CHAPTER 1

Core Syntax Reference

JSP 1.1 Tomcat Beta

HTML Comment

Generates a comment that is sent to the client.

JSP Syntax

```
<!-- comment [ <%= expression %> ] -->
```

Example 1

```
<!-- This file displays the user login screen -->
```

Displays in the page source:

```
<!-- This file displays the user login screen -->
```

Example 2

```
<!-- This page was loaded on
   <%= (new java.util.Date()).toLocaleString() %> -->
```

Displays in the page source:

```
<!-- This page was loaded on January 1, 2000 -->
```

Description

An HTML comment in a JSP file is very similar to any other HTML comment. It documents the file and can be viewed in the page source from your Web browser.

The one difference is that you can use an expression in an HTML comment in a JSP file. The expression is dynamic and is evaluated when the page is loaded or reloaded in the Web browser. You can use any expression that is valid in the page scripting language; for more information, see Expression.

See Also

- Hidden Comment
- Expression

Hidden Comment

Documents the JSP page but is not sent to the client.

JSP Syntax

<%-- *comment* --%>

Examples

```
<%@ page language="java" %>
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</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 hide or “comment out” part of your JSP page. The JSP container does not process anything within the <%-- and --%> characters. A hidden comment is not sent to the client, either in the displayed JSP page or the page source.

You can use any characters in the body of the comment except the closing --%> combination. If you need to use --%> in your comment, you can escape it by typing --%\>.

See Also

- HTML Comment

Declaration

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

JSP Syntax

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

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 Java™ code later in the JSP file. You must declare the variable or method *before* you use it in the JSP file.

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 file, remember these rules:

- You must end the declaration with a semicolon (the same rule as for a Scriptlet, but the opposite of an Expression).
- 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 file included with a <jsp:include> element. The scope of a declaration does not include dynamic files included with <jsp:include>.

See Also

- Scriptlet
- Expression

Expression

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

JSP Syntax

<%= *expression* %>

Examples

```
The map file has <font color="blue"><%= map.size() %></font> entries.
Good guess, but nope. Try <b><%= numguess.getHint() %></b>.
```

Description

An expression element contains a scripting language expression that is evaluated, converted to a `String`, and inserted where the expression appears in the JSP file. 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 file.

When you write expressions in a JSP file, remember these points:

- You cannot use a semicolon to end an expression (however, the same expression within a scriptlet requires the semicolon; see Scriptlet).
- The expression element can contain any expression that is valid according to the *Java Language Specification*.

You can sometimes use expressions as attribute values in JSP elements (see the *JavaServer Pages™ 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
- Scriptlet

Scriptlet

Contains a code fragment valid in the page scripting language.

JSP Syntax

<% *code fragment* %>

Examples

```
<%
   String name = null;
   if (request.getParameter("name") == null) {
%>

<%@ include file="error.html" %>

<%
   } else {
   foo.setName(request.getParameter("name"));
   if (foo.getName().equalsIgnoreCase("integra"))
      name = "acura";
   if (name.equalsIgnoreCase( "acura" )) {
%>
```

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 file (see also Declaration).
- Write expressions valid in the page scripting language (see also Expression).
- Use any of the implicit objects or any object declared with a `<jsp:useBean>` element.
- Write any other statement valid in the scripting language used in the JSP page (if you use the Java programming language, the statements must conform to the *Java Language Specification*).

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 client request. If the scriptlet produces output, the output is stored in the `out` object.

See Also

- Declaration
- Expression

Include Directive

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

JSP Syntax

```
<%@ 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:

```
The current date and time are
Aug 30, 1999 2:38:40
```

Description

The `<%@ include %>` directive inserts a file of text or code in a JSP file at translation time, when the JSP file 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 file. The included file can be a JSP file, HTML file, or text file. If the included file is a JSP file, its JSP elements are parsed and their results included (along with any other text) in the JSP file.

You can only use `include` to include static files. This means that the parsed result of the included file is added to the JSP file where the `<%@ include %>` directive is placed. Once the included file is parsed and included, processing resumes with the next line of the calling JSP file.

The included file can be an HTML file, a JSP file, a text file, or a code file written in the Java programming language. Be careful, though, that the included file does not contain `<html>`, `</html>`, `<body>`, or `</body>` tags. Because the entire content of the included file is added at that location in the JSP file, these tags would conflict with the same tags in the calling JSP file, 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. Simply put, 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 `javax.servlet.ServletContext` object that is in turn stored in the `application` object. If the relative URL starts with a directory or file name, the path is relative to the JSP file.

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.

See Also

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

Page Directive

Defines attributes that apply to an entire JSP page.

JSP Syntax

```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import="{package.class | package.*}, ..." ]
  [ session="true|false" ]
  [ buffer="none|8kb|sizekb" ]
  [ autoFlush="true|false" ]
  [ isThreadSafe="true|false" ]
  [ info="text" ]
  [ errorPage="relativeURL" ]
  [ contentType="mimeType [ ;charset=characterSet ]" |
    "text/html ; charset=ISO-8859-1" ]
  [ isErrorPage="true|false" ]
%>
```

Examples

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

Description

The `<%@ page %>` directive applies to an entire JSP file 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 file. The `<%@ page %>` directive does not apply to any dynamic include files; see `<jsp:include>` for more information.

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 file or translation unit.

No matter where you position the <%@ page %> directive in a JSP file 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 file.

Attributes

- language="**java**"

  The scripting language used in scriptlets, declarations, and expressions in the JSP file and any included files. In this release, the only allowed value is java.

- extends="*package.class*"

  The fully qualified name of the superclass of the Java class file this JSP file 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 file.

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

  A comma-separated list of Java packages that the JSP file should import. The packages (and their classes) are available to scriptlets, expressions, and declarations within the JSP file. 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 file.

  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 file. Either of these usages would cause a translation-time error.

  The default value is true.

- buffer="none|**8kb**|*size*kb"

  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 file. 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 file that this JSP file 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 file.

- isErrorPage="true|**false**"

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

- contentType="*mimeType* [; charset=*characterSet*]" |
  "**text/html;charset=ISO-8859-1**"

  The MIME type and character encoding the JSP file uses for the response it sends to the client. 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`.

Tip

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

Taglib Directive

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

JSP Syntax

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

Examples

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

<public:loop>
.
.
</public:loop>
```

Description

The `<%@ taglib %>` directive declares that the JSP file uses custom tags, names the tag library that defines them, and specifies their tag prefix.

Here, the term *custom tag* refers to both tags and elements. Because JSP files can be converted to XML, it is important to understand the relationship of tags and elements. A tag is simply a short piece of markup that is part of a JSP element. A JSP element is a unit of JSP syntax that has an XML equivalent with a start tag and an end tag. An element can also contain other text, tags, or elements. For example, a `jsp:plugin` element always has a `<jsp:plugin>` start tag and a `</jsp:plugin>` end tag and may have a `<jsp:params>` element and a `<jsp:fallback>` element.

You must use a `<%@ taglib %>` directive *before* you use the custom tag in a JSP file. You can use more than one `<%@ taglib %>` directive in a JSP file, but the prefix defined in each must be unique.

The technique for creating custom tags is described in the *JavaServer Pages Specification* for version 1.1.

Attributes

- `uri="`*`URIToTagLibrary`*`"`

  The Uniform Resource Identifier (URI) that uniquely names 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

- `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.

<jsp:forward>

Forwards a client request to an HTML file, JSP file, or servlet for processing.

JSP Syntax

```
<jsp:forward page="{relativeURL | <%= expression %>}" />
```

or

```
<jsp:forward page="{relativeURL | <%= expression %>}" >
   <jsp:param name="parameterName"
       value="{parameterValue | <%= expression %>}" />+
</jsp:forward>
```

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 file to another file. The target file can be an HTML file, another JSP file, or a servlet, as long as it is in the same application context as the forwarding JSP file. The lines in the source JSP file after the `<jsp:forward>` element are not processed.

You can pass parameter names and values to the target file 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 login file as part of the request. If you use `<jsp:param>`, the target file should be a dynamic file 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 file should not be buffered, and if the JSP file 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 file to which you are forwarding the request. The file can be another JSP file, a servlet, or any other dynamic file that can handle a `request` object.

  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 file. If it is absolute (beginning with a /), the path is resolved by your Web or application server.

- `<jsp:param name="`*parameterName*`"`
  `value="{`*parameterValue* `|` `<%=` *expression* `%>}" />+`

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

  You can use more than one `<jsp:param>` clause if you need to send more than one parameter to the target file. 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.

See Also

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

<jsp:getProperty>

Gets the value of a Bean property so that you can display it in a result page.

JSP 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 displays the property value in a JSP page. 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 the *JavaBeans API Specification*).

Tips

- In Sun's JSP reference implementation, 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.

See Also

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

<jsp:include>

Includes a static file or sends a request to a dynamic file.

JSP Syntax

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

or

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

Examples

```
<jsp:include page="scripts/login.jsp" />
<jsp:include page="copyright.html" />
<jsp:include page="/index.html" />
```

```
<jsp:include page="scripts/login.jsp">
   <jsp:param name="username" value="jsmith" />
</jsp:include>
```

Description

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

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

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

Attributes

- `page="{`*relativeURL* `|` `<%=` *expression* `%>}"`

  The relative URL that locates the file 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 file. If it is absolute (beginning with a /), the pathname is resolved by your Web or application server.

- `flush="true"`

  You must include `flush="true"`, as it is not a default value. You cannot use a value of `false`. Use the `flush` attribute exactly as it is given here.

- `<jsp:param name="`*parameterName*`"`
  `value="{`*parameterValue* `|` `<%=` *expression* `%>}" />+`

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

See Also

- Include Directive
- `<jsp:forward>`

<jsp:plugin>

Executes an applet or Bean and, if necessary, downloads a Java plug-in to execute it.

JSP Syntax

```
<jsp:plugin
   type="bean|applet"
   code="classFileName"
   codebase="classFileDirectoryName"
   [ name="instanceName" ]
   [ archive="URIToArchive, ..." ]
   [ align="bottom|top|middle|left|right" ]
   [ height="displayPixels" ]
   [ width="displayPixels" ]
   [ hspace="leftRightPixels" ]
   [ vspace="topBottomPixels" ]
   [ jreversion="JREVersionNumber | 1.1" ]
   [ nspluginurl="URLToPlugin" ]
   [ iepluginurl="URLToPlugin" ] >

   [ <jsp:params>
      [ <jsp:param name="parameterName"
        value="{parameterValue | <%= expression %>}" /> ]+
   </jsp:params> ]

   [ <jsp:fallback> text message for user </jsp:fallback> ]

</jsp:plugin>
```

Examples

```
<jsp:plugin type=applet code="Molecule.class" codebase="/html">
   <jsp:params>
      <jsp:param name="molecule" value="molecules/benzene.mol" />
   </jsp:params>
   <jsp:fallback>
      <p>Unable to load applet</p>
   </jsp:fallback>
</jsp:plugin>
```

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 file 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 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 file 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 file 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 file 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.
`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*`"`
  `width="`*displayPixels*`"`

  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*`|`**`1.1`**`"`

  The version of the Java Runtime Environment (JRE) the applet or Bean requires. The default value is 1.1.

- `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.

- <jsp:params>
  [<jsp:param name="*parameterName*"
  value="{*parameterValue* | <%= *expression* %>}" />]+
  </jsp:params>

  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 file you are passing the parameter to is an applet, it reads the parameter with the java.applet.Applet.getParameter 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.

See Also

- The official HTML 3.2 specification: *http://www.w3.org/TR/REC-html32.html*
- The official 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="{string | <%= 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:seProperty>` in the JSP file.

- `property="*"`

  Stores all of the values the user enters in the viewable JSP page (called **request parameters**) in matching Bean properties. The names of the properties in the Bean must match the names of the request parameters, which are usually the elements of an HTML form. A Bean property is usually defined by a variable declaration with matching getter and setter methods (for more information, see the JavaBeans API Specification available at `http://java.sun.com/beans`).

  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 so they can be stored in Bean properties. The allowed Bean property types and their conversion methods are shown in TABLE 1-1.

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

Property Type	String Is Converted Using
boolean or Boolean	java.lang.Boolean.valueOf(String)
byte or Byte	java.lang.Byte.valueOf(String)
char or Character	java.lang.Character.valueOf(String)
double or Double	java.lang.Double.valueOf(String)
integer or Integer	java.lang.Integer.valueOf(String)
float or Float	java.lang.Float.valueOf(String)
long or Long	java.lang.Long.valueOf(String)

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 1-1. 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 1-1. 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.

See Also

- `<jsp:useBean>`
- `<jsp:getProperty>`

Tips

- 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.

  In Sun's JSP 1.0 or JSP 1.1 Tomcat, the Bean properties are set in the order in which they are presented to the JSP container by the Beans introspector. 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" |
     class="package.class" type="package.class" |
     beanName="{package.class | <%= expression %>}" type="package.class"
  }
  { /> |
     > other elements
     </jsp:useBean>
  }
```

Examples

```
<jsp:useBean id="cart" scope="session" class="session.Carts" />
<jsp:setProperty name="cart" property="*" />
```

```
<jsp:useBean id="checking" scope="session" class="bank.Checking" >
<jsp:setProperty name="checking" property="balance" value="0.0" />
</jsp:useBean>
```

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.

In this release, you can use a `<jsp:useBean>` element to locate or instantiate a Bean, 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 file.

  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 file.
`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 file. 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, no-argument 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. For more information, see the *JavaBeans API Specification*.

  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.

See Also

- `<jsp:setProperty>`
- `<jsp:getProperty>`
- Javadoc API reference for `java.beans.Beans`
- *JavaBeans API Specification*