Java Server Pages

Sandeep, SunBeam

JSP Actions

- JSP actions are special XML tags which control the behavior of servlet engine.
- JSP provides bunch of standard actions that we can use for specific task such as working with java bean instances, including other resources, forwarding request to other resource etc.
- Syntax : **<jsp:actionName attribute="value" />**
- Following are some of the standard actions:

  1. <jsp:useBean>
  2. <jsp:getProperty>
  3. <jsp:setProperty>
  4. <jsp:include>
  5. <jsp:forward>

<jsp:useBean>

- If we want to use java bean instance in JSP then we should use useBean action.
- Once declared, the bean instance becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.
- The syntax for the useBean tag is as follows:

```
<jsp:useBean id="" class="" scope=""/>
```

  - id : beans name
  - class : F.Q. Bean Class Name.
  - scope : Beans scope(page, request, session or application).
- Consider following example:

```
<jsp:useBean id="date"
class="java.util.Date"
scope="session"/>
```

Equivalent java code is:

```
Date date = null;
synchronized (session){
        date = (Date) pageContext.getAttribute("date", PageContext.SESSION_SCOPE);
        if (date == null){
                date = new Date();
                pageContext.setAttribute("date", date, PageContext.SESSION_SCOPE);
        }
}
```

- From above code we mean that, web container first check java bean attribute in specified scope. If it is not found then web container locate,load,instantiate & initialize java bean instance and then add it into specified scope.
- If we do not specify the scope then it will search/add java bean attribute in page scope. i.e. default scope is page scope.

<jsp:setProperty>

- The setProperty action is used to set values of a bean.
- The syntax for the setProperty action is as follows:

```
<jsp:setProperty name="bean refernce name"
property="property name"
value="static/dynamic value" />
```

- Consider the following code snippet:

```
<jsp:useBean id="user" class="beans.UserBean" scope="session"/>
<jsp:setProperty name="user" property="email" value="sandeep@gmail.com"/>
```

- We know, how web container use setAttribute method to set java bean attribute. If we use setProperty then first WC invokes getAttribute() method and then it invokes setter method on it. Consider the following code:

```
session.getAttribute("user").setEmail("sandeep@gmail.com");
```

<jsp:setProperty>

- Suppose "Login.jsp" contains following code:

  <input type="email" name="email"/>

  then we can set email property as follows:

1. <jsp:setProperty name="user" property="email"
   value="<%=request.getParameter("email")%>" />
2. <jsp:setProperty name="user" property="email" value="${param.email}" />
3. <jsp:setProperty name="user" property="email" param="email" />
4. **<jsp:setProperty name="user" property="*" />**

- If name of request parameter and java bean property is same then we should use above syntax.

<jsp:getProperty>

- The getProperty action is used to get values of a bean.
- The getProperty action provides a way of removing many scriptlets and expressions.
- This action is used to invoke a get method of a JavaBean and uses the following syntax:

```
<jsp:getProperty name="bean reference name" property="property name"/>
```

- Consider the following code snippet:

```
<jsp:useBean id="user" class="beans.UserBean" scope="session"/>

<jsp:getProperty name="user" property="email"/>
```

- If we use getProperty then first WC invokes getAttribute method and then it invokes getter method on it.
- Consider the following code:

```
session.getAttribute("User").getEmail();
```

<jsp:forward>

- It is used to forward current request to another resource such as HTML page, JSP or servlet.

- Syntax :

```
<jsp:forward page="relative URL"/>
```

- Example:

```
<jsp:forward page="Next.jsp"/>
```

- It is 100% same as ReuestDispatcher: forward scenario.

```
RequestDispatcher rd = request.getRequestDispatcher("Next.jsp");
rd.forward( request,response);
```

<jsp:include>

- It is used to include resources at request processing time.

- Syntax:

```
<jsp:include page="relative URL"/>
```

- Example:

```
<jsp:include page="Next.jsp"/>
```

- It is 100% same as ReuestDispatcher : include scenario.

```
RequestDispatcher rd = request.getRequestDispatcher("Next.jsp");
rd.include( request,response);
```

JSTL

- JSP Standard Tag Library.
- If JSP standard tags/actions are insufficient to provide business logic then we should use JSTL.
- Initially JSTL was not a part of J2EE. Since Java EE 5 it is considered as a standard part.

How to use JSTL?

1. Copy "jstl-1.2.jar" file either in /WEB-INF/lib of eclipse or tomcat/lib. It contains Tag implementation classes and Tag Library Descriptor(TLD) which describes how to use tag.
2. Use taglib directive to include to include JSTL tag into JSP.

- Syntax:

  <%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

- Core JSTL tags all share the same URI

  http://java.sun.com/jsp/jstl/core

- General-Purpose Tags
  - <c:out>, <c:set>, <c:remove>, <c:if>, <c:forEach>
  - <c:url>, <c:redirect> etc.

<c:out>

- The tag is helpful as an alternative to the commonly used getProperty action.

- In cases where a JavaBean is not present, or a developer prefers not to use the bean, the out action provides equivalent functionality.

- e.g. **<c:out value="${count}"/>**

<c:set>

- The set action is a JSTL-friendly version of the setProperty action.
- <c:set> sets the specified attribute to specified scope
- Consider following example:

```
<c:set var="count" value="${pageScope.count}" scope="session"/>
```

- It is equivalent to

```
session.setAttriibute("count",pageContext.getAttribute("count"))
```

<c:remove>

- The remove action provides a method of removing a scoped variable.

- Consider Following Example:

```
<c:remove var="count" scope="session"/>
```

- It is equivalent to:

```
session.removeAttribute(count);
```

<c:forEach>

- The forEach tag provides for iteration over a collection of objects.

- Consider following example:

```
<c:forEach var="count" begin="1" end="10" step="1">
      <c:out value="${count}"/>
</c:forEach>
```

- Consider another example:

```
<c:forEach var="book" items="${bb.bookList}">
      <c:out value="${book}"/>
</c:forEach>
```

<c:if>

- The if tag allows the conditional execution of its body depending on the value of a test attribute.

- Consider the following example:

```
<c:forEach var="count" begin="1" end="10">
    <c:if test="${count%2==0}">
        <c:out value="${count}"/>
    </c:if>
</c:forEach>
```

<c:choose>,<c:when> and <c:otherwise>

```
<c:choose>
	<c:when test="${pageScope.Gender==true}">
		<c:out value="Male"/>
	</c:when>
	<c:otherwise>
		<c:out value="Female"/>
	</c:otherwise>
</c:choose>
```

<c:url>

- The url tag is a method the JSTL provides to automatically encode URLs with session information and parameters.
- Syntax:

```
<c:url var="attr Name"
       value="URL to be encoded"
       scope="page|request|session|application"/>
```

- Example:

```
<c:url var="location" value="Subject.jsp"/>
```

- Equivalent Code

```
pageContext.setAttribute("location",
                         response.encodeURL("Subject.jsp"));
```

<c:redirect>

- The functionality of the redirect tag is equivalent to calling the HttpServletResponse sendRedirect() method.

- Example:

```
<c:url var="location" value="Subject.jsp"/>
<c:redirect url="${location}"/>
```

JSP Custom Tag

- When JSP standard actions and JSTL actions are insufficient to solve business logic then we should define custom tag.

- Steps to create/define custom tag
  1. Define a tag handler class to encapsulate business logic.
  2. Create a TLD file to describe tag to the web container so that it can manage life cycle of custom tag.
  3. Import the TLD on JSP using taglib directive.
  4. Invoke the tag.

API to define custom tag

- Methods of SimpleTag Interface
    1. void doTag()throws JspException,IOException
    2. JspTag getParent()
    3. void setParent(JspTag parent)
    4. void setJspContext(JspContext pc)
    5. void setJspBody(JspFragment jspBody)

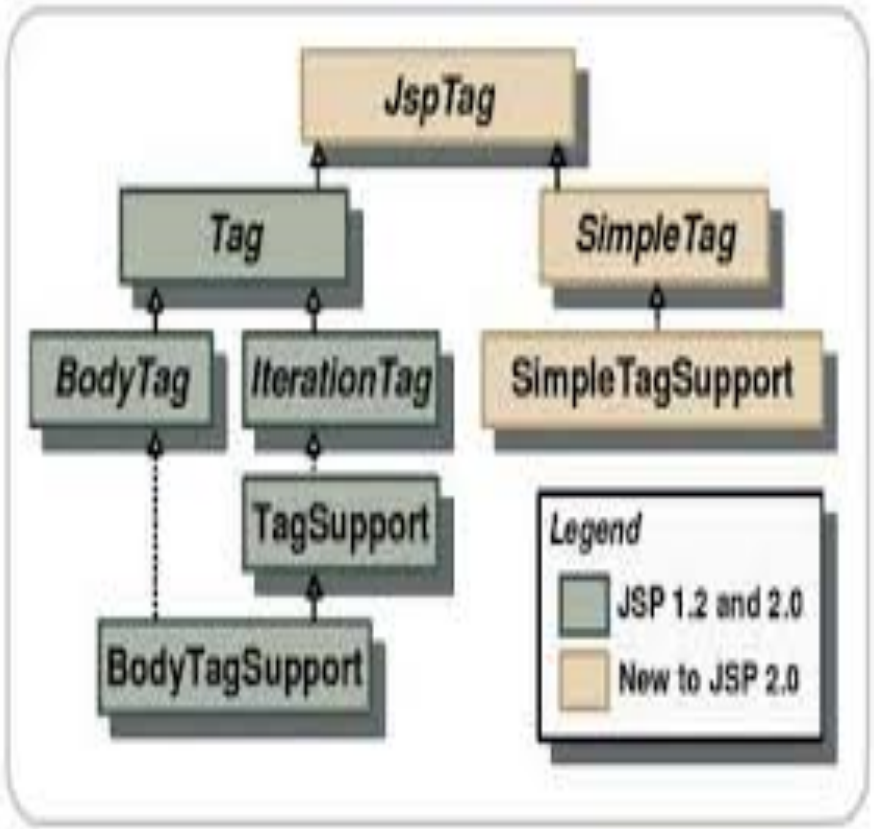

Figure 1 Tag extension class hierarchy

Tag Library Descriptor

- In the abstract sense a Tag Library Descriptor is the mechanism that binds custom tag code to the simple mark-up that appears in a JSP.

- **Tag Handler class <--> TLD <--> Markup(tag)**

- TLD is an XML file that usually appears in the /WEB-INF directory of a Web Application.

- Extension of TLD file should be .tld.

taglib directive

- The taglib directive informs a container what bits of markup on the page should be considered custom code and what code the markup links to.

- The taglib directive always follows the same syntax:

  <%@ taglib prefix="prefixOfTag" uri="uri" %>

  - prefix attribute informs a container what bits of markup are custom actions.
  - where the uri attribute value resolves to a location the container understands
  - Example :

    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

Life Cycle Of Simple Custom Tag

1. When we use custom tag on JSP page then web container(WC) starts life cycle of custom tag.
2. Using taglib directive, web container locates .tld file which is in WEB-INF/tld directory.
3. Once .tld file located, WC try to search matching tag name i.e tag suffix. As shown above it is under tag-name element.
4. Using tag name/suffix Web Container gets F.Q. class name associated with custom tag. Once class name is found web container load that class into WC's memory and instantiate it by calling parameterless constructor.
5. Web Container invokes setJspContext(...) method to pass pageContext object to the tag handler class.
6. The setters for each attribute defined for the tag are called by the container.
7. If a body exists, the setJspBody() method is called by the container to set the body of this tag. If the action element is empty in the page, this method is not called at all
8. Finally Web Container invokes doTag() method which contains tag logic.

Thank you