JavaServer PagesTM (JSPTM) v1.2 Syntax Reference

<jsp:root></jsp:root>	Defines standard elements and namespace attributes of tag libraries.		
Hidden Comment	Documents the JSP page but is not inserted into the response.		
<u>Declaration</u>	Declares a variable or method valid in the scripting language used in the page.		
Expression	Contains an expression valid in the scripting language used in the page.		
<u>Scriptlet</u>	Contains a code fragment valid in the scripting language used in the page.		
<jsp:text></jsp:text>	Encloses template data.		
Include Directive	Includes a resource of text or code when the JSP page is translated.		
Page Directive	Defines attributes that apply to an entire JSP page.		
<u>Taglib Directive</u>	Defines a tag library and prefix for the custom tags used in the JSP page.		
<jsp:forward></jsp:forward>	Forwards a request to an HTML file, JSP page, or servlet.		
<jsp:getproperty></jsp:getproperty>	Inserts the value of a bean property into the response.		
<jsp:include></jsp:include>	Includes a static resource or the result from another web component		
<jsp:plugin></jsp:plugin>	Causes the execution of an applet or bean. The applet or bean executes in the specified plugin. If the plugin is not available, the client displays a dialog to initiate the download of the plugin software.		
<jsp:setproperty></jsp:setproperty>	Sets a bean property value or values.		
<jsp:usebean></jsp:usebean>	Instantiates or references a bean with a specific name and scope.		

Preface

All tags are case sensitive. A pair of single quotes is equivalent to a pair of double quotes. Spaces are not allowed between an equals sign and an attribute value.

The elements in a JSP page can be expressed in JSP syntax or XML syntax. The following rules apply:

- JSP and XML syntax cannot be mixed within a page.
- A page in one syntax can include or forward to a page in the other syntax.
- Some elements have attributes whose value can be computed at request time. In JSP syntax, the format of a value is the same as a JSP expression: <%= expression %>. In XML syntax, the format of the value is %= expression %.

Quoting Conventions

The following outlines quoting conventions for JSP pages expressed in JSP syntax.

Scripting Elements

■ %> by %\>

Template Text

■ <% by <\%

Attributes

- 'as \'. This is required within a single quote-delimited attribute value.
- "as \". This is required within a double quote-delimited attribute value.
- \ as \\
- %> as %\>
- <% as <\%
- ' and " can be used to indicate single and double quotes.

Typographic Conventions

code = fixed	bold = default	italics = user-defined	= or
[] = optional	{ } = required choice	= list of items	+ = can repeat

<jsp:root>

Defines standard JSP elements and namespace attributes of tag libraries.

JSP Syntax

None. However, see Taglib Directive.

XML Syntax

```
<jsp:root
  xmlns:jsp="http://java.sun.com/JSP/Page"
  [ xmlns:taglibPrefix="URI" ]+ ...
  version="1.2">
       JSP Page
</jsp:root>
```

Example

Description

A JSP page in XML syntax has jsp:root as its root element. Tag libraries used within the JSP page are represented in the root element through xmlns attributes. The xmlns:jsp and version attributes are mandatory.

Attributes

xmlns:jsp="http://java.sun.com/JSP/Page"

Enables the use of the standard elements defined in the JSP specification.

■ version="1.2"

Specifies the version of the JSP specification the page is using.

■ xmlns:taglibPrefix="URI"

Specifies a tag library prefix and URI.

taglibPrefix precedes the custom tag name, for example, public in <public:loop>. Empty prefixes are illegal. If you are developing or using custom tags, you cannot use the tag prefixes jsp, jspx, java, javax, servlet, sun, and sunw, as they are reserved by Sun Microsystems.

URI uniquely locates the TLD that describes the set of custom tags associated with the named prefix. URI may be of one of two forms:

- A URI:
 - ■A Uniform Resource Locator (URL), as defined in RFC 2396, available at http://www.rfc-editor.org/rfc/rfc2396.txt
 - ■A Uniform Resource Name (URN), as defined in RFC 2396
- urn:jsptld:path.

If the <code>URI</code> is a URI, then the TLD is located by consulting the mapping indicated in <code>web.xml</code> extended using the implicit maps in the packaged tag libraries. If <code>URI</code> is of the form <code>urn:jsptld:path</code>, <code>path</code> is interpreted relative to the root of the web application and should resolve to a TLD file directly, or to a JAR file that has a TLD file at location META-INF/taglib.tld.

See Also

■ Taglib Directive

Hidden Comment

Documents the JSP page but is not inserted into the response.

JSP Syntax

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

XML Syntax

None.

Example

```
<%@ page language="java" %>
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be inclueded in the response --%>
</body>
</html>
```

Description

A hidden comment marks text or lines that the JSP container should ignore. A hidden comment is useful when you want to comment out part of your JSP page. The JSP container does not process anything within the <\-- and --*> characters. A hidden comment is not inserted into the response.

Declaration

Declares a variable or method valid in the scripting language used in the JSP page.

JSP Syntax

```
<%! declaration; [ declaration; ]+ ... %>
```

XML Syntax

```
<jsp:declaration>
  declaration; [ declaration; ]+ ...
</jsp:declaration</pre>
```

Examples

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

Description

A declaration declares one or more variables or methods that you can use in JavaTM code later in the JSP page. You must declare the variable or method *before* you use it in the JSP page.

You can declare any number of variables or methods within one declaration element, as long as you end each declaration with a semicolon. The declaration must be valid in the Java programming language.

When you write a declaration in a JSP page, remember these rules:

- You must end the declaration with a semicolon (the same rule as for a <u>Scriptlet</u>, but the opposite of an <u>Expression</u>).
- You can already use variables or methods that are declared in packages imported by the page directive, without declaring them in a declaration element.

A declaration has translation unit scope, so it is valid in the JSP page and any of its static include files. A static include file becomes part of the source of the JSP page and is any file included with an include directive or a static resource included with a <jsp:include> element. The scope of a declaration does not include dynamic resources included with <jsp:include>.

- <u>Scriptlet</u>
- <u>Expression</u>

Expression

Contains an expression valid in the scripting language used in the JSP page.

JSP Syntax

<%= expression %>

XML Syntax

```
<jsp:expression>
  expression
</jsp:expression>
```

Examples

```
The map file has <font color="blue"><%= map.size() %></font> entries.

Good guess, but nope. Try
<b><jsp:expression>numquess.getHint()</jsp:expression></b>.
```

Description

An expression element contains a scripting language expression that is evaluated, converted to a String, and inserted into the response where the expression appears in the JSP page. Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP page.

The expression element can contain any expression that is a valid page scripting language. When the Java programming language is the scripting language you do not use a semicolon to end the expression. However, the same expression within a scriptlet requires the semicolon; see <u>Scriptlet</u>).

You can sometimes use expressions as attribute values in JSP elements (see the $\underline{\textit{JavaServer Pages}^{TM} \textit{Syntax Card}}$). An expression can be complex and composed of more than one part or expression. The parts of an expression are evaluated in left-to-right order.

See Also

■ Declaration

■ <u>Scriptlet</u>

Scriptlet

Contains a code fragment valid in the page scripting language.

JSP Syntax

```
<% code fragment %>
```

XML Syntax

```
<jsp:scriptlet>
  code fragment
</jsp:scriptlet>
```

Examples

Description

A scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Within a scriptlet, you can do any of the following:

- Declare variables or methods to use later in the JSP page (see also <u>Declaration</u>).
- \blacksquare Write expressions valid in the page scripting language (see also $\underline{\text{Expression}}).$
- Use any of the implicit objects or any object declared with a <jsp:useBean> element.
- $\,\blacksquare\,$ Write any other statement valid in the scripting language used in the JSP page.

Any text, HTML tags, or JSP elements you write must be *outside* the scriptlet.

Scriptlets are executed at request time, when the JSP container processes the request. If the scriptlet produces output, the output is stored in the out-object.

- <u>Declaration</u>
- **■** Expression

<jsp:text>

Encloses template data.

JSP Syntax

None.

XML Syntax

```
<jsp:text>
   template data
</jsp:text>
```

Example

```
<hello><jsp:scriptlet>i=3;</jsp:scriptlet>
<hi><ipsp:text> hi you all
</jsp:text><jsp:expression>i</jsp:expression>
</hi>
</hello>
```

The output is:

```
<hello> <hi> hi you all </hi></hello>
```

Description

A jsp:text element is used to enclose template data in the XML representation. A jsp:text element has no attributes and can appear anywhere that template data can. The interpretation of a jsp:text element is to pass its content through to the current value of out.

XML syntax allows an XML element that does not represent a standard or custom action to appear anywhere a <code>jsp:text</code> can appear. Such an element is passed to the current out.

Include Directive

Includes a static file in a JSP page, parsing the file's JSP elements.

JSP Syntax

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

XML Syntax

```
<jsp:directive.include file="relativeURL" />
```

Examples

```
include.jsp:
<html>
<head><title>An Include Test</title></head>
<body bgcolor="white">
<font color="blue">
The current date and time are
<%@ include file="date.jsp" %>
</font>
</body>
</html>
date.jsp:
<%@ page import="java.util.*" %>
<%= (new java.util.Date() ).toLocaleString() %>

Displays in the page:
```

Description

The current date and time are

Aug 30, 1999 2:38:40

An include directive inserts a file of text or code in a JSP page at translation time, when the JSP page is compiled. When you use the include directive, the include process is *static*. A static include means that the text of the included file is added to the JSP page. The included file can be a JSP page, HTML file, XML document, or text file. If the included file is a JSP page, its JSP elements are translated and included (along with any other text) in the JSP page. Once the included file is translated and included, the translation process resumes with the next line of the including JSP page.

The included file can be an HTML file, a JSP page, a text file, XML document, or a code file written in the Java programming language. Be careful that the included file does not contain <html>, </html>, <body>, or </body> tags. Because the entire content of the included file is added to the including JSP page, these tags would conflict with the same tags in the including JSP page, causing an error.

Some of the behaviors of the include directive depend on the particular JSP container you are using, for example:

- The included file might be open and available to all requests, or it might have security restrictions.
- The JSP page might be recompiled if the included file changes.

Attributes

■ file="relativeURL"

The pathname to the included file, which is always a relative URL. A relative URL is just the path segment of an URL, without a protocol, port, or domain name, like this:

```
"error.jsp"
"/templates/onlinestore.html"
"/beans/calendar.jsp"
```

If the relative URL starts with /, the path is relative to the JSP application's context, which is a <code>javax.servlet.ServletContext</code> object that is in turn stored in the <code>application</code> object. If the relative URL starts with a directory or file name, the path is relative to the JSP page.

Tip

If you are including a text file and do not want the text to be displayed in the JSP page, place the text in a comment element.

- <jsp:include>
- <jsp:forward>

Page Directive

Defines attributes that apply to an entire JSP page.

JSP Syntax

XML Syntax

```
<jsp:directive.page pageDirectiveAttrList />
```

where pageDirectiveAttrList is the same as the list in the JSP syntax.

Examples

```
<%@ page import="java.util.*, java.lang.*" %>
<%@ page buffer="5kb" autoFlush="false" %>
<jsp:directive.page errorPage="error.jsp" />
```

Description

The page directive applies to an entire JSP page and any of its *static include files*, which together are called a *translation unit*. A static include file is a file whose content becomes part of the calling JSP page. The page directive does not apply to any dynamic resources; see <i pre>specific page include

You can use the page directive more than once in a translation unit, but you can only use each attribute, except import, once. Because the import attribute is similar to the import statement in the Java programming language, you can use a page directive with import more than once in a JSP page or translation unit.

No matter where you position the page directive in a JSP page or included files, it applies to the entire translation unit. However, it is often good programming style to place it at the top of the JSP page.

Attributes

■ language="java"

The scripting language used in scriptlets, declarations, and expressions in the JSP page and any included files. In v1.2, the only allowed value is java.

■ extends="package.class"

The fully qualified name of the superclass of the Java class this JSP page will be compiled to. Use this attribute cautiously, as it can limit the JSP container's ability to provide a specialized superclass that improves the quality of the compiled class.

■ import="{package.class | package.*}, ..."

A comma-separated list of Java packages that the JSP page should import. The packages (and their classes) are available to scriptlets, expressions, and declarations within the JSP page. If you want to import more than one package, you can specify a comma-separated list after import or you can use import more than once in a JSP page.

The following packages are implicitly imported, so you don't need to specify them with the import attribute:

```
java.lang.*
javax.servlet.*
javax.servlet.jsp.*
javax.servlet.http.*
```

You must place the import attribute before the element that calls the imported class.

■ session="true | false"

Whether the client must join an HTTP session in order to use the JSP page. If the value is true, the session object refers to the current or new session.

If the value is false, you cannot use the session object or a <jsp:useBean> element with scope=session in the JSP page. Either of these usages would cause a translation-time error.

The default value is true.

■ buffer="none | 8kb | sizekb"

The buffer size in kilobytes used by the out object to handle output sent from the compiled JSP page to the client web browser. The default value is 8kb. If you specify a buffer size, the output is buffered with at least the size you specified.

■ autoFlush="true false"

Whether the buffered output should be flushed automatically when the buffer is full. If set to true (the default value), the buffer will be flushed. If set to false, an exception will be raised when the buffer overflows. You cannot set autoFlush to false when buffer is set to none.

■ isThreadSafe="true false"

Whether thread safety is implemented in the JSP page. The default value is true, which means that the JSP container can send multiple, concurrent client requests to the JSP page. You must write code in the JSP page to synchronize the multiple client threads. If you use false, the JSP container sends client requests one at a time to the JSP page.

■ info="text"

A text string that is incorporated verbatim into the compiled JSP page. You can later retrieve the string with the Servlet.getServletInfo() method.

■ errorPage="relativeURL"

A pathname to a JSP page that this JSP page sends exceptions to. If the pathname begins with a /, the path is relative to the JSP application's document root directory and is resolved by the web server. If not, the pathname is relative to the current JSP page.

■ isErrorPage="true | false"

Whether the JSP page displays an error page. If set to true, you can use the exception object in the JSP page. If set to false (the default value), you cannot use the exception object in the JSP page.

■ contentType="mimeType [; charset=characterSet]" |

```
"text/html;charset=ISO-8859-1"
```

The MIME type and character encoding the JSP page uses for the response. You can use any MIME type or character set that are valid for the JSP container. The default MIME type is text/html, and the default character set is ISO-8859-1.

■ pageEncoding="characterSet | **ISO-8859-1**"

The character encoding the JSP page uses for the response. The default character set is $\sc 180-8859-1$.

Tip

If you need to include a long list of packages or classes in more than one JSP page, you can create a separate JSP page with a page directive that contains the import list and include that file in the main JSP page.

Taglib Directive

Defines a tag library and prefix for the custom tags used in the JSP page.

JSP Syntax

```
<%@ taglib uri="URIForLibrary" prefix="tagPrefix" %>
```

XML Syntax

None. However, see \leq jsp:root \geq .

Examples

```
<%@ taglib uri="http://www.jspcentral.com/tags" prefix="public" %>
<public:loop>
    ...
</public:loop>
```

Description

The taglib directive declares that the JSP page uses custom tags, names the tag library that defines them, and specifies their tag prefix.

You must use a taglib directive *before* you use the custom tag in a JSP page. You can use more than one taglib directive in a JSP page, but the prefix defined in each must be unique.

Tutorials on creating custom tags are available at http://java.sun.com/products/jsp/taglibraries.html#tutorials.

Attributes

■ uri="URIForLibrary"

The Uniform Resource Identifier (URI) that uniquely locates the TLD that describes the set of custom tags associated with the named tag prefix. A URI can be any of the following:

- A Uniform Resource Locator (URL), as defined in RFC 2396, available at http://www.hut.fi/u/jkorpela/rfc/2396/full.html
- A Uniform Resource Name (URN), as defined in RFC 2396

■ An absolute or relative pathname

If the ${\tt URI}$ is a URL or URN, then the TLD is located by consulting the mapping indicated in web.xml extended using the implicit maps in the packaged tag libraries. If ${\tt URI}$ is pathname, it is interpreted relative to the root of the web application and should resolve to a TLD file directly, or to a JAR file that has a TLD file at location META-INF/taglib.tld.

■ prefix="tagPrefix"

The prefix that precedes the custom tag name, for example, public in <public:loop>. Empty prefixes are illegal. If you are developing or using custom tags, you cannot use the tag prefixes jsp, jspx, java, javax, servlet, sun, and sunw, as they are reserved by Sun Microsystems.

See Also

■ <jsp:root>

<jsp:forward>

Forwards a request to a web resource.

JSP Syntax

XML Syntax

Examples

```
<jsp:forward page="/servlet/login" />
<jsp:forward page="/servlet/login">
    <jsp:param name="username" value="jsmith" />
</jsp:forward>
```

Description

The <jsp:forward> element forwards the request object containing the client request information from one JSP page to another resource. The target resource can be an HTML file, another JSP page, or a servlet, as long as it is in the same application context as the forwarding JSP page. The lines in the source JSP page after the <jsp:forward> element are not processed.

You can pass parameter names and values to the target resource by using a <jsp:param> clause. An example of this would be passing the parameter name username (with name="username") and the value scott (with value="scott") to a servlet as part of the request. If you use <jsp:param>, the target resource should be a dynamic resource that can handle the parameters.

Be careful when using <jsp:forward> with unbuffered output. If you have used
the page directive with buffer="none" to specify that the output of your JSP page
should not be buffered, and if the JSP page has any data in the out object, using
<jsp:forward> will cause an IllegalStateException.

Attributes

■ page="{relativeURL | <%= expression %>}"

A String or an expression representing the relative URL of the component to which you are forwarding the request. The component can be another JSP page, a servlet, or any other object that can respond to a request.

The relative URL looks like a path—it cannot contain a protocol name, port number, or domain name. The URL can be absolute or relative to the current JSP page. If it is absolute (beginning with a /), the path is resolved by your web or application server.

Sends one or more name/value pairs as parameters to a dynamic resource. The target resource should be dynamic, that is, a JSP page, servlet, or other resource that can process the data that is sent to it as parameters.

You can use more than one sparam> clause if you need to send more than one parameter to the target resource. The name attribute specifies the parameter name and takes a case-sensitive literal string as a value. The value attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at request time.

- Include Directive
- <jsp:include>
- Page Directive

<jsp:getProperty>

Inserts the value of a bean property into the result.

JSP Syntax

```
<jsp:getProperty name="beanInstanceName" property="propertyName" />
```

XML Syntax

```
<jsp:getProperty name="beanInstanceName" property="propertyName" />
```

Examples

```
<jsp:useBean id="calendar" scope="page" class="employee.Calendar" />
<h2>
Calendar of <jsp:getProperty name="calendar" property="username" />
</h2>
```

Description

The <jsp:getProperty> element gets a bean property value using the property's getter methods and inserts the value into the response. You must create or locate a bean with <jsp:useBean> before you use <jsp:getProperty>.

The <jsp:getProperty> element has a few limitations you should be aware of:

- You cannot use <jsp:getProperty> to retrieve the values of an indexed property.
- You can use <jsp:getProperty> with JavaBeans components, but not with enterprise beans. As alternatives, you can write a JSP page that retrieves values from a bean that in turn retrieves values from an enterprise bean, or you can write a custom tag that retrieves values from an enterprise bean directly.

Attributes

■ name="beanInstanceName"

The name of an object (usually an instance of a bean) as declared in a <jsp:useBean> element.

■ property="propertyName"

The name of the bean property whose value you want to display. The property is declared as a variable in a bean and must have a corresponding getter method (for more information on declaring variables and writing getter methods in beans, see http://java.sun.com/products/javabeans/docs/).

Tip

If you use <jsp:getProperty> to retrieve a property value that is null, a NullPointerException is thrown. However, if you use a scriptlet or expression to retrieve the value, the string null is displayed in the browser; see Scriptlet or Expression for more information.

- <jsp:useBean>
- <jsp:setProperty>

<jsp:include>

Includes a static file or the result from another web component.

JSP Syntax

XML Syntax

```
<jsp:include page="{relativeURL | %= expression %}"
  [ flush="true | false" ] />

or
<jsp:include page="{relativeURL | %= expression %}"
  [ flush="true | false" ] >
  [ <jsp:param name="parameterName"
     value="{parameterValue | %= expression %}" /> ] +
</jsp:include> }
```

Examples

Description

The <jsp:include> element allows you to include either a *static* or *dynamic* resource in a JSP page. The results of including static and dynamic resources are quite different. If the resource is static, its content is included in the calling JSP page. If the

resource is dynamic, it acts on a request and sends back a result that is included in the JSP page. When the include action is finished, the JSP container continues processing the remainder of the JSP page.

You cannot always determine from a pathname if a resource is static or dynamic. For example, http://server:8080/index.html might map to a servlet through a server alias. The jsp:include> element handles both types of resources, so it is convenient to use when you don't know whether the resource is static or dynamic.

If the included resource is dynamic, you can use a <jsp:param> clause to pass the name and value of a parameter to the resource. As an example, you could pass the string username and a user's name to a login form that is coded in a JSP page.

Attributes

■ page="{ relativeURL | <%= expression %> }"

The relative URL that locates the resource to be included, or an expression that evaluates to a String equivalent to the relative URL.

The relative URL looks like a pathname—it cannot contain a protocol name, port number, or domain name. The URL can be absolute or relative to the current JSP page. If it is absolute (beginning with a /), the pathname is resolved by your web or application server.

■ flush="true | false"

If the page output is buffered and the flush attribute is given a true value, the buffer is flushed prior to the inclusion, otherwise the buffer is not flushed. The default value for the flush attribute is false.

The <jsp:param> clause allows you to pass one or more name/value pairs as parameters to an included resource. The included resource should be dynamic, that is, a JSP page, servlet, or other resource that can process the parameter.

You can use more than one <jsp:param> clause if you want to send more than one parameter to the included resource. The name attribute specifies the parameter name and takes a case-sensitive literal string. The value attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at request time.

- Include Directive
- <jsp:forward>

<jsp:plugin>

Causes the execution of an applet or bean. The applet or bean executes in the specified plugin. If the plugin is not available, displays a dialog to initiate the download of the plugin software.

JSP Syntax

```
<jsp:plugin
  type="bean|applet"
  code="classFileName"
  codebase="classFileDirectoryName"
  [ name="instanceName" ]
  [ archive="URIToArchive, ..." ]
  [ align="bottom|top|middle|left|right" ]
  [ height="{displayPixels | <%=expression%>}"]
  [ width="{displayPixels | <%= expression %>}"]
  [ hspace="leftRightPixels" ]
  [ vspace="topBottomPixels" ]
  [ jreversion="JREVersionNumber | 1.2" ]
  [ nspluginurl="URLToPlugin" ]
  [ iepluginurl="URLToPlugin" ] >
  [ <jsp:params>
     [ <jsp:param name="parameterName"</pre>
         value="{parameterValue | <%= expression %>}" /> ]+
  </isp:params> ]
  [ <jsp:fallback> text message for user </jsp:fallback> ]
</jsp:plugin>
```

XML Syntax

```
<jsp:plugin
  type="bean|applet" code="classFileName"
  codebase="classFileDirectoryName"
  [ name="instanceName" ]  [ archive="URIToArchive, ..." ]
  [ align="bottom|top|middle|left|right" ]
  [ height="{displayPixels | %= expression %}" ]
  [ width="{displayPixels | %= expression %}" ]
  [ hspace="leftRightPixels" ]  [ vspace="topBottomPixels" ]
  [ jreversion="JREVersionNumber | 1.2" ]
  [ nspluginurl="URLToPlugin" ]
  [ iepluginurl="URLToPlugin" ] >
```

```
[ <jsp:params>
        [ <jsp:param name="parameterName"
            value="{parameterValue | %= expression %}" /> ]+
        </jsp:params> ]
        [ <jsp:fallback> text message if plugin download fails
        </jsp:fallback> ]
</jsp:plugin>
```

Examples

Description

The <jsp:plugin> element plays or dispays an object (typically an applet or bean) in the client web browser, using a Java plug-in that is built in to the browser or downloaded from a specified URL.

When the JSP page is translated and compiled and Java and sends back an HTML response to the client, the <jsp:plugin> element is replaced by either an <object> or <embed> element, according to the browser version. The <object> element is defined in HTML 4.0 and <embed> in HTML 3.2.

In general, the attributes to the <jsp:plugin> element specify whether the object is a bean or an applet, locate the code that will be run, position the object in the browser window, specify an URL from which to download the plug-in software, and pass parameter names and values to the object. The attributes are described in detail in the next section.

Attributes

■ type="bean|applet"

The type of object the plug-in will execute. You must specify either bean or applet, as this attribute has no default value.

■ code="classFileName"

The name of the Java class file the plug-in will execute. You must include the . class extension in the name. The class file you specify should be in the directory named in the codebase attribute.

■ codebase="classFileDirectoryName"

The directory (or path to the directory) that contains the Java class file the plug-in will execute. If you do not supply a value, the path of the JSP page that calls <jsp:plugin> is used.

■ name="instanceName"

A name for the instance of the bean or applet, which makes it possible for applets or Beans called by the same JSP page to communicate with each other.

■ archive="URIToArchive, ..."

A comma-separated list of pathnames that locate archive files that will be preloaded with a class loader located in the directory named in codebase. The archive files are loaded securely, often over a network, and typically improve the applet's performance.

■ align="bottom|top|middle|left|right"

The position of the image, object, or applet. The position descriptions listed below use the term *text line* to mean the line in the viewable JSP page that corresponds to the line in the JSP page where the <jsp:plugin> element appears. The allowed values for align are listed below:

bottom Aligns the bottom of the image with the baseline of the text line.

top Aligns the top of the image with the top of the text line.

 ${\tt middle}$ Aligns the vertical center of the image with the baseline of the text

line.

left Floats the image to the left margin and flows text along the image's

right side.

right Floats the image to the right margin and flows text along the image's

left side.

```
■ height="{displayPixels | <%= expression %>}"
width="{displayPixels | <%= expression %>}"
```

The initial height and width, in pixels, of the image the applet or bean displays, not counting any windows or dialog boxes the applet or bean brings up.

hspace="leftRightPixels"
vspace="topBottomPixels"

The amount of space, in pixels, to the left and right (or top and bottom) of the image the applet or bean displays. The value must be a nonzero number. Note that hspace creates space to both the left *and* right and vspace creates space to both the top *and* bottom.

■ jreversion="JREVersionNumber | 1.2"

The version of the Java Runtime Environment (JRE) the applet or bean requires. The default value is 1.2.

■ nspluginurl="URLToPlugin"

The URL where the user can download the JRE plug-in for Netscape Navigator. The value is a full URL, with a protocol name, optional port number, and domain name.

■ iepluginurl="URLToPlugin"

The URL where the user can download the JRE plug-in for Internet Explorer. The value is a full URL, with a protocol name, optional port number, and domain name.

The parameters and values that you want to pass to the applet or bean. To specify more than one parameter value, you can use more than one <jsp:param> element within the <jsp:params> element.

The name attribute specifies the parameter name and takes a case-sensitive literal string. The value attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at runtime.

If the dynamic resource you are passing the parameter to is an applet, it reads the parameter with the <code>java.applet.Applet.getParameter</code> method.

■ <jsp:fallback> text message for user </jsp:fallback>

A text message to display for the user if the plug-in cannot be started. If the plug-in starts but the applet or bean does not, the plug-in usually displays a popup window explaining the error to the user.

- The HTML 3.2 specification: http://www.w3.org/TR/REC-html32.html
- The HTML 4.0 specification: http://www.w3.org/TR/REC-html40/

<jsp:setProperty>

Sets a property value or values in a bean.

JSP Syntax

```
<jsp:setProperty name="beanInstanceName"
{
  property="*" |
  property="propertyName" [ param="parameterName" ] |
  property="propertyName" value="{stringLiteral| <%= expression %>}"
}
/>
```

XML Syntax

```
<jsp:setProperty name="beanInstanceName"
{
  property="*" |
  property="propertyName" [ param="parameterName" ] |
  property="propertyName" value="{stringLiteral | %= expression %}"
}
/>
```

Examples

```
<jsp:setProperty name="mybean" property="*" />
<jsp:setProperty name="mybean" property="username" />
<jsp:setProperty name="mybean" property="username" value="Steve" />
```

Description

The <jsp:setProperty> element sets the value of one or more properties in a bean, using the bean's setter methods. You must declare the bean with <jsp:useBean> before you set a property value with <jsp:setProperty>. Because <jsp:useBean> and <jsp:setProperty> work together, the bean instance names they use must match (that is, the value of name in <jsp:setProperty> and the value of id in <jsp:useBean> must be the same).

You can use <jsp:setProperty> to set property values in several ways:

 By passing all of the values the user enters (stored as parameters in the request object) to matching properties in the bean

- By passing a specific value the user enters to a specific property in the bean
- By setting a bean property to a value you specify as either a String or an expression that is evaluated at runtime

Each method of setting property values has its own syntax, as described in the next section.

Attributes and Usage

■ name="beanInstanceName"

The name of an instance of a bean that has already been created or located with a <jsp:useBean> element. The value of name must match the value of id in <jsp:useBean>. The <jsp:useBean> element must appear before <jsp:set-Property> in the JSP page.

■ property="*"

Stores all of the values of request parameters in bean properties. The names of the bean properties must match the names of the request parameters. A bean property is usually defined by a variable declaration with matching getter and setter methods (for more information, see http://java.sun.com/products/javabeans/docs/).

The values of the request parameters sent from the client to the server are always of type String. The String values are converted to other data types when stored in bean properties. If a property has a PropertyEditor class as indicated in the JavaBeans specification, the setAsText(String) method is used. A conversion failure arises if the method throws an IllegalArgumentException. The allowed bean property types and their conversion methods are shown in TABLE 1.

TABLE 1 How < isp:setProperty> Converts Strings to Other Values

Property Type	String Is Converted Using	
Bean Property	Use setAsText(stringLiteral)	
boolean or Boolean	java. lang. Boolean. value Of (String)	
byte or Byte	java.lang.Byte.valueOf(String)	
char or Character	java.lang.String.charAt(0)	
double or Double	java.lang.Double.valueOf(String)	
integer or Integer	java.lang.Integer.valueOf(String)	
float or Float	java.lang.Float.valueOf(String)	

TABLE 1 How < isp:setProperty> Converts Strings to Other Values

Property Type	String Is Converted Using	
long or Long	java.lang.Long.valueOf(String)	
short or Short	java.lang.Short.valueOf(String)	
Object	new String(string-literal)	

You can also use <jsp:setProperty> to set the value of an indexed property in a bean. The indexed property must be an array of one of the data types shown in TABLE 1. The array elements are converted using the conversion methods shown in the table.

If a request parameter has an empty or null value, the corresponding bean property is not set. Likewise, if the bean has a property that does not have a matching request parameter, the property value is not set.

■ property="propertyName" [param="parameterName"]

Sets one bean property to the value of one request parameter. In the syntax, property specifies the name of the bean property and param specifies the name of the request parameter by which data is being sent from the client to the server.

If the bean property and the request parameter have different names, you must specify both property and param. If they have the same name, you can specify property and omit param.

If a parameter has an empty or null value, the corresponding bean property is not set.

■ property="propertyName" value="{string | <%= expression %>}"

Sets one bean property to a specific value. The value can be a String or an expression that is evaluated at runtime. If the value is a String, it is converted to the bean property's data type according to the conversion rules shown above in TABLE 1. If it is an expression, its value must have a data type that matches the the data type of the value of the expression must match the data type of the bean property.

If the parameter has an empty or null value, the corresponding bean property is not set. You cannot use both the param and value attributes in a <jsp:setProperty> element.

- <jsp:useBean>
- <isp:getProperty>

Tip

When you use property="*", the bean properties are not necessarily set in the order in which they appear in the HTML form or the bean. If the order in which the properties are set is important to how your bean works, use the syntax form property="propertyName" [param="parameterName"]. Better yet, rewrite your bean so that the order of setting properties is not important.

<jsp:useBean>

Locates or instantiates a bean with a specific name and scope.

JSP Syntax

```
<jsp:useBean id="beanInstanceName"
  scope="page|request|session|application"
{
  class="package.class" [ type="package.class" ]|
  beanName="{package.class | <%= expression %>}"
       type="package.class" |
       type="package.class" |
       type="package.class" |
       type="package.class" |
```

XML Syntax

```
<jsp:useBean id="beanInstanceName"
   scope="page|request|session|application"
{
   class="package.class" [ type="package.class" ] |
   beanName="{package.class | %= expression %}"
        type="package.class" |
   type="package.class" |
   type="package.class"
}
{ /> | > other elements </jsp:useBean> }
```

Examples

Description

The <jsp:useBean> element locates or instantiates a JavaBeans component. <jsp:useBean> first attempts to locate an instance of the bean. If the bean does not exist, <jsp:useBean> instantiates it from a class or serialized template.

To locate or instantiate the bean, <jsp:useBean> takes the following steps, in this order:

- 1. Attempts to locate a bean with the scope and name you specify.
- 2. Defines an object reference variable with the name you specify.
- 3. If it finds the bean, stores a reference to it in the variable. If you specified type, gives the bean that type.
- 4. If it does not find the bean, instantiates it from the class you specify, storing a reference to it in the new variable. If the class name represents a serialized template, the bean is instantiated by java.beans.Beans.instantiate.
- 5. If <jsp:useBean> has instantiated (rather than located) the bean, and if it has body tags or elements (between <jsp:useBean> and </jsp:useBean>), executes the body tags.

The body of a <jsp:useBean> element often contains a <jsp:setProperty> element that sets property values in the bean. As described in Step 5, the body tags are only processed if <jsp:useBean> instantiates the bean. If the bean already exists and <jsp:useBean> locates it, the body tags have no effect.

You can use a <jsp:useBean> element to locate or instantiate a JavaBeans component, but not an enterprise bean. To create enterprise beans, you can write a <jsp:useBean> element that calls a bean that in turn calls the enterprise bean, or you can write a custom tag that calls an enterprise bean directly.

Attributes and Usage

■ id="beanInstanceName"

A variable that identifies the bean in the scope you specify. You can use the variable name in expressions or scriptlets in the JSP page.

The name is case sensitive and must conform to the naming conventions of the scripting language used in the JSP page. If you use the Java programming language, the conventions in the Java Language Specification. If the bean has already been created by another <jsp:useBean> element, the value of id must match the value of id used in the original <jsp:useBean> element.

■ scope="page|request|session|application"

The scope in which the bean exists and the variable named in id is available. The default value is page. The meanings of the different scopes are shown below:

page

You can use the bean within the JSP page with the <jsp:useBean> element or any of the page's static include files, until the page sends a response back to the client or forwards a request to another resource.

request You can use the bean from any JSP page processing the same

request, until a JSP page sends a response to the client or forwards the request to another resource. You can use the request object to access the bean, for example, request.

getAttribute(beanInstanceName).

session You can use the bean from any JSP page in the same session as

the JSP page that created the bean. The bean exists across the entire session, and any page that participates in the session can use it. The page in which you create the bean must have a page

directive with session="true".

application You can use the bean from any JSP page in the same application

as the JSP page that created the bean. The bean exists across an entire JSP application, and any page in the application can use

the bean.

■ class="package.class"

Instantiates a bean from a class, using the new keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor. The package and class name are case sensitive.

■ type="package.class"

If the bean already exists in the scope, gives the bean a data type other than the class from which it was instantiated. The value of type must be a superclass of class or an interface implemented by class.

If you use type without class or beanName, no bean is instantiated. The package and class name are case sensitive.

■ class="package.class" type="package.class"

Instantiates a bean from the class named in class and assigns the bean the data type you specify in type. The value of type can be the same as class, a superclass of class, or an interface implemented by class.

The class you specify in class must not be abstract and must have a public, noargument constructor. The package and class names you use with both class and type are case sensitive.

beanName="{package.class | <%= expression %>}" type="package.
class"

Instantiates a bean from a class, a serialized template, or an expression that evaluates to a class or serialized template. When you use beanName, the bean is instantiated by the java.beans.Beans.instantiate method. The Beans.instantiate method checks whether the package and class you specify represents a class or a serialized template. If they represent a serialized template, Beans.instantiate reads the serialized form (which has a name like package.class.ser) using a class loader.

The value of type can be the same as beanName, a superclass of beanName, or an interface implemented by beanName. The package and class names you use with both beanName and type are case sensitive.

- <jsp:setProperty>
- <jsp:getProperty>
- http://java.sun.com/products/javabeans/docs/