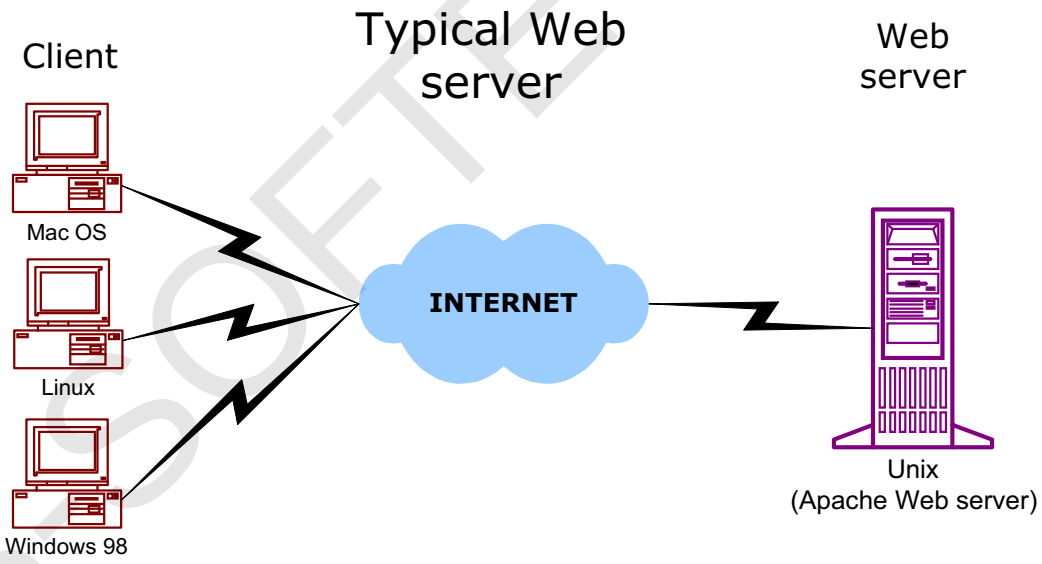
JSP tutorial

Introduction to JSP

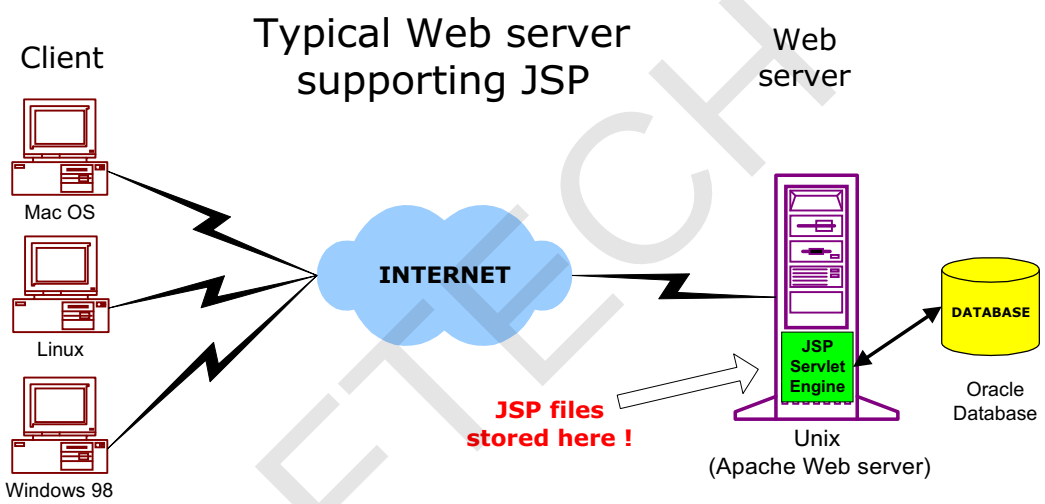
JavaServer Pages (JSP) is a technology based on the Java language and enables the development of dynamic web sites. JSP was developed by Sun Microsystems to allow server side development. JSP files are HTML files with special Tags containing Java source code that provide the dynamic content.

The following shows the Typical Web server, different clients connecting via the Internet to a Web server. In this example, the Web server is running on Unix and is the very popular Apache Web server.



First static web pages were displayed. Typically these were people's first experience with making web pages so consisted of My Home Page sites and company marketing information. Afterwards Perl and C were languages used on the web server to provide dynamic content. Soon most languages including Visualbasic, Delphi, C++ and Java could be used to write applications that provided dynamic content using data from text files or database requests. These were known as CGI server side applications. ASP was developed by Microsoft to allow HTML developers to easily provide dynamic content supported as standard by Microsoft's free Web Server, Internet Information Server (IIS). JSP is the equivalent from Sun Microsystems, a comparison of ASP and JSP will be presented in the following section.

The following diagram shows a web server that supports JSP files. Notice that the web server also is connected to a database.



JSP source code runs on the web server in the JSP Servlet Engine. The JSP Servlet engine dynamically generates the HTML and sends the HTML output to the client's web browser.

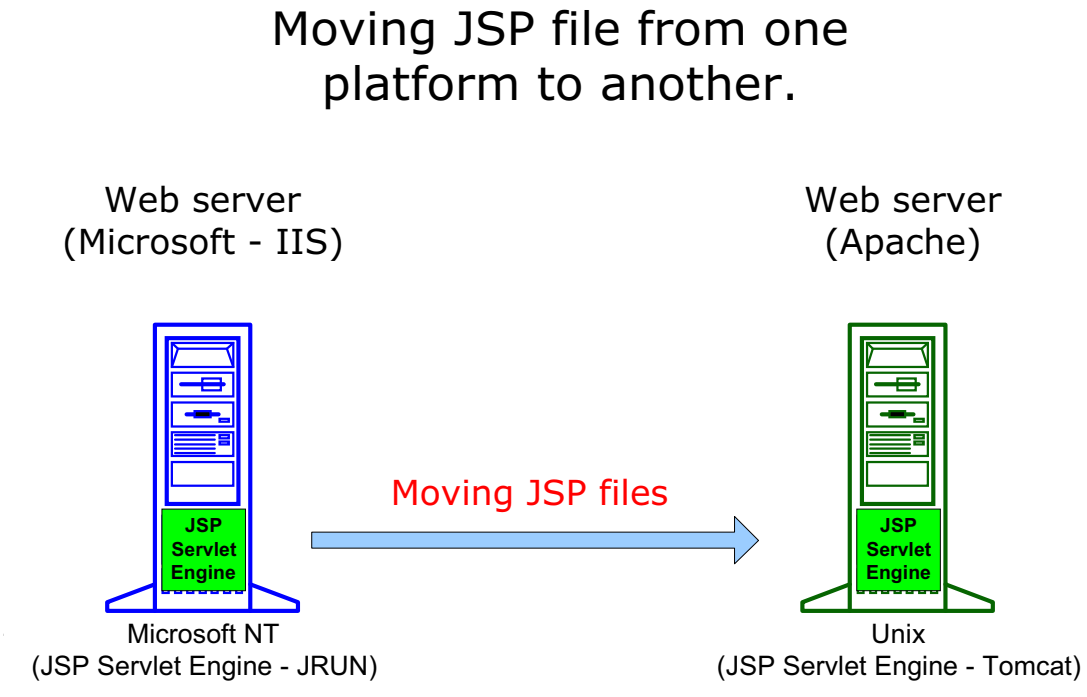
Why use JSP?

JSP is easy to learn and allows developers to quickly produce web sites and applications in an open and standard way. JSP is based on Java, an object-oriented language. JSP offers a robust platform for web development.

Main reasons to use JSP:

1. Multi platform
2. Component reuse by using Javabeans and EJB.
3. Advantages of Java.

You can take one JSP file and move it to another platform, web server or JSP Servlet engine.



HTML and graphics displayed on the web browser are classed as the presentation layer. The Java code (JSP) on the server is classed as the implementation.
By having a separation of presentation and implementation, web designers work only on the presentation and Java developers concentrate on implementing the application.

JSP compared to ASP

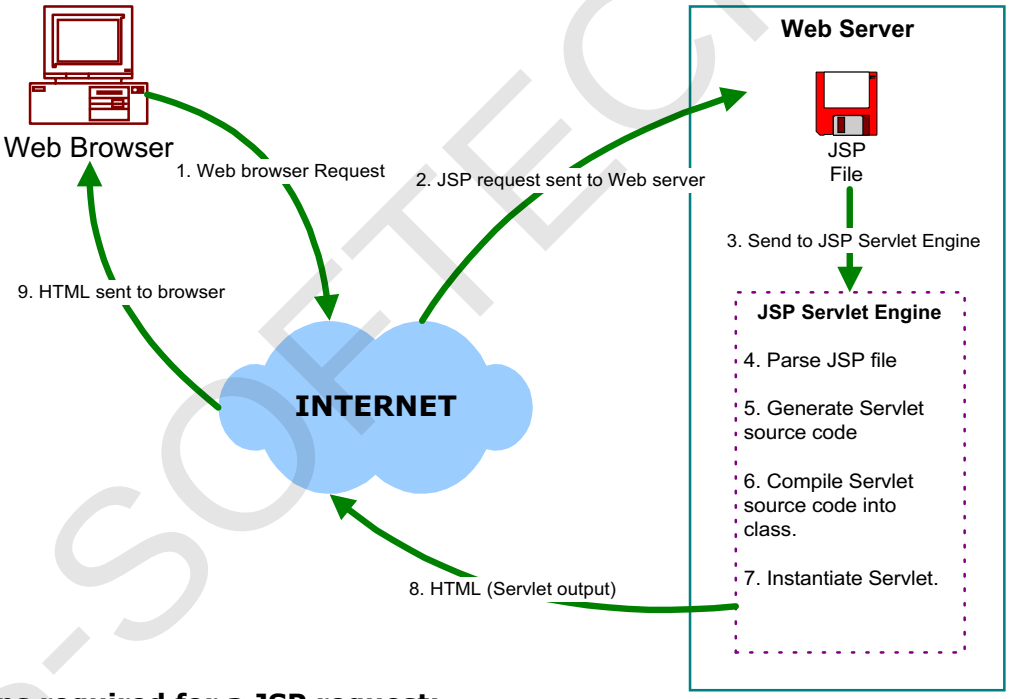
JSP and ASP are fairly similar in the functionality that they provide. JSP may have slightly higher learning curve. Both allow embedded code in an HTML page, session variables and database access and manipulation. Whereas ASP is mostly found on Microsoft platforms i.e. NT, JSP can operate on any platform that conforms to the J2EE specification. JSP allow component reuse by using Javabeans and EJBs. ASP provides the use of COM / ActiveX controls.

JSP compared to Servlets

A Servlet is a Java class that provides special server side service. It is hard work to write HTML code in Servlets. In Servlets you need to have lots of println statements to generate HTML.

JSP architecture

JSPs are built on top of SUN's servlet technology. JSPs are essential an HTML page with special JSP tags embedded. These JSP tags can contain Java code. The JSP file extension is .jsp rather than .htm or .html. The JSP engine parses the .jsp and creates a Java servlet source file. It then compiles the source file into a class file, this is done the first time and this why the JSP is probably slower the first time it is accessed. Any time after this the special compiled servlet is executed and is therefore returns faster.



Steps required for a JSP request:

1. The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). The web browser makes the request via the Internet.
2. The JSP request gets sent to the Web server.
3. The Web server recognises that the file required is special (.jsp), therefore passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time, the JSP file is parsed, otherwise go to step 7.
5. The next step is to generate a special Servlet from the JSP file. All the HTML required is converted to println statements.
6. The Servlet source code is compiled into a class.
7. The Servlet is instantiated, calling the *init* and *service* methods.
8. HTML from the Servlet output is sent via the Internet.
9. HTML results are displayed on the user's web browser.

Creating your first JSP page

```
<html>
<head>
<title>My first JSP page
</title>
</head>
<body>
<%@ page language="java" %>

<% System.out.println("Hello World"); %>
</body>
</html>
```

Using JSP tags

There are five main tags:

1. Declaration tag
2. Expression tag
3. Directive Tag
4. Scriptlet tag
5. Action tag

Declaration tag (<%! %>)

This tag allows the developer to declare variables or methods.

Before the declaration you must have <%!
At the end of the declaration, the developer must have %>

Code placed in this tag must end in a semicolon (;).

Declarations do not generate output so are used with JSP expressions or scriptlets.

For Example,

```
<%!
        private int counter = 0 ;
        private String get Account ( int accountNo) ;
%>
```

Expression tag (<%= %>)

This tag allows the developer to embed any Java expression and is short for out.println().

A semicolon (;) does not appear at the end of the code inside the tag.

For example, to show the current date and time.

```
Date : <%= new java.util.Date() %>
```

Directive tag (<%@ directive ... %>)

A JSP directive gives special information about the page to the JSP Engine.

There are three main types of directives:

1) page – processing information for this page.
2) Include – files to be included.
3) Tag library – tag library to be used in this page.

Directives do not produce any visible output when the page is requested but change the way the JSP Engine processes the page.

For example, you can make session data unavailable to a page by setting a page directive (session) to false.

1. Page directive

This directive has 11 optional attributes that provide the JSP Engine with special processing information. The following table lists the 11 different attributes with a brief description:

language	Which language the file uses.	<%@ page language = "java" %>
extends	Superclass used by the JSP engine for the translated Servlet.	<%@ page extends = "com.taglib…" %>
import	Import all the classes in a java package into the current JSP page. This allows the JSP page to use other java classes. The following packages are implicitly imported. java.lang.* javax.servlet.* javax.servlet.jsp.* javax.servlet.http.*	<%@ page import = "java.util.*" %>
session	Does the page make use of sessions. By default all JSP pages have session data available. There are performance benefits to switching session to false.	Default is set to true.
buffer	Controls the use of buffered output for a JSP page. Default is 8kb	<%@ page buffer = "none" %>
autoFlush	Flush output buffer when full.	<%@ page autoFlush = "true" %>
isThreadSafe	Can the generated Servlet deal with multiple requests? If true a new thread is started so requests are handled simultaneously.	
info	Developer uses info attribute to add information/document for a page. Typically used to add author, version, copyright and date info.	<%@ page info = "visualbuilder.com test page, copyright 2001. " %>
errorPage	Different page to deal with	<%@ page errorPage =

	errors. Must be URL to error page.	"/error/error.jsp" %>
IsErrorPage	This flag is set to true to make a JSP page a special Error Page. This page has access to the implicit object exception (see later).	
contentType	Set the mime type and character set of the JSP.	

2. Include directive

Allows a JSP developer to include contents of a file inside another. Typically include files are used for navigation, tables, headers and footers that are common to multiple pages.

Two examples of using include files:

This includes the html from privacy.html found in the include directory into the current jsp page.

<%@ include file = "include/privacy.html %>

or to include a naviagation menu (jsp file) found in the current directory.

<%@ include file = "navigation.jsp %>

Include files are discussed in more detail in the later sections of this tutorial.

3. Tag Lib directive

A tag lib is a collection of custom tags that can be used by the page.

```
<%@ taglib uri = "tag library URI" prefix = "tag Prefix" %>
```

Custom tags were introduced in JSP 1.1 and allow JSP developers to hide complex server side code from web designers.

Scriptlet tag (<% ... %>)

Between <% and %> tags, any valid Java code is called a Scriptlet. This code can access any variable or bean declared.

For example, to print a variable.

```
<%

        String username = "PS-SOFTECH" ;
        out.println ( username ) ;

%>
```

Action tag

There are three main roles of action tags :

1) enable the use of server side Javabeans
2) transfer control between pages
3) browser independent support for applets.

Javabeans

A Javabean is a special type of class that has a number of methods. The JSP page can call these methods so can leave most of the code in these Javabeans. For example, if you wanted to make a feedback form that automatically sent out an email. By having a JSP page with a form, when the visitor presses the submit button this sends the details to a Javabean that sends out the email. This way there would be no code in the JSP page dealing with sending emails (JavaMail API) and your Javabean could be used in another page (promoting reuse).

To use a Javabean in a JSP page use the following syntax:

<jsp : usebean id = " …. " scope = "application" class = "com…" />

The following is a list of Javabean scopes:

page – valid until page completes.
request – bean instance lasts for the client request
session – bean lasts for the client session.
application – bean instance created and lasts until application ends.

Javabeans are discussed in detail later in this tutorial.

Dynamic JSP Include

You have seen how a file can be included into a JSP using an Include Directive:

```
<%@ include file = "include/privacy.html %>
```

This is useful for including common pages that are shared and is included at compile time.

To include a page at run time you should use dynamic JSP includes.

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

Creating your second JSP page

For the second example, we will make use of the different tags we have learnt. This example will declare two variables; one string used to stored the name of a website and an integer called counter that displays the number of times the page has been accessed. There is also a private method declared to increment the counter. The website name and counter value are displayed.

```
<HTML>
<HEAD>
<TITLE> JSP Example 2</TITLE>
</HEAD>
<BODY> JSP Example 2
<BR>
<%!
        String sitename ="PS-SOFTECH.COM";
        int counter = 0;

        private void increment Counter()
        {
                counter ++;
        }
%>

Website of the day is
<%= sitename %>
<BR>
page accessed
<%= counter %>
</BODY>
</HTML>
```

Implicit Objects

So far we know that the developer can create Javabeans and interact with Java objects. There are several objects that are automatically available in JSP called implicit objects.

The implicit objects are

Variable	Of type
request	javax.servlet.http.httpservletrequest
response	javax.servlet.http. httpservletresponse
out	javax.servlet.jsp.jspwriter
session	javax.servlet.http.httpsession
pagecontent	javax.servlet.jsp.pagecontext
application	javax.servlet.http.servletcontext
config	javax.servlet.http.servletconfig
page	java.lang.object
exception	java.lang.throwable

page object

Represents the JSP page and is used to call any methods defined by the servlet class.

config object

Stores the Servlet configuration data.

request object

Access to information associated with a request. This object is normally used in looking up parameter values and cookies.

```
<% String devStr = request.getParameter("dev"); %>
Development language = <%= devStr %>
```

This code snippet is storing the parameter "dev" in the string devStr. The result is displayed underneath.

The session object is covered in detail in the next section.

Session Tracking in JSP (Session Object)

Say for example, you would like to implement a shopping cart using JSP. There are several options you could consider:

- **Cookies** – a small text file stored on the client's machine. Cookies can be disabled in the browser settings so are not always available.
- **URL rewriting** – store session information in the URL. Works when cookies are not supported but can make bookmarking of web pages a problem because they have session specific information at the end of a URL.
- **Hidden form fields** – HTML hidden edit boxes such as <INPUT TYPE="HIDDEN" NAME="USERNAME" VALUE=" … " >. Every page has to be dynamically produced with the values in the hidden field.
- **Session objects** – JSP Implicit object.

A session object uses a key/value combination to store information.

To retrieve information from a session,

```
session.getValue("visitcounter")
```

The return type of the method getValue is Object, so you will need to typecast to get the required value. If there is not a session key with that name, a null is returned.

To set a session key with a value,

```
session.putValue("visitcounter", totalvisits)
```

The third JSP example in this tutorial demonstrates the use of the session object.

JSP Comments <%-- JSP comment --%>

JSP comments are similar to HTML comments <!-- HTML Comment --> except JSP comments are never sent to the user's browser. HTML comments are visible in the page source.

```
<html>
  <head>
    <title>
      HTML and JSP Comments
    </title>
  </head>
  <body>
    <h2>
      comments
    </h2>

    <!-- This HTML Comment -  visible in the page source -->

    <%-- This JSP comment - not visible in the page source --%>

  </body>
</html>
```

Creating your third JSP page

This third example counts how many times a particular user visits a page. It uses the Session object that was presented in the Implicit object section of this tutorial.

The following are the main steps involved:

1. get the value of the session variable - visitcounter
2. if the session variable (visitcounter) is null
set the session variable to 0 and welcome the visitor.
3. if the session variable is not null (after step 2),
increment the session varaible and display the number of
visits.

```
<!-- session.jsp
    checks to see if you have visited a page and keeps a counter.
    visualbuilder.com
-->
<html>
<head>
</head>

<body>

<%
 // get the value of the session variable - visitcounter
 Integer totalvisits = (Integer)session.getValue("visitcounter");

 // if the session variable (visitcounter) is null
 if (totalvisits == null)
 {
    // set session variable to 0
    totalvisits = new Integer(0);
    session.putValue("visitcounter", totalvisits);

    // print a message to out visitor
    out.println("Welcome, visitor");
 }
 else
 {
    // if you have visited the page before then add 1 to the visitcounter
    totalvisits = new Integer(totalvisits.intValue() + 1);
    session.putValue("visitcounter", totalvisits);
    out.println("You have visited this page " + totalvisits + " time(s)!");
 }
%>

</body>
</html>
```

Error pages

Eventually there will come a time when sometime unexpected happens. In Java terms this is when an exception gets thrown. JSP can handle these situations so when an exception is thrown, a default error page is sent to the browser.

So what makes an error page different from other JSP pages?

Well one of the first lines in an error page must be the page directive isErrorPage=”true”.

Inside your default error page (errorPage.jsp), above the <HTML> tag type:

```
<%@ page isErrorPage="true" import=”java.util.*” %>
<HTML>
<BODY>
Error Occurred
<%= exception.toString() %>
</BODY>
</HTML>
```

Our error page also uses the exception object and the toString() method to display a brief description of the error.

To use a specific error page in your JSP pages, again above the <HTML> tag type:

```
<%@ page errorPage="errorPage.jsp" %>
<HTML>
…
```

This code will go to errorPage.jsp if an error occurs. Even after an error, the HTTP session remains available.

You should now understand how to create an error page for your JSP applications.

Using JavaBeans with JSP

We have already mentioned JavaBeans in the Action Tag section. This section will provide a detailed look into how to use JavaBeans with JSP.

What is a JavaBean?

A JavaBean is a Java class with a few constraints:

- Must have a no argument constructor.
- Must follow a naming convention for get/set methods.
- Must implement the Serializable interface (not required for JSP)

Let's create a simple JavaBean that stores Employee data.

```
package com.visualbuilder.beans;

public class Employee
{
   private String _name;

   public String getName()
   {
     return _name;
   }

   public void setName(String name)
   {
     _name = name;
   }
}
```

Next we will access the properties of the JavaBean from a JSP.

```
<html>
  <head>
    <title>
      Using JavaBeans from JSP
    </title>
  </head>
  <body>

  <jsp:useBean id=”staff” class=” com.visualbuilder.beans.Employee ” />

  <jsp: setProperty name=”staff” property=”name” value=”James Brown” />

  Welcome to the company, <jsp:getProperty name=”staff” property=”name” />

  <%= staff.getName() %>, please visit VisualBuilder.com to get more out of life!

  </body>
</html>
```


The benefit of using JavaBeans is that you can easily reuse the code in other applications. It also minimises the amount of code in the JSP, allowing designers to use their favorite web design editor without destroying the Java code. This follows a component centric approach to developing applications.

Naming convention

A common convention is that property names are mixed-case, beginning with a lowercase letter and uppercasing the first letter of each word in the property name.

getName()
setName(String name)

The data from an HTML form can be passed straight into a JavaBean. In order to do this, use <jsp:setProperty> to define properties in the JavaBean with names that match the names of the HTML form elements. You would also define corresponding set methods for each property.

In our example, if the form element were called name, you would define a property called user and methods getName and setName in the JavaBean. This allows you to get data from the form into the JavaBean.

You should now understand how to use JavaBeans with JSP.