Java Server Pages

Sandeep, SunBeam

Directives

- Directives are messages to a JSP container. They do not send output to a client.

- All directives use the following syntax:

```
<%@ directive {attribute="value"} %>
```

- There are three different JSP directives for use on a page:

1. **page**
2. **include**
3. **taglib**

Page Directive(<%@ page %>)

- The page directive provides page-specific information to a JSP container.

- Attributes for the page directive are as follows:
  1. language
  2. import
  3. session
  4. isErrorPage
  5. errorPage
  6. isThreadSafe

Include Directive(<%@ include %>)

- The include directive is used to include text and/or code at translation time of a JSP.
- The include directive always follows the same syntax:

```
<%@ include file="relativeURL" %>
```

Files included must be part of a Web Application.

- Since include directives take place at translation time, they are equivalent of directly including the source code in the JSP before compilation and do not result in performance loss at runtime.
- A good example to use is including a common header and footer with multiple pages of content.
- Example:

```
<body>
        <%@ include file="Header.jsp" %>
        <h2 align="center">Body</h2>
        <%@ include file="Footer.jsp" %>
</body>
```

Taglib Directive(<%@ taglib %>)

- If we want to use custom tag library then we should use taglib directive.

- The **taglib** directive has following syntax

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

  - **uri** attribute value resolves to a location the container understands.
  - The **prefix** attribute informs a container what markup are custom actions.

Exception Handling

- There are three generic types of Throwable objects thrown by Servlets and JSP
    1. IOException
    2. ServletException
    3. JspException
- Exceptions thrown from a Servlet or JSP can be handled on an individual basis or on an application-wide basis.
- In addition to the try-catch-finally statement, JSP can use the page directive to specify a page that uncaught exceptions are passed to.
- The page directive's "**errorPage**" attribute can be assigned a relative URL value representing a JSP or Servlet especially designed to be an error page.

```
<%@ page errorPage="ErrorPage.jsp" %>
```

- A JSP designed to be an error page can set the page directive isErrorPage attribute to true; this makes the exception implicit object automatically available to represent a passed exception.

```
<%@ page isErrorPage="true" %>
```

Exception Handling

- Error pages can be defined on a per Web Application basis by a Web Application Deployment Descriptor, web.xml.
- web.xml Error Page for All Checked Exceptions:

```
<error-page>
        <exception-type> java.lang.Exception </exception-type>
        <location> /ErrorPage.jsp </location>
</error-page>
```

- web.xml Error Page Entry for HTTP Status Code:

```
<error-page>
        <error-code>404</error-code>
        <location>/FileNotFound.jsp</location>
</error-page>
```

Exception Handling

- **ErrorData** is a final class declared in **javax.servlet.jsp** package.

- It contains information about an error, for error pages.

- Methods:
    1. public String **getRequestURI**()
    2. public int **getStatusCode**()
    3. public String **getServletName**()
    4. public Throwable **getThrowable**()

Scope in JSP

- There are 4 types of scope in JSP:
    1. page
    2. request
    3. session
    4. application.

JSP Expression Language

- JSP EL is a specification.
- tomcat/lib contains "el-api.jar" which contains implementation of JSP EL specification.
- The JSP EL is an alternative to the JSP expressions because it provides a cleaner syntax.
- The benefits of the JSP EL are easy to illustrate.
- We can use JSP expression to access a runtime value. The syntax works, but it is awkward when embedded as a tag's attribute value : **<ex:tag attribute="<%= pageContext.getAttribute("value") %>">**
- Using the JSP EL, it would look like this : **<ex:tag attribute="${value}">**
- The functionality is the same, but the code is much simpler. Another feature of the JSP EL is that it can be used anywhere in a JSP, not just in a custom tag attribute. This makes the JSP EL a simple, powerful alternative to scripting elements.

JSP EL Syntax

- Expressions are always enclosed by the ${ } characters.
- Attributes in the EL are accessed by name, with an optional scope.
- Member, getter method, and array items are all treated in the same fashion and replaced with a “dot(.)”.
- For example:
    1. an object a with a member b would be accessed as ${a.b}.
    2. an array a could have the b member accessed with the same expression in other words, a[b] could be written as ${a.b}.
    3. a JavaBean with a getter method for b would use the same expression in other words, a.getB() could also be written as ${a.b}.

The EL implicit objects(11)

1. pageScope
2. requestScope
3. sessionScope
4. applicationScope
5. param
6. paramValues
7. header
8. headerValues
9. cookie
10. initParam
11. pageContext

The EL : Example

- In "First.jsp", we have set "Name" attribute in diff scope:

```
<%
        String name = "Sandeep";
        pageContext.setAttribute("Name", name);
        request.setAttribute("Name", name);
        session.setAttribute("Name", name);
        application.setAttribute("Name", name);
        RequestDispatcher rd = request.getRequestDispatcher("Second.jsp");
        rd.forward(request, response);
%>
```

The EL : Example

- We can access these attributes on "Second.jsp”
- Using JSP Scriptlet

```
<%
        String name = "";
        name = ( String )pageContext.getAttribute("Name");
        out.println("Page Scope : "+name);
        name = ( String )request.getAttribute("Name");
        out.println("Request Scope : "+name);
        name = ( String )session.getAttribute("Name");
        out.println("Session Scope : "+name);
        name = ( String )application.getAttribute("Name");
        out.println("Application Scope : "+name);
%>
```

The EL : Example

- We can access these attributes on "Second.jsp”

- Using JSP Expression:

```
Page Scope:<%=pageContext.getAttribute("Name") %><br/>
Request Scope:<%=request.getAttribute("Name") %><br/>
Session Scope:<%=session.getAttribute("Name") %><br/>
Application Scope:<%=application.getAttribute("Name") %>
```

The EL : Example

- We can access these attributes on "Second.jsp”

- Using JSP EL:

```
Page Scope:${pageScope.Name}<br/>
Request Scope:${requestScope.Name}<br/>
Session Scope:${sessionScope.Name}<br/>
Application Scope:${applicationScope.Name}<br/>
```

The EL : Example

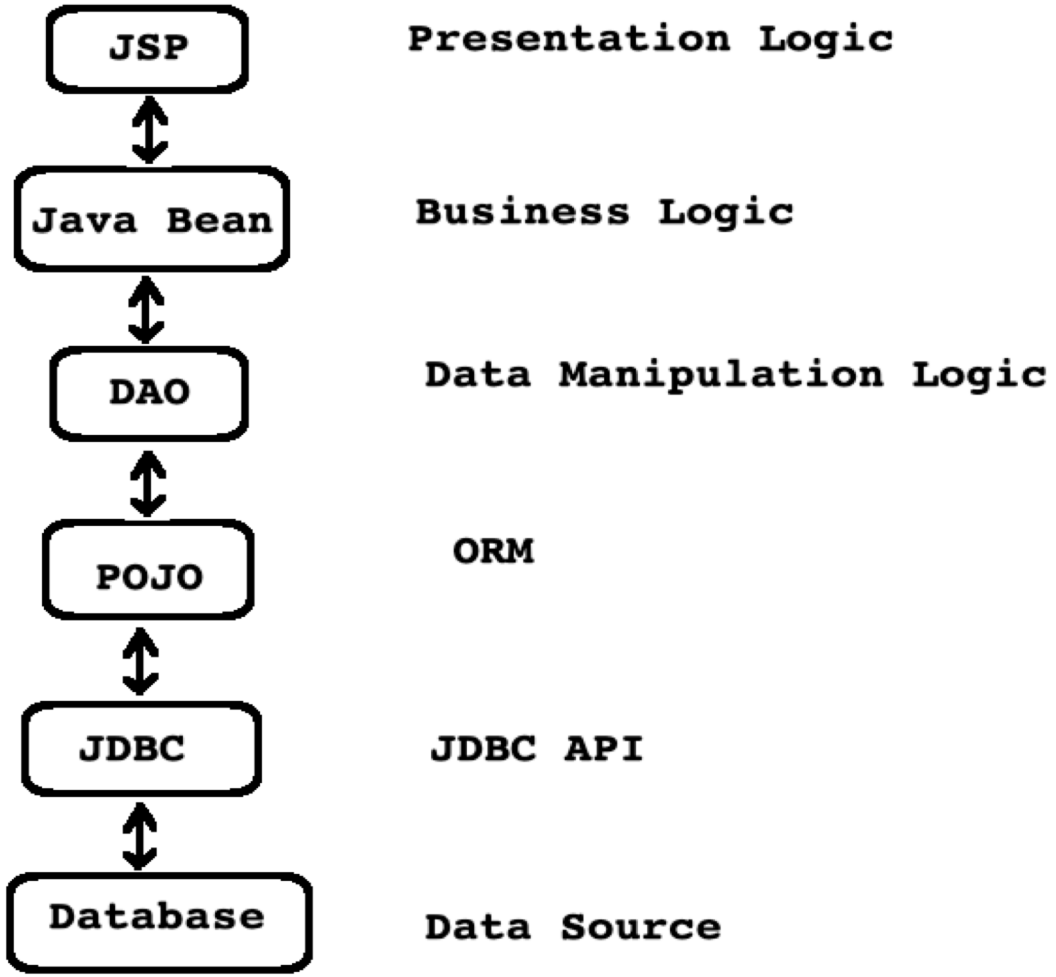
- ${value} will look for the attribute named value by searching the page, request, session, and application scopes in that order. If the attribute is not found, null is returned.

- How ${value} gets evaluated:
  - if pageContext.getAttribute("value") not null them return string
  - else if request.getAttribute("value") not null them return string
  - else if session.getAttribute("value") not null them return string
  - else if application.getAttribute("value") not null them return string
  - else return blank to the browser.

Java Bean

1. It must be packaged public class.
2. It should implement the java.io.Serializable interface.
3. It must have default constructor.
4. Private data member of the java bean is called property. Java Bean property must be non static and non transient.
5. For every property, getter and setter method should be exist inside a class.
6. It can contain business logic methods.

Layers


JSP
Java Bean
DAO
POJO
JDBC
Database
Presentation Logic
Business Logic
Data Manipulation Logic
ORM
JDBC API
Data Source

Why to use Java Beans?

1. It allows us to separate business logic from JSP.

2. We can share java bean instances on multiple pages i.e. we can achieve reusability.

3. If java bean properties are of primitive type then web container automatically translate request parameters into java bean properties.

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);
```

Thank you