `<jsp:setProperty name="personID" property="firstName" param="name"/>.`

Retrieving Value from JavaBeans 1-2

- The `<jsp:getProperty>` element retrieves a bean property value using the getter methods and displays the output in a JSP page.
- Before using `<jsp:getProperty>` element, the user must create or locate a bean with `<jsp:useBean>`.
- **Syntax**:

```
<jsp:getProperty name="BeanAlias"
property="PropertyName" />
```

where,

- `name` specifies the id of the bean specified in the `jsp:useBean` action.
- `property` specifies the property name from which the value is to be retrieved.

Retrieving Value from JavaBeans 2-2

- The code snippet demonstrates how to retrieve value of the **firstName** property of **Person** class in JSP.

```
<html>
 ...

  <body>
 <jsp:useBean id="personID" class="
com.bean.Person" scope="request"/>

 Name is:  <jsp:getProperty name="personID"
property="firstName"/>

</body>
</html>
```

- In the code:
  - `useBean` is used to access the `Person` class instance in the current JSP page.
  - `jsp:getProperty` action is used to retrieve the value of the `name` property from the JavaBean instance.
- The equivalent Java code for the `jsp:getProperty` is as follows:

```
out.print(personID.getFirstName());
```

Accessing JavaBeans within Scriptlets

- A user can access JavaBeans from scripting element in different ways.
- The `jsp:getProperty` and expression convert the value into a string and insert it into an implicit out object.
- To retrieve the value of a property without converting it and insert it into the `out` object, user must have to use a scriptlet.
- Scriptlets are very useful for dynamic processing.
- Using custom tags is a better approach to access object properties and perform flow control.
- The code snippet demonstrates how to create the JavaBean instance and access its properties in JSP.

```
<html>
. . .
 <body>

 // Instantiating Book bean of package pkg using scriptlet
    <% pkg.Book book1 = new pkg.Book(); %>
 // Retrieves the title property using bean getter method
    <%= book1.getTitle() %>
 // Sets title property to a new value
    <% book1.setTitle("Guide to Servlets and JSP"); %>
. . .
</body>
</html>
```

Accessing Non-string Data Type Properties

- The `jsp:setProperty` action is used to set the properties of a bean by the values of the request parameter.
- The JSP container converts the string values into non-string values by the attribute values that evaluate the correct data type to set the property value.
- Some of the conversions from string to appropriate data type is done by using Java wrapper classes. For example:

```
String ageStr = request.getParameter("age");
int ageInt = Integer.valueOf(ageStr);

String amtStr = request.getParameter("Amount");
double amtDouble = Double.valueOf(amtStr);
```

Accessing Indexed Properties

- An indexed property is an array of properties or objects that supports a range of values.
- It enables the accessor to specify an element of a property to read or write.
- If there is an indexed property `'Tomy'` of type string, it may be possible from a scripting environment to access an individual indexed value using the index.
- For example, `b.Tomy[3]` and also to access the same property as an array using `b.Tomy`.
- The indexed getter and setter methods throw a runtime exception `java.lang.ArrayIndexOutOfBoundsException`, if an index is used that is outside the current array bounds.
- The value assigned to an indexed property must be an array.
- The code snippet shows how to declare the getter and setter method for the indexed property **studentRegister**.

```
// Retrieves indexed properties
   public studentRegister[] getStudentRegister()

// Sets indexed properties
   public int setStudentRegister(StudentRegister
int[])
```

Summary

- JSP standard actions are XML like tags that are processed when a browser requests for a JSP page.
- Standard actions in the JSP standard library use the <jsp> prefix. Various standard actions are available that include <jsp:include>, <jsp:forward>, <jsp:param>, <jsp:params>, <jsp:plugin>, <jsp:fallback>, and <jsp:text>.
- JavaBeans are reusable components that can be developed in Java. These are platform independent and define the interactivity of Java objects in the applications.
- JavaBeans component controls access to the private properties of a class by providing public methods.
- The jsp:useBean action is used to create a reference and include an existing JavaBean component in JSP.
- The jsp:getProperty and jsp:setProperty actions act as getter and setter methods to access and retrieve values from the JavaBean components.
- An indexed property is an array of properties or objects that supports a range of values. This property enables the accessor to specify an element of a property to read or write.