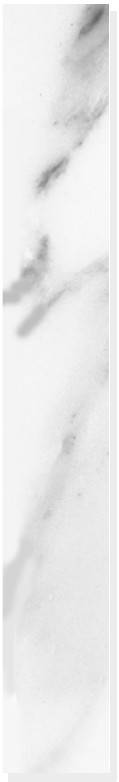
# THE JSP PAGE DIRECTIVE: STRUCTURING GENERATED SERVLETS



**Topics in This Chapter**

- The purpose of the `page` directive
- Designating which classes are imported
- Using custom classes
- Specifying the MIME type of the page
- Generating Excel documents
- Controlling threading behavior
- Participating in sessions
- Setting the size and behavior of the output buffer
- Designating pages to process JSP errors
- XML-compatible syntax for directives

# Chapter 11

A JSP *directive* affects the overall structure of the servlet that results from the JSP page. The following templates show the two possible forms for directives. Single quotes can be substituted for the double quotes around the attribute values, but the quotation marks cannot be omitted altogether. To obtain quote marks within an attribute value, precede them with a back slash, using `\'` for ' and `\"` for ".

```
<%@ directive attribute="value" %>

<%@ directive attribute1="value1"
              attribute2="value2"
              ...
              attributeN="valueN" %>
```

In JSP, there are three types of directives: `page`, `include`, and `taglib`. The `page` directive lets you control the structure of the servlet by importing classes, customizing the servlet superclass, setting the content type, and the like. A `page` directive can be placed anywhere within the document; its use is the topic of this chapter. The second directive, `include`, lets you insert a file into the servlet class at the time the JSP file is translated into a servlet. An `include` directive should be placed in the document at the point at which you want the file to be inserted; it is discussed in Chapter 12 (Including Files and Applets in JSP Documents) for inserting files into JSP pages. JSP 1.1 introduces a third directive, `taglib`, which can be used to define custom

markup tags; it is discussed in Chapter 14 (Creating Custom JSP Tag Libraries).

The `page` directive lets you define one or more of the following case-sensitive attributes: `import`, `contentType`, `isThreadSafe`, `session`, `buffer`, `autoflush`, `extends`, `info`, `errorPage`, `isErrorPage`, and `language`. These attributes are explained in the following sections.

# 11.1 The import Attribute

The `import` attribute of the `page` directive lets you specify the packages that should be imported by the servlet into which the JSP page gets translated. If you don't explicitly specify any classes to import, the servlet imports `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`, and possibly some number of server-specific entries. Never write JSP code that relies on any server-specific classes being imported automatically. Use of the `import` attribute takes one of the following two forms:

```
<%@ page import="package.class" %>
<%@ page import="package.class1,...,package.classN" %>
```

For example, the following directive signifies that all classes in the `java.util` package should be available to use without explicit package identifiers.

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

The `import` attribute is the only `page` attribute that is allowed to appear multiple times within the same document. Although `page` directives can appear anywhere within the document, it is traditional to place `import` statements either near the top of the document or just before the first place that the referenced package is used.

## *Directories for Custom Classes*

If you import classes that are not in any of the standard `java` or `javax.servlet` packages, you need to be sure that those classes have been properly installed on your server. In particular, most servers that support automatic servlet reloading do not permit classes that are in the auto-reloading directories to be referenced by JSP pages. The particular locations used for servlet classes vary from server to server, so you should consult your server's documentation for definitive guidance. The locations used by Apache Tomcat 3.0, the JSWDK 1.0.1, and the Java Web Server 2.0 are summarized

in Table 11.1. All three of these servers also make use of JAR files in the `lib` subdirectory, and in all three cases you must restart the server whenever you change files in this directory.

Table 11.1 Class Installation Directories

| *Server* | *Location Relative to Installation Directory* | *Use* | *Automatically Reloaded When Class Changes?* | *Available from JSP Pages?* |
|---|---|---|---|---|
| Tomcat 3.0 | `webpages/WEB-INF/classes` | Standard location for servlet classes | No | Yes |
| Tomcat 3.0 | `classes` | Alternative location for servlet classes | No | Yes |
| JSWDK 1.0.1 | `webpages/WEB-INF/servlets` | Standard location for servlet classes | No | Yes |
| JSWDK 1.0.1 | `classes` | Alternative location for servlet classes | No | Yes |
| Java Web Server 2.0 | `servlets` | Location for frequently changing servlet classes | **Yes** | **No** |
| Java Web Server 2.0 | `classes` | Location for infrequently changing servlet classes | No | Yes |

## Example

Listing 11.1 presents a page that uses three classes not in the standard JSP import list: `java.util.Date`, `coreservlets.ServletUtilities` (see Listing 8.3), and `coreservlets.LongLivedCookie` (see Listing 8.4). To simplify references to these classes, the JSP page uses

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

Figures 11–1 and 11–2 show some typical results.

## Listing 11.1 ImportAttribute.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>The import Attribute</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<H2>The import Attribute</H2>
<%-- JSP page directive --%>
<%@ page import="java.util.*,coreservlets.*" %>

<%-- JSP Declaration (see Section 10.4) --%>
<%!
private String randomID() {
  int num = (int)(Math.random()*10000000.0);
  return("id" + num);
}

private final String NO_VALUE = "<I>No Value</I>";
%>

<%-- JSP Scriptlet (see Section 10.3) --%>
<%
Cookie[] cookies = request.getCookies();
String oldID =
  ServletUtilities.getCookieValue(cookies, "userID", NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) {
  newID = randomID();
} else {
  newID = oldID;
}
LongLivedCookie cookie = new LongLivedCookie("userID", newID);
response.addCookie(cookie);
%>

<%-- JSP Expressions (see Section 10.2) --%>
This page was accessed at <%= new Date() %> with a userID
cookie of <%= oldID %>.

</BODY>
</HTML>
```

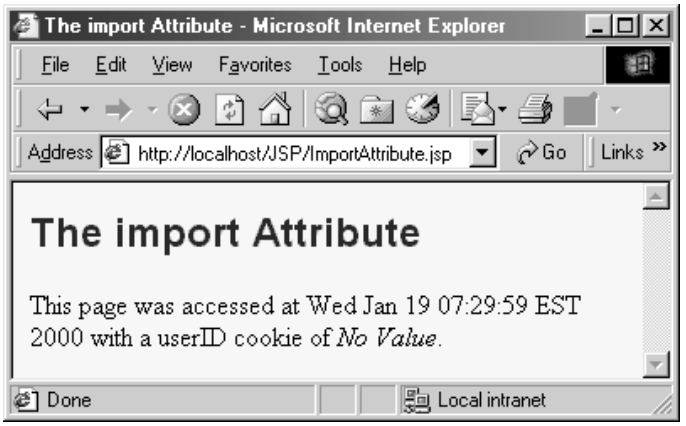



***Figure 11–1*** `ImportAttribute.jsp` when first accessed.

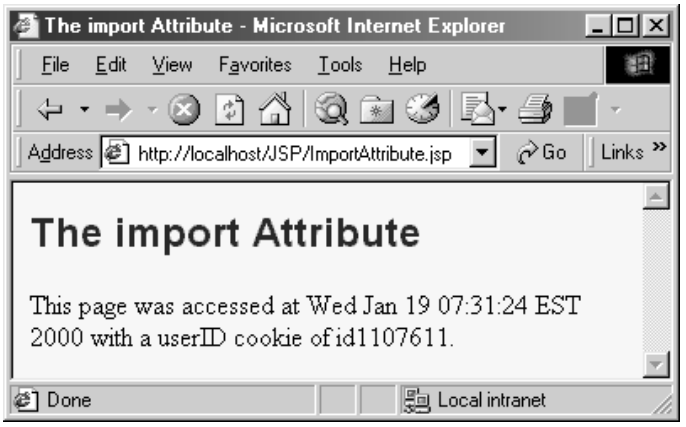



***Figure 11–2*** `ImportAttribute.jsp` when accessed in a subsequent visit.

# 11.2 The contentType Attribute

The `contentType` attribute sets the `Content-Type` response header, indicating the MIME type of the document being sent to the client. For more information on MIME types, see Table 7.1 (Common MIME Types) in Section 7.2 (HTTP 1.1 Response Headers and Their Meaning).

Use of the `contentType` attribute takes one of the following two forms:

```
<%@ page contentType="MIME-Type" %>
<%@ page contentType="MIME-Type; charset=Character-Set" %>
```

For example, the directive

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

has the same effect as the scriptlet

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

Unlike regular servlets, where the default MIME type is `text/plain`, the default for JSP pages is `text/html` (with a default character set of `ISO-8859-1`).

## Generating Plain Text Documents

Listing 11.2 shows a document that appears to be HTML but has a `contentType` of `text/plain`. Strictly speaking, browsers are supposed to display the raw HTML content in such a case, as shown in Netscape in Figure 11–3. Internet Explorer, however, interprets the document as though it were of type `text/html`, as shown in Figure 11–4.

**Listing 11.2** `ContentType.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>The contentType Attribute</TITLE>
</HEAD>
<BODY>

<H2>The contentType Attribute</H2>
<%@ page contentType="text/plain" %>
This should be rendered as plain text,
<B>not</B> as HTML.

</BODY>
</HTML>
```

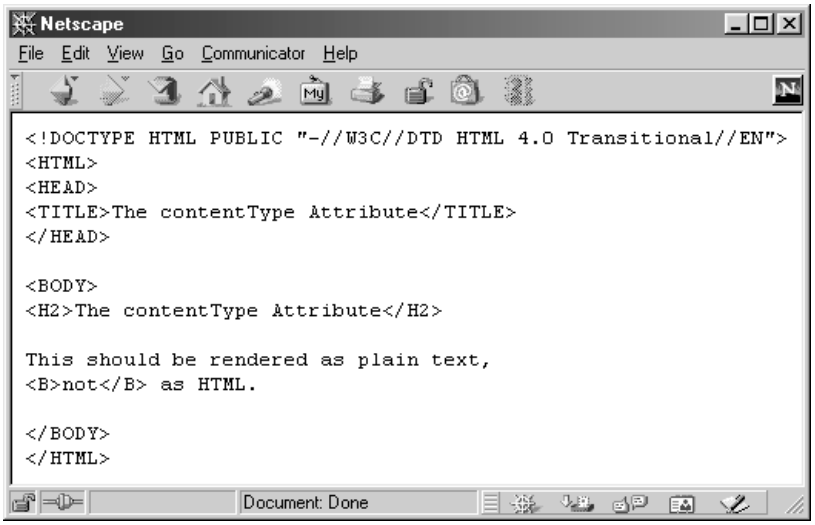



***Figure 11–3*** For plain text documents, Netscape does not try to interpret HTML tags.

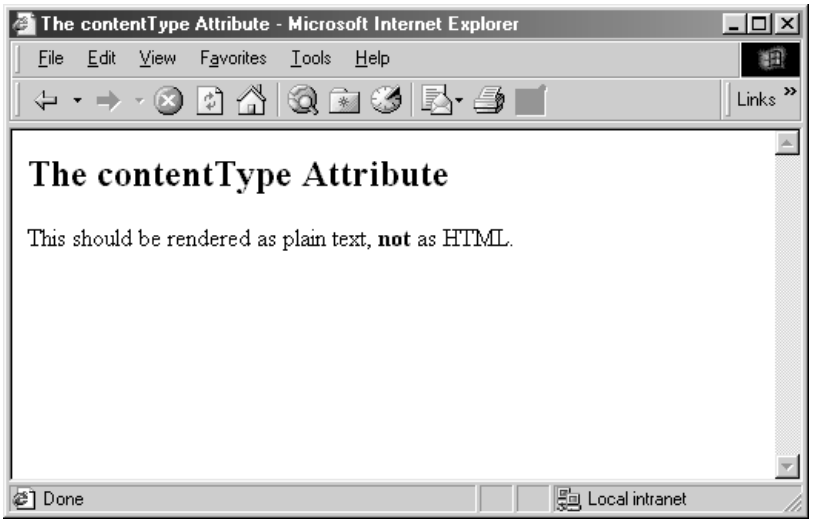



***Figure 11–4*** Internet Explorer interprets HTML tags in plain text documents.

## Generating Excel Spreadsheets

You can create simple Microsoft Excel spreadsheets by specifying `application/vnd.ms-excel` as the content type and then formatting the spreadsheet entries in one of two ways.

One way to format the content is to put rows on separate lines of the document and to use tabs between each of the columns. Listing 11.3 shows a simple example, and Figures 11–5 and 11–6 show the results of loading the page in Netscape on a system with Excel installed. Of course, in a real application, the entries would probably be generated dynamically, perhaps by a JSP expression or scriptlet that refers to database values that were accessed with JDBC (see Chapter 18).

**Listing 11.3** `Excel.jsp`

```
<%@ page contentType="application/vnd.ms-excel" %>
<%-- Note that there are tabs, not spaces, between columns. --%>
1997   1998   1999   2000   2001 (Anticipated)
12.3   13.4   14.5   15.6   16.7
```

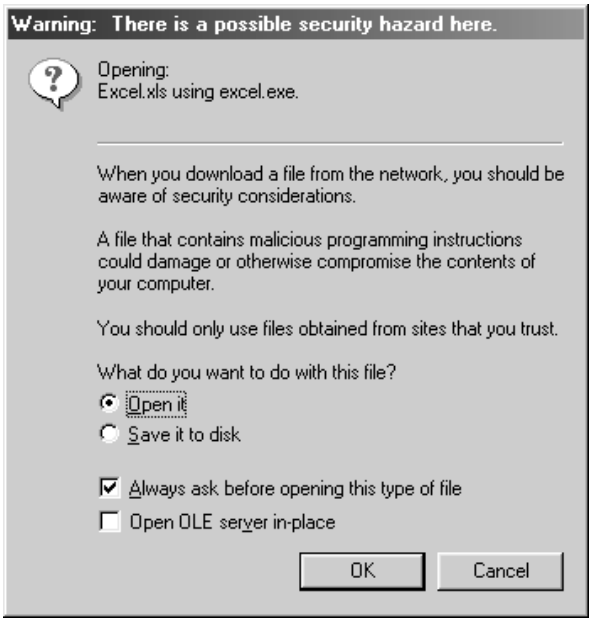



***Figure 11–5*** With the default browser settings, Netscape prompts you before allowing Excel content.

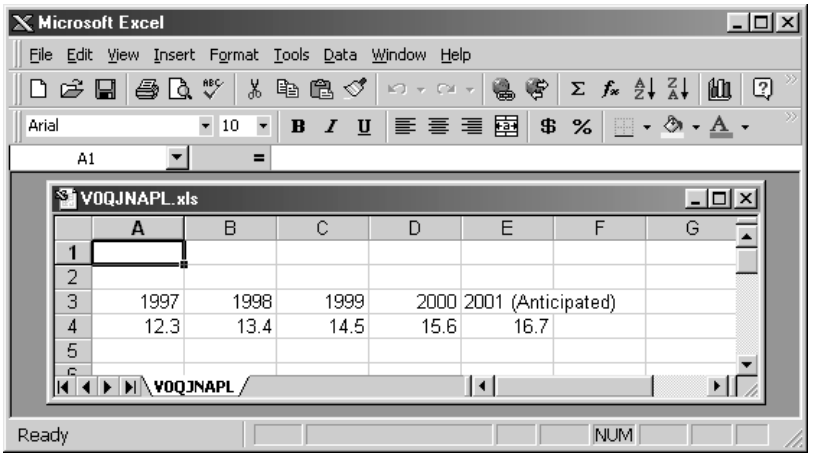



***Figure 11–6*** Result of `Excel.jsp` on system that has Excel installed.

A second way to format Excel content is to use a normal HTML table, which recent versions of Excel can interpret properly as long as the page is marked with the proper MIME type. This capability suggests a simple method of returning either HTML or Excel content, depending on which the user prefers: just use an HTML table and set the content type to `application/vnd.ms-excel` only if the user requests the results in Excel. Unfortunately, this approach brings to light a small deficiency in the `page` directive: attribute values cannot be computed at run time, nor can `page` directives be conditionally inserted as can template text. So, the following attempt results in Excel content regardless of the result of the `checkUserRequest` method.

```
<% boolean usingExcel = checkUserRequest(request); %>
<% if (usingExcel) { %>
<%@ page contentType="application/vnd.ms-excel" %>
<% } %>
```

Fortunately, there is a simple solution to the problem of conditionally setting the content type: just use scriptlets and the normal servlet approach of `response.setContentType`, as in the following snippet:

```
<%
String format = request.getParameter("format");
if ((format != null) && (format.equals("excel"))) {
  response.setContentType("application/vnd.ms-excel");
}
%>
```

Listing 11.4 shows a page that uses this approach; Figures 11–7 and 11–8 show the results in Internet Explorer. Again, in a real application the data would almost certainly be dynamically generated. For example, see Section 18.3 (Some JDBC Utilities) for some very simple methods to create an HTML table (usable in HTML or as an Excel spreadsheet) from a database query.

**Listing 11.4** `ApplesAndOranges.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Comparing Apples and Oranges</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<CENTER>
<H2>Comparing Apples and Oranges</H2>

<%
String format = request.getParameter("format");
if ((format != null) && (format.equals("excel"))) {
  response.setContentType("application/vnd.ms-excel");
}
%>

<TABLE BORDER=1>
  <TR><TH></TH><TH>Apples<TH>Oranges
  <TR><TH>First Quarter<TD>2307<TD>4706
  <TR><TH>Second Quarter<TD>2982<TD>5104
  <TR><TH>Third Quarter<TD>3011<TD>5220
  <TR><TH>Fourth Quarter<TD>3055<TD>5287
</TABLE>

</CENTER>
</BODY>
</HTML>
```

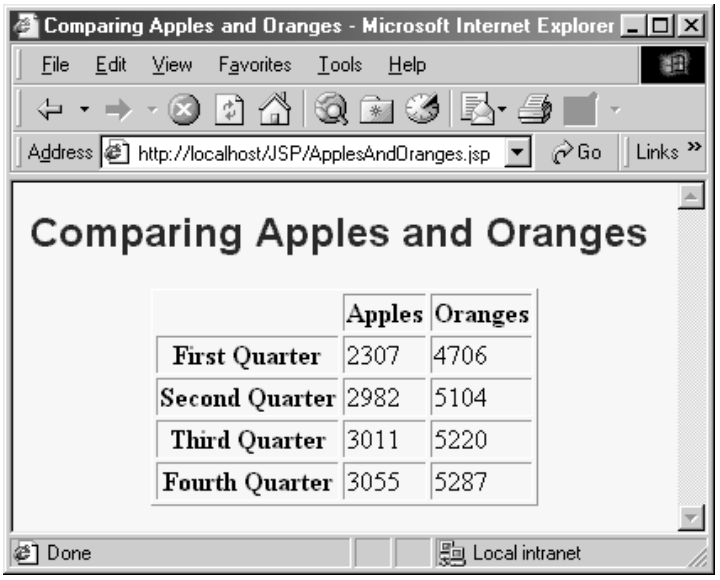



***Figure 11–7*** The default result of `ApplesAndOranges.jsp` is HTML content.

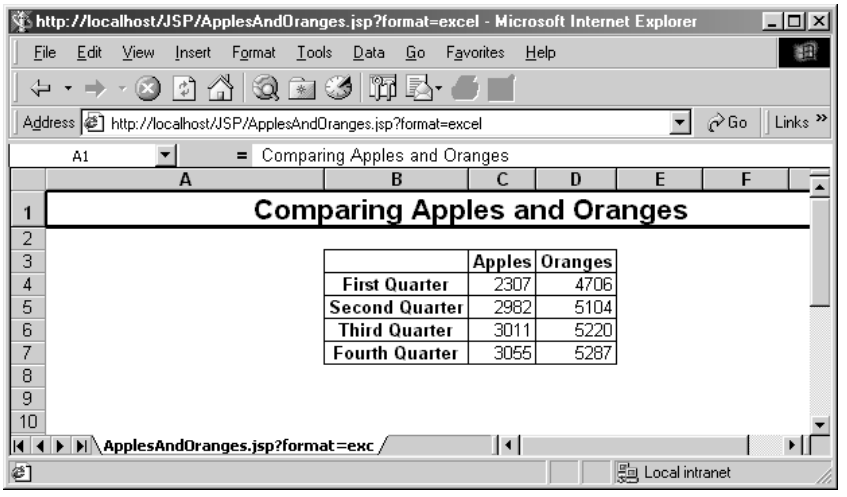



***Figure 11–8*** Specifying `format=excel` for `ApplesAndOranges.jsp` results in Excel content.

# 11.3 The isThreadSafe Attribute

The `isThreadSafe` attribute controls whether or not the servlet that results from the JSP page will implement the `SingleThreadModel` interface. Use of the `isThreadSafe` attribute takes one of the following two forms:

```
<%@ page isThreadSafe="true" %> <%!-- Default --%>
<%@ page isThreadSafe="false" %>
```

With normal servlets, simultaneous user requests result in multiple threads concurrently accessing the `service` method of the same servlet instance. This behavior assumes that the servlet is *thread safe*; that is, that the servlet synchronizes access to data in its fields so that inconsistent values will not result from an unexpected ordering of thread execution. In some cases (such as page access counts), you may not care if two visitors occasionally get the same value, but in other cases (such as user IDs), identical values can spell disaster. For example, the following snippet is not thread safe since a thread could be preempted after reading `idNum` but before updating it, yielding two users with the same user ID.

```
<%! private int idNum = 0; %>
<%
String userID = "userID" + idNum;
out.println("Your ID is " + userID + ".");
idNum = idNum + 1;
%>
```

The code should have used a `synchronized` block. This construct is written

```
synchronized(someObject) { ... }
```

and means that once a thread enters the block of code, no other thread can enter the same block (or any other block marked with the same object reference) until the first thread exits. So, the previous snippet should have been written in the following manner.

```
<%! private int idNum = 0; %>
<%
synchronized(this) {
  String userID = "userID" + idNum;
  out.println("Your ID is " + userID + ".");
  idNum = idNum + 1;
}
%>
```

That's the normal servlet behavior: multiple simultaneous requests are dispatched to multiple threads concurrently accessing the same servlet instance. However, if a servlet implements the `SingleThreadModel` interface, the sys-

tem guarantees that there will not be simultaneous access to the same servlet instance. The system can satisfy this guarantee either by queuing up all requests and passing them to the same servlet instance or by creating a pool of instances, each of which handles a single request at a time.

You use `<%@ page isThreadSafe="false" %>` to indicate that your code is *not* thread safe and thus that the resulting servlet should implement `SingleThreadModel`. (See Section 2.6 (The Servlet Life Cycle.) The default value is `true`, which means that the system assumes you made your code thread safe, and it can consequently use the higher-performance approach of multiple simultaneous threads accessing a single servlet instance. Be careful about using `isThreadSafe="false"` when your servlet has instance variables (fields) that maintain persistent data. In particular, note that servlet engines are permitted (but not required) to create multiple servlet instances in such a case and thus instance variables are not necessarily unique. You could still use `static` fields in such a case, however.

## 11.4 The session Attribute

The `session` attribute controls whether or not the page participates in HTTP sessions. Use of this attribute takes one of the following two forms:

```
<%@ page session="true" %> <%-- Default --%>
<%@ page session="false" %>
```

A value of `true` (the default) indicates that the predefined variable `session` (of type `HttpSession`) should be bound to the existing session if one exists; otherwise, a new session should be created and bound to `session`. A value of `false` means that no sessions will be used automatically and attempts to access the variable `session` will result in errors at the time the JSP page is translated into a servlet.

## 11.5 The buffer Attribute

The `buffer` attribute specifies the size of the buffer used by the `out` variable, which is of type `JspWriter` (a subclass of `PrintWriter`). Use of this attribute takes one of two forms:

```
<%@ page buffer="sizekb" %>
<%@ page buffer="none" %>
```

Servers can use a larger buffer than you specify, but not a smaller one. For example, `<%@ page buffer="32kb" %>` means the document content should be buffered and not sent to the client until at least 32 kilobytes have been accumulated or the page is completed. The default buffer size is server specific, but must be at least 8 kilobytes. Be cautious about turning off buffering; doing so requires JSP entries that set headers or status codes to appear at the top of the file, before any HTML content.

## 11.6 The autoflush Attribute

The `autoflush` attribute controls whether the output buffer should be automatically flushed when it is full or whether an exception should be raised when the buffer overflows. Use of this attribute takes one of the following two forms:

```
<%@ page autoflush="true" %> <%-- Default --%>
<%@ page autoflush="false" %>
```

A value of `false` is illegal when also using `buffer="none"`.

## 11.7 The extends Attribute

The `extends` attribute indicates the superclass of the servlet that will be generated for the JSP page and takes the following form:

```
<%@ page extends="package.class" %>
```

Use this attribute with extreme caution since the server may be using a custom superclass already.

## 11.8 The info Attribute

The `info` attribute defines a string that can be retrieved from the servlet by means of the `getServletInfo` method. Use of `info` takes the following form:

```
<%@ page info="Some Message" %>
```

# 11.9 The errorPage Attribute

The `errorPage` attribute specifies a JSP page that should process any exceptions (i.e., something of type `Throwable`) thrown but not caught in the current page. It is used as follows:

```
<%@ page errorPage="Relative URL" %>
```

The exception thrown will be automatically available to the designated error page by means of the `exception` variable. See Listings 11.5 and 11.6 for examples.

# 11.10 The isErrorPage Attribute

The `isErrorPage` attribute indicates whether or not the current page can act as the error page for another JSP page. Use of `isErrorPage` takes one of the following two forms:

```
<%@ page isErrorPage="true" %>
<%@ page isErrorPage="false" %> <%!-- Default --%>
```

For example, Listing 11.5 shows a JSP page to compute speed based upon distance and time parameters. The page neglects to check if the input parameters are missing or malformed, so an error could easily occur at run time. However, the page designated `SpeedErrors.jsp` (Listing 11.6) as the page to handle errors that occur in `ComputeSpeed.jsp`, so the user does not receive the typical terse JSP error messages. Figures 11–9 and 11–10 show results when good and bad input parameters are received, respectively.

**Listing 11.5** `ComputeSpeed.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Computing Speed</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<%@ page errorPage="SpeedErrors.jsp" %>
```

## Listing 11.5 ComputeSpeed.jsp (continued)

```
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Computing Speed</TABLE>

<%!
// Note lack of try/catch for NumberFormatException if
// value is null or malformed.

private double toDouble(String value) {
  return(Double.valueOf(value).doubleValue());
}
%>

<%
double furlongs = toDouble(request.getParameter("furlongs"));
double fortnights = toDouble(request.getParameter("fortnights"));
double speed = furlongs/fortnights;
%>

<UL>
  <LI>Distance: <%= furlongs %> furlongs.
  <LI>Time: <%= fortnights %> fortnights.
  <LI>Speed: <%= speed %> furlongs per fortnight.
</UL>

</BODY>
</HTML>
```

## Listing 11.6 SpeedErrors.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Error Computing Speed</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>

<%@ page isErrorPage="true" %>

<TABLE BORDER=5 ALIGN="CENTER">
```

Listing 11.6 SpeedErrors.jsp (continued)

```
  <TR><TH CLASS="TITLE">
      Error Computing Speed</TABLE>
<P>
ComputeSpeed.jsp reported the following error:
<I><%= exception %></I>. This problem occurred in the
following place:
<PRE>
<% exception.printStackTrace(new PrintWriter(out)); %>
</PRE>

</BODY>
</HTML>
```

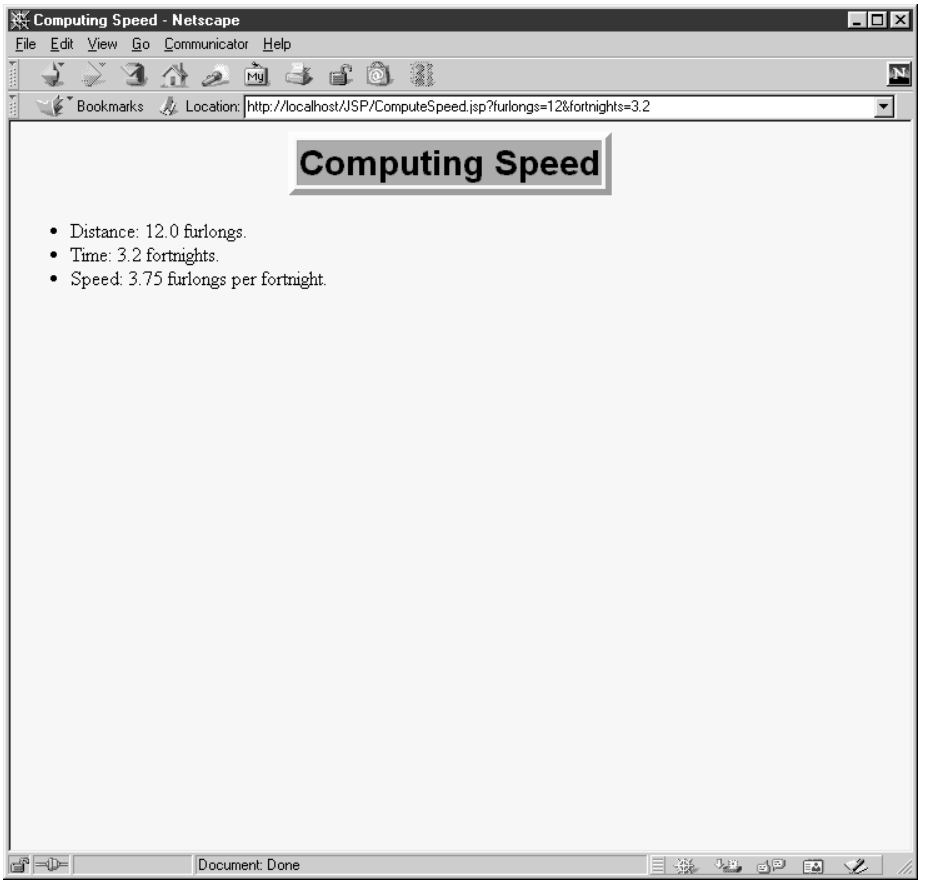



***Figure 11–9*** ComputeSpeed.jsp when it receives legal values.

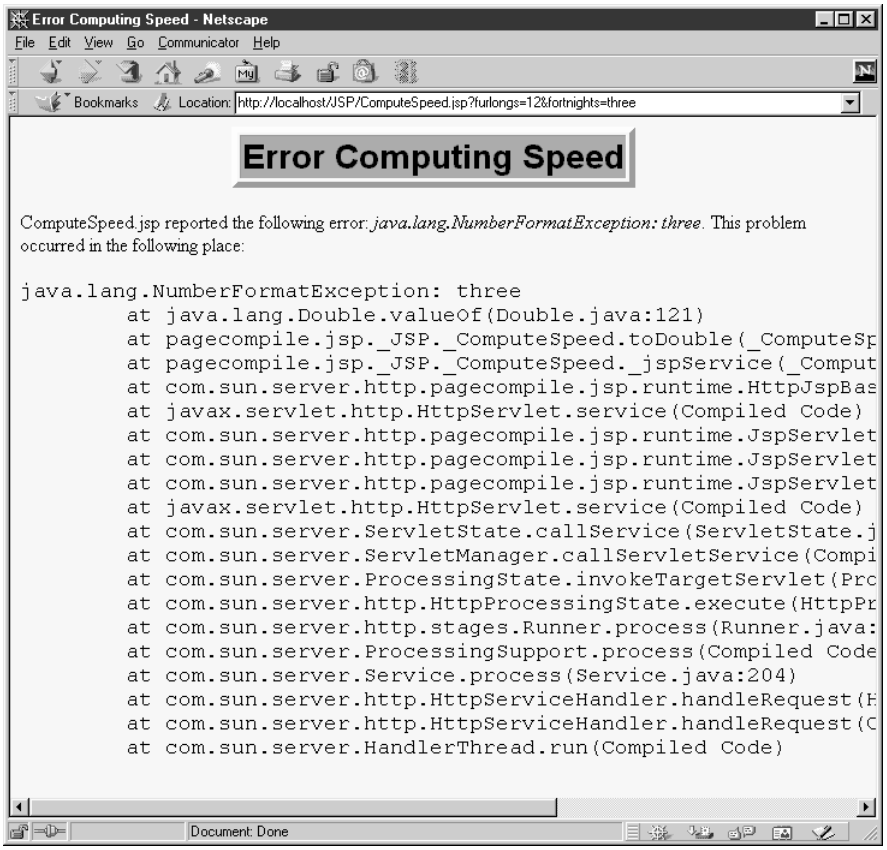



***Figure 11–10*** `ComputeSpeed.jsp` when it receives illegal values.

# 11.11 The language Attribute

At some point, the `language` attribute is intended to specify the underlying programming language being used, as below.

```
<%@ page language="cobol" %>
```

For now, don't bother with this attribute since `java` is both the default and the only legal choice.

# 11.12 XML Syntax for Directives

JSP permits you to use an alternative XML-compatible syntax for directives. These constructs take the following form:

```
<jsp:directive.directiveType attribute="value" />
```

For example, the XML equivalent of

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

is

```
<jsp:directive.page import="java.util.*" />
```