Java Server Pages, JSP

Java Server Pages, JSP

Java server pages is a technology for developing web pages that include dynamic content.

A JSP page can change its output based on variable items, identity of the user, the browsers type and other information.
A JSP contains

- HTML
- JSP elements
- Scriptlets
- User defined tags, including JSTL, Java Standard Tag Library

JSP elements can be used to access JavaBeans, passing control to other pages and servlets, sharing information between requests and users.

JSTL adds functionality such as selections, iterations, translations etc.

A user defined tag can do whatever you like. It is used to extend the functionality of a JSP and to hide implementation details of some algorithm.
Other techniques:

- **CGI, Common Gateway Interface**
  This is an executable program that is started by the server and that outputs HTML. A CGI is written in any language and compiled and built, this means that it is platform dependent. Requires more resources because it is started for each request, runs and terminates.

- **ASP and .NET**
  Similar to JSP but based on Visual Basic. Active X components written in other languages (eg C++) can be invoked by scripts. Database access tools. Newer versions (.NET) compiles and is more efficient, uses languages like C#, JScript.NET etc. Limited support for nonWindows systems.
PHP
PHP is a freeware scripting language. Looks like a mixture of Perl, C++ and Java. Can communicate with the mySQL database, send mail etc. Widely supported.

Coldfusion.
Same principles as PHP, uses CFML, ColdFusion Markup Language. Can communicate with Java Servlets and EJB’s. C++ or Java can be used to customize it.
JSP advantages:

- Can be customized and extended using user defined tags.
- Compiled for increased efficiency.
- Used in combination with servlets and beans.
- JSP is not a product but a specification. There are several implementations.
- Widely used, will live long.
- An integral part of Java EE, the complete platform to develop web applications.
What you need:

- A Java 2 Software Development Kit, Java SE.
- The Servlet API
- A JSP enabled web server like Tomcat or Glassfish.
A JSP is either a plain JSP page using JSP elements or a well formed, namespace aware XML-document. These are called JSP Documents and uses a somewhat different syntax to adhere to the XML rules.

We will see examples of JSP documents later in the course. Here we will only describe plain JSP pages.
Anatomy of a JSP page

The main parts are JSP elements and template texts.

```html
<%@ page language="java" contentType="text/html" %>
<html>
<body bgcolor="white">
<jsp:useBean id = "userInfo" class= "com.ora.jsp.beans.userInfoBean">
   <jsp:setProperty name="userInfo" property="*"/>
</jsp:useBean>

The following information was saved:
<ul>
   <li> User Name: 
       <jsp:getProperty name="userInfo" property="userName"/>
   </li>
   <li> Email Address: 
       <jsp:getProperty name="userInfo" property="emailAddr"/>
   </li>
</ul>
</body>
</html>
```
JSP processing

Once compiled and loaded it remains loaded until the connection is closed or until the JSP is changed. A JSP is usually considered to be threadsafe.
JSP elements

Directives:

- `<%@page ... %>` Defines page specific attributes
- `<%@include ... %>` Includes a file during translation
- `<%@taglib ... %>` Declares a tag library
Standard action elements:

- `<jsp:useBean>` Make a JavaBean available
- `<jsp:getProperty>` Get a property from a Bean
- `<jsp:setProperty>` Set a property in a Bean
- `<jsp:include>` Include the response from a servlet or JSP during request processing
- `<jsp:forward>` Forward the request
- `<jsp:param>` Add a parameter value to a request handed off to another servlet or JSP
- `<jsp:plugin>` generate HTML that is needed to execute an applet in a browser
Scripting elements:

- `<% ... %>` Scriplet to embed code
- `<%= ... %>` Embeds an expression that produces a value. That value, a String, is part of the response
- `<%! ... %>` Declaration of instance variables and methods.
- `<%-- ... --%>` Comment

Note that you cannot have instance variables in a threadsafe page.

In Java code you can also use normal Java comments. They are ignored.

In the HTML code you can use HTML comments. They are processed and sent to the browser, that ignores them.
JSP environment:

There are some objects that are available to a JSP

- `request`, The request object
- `response`, The response object
- `pageContext`, attributes specific to this page
- `session`, attributes specific to this session
- `application`, attributes for this application
- `out`, a PrintWriter that can used for output in scriplets

etc

more about these later on
Where should my .jsp file be stored?

Tomcat searches for JSP’s in the root catalogue of the web application. This is usually called webapps/nnnn where nnnn is the name of your application.

You can use the JSP using the URL http://localhost:8080/nnnn/xxx.jsp
An example of a JSP is:

```html
<%@ page contentType="text/html" %>
<HTML>
<HEAD><TITLE>
    Calendar: A JSP APPLICATION
</TITLE></HEAD>
<BODY BGCOLOR="white">
    <jsp:useBean id="table" scope="session" class="cal.TableBean" />
    <% String time = request.getParameter("time");%>
    <FONT SIZE=5> Please add the following event:
    <BR> <h3> Date <%= table.getDate() %>
    <BR> Time <%= time %> </h3>
</FONT>
<FORM METHOD="POST" ACTION="cal1.jsp">
    <BR>
    <BR> <INPUT NAME="date" TYPE="HIDDEN" VALUE="current">
    <BR> <INPUT NAME="time" TYPE="HIDDEN" VALUE=<%= time %>
    <BR> <h2> Description of the event </h2>
    <INPUT NAME="description" TYPE="TEXT" SIZE="20">
    </h2>
    <BR> <INPUT TYPE="SUBMIT" VALUE="submit">
</FORM>
</BODY></HTML>
```
The page directive:

```jsp
<%@ page name="value"%>
```

parameters:

- `language` default is java
- `extends` default is the JSP class
- `import` imports packages or classes
- `session` True or False, determines if a session should be automatically created. True is default
- `buffer` The output buffer size
- `autoFlush` True or False, whether or not output should be flushed to the client, True is default
- **isThreadSafe** True or false, True is default
- **info** descriptive text
- **errorPage** The page to be used in case of errors
- **contentType** MIME-coding of output
- **isErrorPage** True or False. Whether this is an errorpage or not. False is default
- **pageEncoding** The character set used in the page
The forward action:

```jsp:forward page="contextrelative URL"/>
```

Forward the request to another component with the given URL. Note that the current component continue to execute but must not touch the response any longer.
The include directive:

```jsp
<%@ include file="contextrelative URL" %>
```

and the include action:

```jsp
<jsp:include page="contextrelative URL" flush = "True/False" />
```

The parameter specifies whether or not to flush the buffer before including the target.

The param action is used in the body of include or forward to add additional parameters

```jsp
<jsp:include page="navigation.jsp">
  <jsp:param name="bgColor" value="#CC0000"/>
</jsp:include>
```
The useBean action

Instantiates and/or gives access to a Bean object. If the Bean object exists it is not instantiated.

```jsp
<jsp:useBean id = "instance name"
           scope="page|session|request|application"
           class="packagepath.classname"/>
```

- `id` is the identification used when accessing the Bean i.e. a variable that holds a reference to the object.
- `scope` is the desired scope.
- `class` is the fully qualified class name, not the file path. This class name is relative to context-path/WEB-INF/classes.
The `getProperty/setProperty` action

```jsp
<jsp:getProperty name="beanID" property="propertyname"/>
<jsp:setProperty name="beanID" property="propertyname"
    value = "value" | param = "requestparameter"/>
```

Either you give an explicit value or the name of a request parameter.

A property name can be "*"", this means that request parameters that matches properties by name will be copied. This is a way to copy a lot of data from the request object to a bean object.
An example of a JSP

```html
<html><head>
<title>Login Page for the Bookshop</title>
</head>
<body bgcolor="white">
<form method="POST" action='<%=response.encodeURL("j_security_check") %>' >
  <table border="0" cellspacing="5" align="center">
    <tr>
      <td colspan="2" bgcolor="#FFDC75">
        <h2>Log in to the Bookshop</h2>
      </td>
    </tr>
    <tr>
      <td colspan="2"></td>
    </tr>
    <tr>
      <th align="right">Username:</th>
      <td align="left"> <input type="text" name="j_username"></td>
    </tr>
  </table>
</form>
</body>
</html>
```
<tr>
  <th align="right">Password:</th>
  <td align="left"><input type="password" name="j_password"></td>
</tr>
<tr>
  <td align="right"><input type="submit" value="Log In"></td>
  <td align="left"><input type="reset"></td>
</tr>
Another example

<%@page contentType="text/html" isErrorPage="true" %>
<html>
<head><title>Errorpage</title></head>
<body>
    An error has occurred:<%= exception.getMessage() %>
</body>
</html>
A third example

```html
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Hello World</title></head>
<body>
<h2>The current time is: </h2>
<p>
<%= new java.util.Date() %></p>
<jsp:useBean id="tb" scope="request" class="com.mimer.fredrik.TestBean">
  Error, the instance is created elsewhere
</jsp:useBean>
<h1>
<jsp:getProperty name="tb" property="name" />
</h1>
</body>
</html>
```
A simple bean class.

```java
package com.mimer.fredrik;

public class TestBean {
    private String name;
    private Integer age;

    public TestBean() {} 
    public String getName() {
        return name;
    }
    public void setName(String newName) {
        name = newName;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer newAge) {
        age = newAge;
    }
}
```
As said before there are a number of environment objects available to a page. These are maps, i.e. they contain name-value pairs. These are called *attributes*. They are created and maintained within the container, except for the request context that is created by the client.

The request object also contains name-value pairs transmitted from the client. These are called *parameters*. The request, page, session and application scopes are all used to transfer data between different components.
Figure: The JSP context
Parameters are unique to the request scope, otherwise the scopes are similar.

The session is user specific.

The page is page specific

The application contains the global environment, attributes etc., i.e. it is application specific.
The requests lifetime is one request, i.e. one request from a browser. It is used to transfer data from the browser. You can also use it to add attributes that are to be used during the request.

Most important methods are:

- getAttribute(String name)
- setAttribute(String name, Object Attribute)
- getParameter(String param)
- getParameterValues(String param)
- getParameterMap()
- getRequestDispatcher(String path)
Some short examples showing how to retrieve data from an HTML form

```java
String rockCheckBox = request.getParameter("Rock");

// returns a String or null

if (rockCheckBox != null) {
    You checked Rock music

Note the use of start/end markers around the text we do output. If we didn't do this we would need a printstream to output the string.
If you have a multiple select you can get the selected choices like this

```jsp
<% 
String [] selectedCountries = request.getParameterValues("countries");

// return the selected items 
for (int i=0; i < selectedCountries.length; i++) {

<%= selectedCountries[i] %> <br>
}

%>
```
To retrieve all parameters passed from the browser you can do

<%  
Map parameters = request.getParameterMap();  
Iterator i = parameters.keySet().Iterator();

while (i.hasNext()){
    String parameterName = (String) i.next();
    String parameterValue = request.getParameter(parameterName);
%>
<%= parameterName %> has value
<%= parameterValue %>. <br>
<%
}%>
Note that you can use the out IOstream to print, like this

```java
<%  
Map parameters = request.getParameterMap();  
Iterator i = parameters.keySet().Iterator();

while (i.hasNext()){  
    String parameterName = (String) i.next();  
    String parameterValue = request.getParameter(parameterName);  
    out.println(parameterName + " has value " + parameterValue + ".<br>");
}
%>

This is not really recommended, since it sort of violates the idea of a JSP, to abstract away Java.
The expression language is used to extend the "coding capability" of a JSP, thereby reducing the requirement for Java-scriptlets. You reach a higher level of abstraction.

The basic syntax element in EL is

\$\{expression\}

where expression is something that results in some kind of value.
Instead of saying

```
<%= request.getParameter("firstName"); %>
```
you say

```
${param.firstName}
```
Increases the readability alot. You avoid the explicit Java code.
Reserved words in EL:

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

Operators in EL, (in the order of preference)

[], .
()
-, !, not, empty
*, /, div, %, mod
+, -
<, >, <=, >=, lt, gt, le, ge
==, !=, eq, ne
&&, and
||, or
?:
You can use literals of type

- *Boolean*, true or false
- *Integer*, 1 23
- *Floating Point*, 2.4 0.5e-4
- *String*, "Hello" ’Hi’
- *Null*, null
There are a number of implicit (predefined) objects

- `pageContext`, the `pageContext` object
- `pageScope`, a Map that maps attributes to values
- `requestScope`, `-”-
- `sessionScope`, `-”-
- `applicationScope`, `-”-
- `param`, a Map that maps parameters to values
- `initParam`, a Map that maps initialization parameters to values

etc
Examples;

${sessionScope.firstName}
${firstName}
${param.action}
${paramValues['lastName']}
${param.action == 'show' ? paramValues['lastName'] : null}

The sum of ${1} and ${3} is ${1 + 3}

${param.action == 'show' ? paramValues['lastName'] : null}
Why do we need EL?

To reach a higher level of abstraction in our JSP’s, we don’t need the explicit scriptlets to access scoped Java objects.

EL can also be used to simplify the usage of beans. Instead of

```html
<jsp:useBean id="user" class="com.jspbook.User"/>
<jsp:getProperty name="user" property="name"/>
```

You can say

```html
${user.name}
```

provided that the bean is an attribute in some scope.