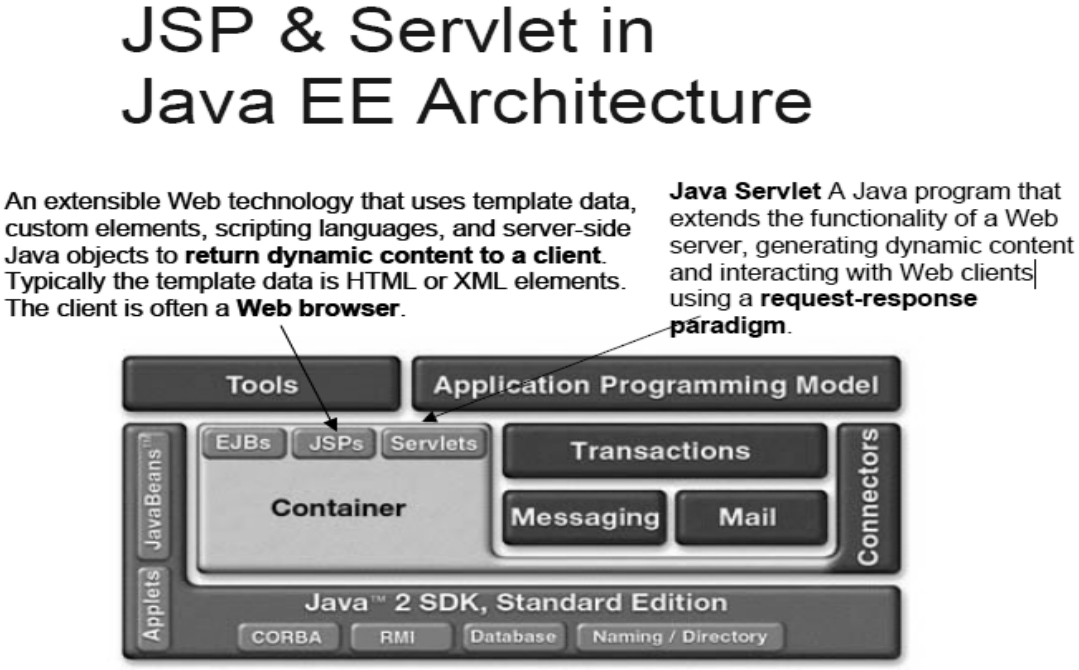
9. JAVA SERVER PAGES (JSP)

Introduction

Java Server Pages (JSP) is a Java Technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web Client request. JSP is an enhancement for the servlet programming that allows us to design the dynamic web pages very easily.



JSP technology allows you to easily create web content that has both static and dynamic components. A JSP page is a text document that contains two types of text:

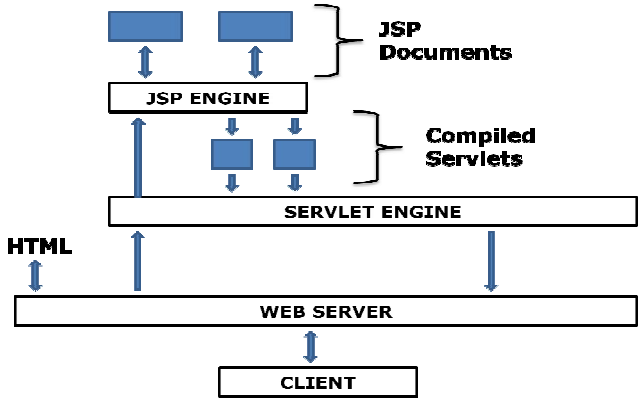
- Static data, which can be expressed in any text-based format (such as HTML, WML and XML)
- JSP elements, which construct dynamic content.

It provides expression language for accessing server-side objects. JSP specifications extend the Java Servlet API.

JSP contains HTML and also JAVA statements. HTML statements can be placed directly whereas all java statements must be enclosed in following tags:

```
<%
      // (java) statements
%>
```

JSPs are compiled into Java Servlets by a JSP compiler. A JSP compiler may generate a servlet in Java code that is then compiled by the Java compiler, or it may generate byte code for the servlet directly. JSPs can also be interpreted on-the-fly reducing the time taken to reload changes.



JSP Benefits

- Content and display logic are separated
- Simplify web application development with JSP, JavaBeans and custom tags
- Supports software reuse through the use of components (JavaBeans, Custom tags)
- Automatic deployment - Recompile automatically when changes are made to JSP pages
- Easier to author web pages
- Platform-independent

Why JSP over Servlet?

- Servlets can do a lot of things, but it is pain to:
  - Use those println() statements to generate HTML page
  - Maintain that HTML page
- No need for compiling, packaging, CLASSPATH setting

There are many ***advantages of JSP over servlet***. They are as follows:

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, JSTL, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Do I have to Use JSP over Servlet or vice-versa?

- No, you want to use both leveraging the strengths of each technology
  - ✍ Servlet's strength is "controlling and dispatching"
  - ✍ JSP's strength is "displaying"
- In a typically production environment, both servlet and JSP are used in a so-called MVC (Model-View-Controller) pattern
  - ➢ Servlet handles controller part
  - ➢ JSP handles view part

Servlets vs JSP

Servlets	**JSP**
HTML code in JAVA	JAVA-like code in HTML
Not easy for author	Very easy for author
	Code is compiled into servlets

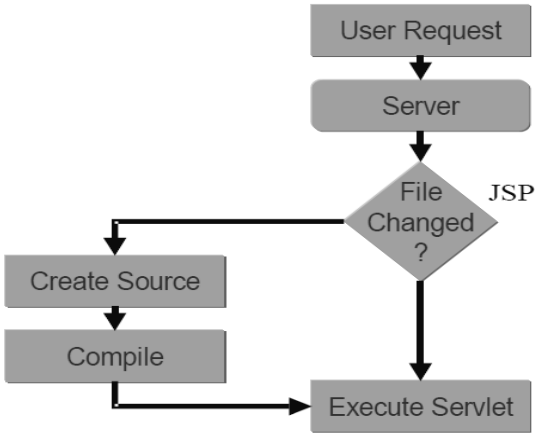
A Simple JSP Page

```
<html>
      <body>
         <h1> Hello World! <br>
         Current time is <%= new java.util.Date() %> </h1>
      </body>
</html>
```

Life Cycle of a JSP Page

A JSP page service requests as a servlet. Thus, the life cycle of JSP pages is determined by Java Servlet Technology. When a request is mapped to a JSP page, the web container first checks whether the JSP page's Servlet is older than the JSP page. If the servlet is older, the web container translates the JSP page into a servlet class and compiles the class.

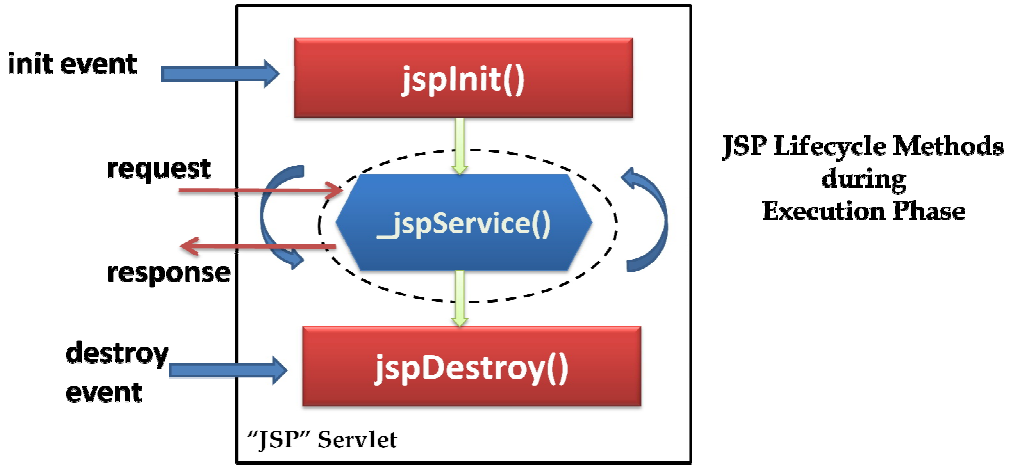
How Does JSP work?



- **Translation and Compilation Phases**
  During the translation phase each type of data in a JSP page is treated differently. Static data is transformed into code that will emit the data into the response stream. JSP elements are treated as follows:

Elements	Description
Directives	Are used to control how the web container translates and executes the JSP page
Scripting elements	Are inserted into the JSP page's servlet class.
Expression language	Expressions are passed as parameters to calls to the JSP expression evaluator.
jsp[set|get] Property	Elements are converted into method calls to JavaBeans components.
jsp [include|forward]	Elements are converted into invocations of the Java Servlet API.
Custom Tags	Are converted into calls to the tag handler that implements the custom tag.

- **Execution Phase**
  After the page has been translated and compiled, the JSP page's servlet follows the servlet life cycle:
  - If an instance of the JSP page's servlet does not exist, the container
    - loads the JSP page's servlet class;
    - instantiates an instance of the servlet class then
    - initializes the servlet instance by calling the **jspInit** method.
  - The container invokes the **_jspService** method, passing request and response objects.
  - If the container needs to remove the JSP page's servlet, it calls the **jspDestroy** method.



Initialization of a JSP page

- Declare methods for performing the following tasks using JSP declaration mechanism
  - Read persistent configuration data
  - Initialize resources
  - Perform any other one-time activities by overriding jspInit() method of JspPage interface

Finalization of a JSP page

- Declare methods for performing the following tasks using JSP declaration mechanism
  - Read persistent configuration data
  - Release resources
  - Perform any other one-time cleanup activities by overriding jspDestroy() method of JspPage interface

Scripting Elements

Declaration Tag

- Declaration is a block of Java code that is used to define class-wide variables and methods in the generated class file.
- Declarations are initialized when the JSP page is initialized.
- Declaration block is enclosed between **<%! and %>**
- Static data members may be defined as well. Also inner classes should be defined here. **Syntax: <%! field or method declaration %>**

Example of JSP declaration tag that declares field

```
<html>
      <body>
      <%! int data=50; %>
      <%= "Value of the variable is:"+data %>
      </body>
</html>
```

Example of JSP declaration tag that declares method

```
<html>
      <body>
      <%!
         int cube(int n) {
            return n*n*n;  }
      %>
      <%= "Cube of 3 is:"+cube(3) %>
      </body>
</html>
```

Scriptlets

It is a block of Java code that is executed when JSP is executed. In JSP, java code can be written inside the jsp page using the scriptlet tag. It is placed in jspService() method. It is enclosed between **<% and %>**

```
<% out.println(“Hello JSP World”); %>
```

Comments: Comments are given using **<%-- and --%>**

```
<%-- comment --%>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Expressions

It is a shorthand notation for a scriptlet that outputs a value in the response stream back to the client. It is enclosed between **<%= and %>**

```
<%=new Date()%>
<input type=“text” value=‘<%=request.getParameter(“age”)%>’ name=‘age’>
```

Example: Program that shows how to invoke JSP from HTML form and send data.

wishform.html

```
<html>
    <head>
        <title>Send Data to another JSP</title>
    </head>
    <body bgcolor="#ffdead">
    <center>
        <h2>Wish Form</h2>
        <form action="wish.jsp">
            <b>Enter your Name: </b>    
            <input type="text" name="uname" size="20"/>
            <input type="submit" value="Submit"/>
        </form>
    </center>
    </body>
</html>
```

wish.jsp

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

```
<html>
    <head>
        <title>Display Data</title>
    </head>
    <body bgcolor="#ffdead">
        <h1>Hello World!</h1>
        <%
            String name=request.getParameter("uname");
            if(name == null) { //no parameter passed
                out.println("<h2>NO PARAMETER PASSED</h2>"); }
            else {
                if(name.equals("")) { //empty string was passed
                    out.println("<h2>EMPTY NAME</h2>"); }
                else {
                    out.println("<h2>WELCOME "+name+"</h2>");
                }
            }
        %>
    </body>
</html>
```

Example: Program to create an HTML form and script to process the input given by the form

When JSP is called first, it displays form. When user submits form, it calls the same JSP again. It is important to differentiate the first call from other calls (post back – JSP calling itself).

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Process Data in Single Page</title>
    </head>
    <body bgcolor="#ffdddd">
        <h1>Process Data in Single JSP</h1>
        <form action="process.jsp">
            <b>Enter a number: </b>    
            <input type="text" name="num" size="20"/>
            <input type="submit" value="Submit"/>
        </form><br/>
        <%
            String no = request.getParameter("num");
            if(no!=null && no.length()>0)
            {
                int n = Integer.parseInt(no);
```

```
                if(n%2 == 0) {
                    %>
      <font face="Verdana" size="4" color="blue"><%=n%> is Even Number</font>
                    <%
                } else {
                    %>
      <font face="Verdana" size="4" color="red"><%=n%> is Odd Number</font>
                    <%
                }
            }
        %>
    </body>
</html>
```

Implicit Objects: JSP implicit objects are exposed by the JSP container and can be referenced by the programmer:

- **out:** The JSPWriter used to write the data to the response stream (output page).
- **page:** The servlet itself.
- **pageContext:** A PageContext instance that contains data associated with the whole page. A given HTML page may be passed among multiple JSPs.
- **request:** The HttpServletRequest object that provides HTTP request information.
- **response:** The HttpServletResponse object that can be used to send data back to the client.
- **session:** The HttpSession object that can be used to track information about a user from one request to another.
- **config:** Provides servlet configuration data.
- **application:** Data shared by all JSPs and servlets in the application.
- **exception:** Exceptions not caught by application code.
- **servletContext:** the context for the JSP page's servlet and any web components contained in the same application.

Example: Program to show usage of Implicit Objects

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ page import="java.util.Date" %>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html;charset=US-ASCII">
 <title>Implicit Objects Demo</title>
 </head>
 <body>
  <h4>Hi There</h4> <%-- out object example --%>
  <strong>Current Time is</strong>: <%out.print(new Date());%><br><br>
```

```
  <%-- request object example --%>
  <strong>Request User-Agent</strong>:
  <%=request.getHeader("User-Agent")%><br><br>
  <%-- response object example --%>
  <%response.addCookie(new Cookie("Test", "Value"));%>
  <%-- config object example --%>
  <strong>User init param value</strong>:
  <%=config.getInitParameter("User")%><br><br>
  <%-- application object example --%>
  <strong>User context param value</strong>:
  <%=application.getInitParameter("User")%><br><br>
  <%-- session object example --%>
  <strong>User Session ID</strong>:<%=session.getId()%><br><br>
  <%-- pageContext object example --%>
  <% pageContext.setAttribute("Test", "Test Value");%>
  <strong>PageContext attribute</strong>: {Name="Test",
        Value="<%=pageContext.getAttribute("Test")%>"}<br><br>
  <%-- page object example --%>
  <strong>Generated Servlet Name</strong>:
        <%=page.getClass().getName()%>
 </body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <context-param>
        <param-name>User</param-name>
        <param-value>Admin</param-value>
    </context-param>
    <servlet>
        <servlet-name>Demo</servlet-name>
        <jsp-file>/implicitdemo.jsp</jsp-file>
        <init-param>
            <param-name>User</param-name>
            <param-value>Srikanth</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>Demo</servlet-name>
        <url-pattern>/home.do</url-pattern>
        <url-pattern>/implicitdemo.jsp</url-pattern>
    </servlet-mapping>
</web-app>
```

Example of session implicit object

index.html

```
<html>
<body>
      <form action="welcome.jsp" method="post">
            <input type="text" name="uname">
            <input type="submit" value="go"><br/>
      </form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
      <%
      String name=request.getParameter("uname");
      out.print("Welcome "+name);
      session.setAttribute("user",name);
      %>
      <a href="second.jsp">second jsp page</a>
</body>
</html>
```

second.jsp

```
<html>
<body>
      <%
       String name=(String)session.getAttribute("user");
       out.print("Hello "+name);
      %>
</body>
</html>
```

JSP Directives

It specifies what JSP container must do. Directives start with @ character within the tag. It control how the JSP compiler generates the servlets. The following are available:

- page
- include
- taglib

- **page Directive**
  - Defines information that will be globally available for that JSP
  - Eg: **<%@ page language="java" %>**

Attribute	Value	Example	Default
Info	Text string	info="Registration form"	None
Language	Language name	language="java"	Java
ContentType	MIME type	contentType="text/html"	None
Import	Class or Package	import = "java.util.*"	None
Session	Boolean flag	session="true" or "false"	True
ErrorPage	Local URL	errorpage="failed.jsp"	None
IsErrorPage	Boolean flag	iserrorpage="false"	False or True

  - *Valid page directive statements:*

```
<%@ page info = "This is a valid set of page directives." %>
// one attribute
<%@ page language="java" import="java.net.*" %>
//multiple attributes in one statement
<%@ page import="java.net.*, java.util.Date" %>
//more than one value for one attribute
<% @ page errorPage = "page-name" %>
<% @ page isErrorPage = "true" %>
```

Example: Program to illustrate page Directive with Attributes

```
<%@ page info="Created by Srikanth" %>
<%@ page contentType="application/vnd.msword"%>
<%@ page import="java.util.Date" %>
<%response.setHeader("Content-Disposition","attachment;filename=demo.doc");%>
<html>
      <body>
            Today is: <%= new Date() %>
      </body>
</html>
```

Exception Handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer.

In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Example of exception handling in JSP by the elements of page directive

index.jsp

```
<html>
      <body>
            <form action="process.jsp" method="post">
                  No1:<input type="text" name="n1" /><br/><br/>
                  No2:<input type="text" name="n2" /><br/><br/>
                  <input type="submit" value="divide"/>
            </form>
      </body>
</html>
```

process.jsp

```
<%@ page errorPage="error.jsp" %>
<%
      String num1=request.getParameter("n1");
      String num2=request.getParameter("n2");
      int a=Integer.parseInt(num1);
      int b=Integer.parseInt(num2);
      int c=a/b;
      out.print("division of numbers is: "+c);
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<h3>Sorry an exception occured!</h3>
Exception is: <%=exception%>
```

- **include Directive**
  - informs the JSP compiler to include a complete file into the current file.
  - Enables programmers to include the contents of one file in another file both at output time and translation time
  - Eg: **<%@ include file="filename.html" %>**
    <% @ include file = "filename.jsp" %>

Example: Program to illustrate include Directive

header.html

```
<!DOCTYPE html>
<html>
    <head><style type="text/css">
            ul {
                list-style-type:none;
                margin:0;
                padding:0;
                overflow:hidden;
            }
```

```
            li {
                float:left;
            }
            a:link,a:visited {
                display:block;
                width:120px;
                font-weight:bold;
                color:#FFFFFF;
                background-color:#98bf21;
                text-align:center;
                padding:4px;
                text-decoration:none;
                text-transform:uppercase;
            }
            a:hover,a:active {
                background-color:#7A991A;
            }
        </style>
    </head>
    <body>
    <center><img src = "banner.jpg" alt="" style="width:100%;height:150px;"/>
        <ul>
            <li><a href="#home">Home</a></li>
            <li><a href="#news">News</a></li>
            <li><a href="#contact">Contact</a></li>
            <li><a href="#about">About</a></li>
        </ul>
    </center>
</body>
</html>
```

jspincludedemo.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <body>
        <%@ include file="header.html" %> <br/>
        <h1> You can include both HTML and JSP pages. <br/><br/>
             <font color="blue">Top is Header (HTML).
             Bottom is Dynamic Content (JSP)</font>
        </h1>
        <%@ include file="sample.jsp"%>
    </body>
</html>
```

- **taglib Directive**
  - indicates that a JSP tag library is to be used. The directive requires that a prefix be specified (much like a namespace in C++) and the URI for the tag library description.
  - Eg: **<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>**
    - uri - References a URI that uniquely name the set of custom tags.
    - prefix – defines the prefix string used to distinguish a custom tag instance.

JavaBeans

JavaBeans component design conventions govern the properties of the class and govern the public methods that give access to the properties.

A JavaBeans component property can be:

- Read/write, read-only or write-only
- Simple, which means it contains a single value or indexed, which means it represents an array of values.
- It provides a default, no-argument constructor.
- It should be serializable and implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.
- A property does not have to be implemented by an instance variable. It must simply be accessible using public methods that conform to the following conventions:

For each readable property, the bean must have a method of form:

type getProperty() { … }

For example, if property name is firstName, your method name would be getFirstName() to read that property. This method is called accessor.

For each writable property, the bean must have a method of form:

type setProperty() { … }

For example, if property name is firstName, your method name would be setFirstName() to write that property. This method is called mutator.

Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

```
public class StudentsBean implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
```

```
   private int age = 0;
   public StudentsBean() {
   }
   //setter and getter methods
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

Using JavaBeans with JSP

In order to use a JavaBean in JSP, you have to use <jsp:useBean> standard action as follows:

<jsp:useBean>

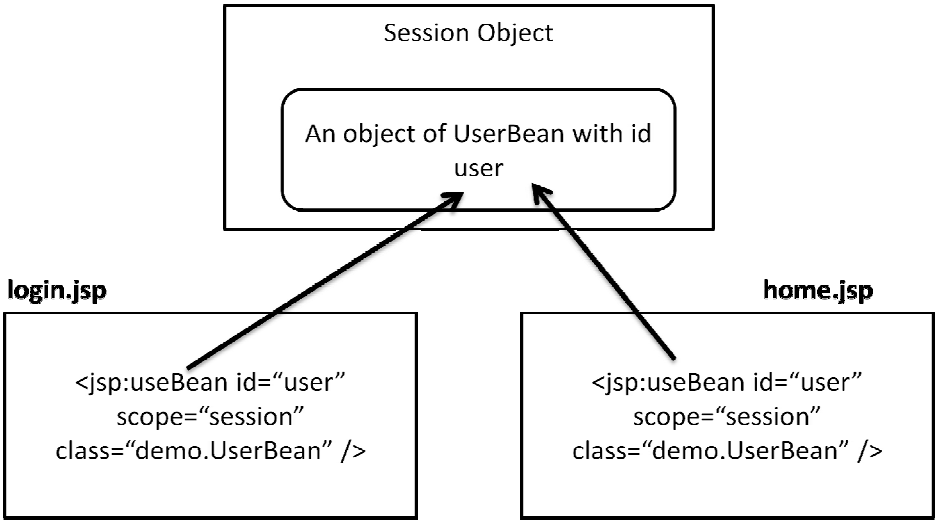
This standard ***action tag*** of JSP is used to create or access an instance of JavaBean. It specifies class, scope and id for JavaBean.

```
<jsp:useBean id="name" scope="scope" class= "fullyqualifiedclassname">
<jsp:useBean id="c" scope="page" class= "test.Count">
```

The above example creates an object of **test.Count** class and assigns id **c**. Its life time is the execution of the page. Remember the class **Count** is to be placed in package **test**. This package starts with **classes** directory (/WEB-INF/classes).

Once a bean is included in JSP, throughout the page the bean can be accessed using **id** of the bean. The scope of the bean can be any of the following:

Scope	Meaning
page	Objects are available throughout the page. They are released when request is over or another page is invoked.
request	Objects are available as long as the request is being processed.
session	Objects are available in the session in which they are created.
application	Objects are available in any session that is in the same application as the session that created the bean.



<jsp:setProperty>

It is used to set property of the bean to the given value or the specified parameter of the form. The jsp:setProperty ***action tag*** sets a property value or values in a bean using the setter method.

```
<jsp:setProperty name="beanname" property="propertyname|*"
                                        value="value" | param="parametername" />
```

The following table shows the meaning of each attribute:

Attribute	Meaning
name	Name of the bean defined by <jsp:useBean> tag
property	Name of the property to be changed. If the property is set to * then the tag iterates over all the parameters of the request, matching parameter names and value types to bean properties, and setting each matched property to the value of the matching parameter.
param	Specifies the name of the request parameter whose value is to be assigned to the bean property.
value	Value to assign to the bean property. This cannot be given if param is already given.

Example:

```
<jsp:setProperty name="emp" property="*"/>
<jsp:setProperty name="emp" property="bs" param="sal"/>
<jsp:setProperty name="emp" property="bs" value="12000"/>
```

<jsp:getProperty>

It is an ***action tag*** used to retrieve the value of the parameter.

```
<jsp:getProperty name="beanname" property="propertyname" />
```

Name is the name of the bean specified by <jsp:useBean> tag and property is the name of the property whose value is to be retrieved.

```
<jsp:getProperty name="emp" property="sal" />
```

Example of JavaBeans

JobBean.java

```
package beans;
import java.sql.*;
public class JobBean {
    private String id, title;
    public JobBean() { }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public int addJob() throws Exception { //business logic
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@Localhost:1521:XE","hr","hr");
        Statement st = con.createStatement();
        PreparedStatement ps = con.prepareStatement
                    ("insert into jobs values(?,?,null,null)");
        ps.setString(1, id);
        ps.setString(2, title);
        int k = ps.executeUpdate();
        ps.close();
        con.close();
        return k;
    }
}
```

addjob.html

```
<html>
    <head>
        <title>Add Job</title>
    </head>
    <body>
        <form action="addjob.jsp">
            <h2>Add Job</h2>
            Id: <input type="text" name="id" size="10" />
            Title: <input type="text" name="title" size="20" />
```

```
            <input type="submit" value="Add" />
        </form>
    </body>
</html>
```

addjob.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<jsp:useBean class="beans.JobBean" scope="page" id="job" />
<jsp:setProperty name="job" property="*" />
<!-- This calls all the setter methods in the Bean setId, setTitle -->
<html>
    <head><title>Add Job</title></head>
    <body>
        <%
            try {
                int r = job.addJob();
                if(r>0) {
                  out.println("<h1><i>"+job.getTitle()+"
                        Added Successfully!</i></h1>");  }
                else
                    out.println("<h1><i>Unsuccessful</i></h1>");
            }
            catch(Exception e) {  out.println(e); }
        %>
    </body>
</html>
```

JSP Actions

JSP actions are XML tags that invoke built-in web server functionality. They are executed at runtime. Some are standard and some are custom (which are developed by Java developers). The following list contains the standard ones:

- **jsp:include**
  - Similar to a subroutine, the Java servlet temporarily hands the request and response off to the specified JSP. Control will then return to the current JSP, once the other JSP has finished. Using this, JSP code will be shared between multiple other JSPs, rather than duplicated.
  - The jsp:include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future.
  - The jsp:include tag can be used to include static as well as dynamic pages.

```
<jsp:include page=“urlspec” flush =“true”>
      <jsp:param name=“name” value=“value”>
</jsp:include>
```

If the page is buffered then the buffer is flushed prior to the inclusion. This action tag may take one or more <jsp:param> tags in its body.

- **jsp:forward**
  - Allows the request to be forwarded to another JSP, a servlet or a static resource.
  - The resource to which the request is being forwarded must be in the same context as the JSP dispatching the request. Execution in the current JSP stops when it encounters a <jsp:forward> tag.

```
<jsp:forward page=“urlspec”>
      <jsp:param name=“name” value=“value”>
</jsp:forward>
```

Example:

includeactiondemo.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Include/Forward Action Tag Demo</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <h1>Name of the Student: </h1>
        <%--<jsp:include page="name.jsp" >--%>
        <jsp:forward page="name.jsp" >
            <jsp:param name="firstname" value="Krishna" />
            <jsp:param name="middlename" value="Srikanth" />
            <jsp:param name="lastname" value="K S V" />
        </jsp:forward>
        <%--</jsp:include>--%>
    </body>
</html>
```

name.jsp

```
<%
    String f = request.getParameter("firstname");
    String m = request.getParameter("middlename");
    String l = request.getParameter("lastname");
    String name = l + ". " + f + m;
%> <h1><%=name%></h1>
```

- **jsp:plugin**
  - Older versions of Netscape Navigator and Internet Explorer used different tags to embed an applet.

- This action generates the browser specific tag needed to include an applet.
- Generates browser-specific codes that makes an OBJECT or EMBED tag for the Java plugin.

```
<jsp:plugin align="middle" height="500" width="500"
     type="applet"  code="MouseDrag.class" name="clock" codebase="."/>
```

Expression Language (EL) in JSP

- The Expression Language (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
- There are many implicit objects, operators and reserve words in EL.
- It is the newly added feature in JSP technology version 2.0.

Syntax for Expression Language (EL)

```
${expression}
```

Implicit Objects in Expression Language (EL)

There are many implicit objects in the Expression Language. They are as follows:

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

Expression language supports the following capabilities:

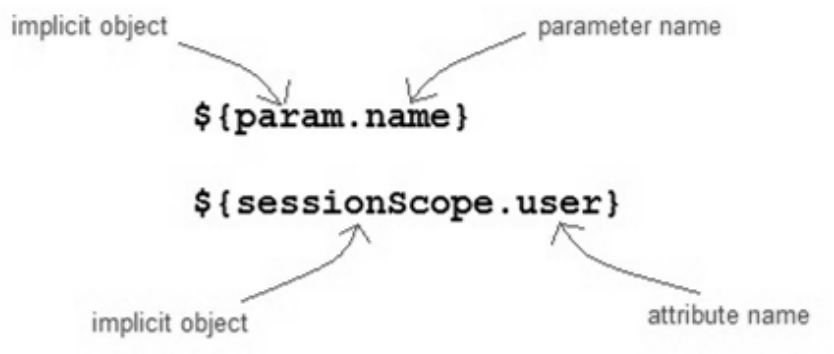
- ❑ Concise access to attributes in objects like session, application
- ❑ Shorthand notation for the bean properties
- ❑ Simple access to collection elements
- ❑ Provides arithmetic, relational, logical operators to manipulate objects within EL.
- ❑ Automatic type conversion.
- ❑ In most cases, missing values result in empty strings and not exceptions.

Precedence of Operators in EL

There are many operators that have been provided in the Expression Language. Their precedence are as follows:

[] .

()
-(unary) not ! empty
* / div % mod
+ - (binary)
< <= > >= lt le gt ge
== != eq ne
&& and
|| or
?:



Reserve words in EL

There are many reserve words in the Expression Language. They are as follows:

lt	le	gt	ge
eq	ne	true	false
and	or	not	instanceof
div	mod	empty	null

Examples:

```
<% request.setAttribute(“name”,“Srikanth”) %>
${name}
${user.name} //user is name of JavaBean
${param.name}
<p>
${header[“User-Agent”]}  //browser
<p>
${cookie.JSESSIONID.value}
<p>
${param.n1 + param.n2}
<p>
${param.n1 == param.n2} //returns true
<p>
${param.n1 > param.n2? “First is big” : “Second is big”}
```

Simple example of Expression Language that prints the name of the user

In this example, we have two files **index.jsp** and **process.jsp**. The index.jsp file gets input from the user and sends the request to the process.jsp which in turn prints the name of the user using EL.

index.jsp

```
<html>
<body>
<form action="process.jsp">
Enter Name:<input type="text" name="name" /><br/><br/>
```

```
<input type="submit" value="go"/>
</form>
</body>
</html>
```

process.jsp

```
Welcome, ${param.name}
```

Example of Expression Language that prints the value set in the session scope

In this example, we printing the data stored in the session scope using EL. For this purpose, we have used sessionScope object.

index.jsp

```
<h3>welcome to index page</h3>
<%
session.setAttribute("user","sonoo");
%>
<a href="process.jsp">visit</a>
```

process.jsp

```
Value is ${ sessionScope.user }
```

How EL expression is use

EL expression can be used in two ways in a JSP page

1. As attribute values in standard and custom tags. `<jsp:include page="${location}">`
2. As a output with HTML tag `<h1>Welcome ${name}</h1>`