JSP SCRIPTING ELEMENTS

Topics in This Chapter

- The purpose of JSP
- How JSP pages are invoked
- Using JSP expressions to insert dynamic results directly into the output page
- Using JSP scriptlets to insert Java code into the method that handles requests for the page
- Using JSP declarations to add methods and field declarations to the servlet that corresponds to the JSP page
- Predefined variables that can be used within expressions and scriptlets

Chapter 10

JavaServer Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content from servlets. You simply write the regular HTML in the normal manner, using familiar Web-page-building tools. You then enclose the code for the dynamic parts in special tags, most of which start with `<%` and end with `%>`. For example, here is a section of a JSP page that results in "Thanks for ordering *Core Web Programming*" for a URL of `http://host/OrderConfirmation.jsp?title=Core+Web+Programming`:

```
Thanks for ordering <I><%= request.getParameter("title") %></I>
```

Separating the static HTML from the dynamic content provides a number of benefits over servlets alone, and the approach used in JavaServer Pages offers several advantages over competing technologies such as ASP, PHP, or ColdFusion. Section 1.4 (The Advantages of JSP) gives some details on these advantages, but they basically boil down to two facts: that JSP is widely supported and thus doesn't lock you into a particular operating system or Web server and that JSP gives you full access to servlet and Java technology for the dynamic part, rather than requiring you to use an unfamiliar and weaker special-purpose language.

The process of making JavaServer Pages accessible on the Web is much simpler than that for servlets. Assuming you have a Web server that supports JSP, you give your file a `.jsp` extension and simply install it in any place you could put a normal Web page: no compiling, no packages, and no user `CLASSPATH` settings. However, although your personal *environment* doesn't need any special settings, the *server* still has to be set up with access to the servlet and JSP class files and the Java compiler. For details, see your server's documentation or Section 1.5 (Installation and Setup).

Although what you write often looks more like a regular HTML file than a servlet, behind the scenes, the JSP page is automatically converted to a normal servlet, with the static HTML simply being printed to the output stream associated with the servlet's `service` method. This translation is normally done the first time the page is requested. To ensure that the first real user doesn't get a momentary delay when the JSP page is translated into a servlet and compiled, developers can simply request the page themselves after first installing it. Many Web servers also let you define aliases so that a URL that appears to reference an HTML file really points to a servlet or JSP page.

Depending on how your server is set up, you can even look at the source code for servlets generated from your JSP pages. With Tomcat 3.0, you need to change the `isWorkDirPersistent` attribute from `false` to `true` in `install_dir/server.xml`. After that, the code can be found in `install_dir/work/port-number`. With the JSWDK 1.0.1, you need to change the `workDirIsPersistent` attribute from `false` to `true` in `install_dir/webserver.xml`. After that, the code can be found in `install_dir/work/%3Aport-number%2F`. With the Java Web Server, 2.0 the default setting is to save source code for automatically generated servlets. They can be found in `install_dir/tmpdir/default/pagecompile/jsp/_JSP`.

One warning about the automatic translation process is in order. If you make an error in the dynamic portion of your JSP page, the system may not be able to properly translate it into a servlet. If your page has such a fatal translation-time error, the server will present an HTML error page describing the problem to the client. Internet Explorer 5, however, typically replaces server-generated error messages with a canned page that it considers friendlier. You will need to turn off this "feature" when debugging JSP pages. To do so with Internet Explorer 5, go to the Tools menu, select Internet Options, choose the Advanced tab, and make sure "Show friendly HTTP error messages" box is not checked.

Core Warning

When debugging JSP pages, be sure to turn off Internet Explorer's "friendly" HTTP error messages.

Aside from the regular HTML, there are three main types of JSP constructs that you embed in a page: *scripting elements*, *directives*, and *actions*. Scripting elements let you specify Java code that will become part of the resultant servlet, directives let you control the overall structure of the servlet, and actions let you specify existing components that should be used and otherwise control the behavior of the JSP engine. To simplify the scripting elements, you have access to a number of predefined variables, such as `request` in the code snippet just shown (see Section 10.5 for more details). Scripting elements are covered in this chapter, and directives and actions are explained in the following chapters. You can also refer to Appendix (Servlet and JSP Quick Reference) for a thumbnail guide summarizing JSP syntax.

This book covers versions 1.0 and 1.1 of the JavaServer Pages specification. JSP changed dramatically from version 0.92 to version 1.0, and although these changes are very much for the better, you should note that newer JSP pages are almost totally incompatible with the early 0.92 JSP engines, and older JSP pages are equally incompatible with 1.0 JSP engines. The changes from version 1.0 to 1.1 are much less dramatic: the main additions in version 1.1 are the ability to portably define new tags and the use of the servlet 2.2 specification for the underlying servlets. JSP 1.1 pages that do not use custom tags or explicitly call 2.2-specific statements are compatible with JSP 1.0 engines, and JSP 1.0 pages are totally upward compatible with JSP 1.1 engines.

10.1 Scripting Elements

JSP scripting elements let you insert code into the servlet that will be generated from the JSP page. There are three forms:

1. *Expressions* of the form `<%= expression %>`, which are evaluated and inserted into the servlet's output
2. *Scriptlets* of the form `<% code %>`, which are inserted into the servlet's `_jspService` method (called by `service`)
3. *Declarations* of the form `<%! code %>`, which are inserted into the body of the servlet class, outside of any existing methods

Each of these scripting elements is described in more detail in the following sections.

Template Text

In many cases, a large percentage of your JSP page just consists of static HTML, known as *template text*. In almost all respects, this HTML looks just like normal HTML, follows all the same syntax rules, and is simply "passed through" to the client by the servlet created to handle the page. Not only does the HTML look normal, it can be created by whatever tools you already are using for building Web pages. For example, I used Allaire's HomeSite for most of the JSP pages in this book.

There are two minor exceptions to the "template text is passed straight through" rule. First, if you want to have `<%` in the output, you need to put `<\%` in the template text. Second, if you want a comment to appear in the JSP page but not in the resultant document, use

```
<%-- JSP Comment --%>
```

HTML comments of the form

```
<!-- HTML Comment -->
```

are passed through to the resultant HTML normally.

10.2 JSP Expressions

A JSP expression is used to insert values directly into the output. It has the following form:

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

The expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at run time (when the page is requested) and thus has full access to information about the request. For example, the following shows the date/time that the page was requested:

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

Predefined Variables

To simplify these expressions, you can use a number of predefined variables. These implicit objects are discussed in more detail in Section 10.5, but for the purpose of expressions, the most important ones are:

- **`request`**, the `HttpServletRequest`
- **`response`**, the `HttpServletResponse`

- **`session`**, the `HttpSession` associated with the request (unless disabled with the `session` attribute of the `page` directive — see Section 11.4)
- **`out`**, the `PrintWriter` (a buffered version called `JspWriter`) used to send output to the client

Here is an example:

```
Your hostname: <%= request.getRemoteHost() %>
```

XML Syntax for Expressions

XML authors can use the following alternative syntax for JSP expressions:

```
<jsp:expression>
Java Expression
</jsp:expression>
```

Note that XML elements, unlike HTML ones, are case sensitive, so be sure to use `jsp:expression` in lower case.

Using Expressions as Attribute Values

As we will see later, JSP includes a number of elements that use XML syntax to specify various parameters. For example, the following example passes `"Marty"` to the `setFirstName` method of the object bound to the `author` variable. Don't worry if you don't understand the details of this code; it is discussed in detail in Chapter 13 (Using JavaBeans with JSP). My purpose here is simply to point out the use of the `name`, `property`, and `value` attributes.

```
<jsp:setProperty name="author"
                 property="firstName"
                 value="Marty" />
```

Most attributes require the value to be a fixed string enclosed in either single or double quotes, as in the example above. A few attributes, however, permit you to use a JSP expression that is computed at request time. The `value` attribute of `jsp:setProperty` is one such example, so the following code is perfectly legal:

```
<jsp:setProperty name="user"
                 property="id"
                 value='<%= "UserID" + Math.random() %>' />
```

Table 10.1 lists the attributes that permit a request-time value as in this example.

Table 10.1 Attributes That Permit JSP Expressions

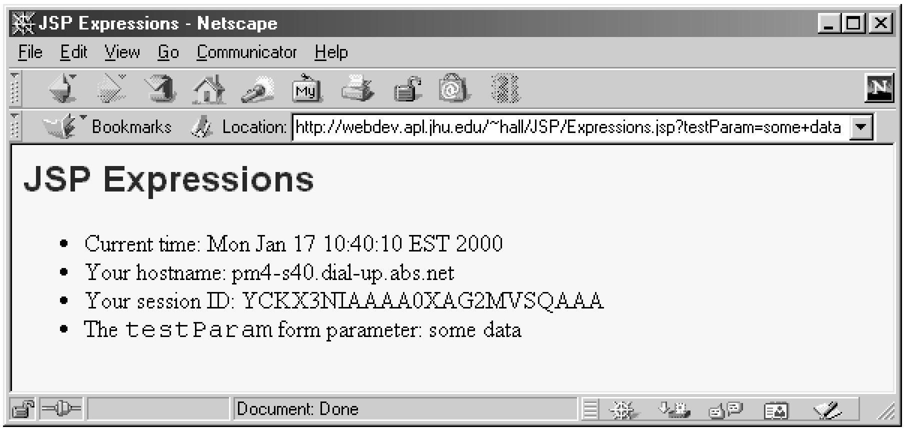
Element Name	*Attribute Name(s)*
`jsp:setProperty` (see Section 13.3, "Setting Bean Properties")	`name` `value`
`jsp:include` (see Chapter 12, "Including Files and Applets in JSP Documents")	`page`
`jsp:forward` (see Chapter 15, "Integrating Servlets and JSP")	`page`
`jsp:param` (see Chapter 12, "Including Files and Applets in JSP Documents")	`value`

Example

Listing 10.1 gives an example JSP page; Figure 10–1 shows the result. Notice that I included `META` tags and a style sheet link in the `HEAD` section of the HTML page. It is good practice to include these elements, but there are two reasons why they are often omitted from pages generated by normal servlets. First, with servlets, it is tedious to generate the required `println` statements. With JSP, however, the format is simpler and you can make use of the code reuse options in your usual HTML building tool. Second, servlets cannot use the simplest form of relative URLs (ones that refer to files in the same directory as the current page) since the servlet directories are not mapped to URLs in the same manner as are URLs for normal Web pages. JSP pages, on the other hand, are installed in the normal Web page hierarchy on the server, and relative URLs are resolved properly. Thus, style sheets and JSP pages can be kept together in the same directory. The source code for the style sheet, like all code shown or referenced in the book, can be downloaded from `http://www.coreservlets.com/`.

Listing 10.1 `Expressions.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
      CONTENT="JSP,expressions,JavaServer,Pages,servlets">
<META NAME="description"
      CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY>
</HTML>
```


JSP Expressions - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://webdev.apl.jhu.edu/~hall/JSP/Expressions.jsp?testParam=some+data
Document: Done

JSP Expressions

- Current time: Mon Jan 17 10:40:10 EST 2000
- Your hostname: pm4-s40.dial-up.abs.net
- Your session ID: YCKX3NIAAAA0XAG2MVSQAAA
- The `testParam` form parameter: some data

Figure 10–1 Typical result of `Expressions.jsp`.

10.3 JSP Scriptlets

If you want to do something more complex than insert a simple expression, JSP scriptlets let you insert arbitrary code into the servlet's `_jspService` method (which is called by `service`). Scriptlets have the following form:

```
<% Java Code %>
```

Scriptlets have access to the same automatically defined variables as expressions (`request`, `response`, `session`, `out`, etc.; see Section 10.5). So, for example, if you want output to appear in the resultant page, you would use the `out` variable, as in the following example.

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
```

In this particular instance, you could have accomplished the same effect more easily by using the following JSP expression:

```
Attached GET data: <%= request.getQueryString() %>
```

In general, however, scriptlets can perform a number of tasks that cannot be accomplished with expressions alone. These tasks include setting response headers and status codes, invoking side effects such as writing to the server log or updating a database, or executing code that contains loops, conditionals, or other complex constructs. For instance, the following snippet specifies that the current page is sent to the client as plain text, not as HTML (which is the default).

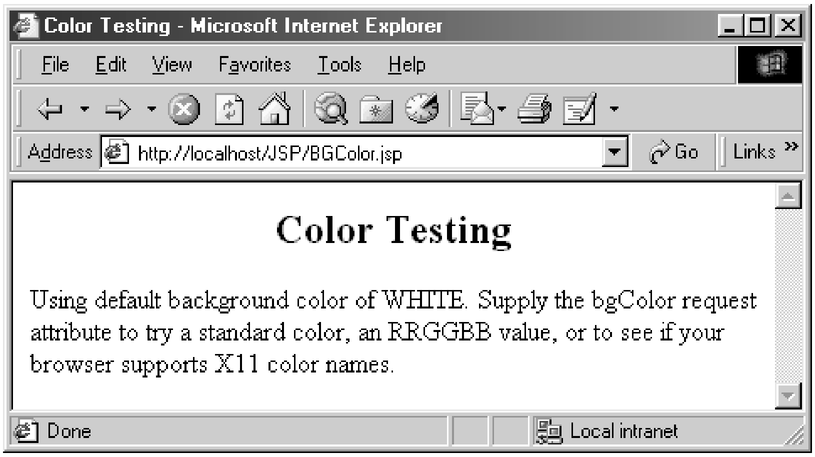
```
<% response.setContentType("text/plain"); %>
```

It is important to note that you can set response headers or status codes at various places within a JSP page, even though this capability appears to violate the rule that this type of response data needs to be specified before any document content is sent to the client. Setting headers and status codes is permitted because servlets that result from JSP pages use a special type of `PrintWriter` (of the more specific class `JspWriter`) that buffers the document before sending it. This buffering behavior can be changed, however; see Section 11.6 for a discussion of the `autoflush` attribute of the `page` directive.

As an example of executing code that is too complex for a JSP expression, Listing 10.2 presents a JSP page that uses the `bgColor` request parameter to set the background color of the page. Some results are shown in Figures 10–2, 10–3, and 10–4.

Listing 10.2 `BGColor.jsp`

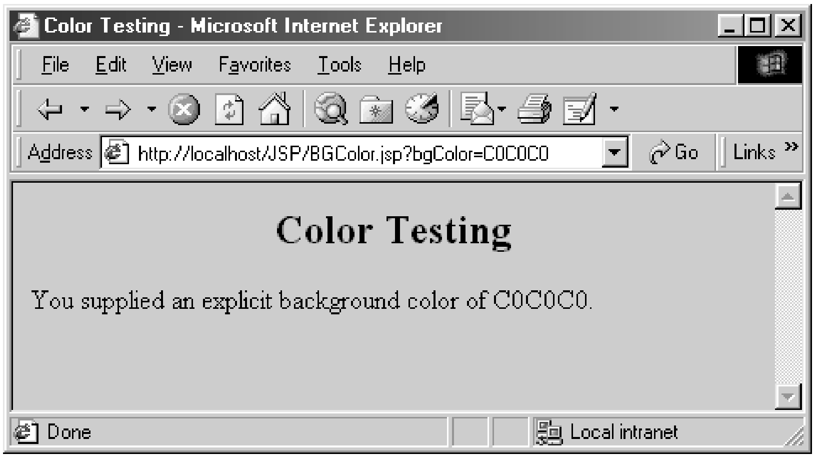
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>

<%
String bgColor = request.getParameter("bgColor");
boolean hasExplicitColor;
if (bgColor != null) {
  hasExplicitColor = true;
} else {
  hasExplicitColor = false;
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Color Testing</H2>

<%
if (hasExplicitColor) {
  out.println("You supplied an explicit background color of " +
              bgColor + ".");
} else {
  out.println("Using default background color of WHITE. " +
              "Supply the bgColor request attribute to try " +
              "a standard color, an RRGGBB value, or to see " +
              "if your browser supports X11 color names.");
}
%>

</BODY>
</HTML>
```


Color Testing - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/JSP/BGColor.jsp
Go
Links
Done
Local intranet

Color Testing

Using default background color of WHITE. Supply the bgColor request attribute to try a standard color, an RRGGBB value, or to see if your browser supports X11 color names.

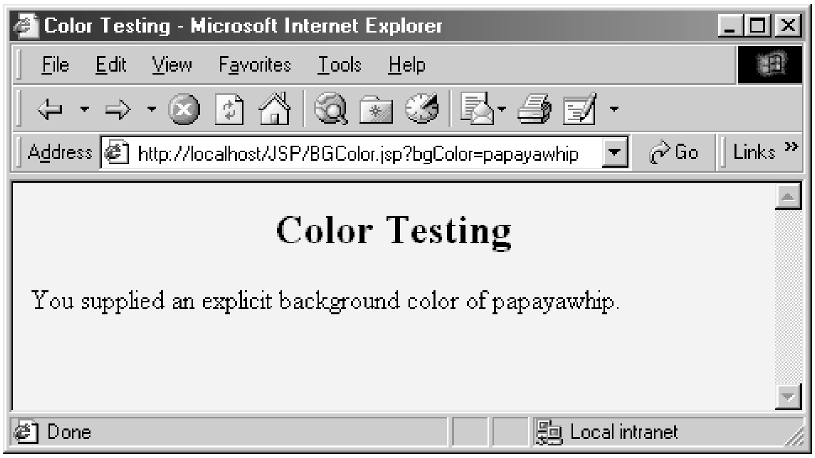
Figure 10–2 Default result of `BGColor.jsp`.


Color Testing - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/JSP/BGColor.jsp?bgColor=C0C0C0
Go
Links
Done
Local intranet

Color Testing

You supplied an explicit background color of C0C0C0.

Figure 10–3 Result of `BGColor.jsp` when accessed with a `bgColor` parameter having the RGB value `C0C0C0`.


Color Testing - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/JSP/BGColor.jsp?bgColor=papayawhip
Go
Links
Done
Local intranet

Color Testing

You supplied an explicit background color of papayawhip.

Figure 10–4 Result of `BGColor.jsp` when accessed with a `bgColor` parameter having the X11 color value `papayawhip`.

Using Scriptlets to Make Parts of the JSP File Conditional

Another use of scriptlets is to conditionally include standard HTML and JSP constructs. The key to this approach is the fact that code inside a scriptlet gets inserted into the resultant servlet's `_jspService` method (called by `service`) *exactly* as written, and any static HTML (template text) before or after a scriptlet gets converted to `print` statements. This means that scriptlets need not contain complete Java statements, and blocks left open can affect the static HTML or JSP outside of the scriptlets. For example, consider the following JSP fragment containing mixed template text and scriptlets.

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

When converted to a servlet by the JSP engine, this fragment will result in something similar to the following.

```
if (Math.random() < 0.5) {
  out.println("Have a <B>nice</B> day!");
} else {
  out.println("Have a <B>lousy</B> day!");
}
```

Special Scriptlet Syntax

There are two special constructs you should take note of. First, if you want to use the characters `%>` inside a scriptlet, enter `%\>` instead. Second, the XML equivalent of `<%` *`Code`* `%>` is

```
<jsp:scriptlet>
Code
</jsp:scriptlet>
```

The two forms are treated identically by JSP engines.

10.4 JSP Declarations

A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (*outside* of the `_jspService` method that is called by `service` to process the request). A declaration has the following form:

```
<%! Java Code %>
```

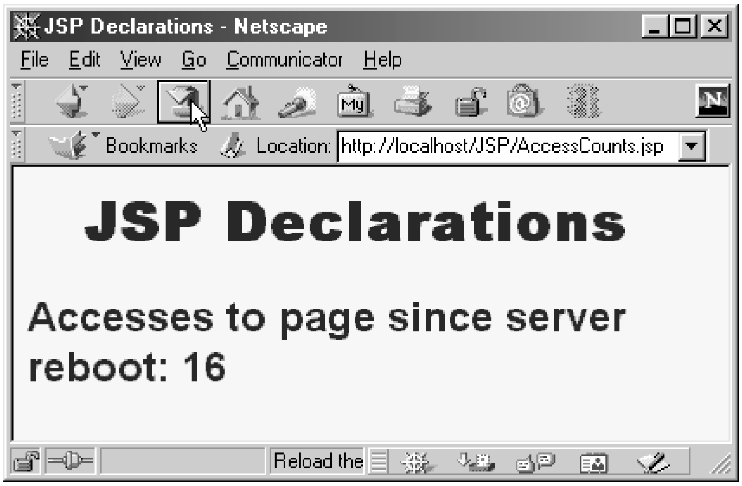
Since declarations do not generate any output, they are normally used in conjunction with JSP expressions or scriptlets. For example, here is a JSP fragment that prints the number of times the current page has been requested since the server was booted (or the servlet class was changed and reloaded). Recall that multiple client requests to the same servlet result only in multiple threads calling the `service` method of a single servlet instance. They do *not* result in the creation of multiple servlet instances except possibly when the servlet implements `SingleThreadModel`. For a discussion of `SingleThreadModel`, see Section 2.6 (The Servlet Life Cycle) and Section 11.3 (The isThreadSafe Attribute). Thus, instance variables (fields) of a servlet are shared by multiple requests and `accessCount` does not have to be declared `static` below.

```
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```

Listing 10.3 shows the full JSP page; Figure 10–5 shows a representative result.

Listing 10.3 `AccessCounts.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Declarations</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
      CONTENT="JSP,declarations,JavaServer,Pages,servlets">
<META NAME="description"
      CONTENT="A quick example of JSP declarations.">
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<H1>JSP Declarations</H1>

<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>

</BODY>
</HTML>
```


JSP Declarations - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/JSP/AccessCounts.jsp
Reload the

JSP Declarations

Accesses to page since server reboot: 16

Figure 10–5 Visiting `AccessCounts.jsp` after it has been requested 15 times by the same or different clients.

Special Declaration Syntax

As with scriptlets, if you want to use the characters `%>`, enter `%\>` instead. Finally, note that the XML equivalent of `<%! Code %>` is

```
<jsp:declaration>
Code
</jsp:declaration>
```

10.5 Predefined Variables

To simplify code in JSP expressions and scriptlets, you are supplied with eight automatically defined variables, sometimes called *implicit objects*. Since JSP declarations (see Section 10.4) result in code that appears outside of the `_jspService` method, these variables are not accessible in declarations. The available variables are `request`, `response`, `out`, `session`, `application`, `config`, `pageContext`, and `page`. Details for each are given below.

request
This variable is the `HttpServletRequest` associated with the request; it gives you access to the request parameters, the request type (e.g., `GET` or `POST`), and the incoming HTTP headers (e.g., cookies). Strictly speaking, if the protocol in the request is something other than HTTP, `request` is allowed to be a subclass of `ServletRequest` other than `HttpServletRequest`. However, few, if any, JSP servers currently support non-HTTP servlets.

response
This variable is the `HttpServletResponse` associated with the response to the client. Note that since the output stream (see `out`) is normally buffered, it is legal to set HTTP status codes and response headers in JSP pages, even though the setting of headers or status codes is not permitted in servlets once any output has been sent to the client.

out
This is the `PrintWriter` used to send output to the client. However, to make the `response` object useful, this is a buffered version of `PrintWriter` called `JspWriter`. You can adjust the buffer size through use of the `buffer` attribute of the `page` directive (see Section 11.5). Also note

that `out` is used almost exclusively in scriptlets, since JSP expressions are automatically placed in the output stream and thus rarely need to refer to `out` explicitly.

session

This variable is the `HttpSession` object associated with the request. Recall that sessions are created automatically, so this variable is bound even if there is no incoming session reference. The one exception is if you use the `session` attribute of the `page` directive (see Section 11.4) to turn sessions off. In that case, attempts to reference the `session` variable cause errors at the time the JSP page is translated into a servlet.

application

This variable is the `ServletContext` as obtained via `getServletConfig().getContext()`. Servlets and JSP pages can store persistent data in the `ServletContext` object rather than in instance variables. `ServletContext` has `setAttribute` and `getAttribute` methods that let you store arbitrary data associated with specified keys. The difference between storing data in instance variables and storing it in the `ServletContext` is that the `ServletContext` is shared by all servlets in the servlet engine (or in the Web application, if your server supports such a capability). For more information on the use of the `ServletContext`, see Section 13.4 (Sharing Beans) and Chapter 15 (Integrating Servlets and JSP).

config

This variable is the `ServletConfig` object for this page.

pageContext

JSP introduced a new class called `PageContext` to give a single point of access to many of the page attributes and to provide a convenient place to store shared data. The `pageContext` variable stores the value of the `PageContext` object associated with the current page. See Section 13.4 (Sharing Beans) for a discussion of its use.

page

This variable is simply a synonym for `this` and is not very useful in the Java programming language. It was created as a place holder for the time when the scripting language could be something other than Java.