CHAPTER 1

Core Syntax Reference

Output Comment

Generates a comment that is sent to the client in the viewable page source.

JSP Syntax

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

Examples

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

The JSP engine handles an output comment as uninterpreted HTML text, returning the comment in the HTML output sent to the client. You can see the comment by viewing the page source from your Web browser.

An Expression included in a comment is dynamic and is evaluated when the Web browser loads the page (that is, when the user first loads the page or reloads it later). You can use any valid JSP 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

The JSP engine ignores a hidden comment, and does not process any code within hidden comment tags. A hidden comment is not sent to the client, either in the displayed JSP page or the HTML page source. The hidden comment is useful when you want to hide or “comment out” part of your JSP page.

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

- Output Comment

Declaration

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

JSP Syntax

<%! *declarations* %>

Examples

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

Description

A declaration declares one or more variables or methods for use later in the JSP source file.

A declaration must contain at least one complete declarative statement. You can declare any number of variables or methods within one declaration tag, as long as they are separated by semicolons. The declaration must be valid in the scripting language used in the JSP file. In JSP 1.0, you must use the Java™ programming language for scripting, and declarations must conform to the *Java Language Specification*.

When you use the Java programming language for scripting, 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 must declare the variable or method *before* you use it in the JSP file.
- You can already use variables or methods that are declared in packages imported by the Page Directive, so there is no need to declare them.

In general, a declaration has page scope, so it is valid in scriptlets, expressions, and other declarations within the same JSP source file. But if a JSP file includes other files, the rules of scope become a bit trickier. The rules of scope for declarations are described below:

- If the JSP source file does not include any other files, declarations have page scope.

- If the JSP source file includes a dynamic file with `<jsp:include>`, declarations have page scope.
- If the JSP source file includes static files with `<jsp:include>` or the Include Directive, declarations have **translation unit** scope. A translation unit is the JSP source file plus all of its static include files, but without any dynamic include files.

See Also

- Scriptlet
- Expression

Expression

Contains an expression valid in the page scripting language.

JSP Syntax

`<%=` *expression* `%>`

Examples

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

```
Good guess, but nope. Try <b><%= numguess.getHint() %></b>.
```

Description

An expression tag 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 text in a JSP file.

When you use the Java language for scripting, remember these points:

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

You can sometimes use expressions as attribute values in JSP tags (see the *JavaServer Pages™ Syntax Card*). If an expression has more than one part, the parts are evaluated in left-to-right order as they appear in the tag.

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 scriptlet tags, 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 JSP implicit objects or any object declared with a `<jsp:useBean>` tag.
- Write any other statement valid in the page scripting language (if you use the Java programming language, the statements must conform to the *Java Language Specification*).

You must write plain text, HTML-encoded text, or other JSP tags *outside* the scriptlet.

Scriptlets are executed at request time, when the JSP engine processes the client request. If the scriptlet produces output, the output is stored in the `out` object, from which you can display it.

See Also

- Declaration
- Expression

Include Directive

Includes a file of text or code in the JSP source file.

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. 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 tags 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 similar tags in the JSP file.

Some of the behaviors of the `include` directive depend on the JSP engine, for example:

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

Attributes

- `file="`*relativeURL*`"`

  The pathname to the included file, which can contain any text or code that is valid in a JSP file. The value of `file` is always a relative URL. Simply put, a relative URL is just the path segment of an URL, without the protocol, port, or domain:

  ```
  "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 comment tags around the text in the text file.

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 static files it includes with the Include Directive or `<jsp:include>`, which together are called a **translation unit**. Note that the `page` directive does not apply to any dynamic included 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 it more than once, just as you would use multiple `import` commands in the Java programming language) No matter where you position the `page` directive in the JSP file or included files, it applies to the entire translation unit. However, it is usually good programming style to place it at the top of the file.

Attributes

- `language="`**`java`**`"`

  The scripting language used in scriptlets, declarations, and expressions in the JSP file and any included files. In JSP 1.0, 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 engine's ability to provide a specialized superclass that improves the quality of the compiled file.

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

  A comma-separated list of one or more packages that the JSP file should import. The packages (and their classes) are available to scriptlets, expressions, declarations, and tags within the JSP file. You must place the `import` attribute *before* the tag that calls the imported class. 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.

- `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 in the JSP file. 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 `true` (the default value), means that the buffer will be flushed. If `false`, means that an exception will be raised when the buffer overflows. You cannot set `autoFlush` to `false` when the value of `buffer` is `none`.

- `isThreadSafe="`**`true`**`|false"`

  Whether thread safety is implemented in the JSP file. The default value is `true`, which means that the JSP engine can send multiple requests to the page concurrently. If you use `true` (the default value), multiple threads can access the JSP page, and you must synchronize them. If you use `false`, the JSP engine sends client requests one at a time to the JSP page.

- `info="`*`text`*`"`

  A text string that is incorporated verbatim in 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 `true`, you can use the `exception` object, which contains a reference to the thrown exception, in the JSP file. If `false` (the default value), means that 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 engine. The default MIME type is `text/html`, and the default character set is `ISO-8859-1`.

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.

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.

In JSP 1.0, the method of creating portable custom tags that can be recognized by any servlet engine is not yet defined. Some JSP engines might allow you to create custom tags, but those tags might not be recognized by other JSP engines.

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
  - 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.

See Also

- RFC 2396, available at `http://www.hut.fi/u/jkorpela/rfc/2396/full.html`

<jsp:forward>

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

JSP Syntax

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

Examples

```
<jsp:forward page="/servlet/login" />
```

Description

The `<jsp:forward>` tag forwards the `request` object sent to the JSP file to another file for processing. The JSP engine does not process any of the remainder of the JSP file.

If the output from the compiled JSP file is buffered (by using a `page` directive with the default value or an explicit size set for `buffer`), the buffer is cleared before the request is forwarded. If the output is not buffered (if you used a page directive with `buffer=none`), and if anything has been written to the buffer, using `<jsp:forward>` results in 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 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.

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

You must create or locate a Bean instance with `<jsp:useBean>` *before* you use `<jsp:getProperty>`.

The `<jsp:getProperty>` tag gets a Bean property value using the property's `get` method, converts the value of a Bean property to a `String` and stores it in the `out` object.

In JSP 1.0, `<jsp:getProperty>` 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. You can get around this limitation by writing a JSP application in which a JSP file gets values from a Bean, and the Bean in turns calls an enterprise bean.

Attributes

- `name="`*`beanInstanceName`*`"`

  The name of the Bean instance as declared in a `<jsp:useBean>` tag.

- `property="`*`propertyName`*`"`

  The name of the Bean property whose value you want to display.

See Also

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

<jsp:include>

Includes either a static or dynamic file in a JSP file.

JSP Syntax

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

Examples

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

Description

The `<jsp:include>` tag allows you to include either a **static** or **dynamic** file. A static file is parsed and its content included in the calling JSP page. A dynamic file acts on the request and sends back a result that is included in the JSP page.

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

When the include action is finished, the JSP engine continues processing the remainder of the JSP file.

Attributes

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

  The relative URL to 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"`

  In JSP 1.0, you must include `flush="true"`, as it is not a default value.

See Also

- Include Directive
- <jsp:forward>

<jsp:plugin>

Downloads a plugin to the client Web browser to execute an applet or Bean.

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" /> ]+
   </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> tag is replaced by either an <object> or <embed> tag, whichever is most appropriate for the client Web browser (the <object> tag is for browsers that use HTML 4.0).

The <jsp:params> element sends parameter names and values to an applet or Bean at startup. The <jsp:fallback> element provides a message for the user if the plugin does not start. If the plugin starts but the applet or Bean does not, the plugin usually displays a popup window explaining the error to the user.

The <jsp:plugin> tag takes most of its attributes from the HTML <applet> and <object> tags (<applet> is defined in HTML 3.2 and <object> in HTML 4.0). You may want to refer to the official HTML specifications in which these tags are introduced:

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

Attributes

- `type="bean|applet"`

  The type of object the plugin 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 that the plugin will execute. You must include the .class extension in the name following `code`. The filename is relative to the directory named in the `codebase` attribute.

- `codebase="`*classFileDirectoryName*`"`

  The absolute or relative path to the directory that contains the applet's code. 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 Bean or applet instance, 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 paths that locate archive files to 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 positioning of the image displayed by the applet or Bean relative to the line in the JSP result page that corresponds to the line in the JSP file containing the `<jsp:plugin>` tag. The results of the different values 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. Must be a small nonzero number.

- `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 plugin 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 plugin 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*" />]+
  </jsp:params>

  The parameters and values that you want to pass to the applet or Bean. To specify more than one name and value, use multiple <jsp:param> tags within the <jsp:params> element. Applets read parameters 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 plugin cannot be started.

<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>` tag sets the value of one or more properties in a JavaBean component, using the Bean's `set` methods. You must use a `<jsp:useBean>` tag to declare the Bean before you use `<jsp:setProperty>`. The value of `name` in `<jsp:setProperty>` must match the value of `id` in `<jsp:useBean>`.

With `<jsp:setProperty>`, you can set property values in several ways:

- By passing all of the values in the user's request (stored as parameters in the `request` object) to matching properties in the Bean
- By passing a specific value in the `request` object to a matching property or a property of a different name in the Bean
- By explicitly setting a Bean property to a value specified as a `String` or the result of an expression

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>` tag. The value of `name` must match the value of `id` in `<jsp:useBean>`. The `<jsp:useBean>` tag must appear before `<jsp:setProperty>` in the same JSP file.

- `property="*"`

  Stores all of the values in the `request` object parameters (called **request parameters**) in matching Bean properties. The property names in the Bean must match the request parameters. The parameter names usually come from the elements of an HTML form, and the values come from the data the user enters.

  The values of the request parameters 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 have one of the types shown in TABLE 1-1, and the request value assigned to it must be an array of the same type. The array elements are converted using the conversion methods shown in TABLE 1-1.

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. The request parameter can have a different name than the Bean property, and if so, you must specify param. If the Bean property and request parameter have the same name, you can omit param.

  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> tag.

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

  Sets one Bean property to a specific value. The value can be a String or an Expression. If you use a String, it is converted to the Bean property's data type, according to the conversion rules shown above in TABLE 1-1. If you use an expression, 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> tag.

See Also

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

<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 tags </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>` tag attempts to locates a Bean, or if the Bean does not exist, 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 it has *instantiated* (rather than located) the Bean, and if it has body tags (between `<jsp:useBean>` and `</jsp:useBean>`), executes the body tags.

The body of a `<jsp:useBean>` tag often contains a `<jsp:setProperty>` tag that defines property values in the object. 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 JSP 1.0, `<jsp:useBean>` works with JavaBeans components, but not with enterprise beans. If you want to use enterprise beans, you can write a JSP file that constructs a JavaBean component, and have the JavaBean component call the enterprise bean.

Attributes and Usage

- `id="`*beanInstanceName*`"`

  Names a variable that identifies the Bean in the scope you specify. You can use the variable name in expressions or scriptlets in the same JSP file.

  The name is case sensitive and must conform to the naming conventions of the page scripting language (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>` tag, the value of `id` must match the value of `id` used in the original `<jsp:useBean>` tag.

- `scope="`**`page`**`|request|session|application"`

  Defines a scope in which the Bean exists and the variable named in `id` is available. The default value is `page`.

- `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.

- `class="`*package.class*`" type="`*package.class*`"`

  Instantiates a Bean from a class, using the `new` keyword and the class constructor, and gives the Bean the type you specify in `type`. 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.

  The value of `type` can be the same as `class`, a superclass of `class`, or an interface implemented by `class`.

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

  Instantiates a Bean from either a class or a serialized template, using the `java.beans.Beans.instantiate` method, and gives the Bean the type specified in `type`. `Beans.instantiate` checks whether a name represents a class or a serialized template. If the Bean is a serialized template, `Beans.instantiate` reads the serialized form (with a name like *package.class.ser*) using a class loader. For more information, see the *JavaBeans API Specification*.

  The value of `beanName` is either a package and class name or an Expression that evaluates to a package and class name, and is passed to `Beans.instantiate`. The value of `type` is a package and class name. The value of `type` can be the same as `beanName`, a superclass of `beanName`, or an interface implemented by `beanName`.

  With both `beanName` and `type`, the package and class name are case sensitive.

- `type="package.class"`

  If the Bean already exists in the specified scope, gives the Bean the type you specify. If you use `type` without `class` or `beanName`, no Bean is instantiated. The package and class name are case sensitive.

See Also

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