



# JavaServer Pages

Juan Cruz  
Kevin Hessels  
Ian Moon

## Table of Contents

1. Introduction
  - o What is JSP?
  - o Alternative Solutions
  - o Why Use JSP?
2. JSP Process
  - o Request
  - o Compilation
  - o Example
3. Object Instantiation and Scope
  - o Scope
  - o Synchronization
  - o Session Objects
4. JSP Syntax
  - o Template Text
  - o Scripting Elements
  - o Directives
  - o Actions
5. End Notes
6. References

## 1. Introduction

The last decade has seen an explosion in the growth of the Internet as a medium for information exchange and, more recently, as a place for commerce. Up until recently, content provided on the web was presented primarily in the form of static pages. These static web pages, while effective for presenting some types of information, lack the functionality that is required to create a truly interactive web experience. With the increasing demand for dynamic content on the web, from applications of on-line store catalogues to portal sites customized to the user's preferences, there came a need to be able change the content of these pages dynamically. Although a number of solutions have been provided, this paper highlights the technology introduced by Sun Microsystems, know as JavaServer Pages (JSP). In the following, we will highlight the key advantages that JSP provides as well as a number of concepts behind the technology. Finally, a number of examples will illustrate how JSP can be used to easily create effective web content.

## What is JSP?

Much like its primary competitors, ASP and PHP, JSP is a server-side scripting language that is used to create dynamic and interactive web content. JSP is based upon the Java programming language, and thus, at JSP's foundation are the numerous, pre-existing classes that comprise the Java platform. These classes, when combined with static HTML using JSP's "XML-like tags and scriptlets" [1] allow for the rapid development of dynamic web pages. JSP is similar, in many respects, to other technologies, such as Microsoft's ASP or the Open Source PHP, in which executable content is embedded within the HTML page and executed at the time the page is served to the client. Because the Java language is able to run on virtually any platform, JSP can also be deployed on any platform. The only requirement is a Java interpreter and the Apache/Tomcat web server. While a number of web server applications exist that support JSP, the combination of Apache and Tomcat is by far one of the most popular. Because most developers are familiar with both HTML and the Java programming language, the use of JSP is a natural solution for delivering interactive web-based applications.

## Alternative Solutions

JSP is one of a number of products/technologies that enable the delivery of dynamic web content. JSP, however, provides a number of advantages over these other technologies. One of the main competitors is Microsoft's ASP technology which provides similar functionality based on the VBScript language. The primary drawback to this solution is that ASP does not provide for vendor independent deployment. Thus, in order to use ASP, you must use Microsoft's IIS web server on the Windows platform. This solution may not be feasible for some organizations due to the high cost of product licensing and incompatibility with pre-existing technology infrastructure.

Another solution in this domain is the JavaScript language, used primarily on the client side, to offer dynamic web pages. There are a number of drawbacks to using JavaScript to create dynamic content. Primarily, because JavaScript is run on the client-side, it does not provide any means in which the server's resources can be accessed [2]. Thus, providing content from a database on the the server, for example, is impossible. This poses a severe limitation of the number of applications in which JavaScript can be used, particularly in the area of on-line commerce. Another disadvantage to using JavaScript for creating dynamic web content is the poor adherence by a number of browsers to the Document Object Model standards. While there has been a movement towards compliance by the major browser vendors, it is still required to write code that supports the myriad of legacy applications. This problem greatly increases the cost and complexity in developing web applications based on this technology.

JSP is very closely related to the Java Servlet technology, also developed by Sun Microsystems, and thus, provides functionality which is very similar to the Servlet and vice versa [2]. The primary difference between these technologies, however, is their intended application. Because the Java Servlet is written and compiled as a normal class, they are intended to deliver content that is based primarily on content generated by the server itself. JSP, on the other hand, is written as HTML with embedded Java, intended for scripting small portions of the page to generate an interactive site. While either technology can be used to create similar effects, it is up to the developer to evaluate which one is most suitable for the requirements at hand.

## Why Use JSP?

The key advantage to using JSP over the alternatives is the fact that JSP is completely platform-independent, enabling them to run on any platform supported by the Apache/Tomcat web server. This platform independence is due to the fact the JSP is based on Sun's Java technology which is available for virtually any operating system. According to Sun, this has the advantage of allowing the content provider to "extend the capabilities of a web server with minimal overhead, maintenance, and support." [1] Although JSP is closely related to Sun's Java Servlet technology, JSP provides additional benefits. The primary advantage that JSP provides over Servlets is the fact that JSP code can be directly embedded within static HTML pages to provide dynamic content from the server [2]. This embedding is provided by an XML markup to define the Java code, a standard which should be familiar to all web developers. Another advantage of JSP is the ability of JSP to integrate with pre-existing applications. Because JSP emphasizes the separation of content and functionality, pre-existing code can be integrated through the use of a JavaBean interface to extend its functionality.

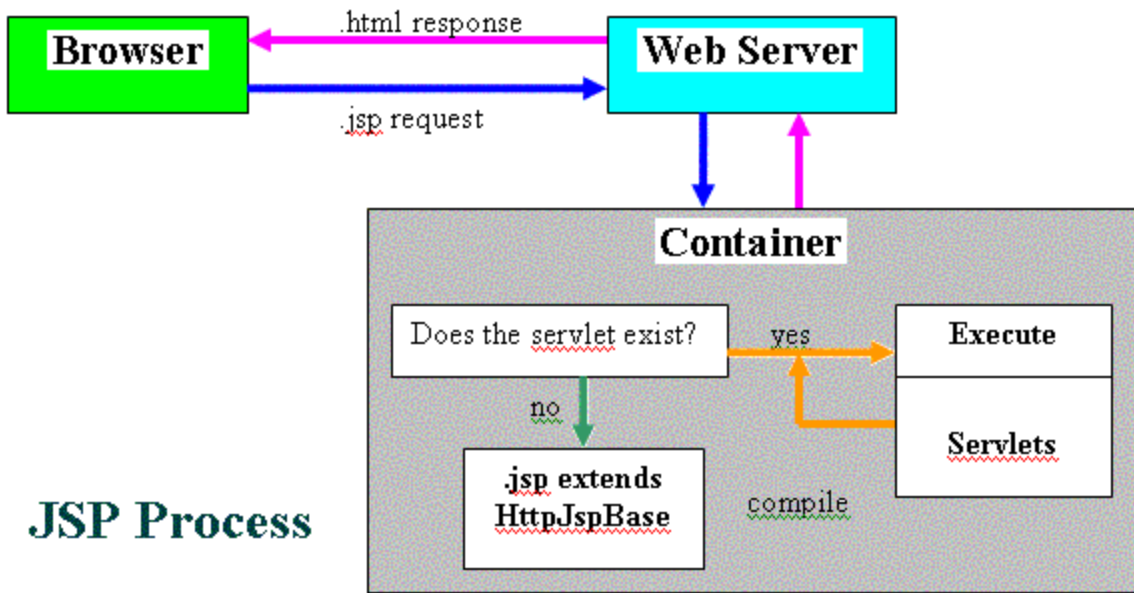
## 2. JSP Process

### Request

There are three fundamental components to the JSP process, the web browser, web server, and a container. A web server receives an HTTP request from a browser, and responds by sending an HTTP response and document to the browser to be displayed. When a web server is sent a request for a .jsp file it passes the request to the container. "Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process." 1 Tomcat is one example of a container but there are many others. The container handles the request and passes HTML back to the web server that sends the response to the browser to be displayed.

### Compilation

JSP is an extension of another Sun technology Java Servlets. The actions of the container demonstrate this relationship. Once a JSP request has been forwarded to the container from the web server, "the container looks for a servlet class with the requested file name in the appropriate package." 2 If the servlet is found it is run. A Java Servlet like a CGI writes out an HTML document that is passed back to the web server. "If the container doesn't find the servlet class, it looks for the same file name, but with a .jsp extension." 3 If it finds a match, the JSP is converted to a class that extends `HttpJspBase`, which in turn implements the `Servlet` interface. This is the core of the relationship between Java Servlets and JavaServer Pages. The new class is compiled and run as a servlet, creating a document for the web server.



### JSP Process

"Note that the JSP 1.1 specification also allows for JSP pages to be pre compiled into class files. Pre- compilation may be especially useful in removing the start-up lag that occurs when a JSP page delivered in source form receives the first request from a client." 4

HttpJspBase consists of mainly three methods, `jspInit()`, `jspDestroy()`, and `_jspService()`. "`_jspService()` cannot be overridden, the developer can describe initialization and destroy events by providing implementations for the `jspInit()` and `jspDestroy()` methods within their JSP pages." 5 Once the class file is compiled and loaded within the servlet container, "the `_jspService()` method is responsible for replying to a client's request. By default, the `_jspService()` method is dispatched on a separate thread by the servlet container in processing concurrent client requests, as shown below:" 6

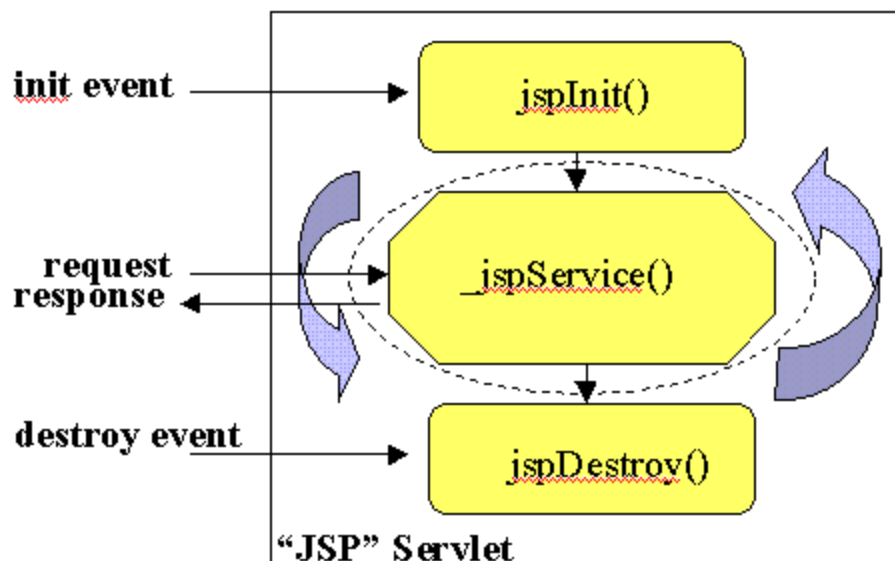


Figure 2 from "JavaServer Pages Fundamentals" [5]

## Example

```

<%@ page import="java.text.*, java.util.*" %>
<html>
<body>
<%
Date d = new Date();
String today = DateFormat.getDateInstance().format(d);
%>
Today is:
<em> <%=today%> </em>
</body>
</html>

```

**Page Compilation**



Figure 1 from "JavaServer Pages Fundamentals" [5]

---

```
package jsp;
```

```

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.util.Vector;
import org.apache.jasper.runtime.*;
import java.beans.*;
import org.apache.jasper.JasperException;
import java.text.*;
import java.util.*;

```

```

public class _0005cjsp_0005cjsptest_0002ejspjsptest_jsp_0 extends HttpJspBase {
    static { }
    public _0005cjsp_0005cjsptest_0002ejspjsptest_jsp_0() { }
    private static boolean _jspx_initd = false;
    public final void _jspx_init() throws JasperException { }
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;

```

```

Object page = this;
String _value = null;
try {
    if (_jspx_inited == false) {
        _jspx_init();
        _jspx_inited = true;
    }
    _jspxFactory = JspFactory.getDefaultFactory();
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response, "", true, 8192,
    true);
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    // begin
    out.write("\r\n<html>\r\n<body>\r\n");
    // end
    // begin [file="E:\jsp\jsptest.jsp";from=(3,2);to=(5,0)]
    Date d = new Date();
    String today = DateFormat.getDateInstance().format(d);
    // end
    // begin
    out.write("\r\nToday is: \r\n<em> ");
    // end
    // begin [file="E:\jsp\jsptest.jsp";from=(7,8);to=(7,13)]
    out.print(today);
    // end
    // begin
    out.write(" </em>\r\n</body>\r\n</html>\r\n");
    // end
} catch (Exception ex) {
    if (out.getBufferSize() != 0)
        out.clear();
    pageContext.handlePageException(ex);
} finally {
    out.flush();
    _jspxFactory.releasePageContext(pageContext);
}
}
}

```

Code Listing 1 from "JavaServer Pages Fundamentals" [5]

---

## 3. Object Instantiation and Scope

### Scope

Member objects of the JSP class that is created can be created implicitly using JSP directives, explicitly through actions, and directly in a scriptlet. The syntax will be discussed later in this document. The instantiated objects can be associated with a scope attribute defining where there is a reference to the object and when that reference is removed. The following diagram indicates the various scopes that can be associated with a newly created object:<sup>7</sup>

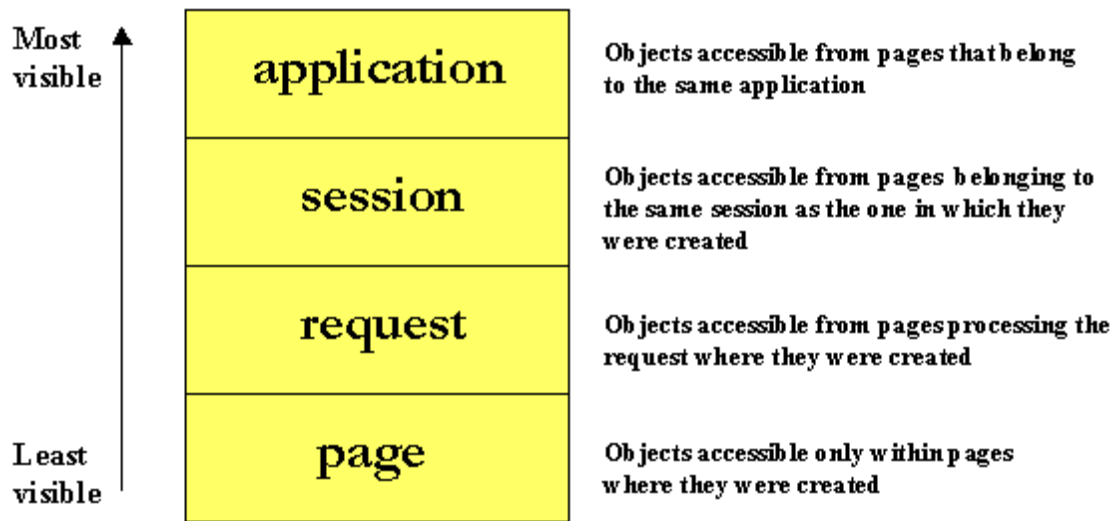


Figure 5 from "JavaServer Pages Fundamentals" [5]

## Synchronization

JSP variables can be created in a `jspPage` by declaring them in the page using the declaration syntax tags, `<%! %>`. These variables, when converted from JSP to servlets, become member variables of the servlet class. Objects created through other means are members of the `_jspService()` method. This is an important differentiation when it comes to synchronization. By default, each servlet is instantiated only once within the container, and rely on multi-threading of `_jspService` to handle multiple requests. Therefore, the member variables of the servlet are shared to multiple pages, causing concurrence problems.<sup>8</sup>

"There are a couple of different ways to ensure that the service methods are thread-safe. The easy approach is to include the JSP page directive:

```
<%@ page isThreadSafe="false" %>
```

This causes the JSP page implementation class to implement the `SingleThreadModel` interface, resulting in the synchronization of the service method, and having multiple instances of the servlet to be loaded in memory. The concurrent client requests are then distributed evenly amongst these instances for processing in a round-robin fashion.<sup>9</sup> This of course can lead to very poor response times while under high request volume. A better approach is to explicitly synchronize access to shared objects (like those instances with application scope, for example) within the JSP page, using scriptlets:<sup>10</sup>

```
<%
    synchronized (application) {
        SharedObject foo = (SharedObject)
```

```

        application.getAttribute("sharedObject");
        foo.update(someValue);
        application.setAttribute("sharedObject",foo);
    }
%>

```

## Session Objects

JSP's participate in sessions with the browser client. A cookie is used to identify the session of the browser. The HttpSession object is an implicit object that does not have to be declared by the developer. However, other objects or cookies can be stored and retrieved from this scope.

```

<%-- Create a foo in the object scope --%>
<%
    Foo foo = new Foo();
    session.putvalue("foo",foo);
%>

<%-- Retrieve a foo in the object scope --%>
<%
    Foo myFoo = (Foo) session.getvalue("foo");
%>

<%-- JSP opt out of a session --%>
<%@ page session="false" %>

```

"There is no limit on the number of objects you can store into the session. However, placing large objects into the session may degrade performance, as they take up valuable heap space. By default, most servers set the lifetime of a session object to 30 minutes, although you can easily reset it on a per session basis by invoking `setMaxInvalidationInterval(int secs)` on the session object. The JSP engine holds a live reference to objects placed into the session as long as the session is valid. If the session is invalidated or encounters a session timeout, then the objects within are flagged for garbage collection."<sup>11</sup>

## 4. JSP Syntax

JSP is composed of HTML or XML code embedded with constructing code written in Java. The HTML code contained in a JSP is called *fixed-template data* or *fixed-template text*. There are three key components to JSPs: *scripting elements*, *directives*, and *actions*. [2]. Scripting Elements enable programmers to insert Java code that interacts with other JSP components, directives control the overall structure of the servlet, and actions specify existing components that can be used with the JSP.

## Template Text

The servlet generated from the JSP code passes the HTML syntax directly to the client with no modifications. This one follows the regular HTML syntax rules. HTML comments of the form `<!-- HTML comment -->` are passed to the client, yet JSP comments of the form `<%-- JSP comment --%>` are not passed.

## Scripting Elements

Scripting elements allows you to insert java code into the generated servlet, and there are three forms:

### Expressions

Expressions evaluate a java statement and converts the result value into a string. The result is inserted directly into the output. The expression is evaluated at request time. Expressions have the following form:

```
<%= Java Expression %>
```

For example, the following expression generates the current date and time:

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

### Scriptlets

Scriptlets are blocks of code delimited by `<%` and `%>` , and they permit the execution of more complex tasks than a simple expressions. Scriptlets can perform tasks like executing loops, conditional statements, updating a database or setting up response headers (i.e.. html or plain text). The scriptlet code is inserted into the `_jspService` method.

For example, the following page `BGColor.jsp` [2] reads the background color of a page from a form field "bgColor" and outputs the background color selected by the user; otherwise, it displays a plain white background as default.

```
<html>
<head>
  <title> Color Testing </title>
</head>

<%
  String bgColor = request.getParameter("bgColor");
  boolean hasExplicitColor;
  if (bgColor != null) {
```

```

        hasExplicitColor = true;
    } else {
        hasExplicitColor = false;
        bgColor = "white";
    }
%>
<body bgcolor="<%= bgColor %>">
<h2 align="center">Color Testing</h2>

<%
    if (hasExplicitColor) {
        out.println("User background color= " + bgColor);
    } else {
        out.println("Using default backgroundcolor of WHITE.");
    }
%>
</body>
</html>

```

Scriptlets are not required to contain complete java statements, they can include static html sections. The following example [2] contains mixed template text and scriptlets:

```

<% if ( Math.random( ) < 0.5 ) { %>
    Have a <b> nice </b> day!
<% } else { %>
    Have a <b> lousy </b> day!
<% } %>

```

**Example Listing 10.2, "Core Servlets and JavaServer Pages"**

As you can see, the example mixes scriptlet code and html output code, this is due to the fact that the code will be converted to servlet output in the end. The same condition also applies to while or for loops.

## Declarations

Declarations allow the programmers to define variables and methods, and they are delimited by `<%!` and `%>`. Declarations get inserted into the main body of the servlet class (outside of the `_jspService` method). The variables become instance variables of the servlet.

Note, that in multiple client requests to the same servlet, a single servlet receives the calls of multiple threads, thus instance variables are shared by multiple requests [2]. The following example displays the number of times that the same page has been accessed by different users, for this, an instance variable `accessCount` keeps track of the number of requests to the servlet:

```
<body>
  <h1> JSP Declarations </h1>
  <%! private int accessCount = 0; %>
  <h2> Access to page since servlet reboot:
  <%= ++accessCount %></h2>
</body>
```

## Predefined Variables

There are eight predefined variables that provide JSP pages with servlet capabilities and they are called *implicit objects*. These objects are accessible through expressions and scriptlets. Yet, they are not accessible through declarations since declarations are defined outside of the `_jspService` method. There are four scopes in which implicit objects are defined, these scopes are: request, page, application and session [3].

Objects with *page scope* exist only in the page that defines them, and they are:

- *response*: this variable sets the response header to the client, for example it can be html or plain text.
- *out*: variable is used to send output to the client and stores the output in a buffer before it is sent.
- *config*: represents the JSP configuration options.
- *pageContext*: provides access to page attributes and provides a place to store shared data.
- *page*: represents the *this* reference for the current JSP instance.

Objects with *request scope*:

- *request*: it gives you access to the request parameters, the request type (GET or POST), and incoming; http headers (i.e. cookies).

Objects with *session scope* exist for the entire client's session:

- *session*: represents the client session information, and it is available only in pages that participate in a session.

Objects with *application scope* can be manipulated by any servlet or JSP container:

- *application*: the data stored in these variables is shared by all servlets in the servlet engine.

## Directives

Directives are messages that enable the programs to set the overall structure of the resulting servlet. They are delimited by the tags `<%@` and `%>`. There are three types of directives: page, include and tag libraries (taglib).

### Page Directive

A page directive specifies global settings for the JSP in the JSP container. The page directive defines the following attributes:

import, contentType, isThreadSafe, session, buffer, autoflush, extends, info, errorPage, isErrorPage and language [2].

Examples of page directives follow:

- import: specifies the library packages that will be imported to the servlet generated from the JSP. The following directive gives access to the classes in the java.util package:

```
<%@ page import ="java.util.*" %>
```

- contentType: sets the Content-Type response header indicating the MIME type of the document. For example, to set the content type to plain text:

```
<%@ page contentType="text/plain" %>
```

Note, that the content type could have been set with an implicit object from a scriptlet:

```
<% response.setContentType("text/plain"); %>
```

## Include Directive

The include directive allows the insertion of external files in to the JSP page at translation time (this is the first time that the document is accessed). These files can be html code or extra jsp functions [2]. The include directive has the following form:

```
<%@ include file="Relative URL" %>
```

## Tag Library Directives

Tag Libraries allow the access of custom tags that provide complex functionality to modify JSP or text context. They are useful for Web page designers who don't know much about java programming[3]. Example:

```
<%@ taglib uri = "advjhttp1-taglib.tld" prefix="advjhttp1" %>
```

The tag specifies the *uri* of the tag library descriptor file and the *prefix* for each tag "advjhttp". The tags can be inserted into the jsp as follows:

```
<advjhttp1:welcome> </advjhttp1:welcome>
```

## Actions

Actions are predefined tasks that are processed by the JSP container at request time. Actions are delimited by `<jsp: action>` and `</jsp:action>`, where action is the predefined task to execute [3].

Standard actions are:

**<jsp: include>** Used to insert other jsp or html pages into the current JSP page. The inserted page is processed at request time (that is when the user access the page).

Example:

```
<jsp: include page="banner.html" flush = "true" />
```

Page indicates the html page to include and flush indicates that the buffer must be flushed.

**<jsp: forward>** Used to forward the request process to other JSP page or servlet. The action terminates the current JSP execution and starts the new one.

**<jsp: plugin>** Used to add a plug-in component to a page as a browser specific object. For a Java-Applet, it allows the downloading and installation of the java plug-in.

Java Bean Manipulation actions:

**<jsp: useBean>** Declares a Java Bean instance for use in the JSP page. It assigns the bean an id to access the bean through scripting elements and it specifies the scope of the bean (ie. page or session). Example:

```
<jsp:useBean id="courseBean" class="coursepack.CourseListBean" />
```

Where the id courseBean is the object name to access the JavaBean, and class points to the package "CourseListBean.class".

**<jsp: getProperty>** Gets a property in the specified JavaBean instance. Example:

```
<jsp:getProperty name="courseBean" property="courseColor" />
```

Other way to get bean properties is using an expression as follows:

```
<%= courseBean.getCourseColor(courseNumber) %>
```

**<jsp: setProperty>** Sets a property in the specified JavaBean instance. Example:

```
<jsp:setProperty name="courseBean" property="courseColor" value="blue" />
```

The above expression is the same as:

```
<% courseBean.setCourseColor("blue")%> ~
```

## 5. End Notes

1. <http://jakarta.apache.org/tomcat/index.html>
2. [http://www.jspinsider.com/content/rcarnes/jspb\\_1.view](http://www.jspinsider.com/content/rcarnes/jspb_1.view)
3. <http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html>
4. Ibid.
5. Ibid.
6. Ibid.
7. Ibid.
8. <http://pdf.coreservlets.com/chapter10.pdf>
9. <http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html>
10. Ibid.
11. Ibid.

## 6. References:

[1] JavaServer Pages: Dynamically Generated Web Content

Sun Microsystems, 14 March 2002, <<http://java.sun.com/products/jsp/>>

[2] Core Servlets and JavaServer Pages, Hall, Marty

Sun Microsystems Press/Prentice Hall, 2000

<[pdf.coreservlets.com](http://pdf.coreservlets.com)>

[3] Internet & World Wide Web: How to Program, 2nd Edition,

Deitel, H. M., Deitel P. J., and Neito T. R., Prentice Hall, Upper Saddle River, N. J., 2002

[4] Bodoff, Stephanie, Java Servlet Technology

<[http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/Servlets.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html)> (13 Mar. 2002)

[5] jGuru.com (1999), JavaServer Pages Fundamentals Short Course

<<http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html>> (24 Mar. 2002)

[6] Saegesser, Marc A. (2002), The Jakarta Site - Jakarta Tomcat

<<http://jakarta.apache.org/tomcat/index.html>> (14 Mar. 2002)

[7] Carnes, Ray (2000), JSP: THE SHORT COURSE - Lesson 1

<[http://www.jspinsider.com/content/rcarnes/jspb\\_1.view](http://www.jspinsider.com/content/rcarnes/jspb_1.view)> (24 Mar. 2002)