37

JavaServer Pages (JSP): Bonus for Java Developers

Objectives

- To be able to create and deploy JavaServer Pages.
- To use JSP's implicit objects and scriptlets to create dynamic Web pages.
- To specify global JSP information with directives.
- To use actions to manipulate JavaBeans in a JSP, to include resources dynamically and to forward requests to other JSPs.

A tomato does not communicate with a tomato, we believe. We could be wrong.
Gustav Eckstein

A donkey appears to me like a horse translated into Dutch.
Georg Christoph Licthtenberg

Talent is a question of quantity. Talent does not write one page: it writes three hundred.
Jules Renard

Every action must be due to one or other of seven causes: chance, nature, compulsion, habit, reasoning, anger, or appetite.
Aristotle

Outline

37.1 Introduction[1]

Our discussion of client–server networking continues in this chapter with **JavaServer Pages (JSP)**—an extension of servlet technology. JavaServer Pages simplify the delivery of dynamic Web content. They enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting. JavaServer Page programmers can reuse JavaBeans and create custom tag libraries that encapsulate complex, dynamic functionality. Custom-tag libraries even enable Web-page designers who are not familiar with Java to enhance Web pages with powerful dynamic content and processing capabilities.

In addition to the types for programming servlets (Chapter 36), classes and interfaces specific to JavaServer Pages programming are located in packages **`javax.servlet.jsp`** and **`javax.servlet.jsp.tagext`**. We discuss many of these classes and interfaces throughout this chapter as we present JavaServer Pages fundamentals. For a complete description of JavaServer Pages, see the JavaServer Pages 1.2 specification, which can be downloaded from `java.sun.com/products/jsp/download.html`. We also include other JSP resources in Section 37.9. [*Note:* The source code, images and the examples in this chapter can be found on the CD that accompanies this book and at `www.deitel.com`.]

1. We include this chapter as a bonus for Java developers who also are familiar with Java Database Connectivity (JDBC). Readers interested in learning to program in Java may want to refer to our books *Java How To Program, Fifth Edition* and *Advanced Java 2 Platform How to Program.*

37.2 JavaServer Pages Overview

There are four key components to JSPs: **directives**, **actions**, **scriptlets** and **tag libraries**. Directives are messages to the JSP container that enable the programmer to specify page settings, to include content from other resources and to specify custom tag libraries for use in a JSP. Actions encapsulate functionality in predefined tags that programmers can embed in a JSP. Actions often are performed based on the information sent to the server as part of a particular client request. They also can create Java objects for use in JSP scriptlets. Scriptlets, or **scripting elements**, enable programmers to insert Java code that interacts with components in a JSP (and possibly other Web application components) to perform request processing. Tag libraries are part of the **tag extension mechanism** that enables programmers to create custom tags. Such tags enable programmers to manipulate JSP content. These JSP component types are discussed in detail in subsequent sections.

In some ways, Java Server Pages look like standard XHTML or XML documents. In fact, JSPs normally include XHTML or XML markup. Such markup is known as **fixed-template data** or **fixed-template text**. Fixed-template data often help a programmer decide whether to use a servlet or a JSP. Programmers tend to use JSPs when most of the content sent to the client is fixed template data and only a small portion of the content is generated dynamically with Java code. Programmers typically use servlets when only a small portion of the content sent to the client is fixed-template data. In fact, some servlets do not produce content. Rather, they perform a task on behalf of the client, then invoke other servlets or JSPs to provide a response. Note that in most cases, servlet and JSP technologies are interchangeable. As with servlets, JSPs normally execute as part of a Web server. The server component that executes them often is referred to as the **JSP container**.

Software Engineering Observation 37.1

Literal text in a JSP becomes string literals in the servlet that represents the translated JSP.

When a JSP-enabled server receives the first request for a JSP, the JSP container translates that JSP into a Java servlet that handles the current request and future requests to the JSP. If there are any errors compiling the new servlet, these errors result in **translation-time errors**. The JSP container places the Java statements that implement the JSP's response in method **`_jspService`** at translation time. If the new servlet compiles properly, the JSP container invokes method `_jspService` to process the request. The JSP may respond directly to the request or may invoke other Web application components to assist in processing the request. Any errors that occur during request processing are known as **request-time errors**.

Performance Tip 37.1

Some JSP containers translate JSPs to servlets at installation time. This eliminates the translation overhead for the first client that requests each JSP.

Overall, the request/response mechanism and life cycle of a JSP is the same as that of a servlet. JSPs can define methods **`jspInit`** and **`jspDestroy`** (similar to servlet methods `init` and `destroy`), which the JSP container invokes when initializing a JSP and terminating a JSP, respectively. JSP programmers can define these methods using JSP **declarations**—part of the JSP scripting mechanism.

37.3 First JavaServer Page Example

We begin our introduction to JavaServer Pages with a simple example (Fig. 37.1) in which the current date and time are inserted into a Web page using a JSP expression.

As you can see, most of `clock.jsp` consists of XHTML markup. In cases like this, JSPs are easier to implement than servlets. In a servlet that performs the same task as this JSP, each line of XHTML markup typically is a separate Java statement that outputs the string representing the markup as part of the response to the client. Writing code to output markup can often lead to errors. Most JSP editors provide syntax coloring to help programmers check that their markup follows proper syntax.

Software Engineering Observation 37.2

JavaServer Pages are easier to implement than servlets when the response to a client request consists primarily of markup that remains constant between requests.

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 37.1: clock.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <meta http-equiv = "refresh" content = "60" />

      <title>A Simple JSP Example</title>

      <style type = "text/css">
         .big { font-family: helvetica, arial, sans-serif;
                font-weight: bold;
                font-size: 2em; }
      </style>
   </head>

   <body>
      <p class = "big">Simple JSP Example</p>

      <table style = "border: 6px outset;">
         <tr>
            <td style = "background-color: black;">
               <p class = "big" style = "color: cyan;">

                  <!-- JSP expression to insert date/time -->
                  <%= new java.util.Date() %>

               </p>
            </td>
         </tr>
      </table>
   </body>
```

Fig. 37.1 JSP expression inserting the date and time into a Web page. (Part 1 of 2.)

```
37
38 </html>
```

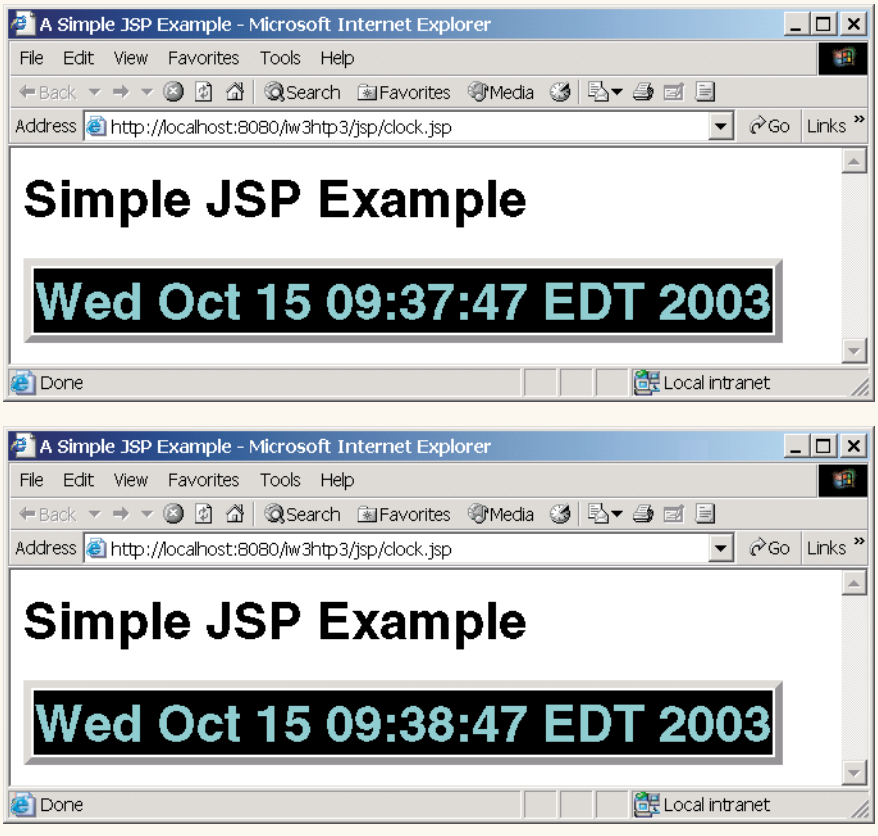


Fig. 37.1 JSP expression inserting the date and time into a Web page. (Part 2 of 2.)

The JSP of Fig. 37.1 generates an XHTML document that displays the current date and time. The key line in this JSP (line 30) is the expression

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

JSP expressions are delimited by <%= and %>. The preceding expression creates a new instance of class `Date` (package `java.util`). When the client requests this JSP, the preceding expression inserts the `String` representation of the date and time in the response to the client. [*Note:* Proper internationalization requires that the JSP return the date in the client locale's format. In this example, the server's locale determines the `String` representation of the `Date`. In Fig. 37.9, `clock2.jsp` demonstrates how to determine the client's locale and uses a `DateFormat` (package `java.text`) object to format the date using that locale.]

Software Engineering Observation 37.3

The JSP container converts the result of every JSP expression into a string that is output as part of the response to the client.

We use the XHTML **`meta` element** in line 10 to set a **refresh interval** of 60 seconds for the document. This causes the browser to request `clock.jsp` every 60 seconds. For each request to `clock.jsp`, the JSP container reevaluates the expression in line 30, creating a new `Date` object with the server's current date and time.

As in Chapter 36, we use Apache Tomcat to test our JSPs in the `iw3htp3` Web application we created previously. For details on creating and configuring the `iw3htp3` Web application, review Section 36.3.1 and Section 36.3.2. To test `clock.jsp`, create a new directory called `jsp` in the `iw3htp3` subdirectory of Tomcat's `webapps` directory. Next, copy `clock.jsp` into the `jsp` directory. Open your Web browser and enter the following URL to test `clock.jsp`:

```
http://localhost:8080/iw3htp3/jsp/clock.jsp
```

When you first invoke the JSP, notice the delay as Tomcat translates the JSP into a servlet and invokes the servlet to respond to your request. [*Note:* It is not necessary to create a directory named `jsp` in a Web application. We use this directory to separate the examples in this chapter from the servlet examples in Chapter 36.]

37.4 Implicit Objects

Implicit objects provide programmers with access to many servlet capabilities in the context of a JavaServer Page. Implicit objects have four scopes: **application**, **page**, **request** and **session**. The JSP and servlet container application owns objects with **application scope**. Any servlet or JSP can manipulate such objects. Objects with **page scope** exist only in the page that defines them. Each page has its own instances of the page-scope implicit objects. Objects with **request scope** exist for the duration of the request. For example, a JSP can partially process a request, then forward the request to another servlet or JSP for further processing. Request-scope objects go out of scope when request processing completes with a response to the client. Objects with **session scope** exist for the client's entire browsing session. Figure 37.2 describes the JSP implicit objects and their scopes. This chapter demonstrates several of these objects.

Implicit object	Description
Application Scope	
`application`	This `javax.servlet.ServletContext` object represents the container in which the JSP executes.
Page Scope	
`config`	This `javax.servlet.ServletConfig` object represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
`exception`	This `java.lang.Throwable` object represents the exception that is passed to the JSP error page. This object is available only in a JSP error page.

Fig. 37.2 JSP implicit objects. (Part 1 of 2.)

Implicit object	Description
`out`	This `javax.servlet.jsp.JspWriter` object writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
`page`	This `java.lang.Object` object represents the `this` reference for the current JSP instance.
`pageContext`	This `javax.servlet.jsp.PageContext` object hides the implementation details of the underlying servlet and JSP container and provides JSP programmers with access to the implicit objects discussed in this table.
`response`	This object represents the response to the client and is normally an instance of a class that implements `HttpServletResponse` (package `javax.servlet.http`). If a protocol other than HTTP is used, this object is an instance of a class that implements `javax.servlet.ServletResponse`.
Request Scope	
`request`	This object represents the client request. The object normally is an instance of a class that implements `HttpServletRequest` (package `javax.servlet.http`). If a protocol other than HTTP is used, this object is an instance of a subclass of `javax.servlet.ServletRequest`.
Session Scope	
`session`	This `javax.servlet.http.HttpSession` object represents the client session information if such a session has been created. This object is available only in pages that participate in a session.

Fig. 37.2 JSP implicit objects. (Part 2 of 2.)

Note that many of the implicit objects extend classes or implement interfaces discussed in Chapter 36. Thus, JSPs can use the same methods that servlets use to interact with such objects, as described in Chapter 36. Most of the examples in this chapter use one or more of the implicit objects in Fig. 37.2.

37.5 Scripting

JavaServer Pages often present dynamically generated content as part of an XHTML document that is sent to the client in response to a request. In some cases, the content is static, but is output only if certain conditions are met during a request (such as providing values in a `form` that submits a request). JSP programmers can insert Java code and logic in a JSP using scripting.

37.5.1 Scripting Components

JSP scripting components include scriptlets, comments, expressions, declarations and escape sequences. This section describes each of these scripting components. Many of these scripting components are demonstrated in Fig. 37.4 at the end of Section 37.5.2.

Scriptlets are blocks of code delimited by **<%** and **%>**. They contain Java statements that the container places in method `_jspService` at translation time.

JSPs support three comment styles: JSP comments, XHTML comments and scripting-language comments. **JSP comments** are delimited by **<%--** and **--%>**. These can be placed throughout a JSP, but not inside scriptlets. **XHTML comments** are delimited with **<!--** and **-->**. These comments can be placed throughout a JSP, but not inside scriptlets. Scripting language comments are currently Java comments, because Java is the only JSP scripting language at the present time. Scriptlets can use Java's end-of-line `//` comments and traditional comments (delimited by `/*` and `*/`). JSP comments and scripting-language comments are ignored and do not appear in the response to a client. When clients view the source code of a JSP response, they will see only the XHTML comments in the source code. The different comment styles are useful for separating comments that the user should be able to see from comments that document logic processed on the server.

Common Programming Error 37.1

Placing a JSP comment or XHTML comment inside a scriptlet is a translation-time syntax error that prevents the JSP from being translated properly.

A JSP expression, delimited by **<%=** and **%>**, contains a Java expression that is evaluated when a client requests the JSP containing the expression. The container converts the result of a JSP expression to a `String` object, then outputs the `String` as part of the response to the client.

Declarations (delimited by **<%!** and **%>**) enable a JSP programmer to define variables and methods for use in a JSP. Variables become instance variables of the servlet class that represents the translated JSP. Similarly, methods become members of the class that represents the translated JSP. Declarations of variables and methods in a JSP use Java syntax. Thus, a variable declaration must end in a semicolon, as in

```
<%! int counter = 0; %>
```

Common Programming Error 37.2

Declaring a variable without using a terminating semicolon is a syntax error.

Software Engineering Observation 37.4

JSPs should not store client state information in instance variables. Rather, JSPs should use the JSP implicit `session` object.

Special characters or character sequences that the JSP container normally uses to delimit JSP code can be included in a JSP as literal characters in scripting elements, fixed template data and attribute values using **escape sequences**. Figure 37.3 shows the literal character or characters and the corresponding escape sequences and discusses where to use the escape sequences.

Literal	Escape sequence	Description
<%	<\%	The character sequence <% normally indicates the beginning of a scriptlet. The <\% escape sequence places the literal characters <% in the response to the client.

Fig. 37.3 JSP escape sequences. (Part 1 of 2.)

Literal	Escape sequence	Description
%>	%\>	The character sequence %> normally indicates the end of a scriptlet. The %\> escape sequence places the literal characters %> in the response to the client.
'	\'	As with string literals in a Java program, the escape sequences for characters ', " and \ allow these characters to appear in attribute values. Remember that the literal text in a JSP becomes string literals in the servlet that represents the translated JSP.
"	\"	
\	\\	

Fig. 37.3 JSP escape sequences. (Part 2 of 2.)

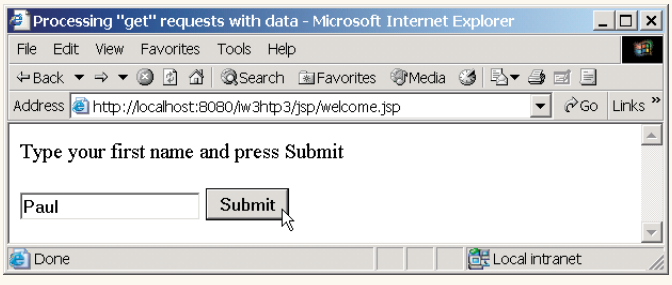
37.5.2 Scripting Example

The JSP of Fig. 37.4 demonstrates responding to `get` requests with basic scripting capabilities. The JSP enables the user to input a first name, then outputs that name in the response. Using scripting, the JSP determines whether a `firstName` parameter was passed as part of the request; if not, the JSP returns an XHTML document containing a `form` through which the user can input a first name. Otherwise, the JSP obtains the `firstName` value and uses it as part of an XHTML document that welcomes the user to JavaServer Pages.

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 37.4: welcome.jsp -->
<!-- JSP that processes a "get" request containing data. -->

<html xmlns = "http://www.w3.org/1999/xhtml">

   <!-- head section of document -->
   <head>
      <title>Processing "get" requests with data</title>
   </head>

   <!-- body section of document -->
   <body>
      <% // begin scriptlet

         String name = request.getParameter( "firstName" );

         if ( name != null ) {

      %> <%-- end scriptlet to insert fixed template data --%>

            <h1>
               Hello <%= name %>, <br />
```

Fig. 37.4 Scripting a JavaServer Page—`welcome.jsp`. (Part 1 of 2.)

```
27                 Welcome to JavaServer Pages!
28              </h1>
29
30        <% // continue scriptlet
31
32           }  // end if
33           else {
34
35        %> <%-- end scriptlet to insert fixed template data --%>
36
37              <form action = "welcome.jsp" method = "get">
38                 <p>Type your first name and press Submit</p>
39
40                 <p><input type = "text" name = "firstName" />
41                    <input type = "submit" value = "Submit" />
42                 </p>
43              </form>
44
45        <% // continue scriptlet
46
47           }  // end else
48
49        %> <%-- end scriptlet --%>
50     </body>
51
52  </html>  <!-- end XHTML document -->
```



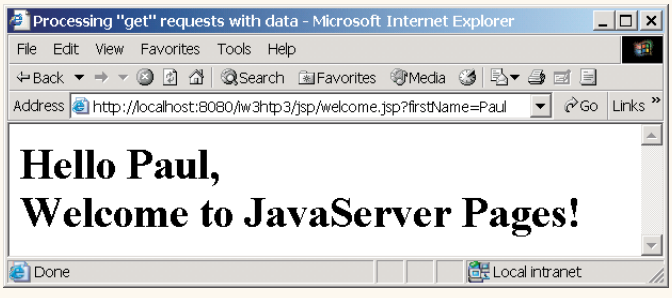


Fig. 37.4 Scripting a JavaServer Page—`welcome.jsp`. (Part 2 of 2.)

Notice that the majority of the code in Fig. 37.4 is XHTML markup (i.e., fixed template data). Throughout the **body** element are several scriptlets (lines 17–23, 30–35 and 45–

49) and a JSP expression (line 26). Note that three comment styles appear in this JSP (in line 5, line17 and line 23).

The scriptlets define an `if`…`else` statement that determines whether the JSP received a value for the first name as part of the request. Line 19 uses method `getParameter` of JSP implicit object `request` (an `HttpServletRequest` object) to obtain the value for parameter `firstName` and assigns the result to variable `name`. Line 21 determines if `name` is not `null`, (i.e., a value for the first name was passed to the JSP as part of the request). If this condition is `true`, the scriptlet terminates temporarily so the fixed template data in lines 25–28 can be output. The JSP expression in line 26 outputs the value of variable `name` (i.e., the first name passed to the JSP as a request parameter. The scriptlet continues in lines 30–35 with the closing brace of the `if` statement's body and the beginning of the `else` part of the `if`…`else` statement. If the condition in line 21 is `false`, lines 25–28 are not output. Instead, lines 37–43 output a `form` element. The user can type a first name in the `form` and press the **Submit** button to request the JSP again and execute the `if` statement's body (lines 25–28).

Software Engineering Observation 37.5

Scriptlets, expressions and fixed template data can be intermixed in a JSP to create different responses based on information in a request to a JSP.

Error-Prevention Tip 37.1

It is sometimes difficult to debug errors in a JSP, because the line numbers reported by a JSP container normally refer to the servlet that represents the translated JSP, not the original JSP line numbers. Program development environments such as Sun Microsystems, Inc.'s Sun One Studio 4 enable JSPs to be compiled in the environment, so you can see syntax error messages. These messages include the statement in the servlet that represents the translated JSP, which can be helpful in determining the error.

Error-Prevention Tip 37.2

Many JSP containers store the servlets representing the translated JSPs. For example, the Tomcat installation directory contains a subdirectory called `work` *in which you can find the source code for the servlets translated by Tomcat.*

To test Fig. 37.4 in Tomcat, copy `welcome.jsp` into the `jsp` directory created in Section 37.3. Open your Web browser and enter the following URL to test `welcome.jsp`:

```
http://localhost:8080/iw3htp3/jsp/welcome.jsp
```

When you first execute the JSP, it displays the `form` in which you can enter your first name, because the preceding URL does not pass a `firstName` parameter to the JSP. After you submit your first name, your browser should appear as shown in the second screen capture of Fig. 37.4. [*Note*: As with servlets, it is possible to pass `get` request arguments as part of the URL.] The following URL supplies the `firstName` parameter to `welcome.jsp`:

```
http://localhost:8080/iw3htp3/jsp/welcome.jsp?firstName=Paul
```

37.6 Standard Actions

We continue our JSP discussion with the **JSP standard actions** (Fig. 37.5). These actions provide JSP implementors with access to several of the most common tasks performed in a JSP, such as including content from other resources, forwarding requests to other resources and interacting with JavaBeans. JSP containers process actions at request time. Actions

Action	Description
`<jsp:include>`	Dynamically includes another resource in a JSP. As the JSP executes, the referenced resource is included and processed.
`<jsp:forward>`	Forwards request processing to another JSP, servlet or static page. This action terminates the current JSP's execution.
`<jsp:plugin>`	Allows a plug-in component to be added to a page in the form of a browser-specific `object` or `embed` HTML element. In the case of a Java applet, this action enables the downloading and installation of the **Java Plug-in**, if it is not already installed on the client computer.
`<jsp:param>`	Used with the `include`, `forward` and `plugin` actions to specify additional name/value pairs of information for use by these actions.
JavaBean Manipulation	
`<jsp:useBean>`	Specifies that the JSP uses a JavaBean instance. This action specifies the scope of the bean and assigns it an ID that scripting components can use to manipulate the bean.
`<jsp:setProperty>`	Sets a property in the specified JavaBean instance. A special feature of this action is automatic matching of request parameters to bean properties of the same name.
`<jsp:getProperty>`	Gets a property in the specified JavaBean instance and converts the result to a string for output in the response.

Fig. 37.5 JSP standard actions.

are delimited by `<jsp:`*action*`>` and `</jsp:`*action*`>`, where *action* is the standard action name. In cases where nothing appears between the starting and ending tags, the XML empty element syntax `<jsp:`*action* `/>` can be used. Figure 37.5 summarizes the JSP standard actions. We use the actions in the next several subsections.

37.6.1 `<jsp:include>` Action

JavaServer Pages support two include mechanisms—the **`<jsp:include>` action** and the **`include` directive**. Action `<jsp:include>` enables dynamic content to be included in a JavaServer Page at request time (not translation time as with the include directive). If the included resource changes between requests, the next request to the JSP containing the `<jsp:include>` action includes the new content of the resource. On the other hand, the `include` directive copies the content into the JSP once, at JSP translation time. If the included resource changes, the new content will not be reflected in the JSP that used the `include` directive unless that JSP is recompiled. Figure 37.6 describes the attributes of action `<jsp:include>`.

Software Engineering Observation 37.6

According to the JavaServer Pages 1.1 specification, a JSP container is allowed to determine whether a resource included with the `include` directive has changed. If so, the container can recompile the JSP that included the resource. However, the specification does not provide a mechanism to indicate a change in an included resource to the container.

Attribute	Description
`page`	Specifies the relative URI path of the resource to include. The resource must be part of the same Web application.
`flush`	Specifies whether the buffer should be flushed after the `include` is performed. In JSP 1.1, this attribute is required to be `true`.

Fig. 37.6 Action `<jsp:include>` attributes.

Performance Tip 37.2

The `<jsp:include>` action is more flexible than the `include` directive, but requires more overhead when page contents change frequently. Use the `<jsp:include>` action only when dynamic content is necessary.

Common Programming Error 37.3

Setting the `<jsp:include>` action's `flush` attribute to `false` is a translation-time error. Currently, the `flush` attribute supports only `true` values.

Common Programming Error 37.4

Not specifying the `<jsp:include>` action's `flush` attribute is a translation-time error. Specifying this attribute is mandatory.

Common Programming Error 37.5

Specifying in a `<jsp:include>` action a page that is not part of the same Web application is a request-time error—the `<jsp:include>` action will not include any content.

The next example demonstrates action `<jsp:include>` using four XHTML and JSP resources that represent both static and dynamic content. JavaServer Page `include.jsp` (Fig. 37.10) includes three other resources: `banner.html` (Fig. 37.7), `toc.html` (Fig. 37.8) and `clock2.jsp` (Fig. 37.9). JavaServer Page `include.jsp` creates an XHTML document containing a `table` in which `banner.html` spans two columns across the top of the `table`, `toc.html` is the left column of the second row and `clock2.jsp` (a simplified version of Fig. 37.1) is the right column of the second row. Figure 37.10 uses three `<jsp:include>` actions (lines 38–39, 48 and 55–56) as the content in `td` elements of the `table`. Using two XHTML documents and a JSP in Fig. 37.10 demonstrates that JSPs can include both static and dynamic content. The output window in Fig. 37.10 demonstrates the result of one request to `include.jsp`.

```
<!-- Fig. 37.7: banner.html                 -->
<!-- banner to include in another document -->
<div style = "width: 580px">
   <p>
      Java(TM), C, C++, Visual Basic(R),
```

Fig. 37.7 Banner (`banner.html`) to include across the top of the XHTML document created by Fig. 37.10. (Part 1 of 2.)

```
      Object Technology, and <br /> Internet and
      World Wide Web Programming Training <br />
      On-Site Seminars Delivered Worldwide
   </p>

   <p>
      <a href = "mailto:deitel@deitel.com">deitel@deitel.com</a>
      <br />978.461.5880<br />12 Clock Tower Place, Suite 200,
      Maynard, MA 01754
   </p>
</div>
```

Fig. 37.7 Banner (`banner.html`) to include across the top of the XHTML document created by Fig. 37.10. (Part 2 of 2.)

```
<!-- Fig. 37.8: toc.html                      -->
<!-- contents to include in another document -->

<p><a href = "http://www.deitel.com/books/index.html">
   Publications/BookStore
</a></p>

<p><a href = "http://www.deitel.com/whatsnew.html">
   What's New
</a></p>

<p><a href = "http://www.deitel.com/books/downloads.html">
   Downloads/Resources
</a></p>

<p><a href = "http://www.deitel.com/faq/index.html">
   FAQ (Frequently Asked Questions)
</a></p>

<p><a href = "http://www.deitel.com/intro.html">
   Who we are
</a></p>

<p><a href = "http://www.deitel.com/index.html">
   Home Page
</a></p>

<p>Send questions or comments about this site to
   <a href = "mailto:deitel@deitel.com">
      deitel@deitel.com
   </a><br />
   Copyright 1995-2003 by Deitel &amp; Associates, Inc.
   All Rights Reserved.
</p>
```

Fig. 37.8 Table of contents (`toc.html`) to include down the left side of the XHTML document created by Fig. 37.10.

Figure 37.9 (`clock2.jsp`) demonstrates how to determine the client's `Locale` (package `java.util`) and uses that `Locale` to format a `Date` with a **`DateFormat`** (package `java.text`) object. Line 14 invokes the `request` object's `getLocale` method, which returns the client's `Locale`. Lines 17–20 invoke `DateFormat static` method `getDateTimeInstance` to obtain a `DateFormat` object. The first two arguments indicate that the date and time formats should each be `LONG` format (other options are `FULL`, `MEDIUM`, `SHORT` and `DEFAULT`). The third argument specifies the `Locale` for which the `DateFormat` object should format the date. Line 25 invokes the `DateFormat` object's `format` method to produce a `String` representation of the `Date`. The `DateFormat` object formats this `String` for the `Locale` specified in lines 17–20. [*Note*: This example works for Western languages that use the ISO-8859-1 character set. However, for languages that do not use this character set, the JSP must specify the proper character set using the JSP `page` directive (Section 37.7.1). At the site `java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html`, Sun provides a list of character encodings. The response's content type defines the character set to use in the response. The content type has the form: "*mimeType*`;charset=`*encoding*" (e.g., `"text/html;charset=ISO-8859-1"`.]

```
1  <!-- Fig. 37.9: clock2.jsp                          -->
2  <!-- date and time to include in another document -->
3
4  <table>
5     <tr>
6        <td style = "background-color: black;">
7           <p class = "big" style = "color: cyan; font-size: 3em;
8              font-weight: bold;">
9
10             <%-- script to determine client local and --%>
11             <%-- format date accordingly               --%>
12             <%
13                // get client locale
14                java.util.Locale locale = request.getLocale();
15
16                // get DateFormat for client's Locale
17                java.text.DateFormat dateFormat =
18                   java.text.DateFormat.getDateTimeInstance(
19                      java.text.DateFormat.LONG,
20                      java.text.DateFormat.LONG, locale );
21
22             %>  <%-- end script --%>
23
24             <%-- output date --%>
25             <%= dateFormat.format( new java.util.Date() ) %>
26          </p>
27       </td>
28    </tr>
29 </table>
```

Fig. 37.9 JSP `clock2.jsp` to include as the main content in the XHTML document created by Fig. 37.10.

To test Fig. 37.10 in Tomcat, copy `banner.html`, `toc.html`, `clock2.jsp`, `include.jsp` and the `images` directory into the `jsp` directory created in Section 37.3. Open your Web browser and enter the following URL to test `include.jsp`:

```
http://localhost:8080/iw3htp3/jsp/include.jsp
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 37.10: include.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <title>Using jsp:include</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td {
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <table>
         <tr>
            <td style = "width: 160px; text-align: center">
               <img src = "images/logotiny.png"
                  width = "140" height = "93"
                  alt = "Deitel & Associates, Inc. Logo" />
            </td>

            <td>

               <%-- include banner.html in this JSP --%>
               <jsp:include page = "banner.html"
                  flush = "true" />

            </td>
         </tr>

         <tr>
            <td style = "width: 160px">

```

Fig. 37.10 JSP `include.jsp` Includes resources with `<jsp:include>`. (Part 1 of 2.)

```
                <%-- include toc.html in this JSP --%>
                <jsp:include page = "toc.html" flush = "true" />

             </td>

             <td style = "vertical-align: top">

                <%-- include clock2.jsp in this JSP --%>
                <jsp:include page = "clock2.jsp"
                   flush = "true" />

             </td>
          </tr>
       </table>
    </body>
</html>
```

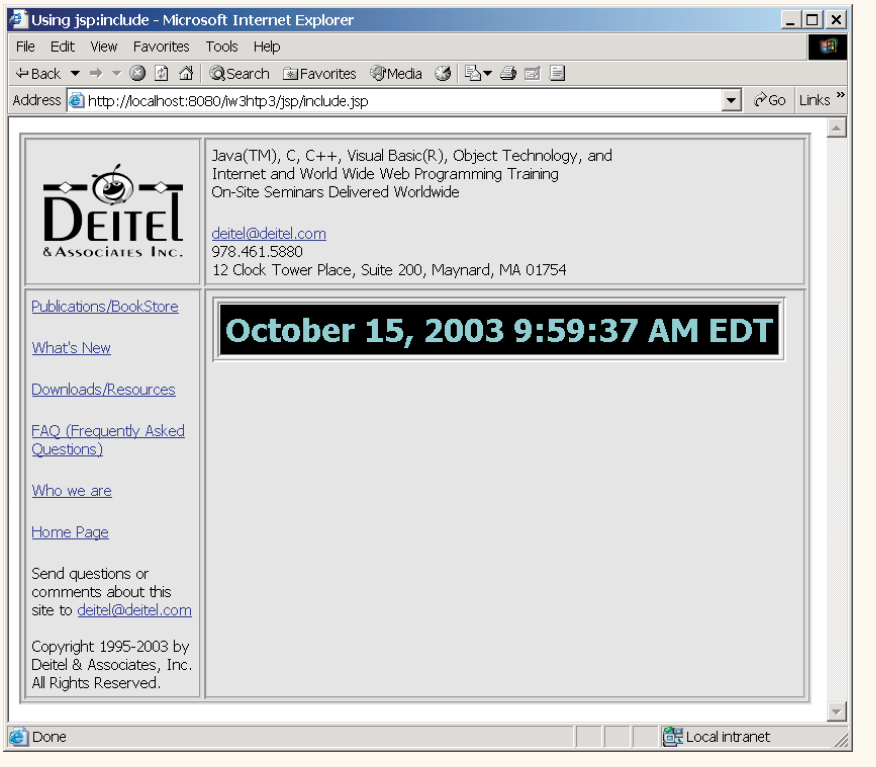


Fig. 37.10 JSP `include.jsp` Includes resources with `<jsp:include>`. (Part 2 of 2.)

37.6.2 `<jsp:forward>` Action

Action **`<jsp:forward>`** enables a JSP to forward request processing to a different resource. Request processing by the original JSP terminates as soon as the JSP forwards the request. Action `<jsp:forward>` has only a `page` attribute that specifies the relative URL of the resource (in the same Web application) to which the request should be forwarded.

Software Engineering Observation 37.7

When using the `<jsp:forward>` action, the resource to which the request will be forwarded must be in the same context (Web application) as the JSP that originally received the request.

JavaServer Page `forward1.jsp` (Fig. 37.11) is a modified version of `welcome.jsp` (Fig. 37.4). The primary difference is in lines 22–25 in which JavaServer Page `forward1.jsp` forwards the request to JavaServer Page `forward2.jsp` (Fig. 37.12). Notice the `<jsp:param>` action in lines 23–24. This action adds a request parameter representing the date and time at which the initial request was received to the request object that is forwarded to `forward2.jsp`.

The `<jsp:param>` action specifies name/value pairs of information that are passed to the `<jsp:include>`, `<jsp:forward>` and `<jsp:plugin>` actions. Every `<jsp:param>` action has two required attributes: `name` and `value`. If a `<jsp:param>` action specifies a parameter that already exists in the request, the new value for the parameter takes precedence over the original value. All values for that parameter can be obtained by using the JSP implicit object `request`'s `getParameterValues` method, which returns an array of `String`s.

JSP `forward2.jsp` uses the `name` specified in the `<jsp:param>` action (`"date"`) to obtain the date and time. It also uses the `firstName` parameter originally passed to `forward1.jsp` to obtain the user's first name. JSP expressions in Fig. 37.12 (lines 23 and 31) insert the request parameter values in the response to the client. The screen capture in Fig. 37.11 shows the initial interaction with the client. The screen capture in Fig. 37.12 shows the results returned to the client after the request was forwarded to `forward2.jsp`.

To test Fig. 37.11 and Fig. 37.12 in Tomcat, copy `forward1.jsp` and `forward2.jsp` into the `jsp` directory created in Section 37.3. Open your Web browser and enter the following URL to test `forward1.jsp`:

```
http://localhost:8080/iw3htp3/jsp/forward1.jsp
```

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 37.11: forward1.jsp -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8
9  <head>
10    <title>Forward request to another JSP</title>
11 </head>
12
13 <body>
14    <% // begin scriptlet
15
16       String name = request.getParameter( "firstName" );
17
18       if ( name != null ) {
19
```

Fig. 37.11 JSP `forward1.jsp` receives a `firstName` parameter, adds a date to the request parameters and forwards the request to `forward2.jsp` for further processing. (Part 1 of 2.)

```
    %> <%-- end scriptlet to insert fixed template data --%>

          <jsp:forward page = "forward2.jsp">
             <jsp:param name = "date"
                value = "<%= new java.util.Date() %>" />
          </jsp:forward>

    <% // continue scriptlet

       }  // end if
       else {

    %> <%-- end scriptlet to insert fixed template data --%>

          <form action = "forward1.jsp" method = "get">
             <p>Type your first name and press Submit</p>

             <p><input type = "text" name = "firstName" />
                <input type = "submit" value = "Submit" />
             </p>
          </form>

    <%  // continue scriptlet

       }  // end else

    %> <%-- end scriptlet --%>
</body>

</html>  <!-- end XHTML document -->
```

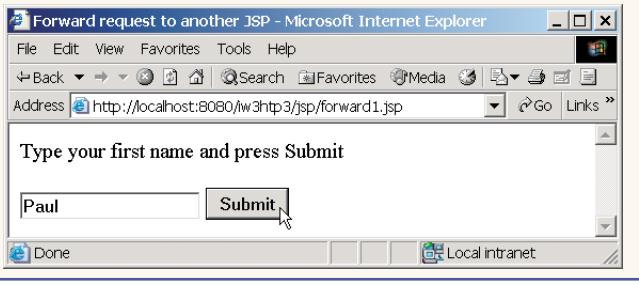


Fig. 37.11 JSP `forward1.jsp` receives a `firstName` parameter, adds a date to the request parameters and forwards the request to `forward2.jsp` for further processing. (Part 2 of 2.)

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

```

Fig. 37.12 JSP `forward2.jsp` receives a request (from `forward1.jsp` in this example) and uses the request parameters as part of the response to the client. (Part 1 of 2.)

```
<!-- forward2.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml"

<head>
   <title>Processing a forwarded request</title>

   <style type = "text/css">
      .big {
         font-family: tahoma, helvetica, arial, sans-serif;
         font-weight: bold;
         font-size: 2em;
      }
   </style>
</head>

<body>
   <p class = "big">
      Hello <%= request.getParameter( "firstName" ) %>, <br />
      Your request was received <br /> and forwarded at
   </p>

   <table style = "border: 6px outset;">
      <tr>
         <td style = "background-color: black;">
            <p class = "big" style = "color: cyan;">
               <%= request.getParameter( "date" ) %>
            </p>
         </td>
      </tr>
   </table>
</body>

</html>
```

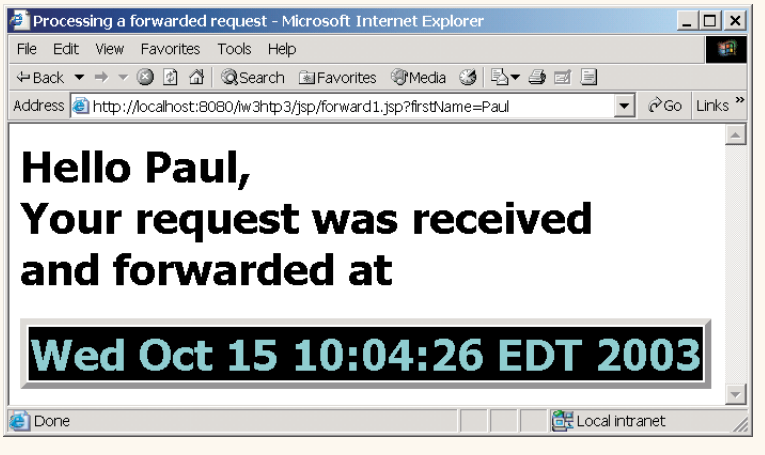


Fig. 37.12 JSP `forward2.jsp` receives a request (from `forward1.jsp` in this example) and uses the request parameters as part of the response to the client. (Part 2 of 2.)

37.6.3 <jsp:useBean> Action

Action <jsp:useBean> enables a JSP to manipulate a Java object. This action creates a Java object or locates an existing object for use in the JSP. Figure 37.13 summarizes action <jsp:useBean>'s attributes. If attributes `class` and `beanName` are not specified, the JSP container attempts to locate an existing object of the type specified in attribute `type`. Like JSP implicit objects, objects specified with action <jsp:useBean> have `page`, `request`, `session` or `application` scope that indicates where they can be used in a Web application. Objects with `page` scope are accessible only by the page in which they are defined. Multiple JSP pages potentially can access objects in other scopes. For example, all JSPs that process a single request can access an object in `request` scope.

Common Programming Error 37.6

One or both of the <jsp:useBean> attributes `class` and `type` must be specified; otherwise, a translation-time error occurs.

Many Web sites place rotating advertisements on their Web pages. Each visit to one of these pages typically results in a different advertisement being displayed in the user's Web browser. Typically, clicking an advertisement takes you to the Web site of the company that placed the advertisement. Our first example of <jsp:useBean> demonstrates a simple advertisement rotator bean that cycles through a list of five advertisements. In this example, the advertisements are covers for some of our books. Clicking a cover takes you to the Amazon.com Web site where you can read about and possibly order the book.

The `Rotator` bean (Fig. 37.14) has three methods: `getImage`, `getLink` and `nextAd`. Method `getImage` returns the image file name for the book cover image. Method `getLink` returns the hyperlink to the book at Amazon.com. Method `nextAd` updates the `Rotator` so the next calls to `getImage` and `getLink` return information for a different advertisement. Methods `getImage` and `getLink` each represent a read-only JavaBean property—`image` and `link`, respectively. `Rotator` keeps track of the current advertisement with its `selectedIndex` variable, which is updated by invoking method `nextAd`.

Attribute	Description
`id`	The name used to manipulate the Java object with actions <jsp:setProperty> and <jsp:getProperty>. A variable of this name is also declared for use in JSP scripting elements. The name specified here is case sensitive.
`scope`	The scope in which the Java object is accessible—`page`, `request`, `session` or `application`. The default scope is `page`.
`class`	The fully qualified class name of the Java object.
`beanName`	The name of a bean that can be used with method `instantiate` of class `java.beans.Beans` to load a JavaBean into memory.
`type`	The type of the JavaBean. This can be the same type as the `class` attribute, a superclass of that type or an interface implemented by that type. The default value is the same as for attribute `class`. A `ClassCastException` occurs if the Java object is not of the type specified with attribute `type`.

Fig. 37.13 Attributes of the <jsp:useBean> action.

```
1  // Fig. 37.14: Rotator.java
2  // A JavaBean that rotates advertisements.
3  package com.deitel.iw3htp3.jsp;
4
5  public class Rotator {
6     private String images[] = { "images/advjHTP1.jpg",
7        "images/cppHTP4.jpg", "images/iw3HTP3.jpg",
8        "images/jwsFEP1.jpg", "images/vbnetHTP2.jpg" };
9
10    private String links[] = {
11       "http://www.amazon.com/exec/obidos/ASIN/0130895601/" +
12          "deitelassociatin",
13       "http://www.amazon.com/exec/obidos/ASIN/0130384747/" +
14          "deitelassociatin",
15       "http://www.amazon.com/exec/obidos/ASIN/0131450913/" +
16          "deitelassociatin",
17       "http://www.amazon.com/exec/obidos/ASIN/0130461342/" +
18          "deitelassociatin",
19       "http://www.amazon.com/exec/obidos/ASIN/0130293636/" +
20          "deitelassociatin" };
21
22    private int selectedIndex = 0;
23
24    // returns image file name for current ad
25    public String getImage()
26    {
27       return images[ selectedIndex ];
28    }
29
30    // returns the URL for ad's corresponding Web site
31    public String getLink()
32    {
33       return links[ selectedIndex ];
34    }
35
36    // update selectedIndex so next calls to getImage and
37    // getLink return a different advertisement
38    public void nextAd()
39    {
40       selectedIndex = ( selectedIndex + 1 ) % images.length;
41    }
42 }
```

Fig. 37.14 `Rotator` bean that maintains a set of advertisements.

Lines 7–8 of JavaServer Page `adrotator.jsp` (Fig. 37.15) obtain a reference to an instance of class `Rotator`. The `id` for the bean is `rotator`. The JSP uses this name to manipulate the bean. The scope of the object is `session`, so that each individual client will see the same sequence of ads during their browsing session. When `adrotator.jsp` receives a request from a new client, the JSP container creates the bean and stores it in JSP that client's `session` (an `HttpSession` object). In each request to this JSP, line 22 uses the `rotator` reference created in line 7 to invoke the `Rotator` bean's `nextAd` method. Thus, each request will receive the next advertisement selected by the `Rotator` bean. Lines

29–34 define a hyperlink to the Amazon.com site for a particular book. Lines 29–30 introduce action **`<jsp:getProperty>`** to obtain the value of the `Rotator` bean's `link` property. Action `<jsp:getProperty>` has two attributes—`name` and `property`—that specify the bean object to manipulate and the property to get. If the JavaBean object uses standard JavaBean naming conventions, the method used to obtain the `link` property value from the bean should be `getLink`. Action `<jsp:getProperty>` invokes `getLink` on the bean referenced with `rotator`, converts the return value into a `String` and outputs the `String` as part of the response to the client. The `link` property becomes the value of the hyperlink's `href` attribute. The hyperlink is represented in the resulting Web page as the book cover image. Lines 32–33 create an `img` element and use another `<jsp:getProperty>` action to obtain the `Rotator` bean's `image` property value.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 37.15: adrotator.jsp -->
6
7  <jsp:useBean id = "rotator" scope = "application"
8     class = "com.deitel.iw3htp3.jsp.Rotator" />
9
10 <html xmlns = "http://www.w3.org/1999/xhtml">
11
12    <head>
13       <title>AdRotator Example</title>
14
15       <style type = "text/css">
16          .big { font-family: helvetica, arial, sans-serif;
17                 font-weight: bold;
18                 font-size: 2em }
19       </style>
20
21       <%-- update advertisement --%>
22       <% rotator.nextAd(); %>
23    </head>
24
25    <body>
26       <p class = "big">AdRotator Example</p>
27
28       <p>
29          <a href = "<jsp:getProperty name = "rotator"
30             property = "link" />">
31
32             <img src = "<jsp:getProperty name = "rotator"
33                property = "image" />" alt = "advertisement" />
34          </a>
35       </p>
36    </body>
37 </html>
```

Fig. 37.15 JSP `adrotator.jsp` uses a `Rotator` bean to display a different advertisement on each request for the page. (Part 1 of 2.)

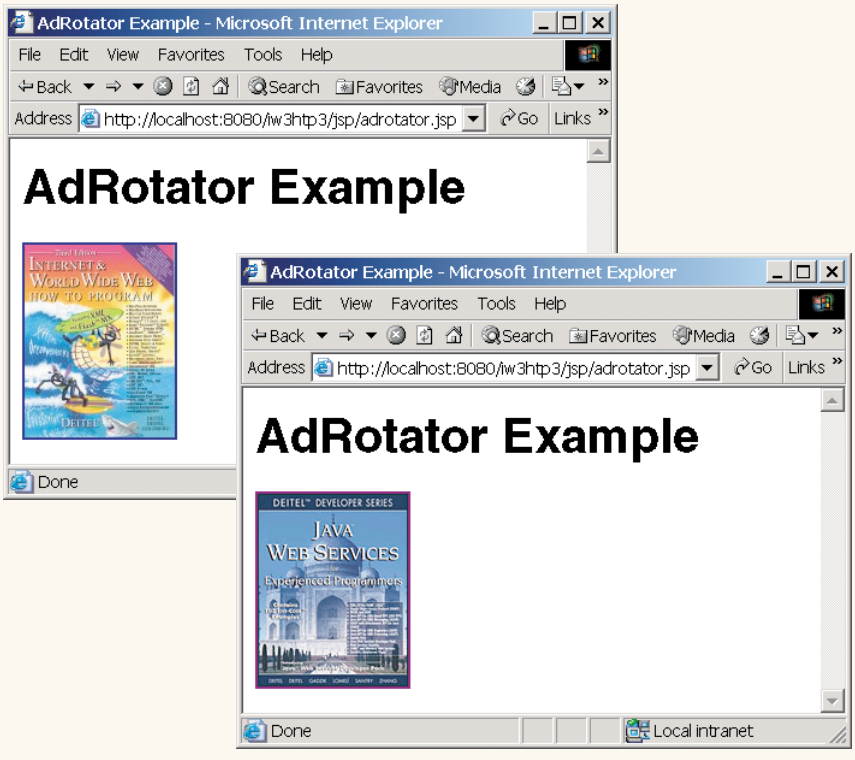


Fig. 37.15 JSP `adrotator.jsp` uses a `Rotator` bean to display a different advertisement on each request for the page. (Part 2 of 2.)

The link and image properties also can be obtained with JSP expressions. For example, action `<jsp:getProperty>` in lines 29–30 can be replaced with the expression

```
<%= rotator.getLink() %>
```

Similarly, action `<jsp:getProperty>` in lines 32–33 can be replaced with the expression

```
<%= rotator.getImage() %>
```

To test `adrotator.jsp` in Tomcat, copy `adrotator.jsp` into the `jsp` directory created in Section 37.3. You should have copied the `images` directory into the `jsp` directory when you tested Fig. 37.10. If not, you must copy the `images` directory there now. Copy `Rotator.class` into the `iw3htp3` Web application's `WEB-INF\classes` directory in Tomcat. [*Note:* This example will work only if the proper package directory structure for `Rotator` is defined in the `classes` directory. `Rotator` is defined in package `com.deitel.iw3htp3.jsp`.] Open your Web browser and enter the following URL to test `adrotator.jsp`:

```
http://localhost:8080/iw3htp3/jsp/adrotator.jsp
```

Try reloading this JSP several times in your browser to see the advertisement change with each request.

Action **`<jsp:setProperty>`** sets JavaBean property values and is particularly useful for mapping request parameter values to JavaBean properties. Request parameters can be used to set properties of primitive types `boolean`, `byte`, `char`, `int`, `long`, `float` and `double` and `java.lang` types `String`, `Boolean`, `Byte`, `Character`, `Integer`, `Long`, `Float` and `Double`. Figure 37.16 summarizes the `<jsp:setProperty>` attributes.

Attribute	Description
`name`	The ID of the JavaBean for which a property (or properties) will be set.
`property`	The name of the property to set. Specifying `"*"` for this attribute causes the JSP to match the request parameters to the properties of the bean. For each request parameter that matches (i.e., the name of the request parameter is identical to the bean's property name), the corresponding property in the bean is set to the value of the parameter. If the value of the request parameter is `""`, the property value in the bean remains unchanged.
`param`	If request parameter names do not match bean property names, this attribute can be used to specify which request parameter should be used to obtain the value for a specific bean property. This attribute is optional. If this attribute is omitted, the request parameter names must match bean property names.
`value`	The value to assign to a bean property. The value typically is the result of a JSP expression. This attribute is particularly useful for setting bean properties that cannot be set using request parameters. This attribute is optional. If this attribute is omitted, the JavaBean property must be of a type that can be set using request parameters.

Fig. 37.16 Attributes of the `<jsp:setProperty>` action.

Common Programming Error 37.7

Use action `<jsp:setProperty>`'s `value` attribute to set JavaBean property types that cannot be set with request parameters; otherwise, conversion errors occur.

Software Engineering Observation 37.8

Action `<jsp:setProperty>` can use request-parameter values to set JavaBean properties only for properties of the following types: `String`s, primitive types (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`) and type wrapper classes (`Boolean`, `Byte`, `Character`, `Short`, `Integer`, `Long`, `Float` and `Double`).

37.7 Directives

Directives are messages to the JSP container that enable the programmer to specify page settings (such as the error page), to include content from other resources and to specify custom-tag libraries for use in a JSP. Directives (delimited by **`<%@`** and **`%>`**) are processed at translation time. Thus, directives do not produce any immediate output, because they are processed before the JSP accepts any requests. Figure 37.17 summarizes the three directive types. These directives are discussed in the next several subsections.

37.7.1 page Directive

The **page directive** specifies global settings for the JSP in the JSP container. There can be many `page` directives, provided that there is only one occurrence of each attribute. The only exception to this is the `import` attribute, which can be used repeatedly to import Java packages used in the JSP. Figure 37.18 summarizes the attributes of the `page` directive.

Directive	Description
`page`	Defines page settings for the JSP container to process.
`include`	Causes the JSP container to perform a translation-time insertion of another resource's content. As the JSP is translated into a servlet and compiled, the referenced file replaces the `include` directive and is translated as if it were originally part of the JSP.
`taglib`	Allows programmers to define new tags in the form of **tag libraries**, which can be used to encapsulate functionality and simplify the coding of a JSP.

Fig. 37.17 JSP directives.

Attribute	Description
`language`	The scripting language used in the JSP. Currently, the only valid value for this attribute is `java`.
`extends`	Specifies the class from which the translated JSP will be inherited. This attribute must be a fully qualified class name.
`import`	Specifies a comma-separated list of fully qualified type names and/or packages that will be used in the current JSP. When the scripting language is `java`, the default import list is `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*` and `javax.servlet.http.*`. If multiple `import` properties are specified, the package names are placed in a list by the container.
`session`	Specifies whether the page participates in a session. The values for this attribute are `true` (participates in a session—the default) or `false` (does not participate in a session). When the page is part of a session, implicit object `session` is available for use in the page. Otherwise, `session` is not available, and using `session` in the scripting code results in a translation-time error.
`buffer`	Specifies the size of the output buffer used with the implicit object `out`. The value of this attribute can be `none` for no buffering, or a value such as `8kb` (the default buffer size). The JSP specification indicates that the buffer used must be at least the size specified.

Fig. 37.18 Attributes of the `page` directive. (Part 1 of 2.)

Attribute	Description
`autoFlush`	When set to `true` (the default), this attribute indicates that the output buffer used with implicit object `out` should be flushed automatically when the buffer fills. If set to `false`, an exception occurs if the buffer overflows. This attribute's value must be `true` if the buffer attribute is set to `none`.
`isThreadSafe`	Specifies if the page is thread safe. If `true` (the default), the page is considered to be thread safe, and it can process multiple requests at the same time. If `false`, the servlet that represents the page implements interface `java.lang.SingleThreadModel` and only one request can be processed by that JSP at a time. The JSP standard allows multiple instances of a JSP to exists for JSPs that are not thread safe. This enables the container to handle requests more efficiently. However, this does not guarantee that resources shared across JSP instances are accessed in a thread-safe manner.
`info`	Specifies an information string that describes the page. This string is returned by the `getServletInfo` method of the servlet that represents the translated JSP. This method can be invoked through the JSP's implicit `page` object.
`errorPage`	Any exceptions in the current page that are not caught are sent to the error page for processing. The error page implicit object `exception` references the original exception.
`isErrorPage`	Specifies if the current page is an error page that will be invoked in response to an error on another page. If the attribute value is `true`, the implicit object `exception` is created and references the original exception that occurred. If `false` (the default), any use of the `exception` object in the page results in a translation-time error.
`contentType`	Specifies the MIME type of the data in the response to the client. The default type is `text/html`.

Fig. 37.18 Attributes of the `page` directive. (Part 2 of 2.)

Common Programming Error 37.8

Providing multiple `page` directives with one or more repeated attributes in common is a JSP translation-time error. Also, providing a `page` directive with an attribute or value that is not recognized is a JSP translation-time error.

Software Engineering Observation 37.9

According to the JSP specification section 2.7.1, the `extends` attribute "should not be used without careful consideration as it restricts the ability of the JSP container to provide specialized superclasses that may improve on the quality of rendered service." Remember that a Java class can extend exactly one other class. If your JSP specifies an explicit superclass, the JSP container cannot translate your JSP into a subclass of one of the container application's own enhanced servlet classes.

Common Programming Error 37.9

Using JSP implicit object `session` in a JSP that does not have its `page` directive attribute `session` set to `true` is a translation-time error.

37.7.2 `include` Directive

The **`include` directive** includes the content of another resource once, at JSP translation time. The `include` directive has only one attribute—`file`—that specifies the URL of the resource to include. The difference between directive `include` and action `<jsp:include>` is noticeable only if the included content changes. For example, if the definition of an XHTML document changes after it is included with directive `include`, future invocations of the JSP will show the original content of the XHTML document, not the new content. In contrast, action `<jsp:include>` is processed in each request to the JSP. Therefore, changes to included content would be apparent in the next request to the JSP that uses action `<jsp:include>`.

JSP `includeDirective.jsp` (Fig. 37.19) reimplements `include.jsp` (Fig. 37.10) using `include` directives. To test `includeDirective.jsp` in Tomcat, copy `includeDirective.jsp` into the `jsp` directory created in Section 37.3. Open your Web browser and enter the following URL to test `includeDirective.jsp`:

```
http://localhost:8080/iw3htp3/jsp/includeDirective.jsp
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 37.19: includeDirective.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <title>Using the include directive</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td {
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <table>
         <tr>
            <td style = "width: 160px; text-align: center">
               <img src = "images/logotiny.png"
                  width = "140" height = "93"
                  alt = "Deitel & Associates, Inc. Logo" />
```

Fig. 37.19 JSP `includeDirective.jsp` demonstrates including content at translation-time with directive `include`. (Part 1 of 2.)

```
            </td>

            <td>

               <%-- include banner.html in this JSP --%>
               <%@ include file = "banner.html" %>

            </td>
         </tr>

         <tr>
            <td style = "width: 160px">

               <%-- include toc.html in this JSP --%>
               <%@ include file = "toc.html" %>

            </td>

            <td style = "vertical-align: top">

               <%-- include clock2.jsp in this JSP --%>
               <%@ include file = "clock2.jsp" %>

            </td>
         </tr>
      </table>
   </body>
</html>
```

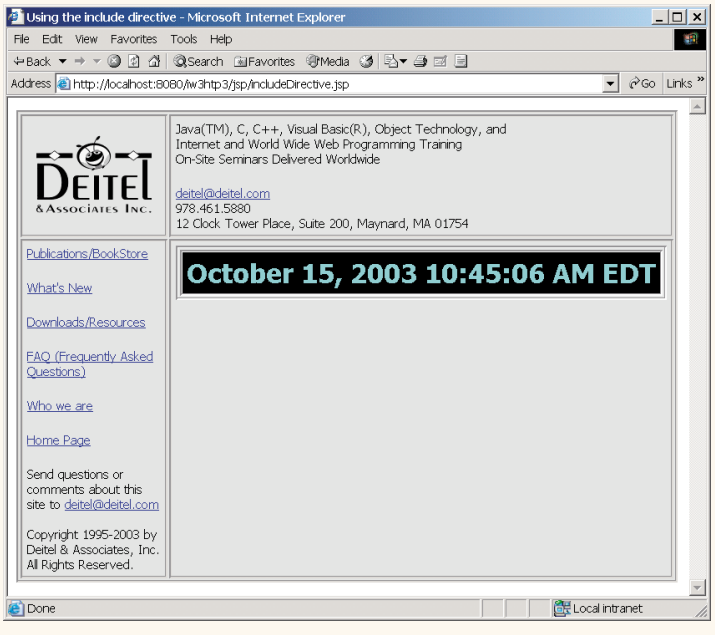


Fig. 37.19 JSP `includeDirective.jsp` demonstrates including content at translation-time with directive `include`. (Part 2 of 2.)

37.8 Case Study: Guest Book

Our next example is a guest book that enables users to place their first name, last name and e-mail address into a guest-book database. After submitting their information, users see a Web page containing all the users in the guest book. Each person's e-mail address is displayed as a hyperlink that allows the user to send an e-mail message to the person. The example demonstrates action `<jsp:setProperty>`, the JSP `page` directive, JSP error pages and the use of JDBC.

The guest book example consists of JavaBeans `GuestBean` (Fig. 37.20) and `GuestDataBean` (Fig. 37.21), and JSPs `guestBookLogin.jsp` (Fig. 37.22), `guestBookView.jsp` (Fig. 37.23) and `guestBookErrorPage.jsp` (Fig. 37.24). Sample outputs from this example are shown in Fig. 37.25. JavaBean `GuestBean` (Fig. 37.20) defines three guest properties: `firstName`, `lastName` and `email`. Each is a read/write property with *set* and *get* methods to manipulate the property.

```
// Fig. 37.20: GuestBean.java
// JavaBean to store data for a guest in the guest book.
package com.deitel.iw3htp3.jsp.beans;

public class GuestBean {
   private String firstName, lastName, email;

   // set the guest's first name
   public void setFirstName( String name )
   {
      firstName = name;
   }

   // get the guest's first name
   public String getFirstName()
   {
      return firstName;
   }

   // set the guest's last name
   public void setLastName( String name )
   {
      lastName = name;
   }

   // get the guest's last name
   public String getLastName()
   {
      return lastName;
   }

   // set the guest's email address
   public void setEmail( String address )
   {
      email = address;
   }
```

Fig. 37.20 `GuestBean` stores information for one guest. (Part 1 of 2.)

```

   // get the guest's email address
   public String getEmail()
   {
      return email;
   }
}
```

Fig. 37.20 GuestBean stores information for one guest. (Part 2 of 2.)

JavaBean GuestDataBean (Fig. 37.21) connects to the guestbook database and provides methods getGuestList and addGuest to manipulate the database. The guestbook database has a single table (guests) containing three columns (firstName, lastName and email). [*Note*: The examples folder for this chapter contains the Access database (guestbook.mdb) used in this example. For information on setting up an ODBC Data Source Name to reference this database, please see www.deitel.com/books/iw3HTP3/iw3htp3.html.].

```
// Fig. 37.21: GuestDataBean.java
// Class GuestDataBean makes a database connection and supports
// inserting and retrieving data from the database.
package com.deitel.iw3htp3.jsp.beans;

// Java core packages
import java.io.*;
import java.sql.*;
import java.util.*;

public class GuestDataBean {
   private Connection connection;
   private PreparedStatement addRecord, getRecords;

   // construct TitlesBean object
   public GuestDataBean() throws Exception
   {
      // load the Cloudscape driver
      Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );

      // connect to the database
      connection = DriverManager.getConnection(
         "jdbc:odbc:guestbook" );

      statement = connection.createStatement();
   }

   // return an ArrayList of GuestBeans
   public ArrayList getGuestList() throws SQLException
   {
      ArrayList guestList = new ArrayList();
```

Fig. 37.21 GuestDataBean performs database access on behalf of guestBookLogin.jsp. (Part 1 of 2.)

```
32
33         // obtain list of titles
34         ResultSet results = statement.executeQuery(
35            "SELECT firstName, lastName, email FROM guests" );
36
37         // get row data
38         while ( results.next() ) {
39            GuestBean guest = new GuestBean();
40
41            guest.setFirstName( results.getString( 1 ) );
42            guest.setLastName( results.getString( 2 ) );
43            guest.setEmail( results.getString( 3 ) );
44
45            guestList.add( guest );
46         }
47
48         return guestList;
49      }
50
51      // insert a guest in guestbook database
52      public void addGuest( GuestBean guest ) throws SQLException
53      {
54         statement.executeUpdate( "INSERT INTO guests ( firstName, " +
55            "lastName, email ) VALUES ( '" + guest.getFirstName() + "', '" +
56            guest.getLastName() + "', '" + guest.getEmail() + "' )" );
57      }
58
59      // close statements and terminate database connection
60      protected void finalize()
61      {
62         // attempt to close database connection
63         try {
64            statement.close();
65            connection.close();
66         }
67
68         // process SQLException on close operation
69         catch ( SQLException sqlException ) {
70            sqlException.printStackTrace();
71         }
72      }
73   }
```

Fig. 37.21 `GuestDataBean` performs database access on behalf of `guestBookLogin.jsp`. (Part 2 of 2.)

`GuestDataBean` method `getGuestList` (lines 29–49) returns an `ArrayList` of `GuestBean` objects representing the guests in the database. Method `getGuestList` creates the `GuestBean` objects from the `ResultSet` returned by `Statement` method `executeQuery` (lines 34–35).

`GuestDataBean` method `addGuest` (lines 52–57) receives a `GuestBean` as an argument and uses the `GuestBean`'s properties as the arguments to `Statement` method `executeUpdate` (lines 54–56). This `Statement` inserts a new guest in the database.

Note that the `GuestDataBean`'s constructor, `getGuestList` and `addGuest` methods do not process potential exceptions. In the constructor, line 19 can throw a `ClassNotFoundException`, and the other statements can throw `SQLException`s. Similarly, `SQLException`s can be thrown from the bodies of methods `getGuestList` and `addGuest`. In this example, we purposely let any exceptions that occur get passed back to the JSP that invokes the `GuestDataBean`'s constructor or methods. This enables us to demonstrate JSP error pages. When a JSP performs an operation that causes an exception, the JSP can include scriptlets that catch the exception and process it. Exceptions that are not caught can be forwarded to a JSP error page for handling.

JavaServer Page `guestBookLogin.jsp` (Fig. 37.22) is a modified version of `forward1.jsp` (Fig. 37.11) that displays a `form` in which users can enter their first name, last name and e-mail address. When the user submits the `form`, `guestBookLogin.jsp` is requested again, so it can ensure that all the data values were entered. If not, the `guestBookLogin.jsp` responds with the `form` again, so the user can fill in missing field(s). If the user supplies all three pieces of information, `guestBookLogin.jsp` forwards the request to `guestBookView.jsp`, which displays the guest book contents.

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 37.22: guestBookLogin.jsp -->

<%-- page settings --%>
<%@ page errorPage = "guestBookErrorPage.jsp" %>

<%-- beans used in this JSP --%>
<jsp:useBean id = "guest" scope = "page"
   class = "com.deitel.iw3htp3.jsp.beans.GuestBean" />
<jsp:useBean id = "guestData" scope = "request"
   class = "com.deitel.iw3htp3.jsp.beans.GuestDataBean"

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <title>Guest Book Login</title>

   <style type = "text/css">
      body {
         font-family: tahoma, helvetica, arial, sans-serif;
      }

      table, tr, td {
         font-size: .9em;
         border: 3px groove;
         padding: 5px;
         background-color: #dddddd;
      }
```

Fig. 37.22 `guestBookLogin.jsp` enables the user to submit a first name, a last name and an e-mail address to be placed in the guest book. (Part 1 of 3.)

```
   </style>
</head>

<body>
   <jsp:setProperty name = "guest" property = "*" />

   <% // start scriptlet

      if ( guest.getFirstName() == null ||
           guest.getLastName() == null ||
           guest.getEmail() == null ) {

   %> <%-- end scriptlet to insert fixed template data --%>

         <form method = "post" action = "guestBookLogin.jsp">
            <p>Enter your first name, last name and email
               address to register in our guest book.</p>

            <table>
               <tr>
                  <td>First name</td>

                  <td>
                     <input type = "text" name = "firstName" />
                  </td>
               </tr>

               <tr>
                  <td>Last name</td>

                  <td>
                     <input type = "text" name = "lastName" />
                  </td>
               </tr>

               <tr>
                  <td>Email</td>

                  <td>
                     <input type = "text" name = "email" />
                  </td>
               </tr>

               <tr>
                  <td colspan = "2">
                     <input type = "submit"
                        value = "Submit" />
                  </td>
               </tr>
            </table>
         </form>
```

Fig. 37.22 `guestBookLogin.jsp` enables the user to submit a first name, a last name and an e-mail address to be placed in the guest book. (Part 2 of 3.)

```
83
84      <% // continue scriptlet
85
86         }  // end if
87         else {
88            guestData.addGuest( guest );
89
90      %> <%-- end scriptlet to insert jsp:forward action --%>
91
92            <%-- forward to display guest book contents --%>
93            <jsp:forward page = "guestBookView.jsp" />
94
95      <% // continue scriptlet
96
97         }  // end else
98
99      %> <%-- end scriptlet --%>
100  </body>
101
102  </html>
```

Fig. 37.22 `guestBookLogin.jsp` enables the user to submit a first name, a last name and an e-mail address to be placed in the guest book. (Part 3 of 3.)

Line 8 of `guestBookLogin.jsp` uses the **page directive**, which defines information that is globally available in a JSP. Directives are delimited by **`<%@`** and **`%>`**. In this case, the `page` directive's **`errorPage` attribute** is set to `guestBookErrorPage.jsp` (Fig. 37.24), indicating that all uncaught exceptions are forwarded to `guestBookErrorPage.jsp` for processing.

Lines 11–14 define two `<jsp:useBean>` actions. Lines 11–12 create an instance of `GuestBean` called `guest`. This bean has `page` scope—it exists for use only in this page. Lines 13–14 create an instance of `GuestDataBean` called `guestData`. This bean has `request` scope—it exists for use in this page and any other page that helps process a single client request. Thus, when `guestBookLogin.jsp` forwards a request to `guestBookView.jsp`, the same `GuestDataBean` instance is still available for use in `guestBookView.jsp`.

Line 36 demonstrates setting properties of the `GuestBean` called `guest` with request parameter values. The `input` elements in lines 55, 63 and 71 have the same names as the `GuestBean` properties. So, we use action `<jsp:setProperty>`'s ability to match request parameters to properties by specifying `"*"` for attribute `property`. Line 36 also can set the properties individually with the following lines:

```
<jsp:setProperty name = "guest" property = "firstName"
   param = "firstName" />

<jsp:setProperty name = "guest" property = "lastName"
   param = "lastName" />

<jsp:setProperty name = "guest" property = "email"
   param = "email" />
```

If the request parameters had names that differed from `GuestBean`'s properties, the `param` attribute in each of the preceding `<jsp:setProperty>` actions could be changed to the appropriate request parameter name.

JavaServer Page `guestBookView.jsp` (Fig. 37.23) outputs an XHTML document containing the guest-book entries in tabular format. Lines 8–10 define three `page` directives. Line 8 specifies that the error page for this JSP is `guestBookErrorPage.jsp`. Line 9 indicates that classes from package `java.util` are used in this JSP, and line 10 indicates that classes from our package `com.deitel.iw3htp3.jsp.beans` also are used.

Lines 13–14 specify a `<jsp:useBean>` action that declares a reference to a `GuestDataBean` object. If a `GuestDataBean` object already exists, the action returns a reference to the existing object. Otherwise, the action creates a `GuestDataBean` for use in this JSP. Lines 50–59 define a scriptlet that gets the guest list from the `GuestDataBean` and begin a loop to output the entries. Lines 61–70 combine fixed template text with JSP expressions to create rows in the table of guest book data that will be displayed on the client. The scriptlet in lines 72–76 terminates the loop.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 37.23: guestBookView.jsp -->
6
7  <%-- page settings --%>
8  <%@ page errorPage = "guestBookErrorPage.jsp" %>
9  <%@ page import = "java.util.*" %>
10 <%@ page import = "com.deitel.iw3htp3.jsp.beans.*"
11
12 <%-- GuestDataBean to obtain guest list --%>
13 <jsp:useBean id = "guestData" scope = "request"
14    class = "com.deitel.iw3htp3.jsp.beans.GuestDataBean"
15
16 <html xmlns = "http://www.w3.org/1999/xhtml">
17
18    <head>
19       <title>Guest List</title>
20
21       <style type = "text/css">
22          body {
23             font-family: tahoma, helvetica, arial, sans-serif;
24          }
25
26          table, tr, td, th {
27             text-align: center;
28             font-size: .9em;
29             border: 3px groove;
30             padding: 5px;
```

Fig. 37.23 `guestBookView.jsp` displays the contents of the guest book. (Part 1 of 2.)

```
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <p style = "font-size: 2em;">Guest List</p>

      <table>
         <thead>
            <tr>
               <th style = "width: 100px;">Last name</th>
               <th style = "width: 100px;">First name</th>
               <th style = "width: 200px;">Email</th>
            </tr>
         </thead>

         <tbody>

         <% // start scriptlet

            List guestList = guestData.getGuestList();
            Iterator guestListIterator = guestList.iterator();
            GuestBean guest;

            while ( guestListIterator.hasNext() ) {
               guest = ( GuestBean ) guestListIterator.next();

         %> <%-- end scriptlet; insert fixed template data --%>

               <tr>
                  <td><%= guest.getLastName() %></td>

                  <td><%= guest.getFirstName() %></td>

                  <td>
                     <a href = "mailto:<%= guest.getEmail() %>">
                        <%= guest.getEmail() %></a>
                  </td>
               </tr>

         <% // continue scriptlet

            } // end while

         %> <%-- end scriptlet --%>

         </tbody>
      </table>
   </body>

</html>
```

Fig. 37.23 `guestBookView.jsp` displays the contents of the guest book. (Part 2 of 2.)

JavaServer Page `guestBookErrorPage.jsp` (Fig. 37.24) outputs an XHTML document containing an error message based on the type of exception that causes this error page to be invoked. Lines 8–10 define several `page` directives. Line 8 sets `page` directive attribute **`isErrorPage`**. Setting this attribute to `true` makes the JSP an error page and enables access to the JSP implicit object `exception` that refers to an exception object indicating the problem that occurred.

Common Programming Error 37.10

JSP implicit object `exception` can be used only in error pages. Using this object in other JSPs results in a translation-time error.

Lines 29–46 define scriptlets that determine the type of exception that occurred and begin outputting an appropriate error message with fixed template data. The actual error message from the exception is output in line 56.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 37.24: guestBookErrorPage.jsp -->
6
7  <%-- page settings --%>
8  <%@ page isErrorPage = "true" %>
9  <%@ page import = "java.util.*" %>
10 <%@ page import = "java.sql.*" %>
11
12 <html xmlns = "http://www.w3.org/1999/xhtml">
13
14    <head>
15       <title>Error!</title>
16
17       <style type = "text/css">
18          .bigRed {
19             font-size: 2em;
20             color: red;
21             font-weight: bold;
22          }
23       </style>
24    </head>
25
26    <body>
27       <p class = "bigRed">
28
29       <% // scriptlet to determine exception type
30          // and output beginning of error message
31          if ( exception instanceof SQLException )
32       %>
33
34             An SQLException
35
```

Fig. 37.24 `guestBookErrorPage.jsp` responds to exceptions in `guestBookLogin.jsp` and `guestBookView.jsp`. (Part 1 of 2.)

```
      <%
         else if ( exception instanceof ClassNotFoundException )
      %>

            A ClassNotFoundException

      <%
         else
      %>

            An exception

      <%-- end scriptlet to insert fixed template data --%>

         <%-- continue error message output --%>
         occurred while interacting with the guestbook database.
      </p>

      <p class = "bigRed">
         The error message was:<br />
         <%= exception.getMessage() %>
      </p>

      <p class = "bigRed">Please try again later</p>
   </body>

</html>
```

Fig. 37.24 `guestBookErrorPage.jsp` responds to exceptions in `guestBookLogin.jsp` and `guestBookView.jsp`. (Part 2 of 2.)

Figure 37.25 shows sample interactions between the user and the JSPs in the guest book example. In the first two rows of output, separate users entered their first name, last name and e-mail. In each case, the current contents of the guest book are returned and displayed for the user. In the final interaction, a third user specified an e-mail address that already existed in the database. The e-mail address is the primary key in the `guests` table of the `guestbook` database, so its values must be unique. Thus, the database prevents the new record from being inserted, and an exception occurs. The exception is forwarded to `guestBookErrorPage.jsp` for processing, which results in the last screen capture.

To test the guest book in Tomcat, copy `guestBookLogin.jsp`, `guestBookView.jsp` and `guestBookErrorPage.jsp` into the `jsp` directory created in Section 37.3. Copy `GuestBean.class` and `GuestDataBean.class` into the `iw3htp3` Web application's `WEB-INF\classes` directory in Tomcat. [*Note:* This example will work only if the proper package directory structure for `GuestBean` and `GuestDataBean` is defined in the `classes` directory. These classes are defined in package `com.deitel.iw3htp3.jsp.beans`.] Open your Web browser and enter the following URL to test `guestBookLogin.jsp`:

```
http://localhost:8080/iw3htp3/jsp/guestBookLogin.jsp
```

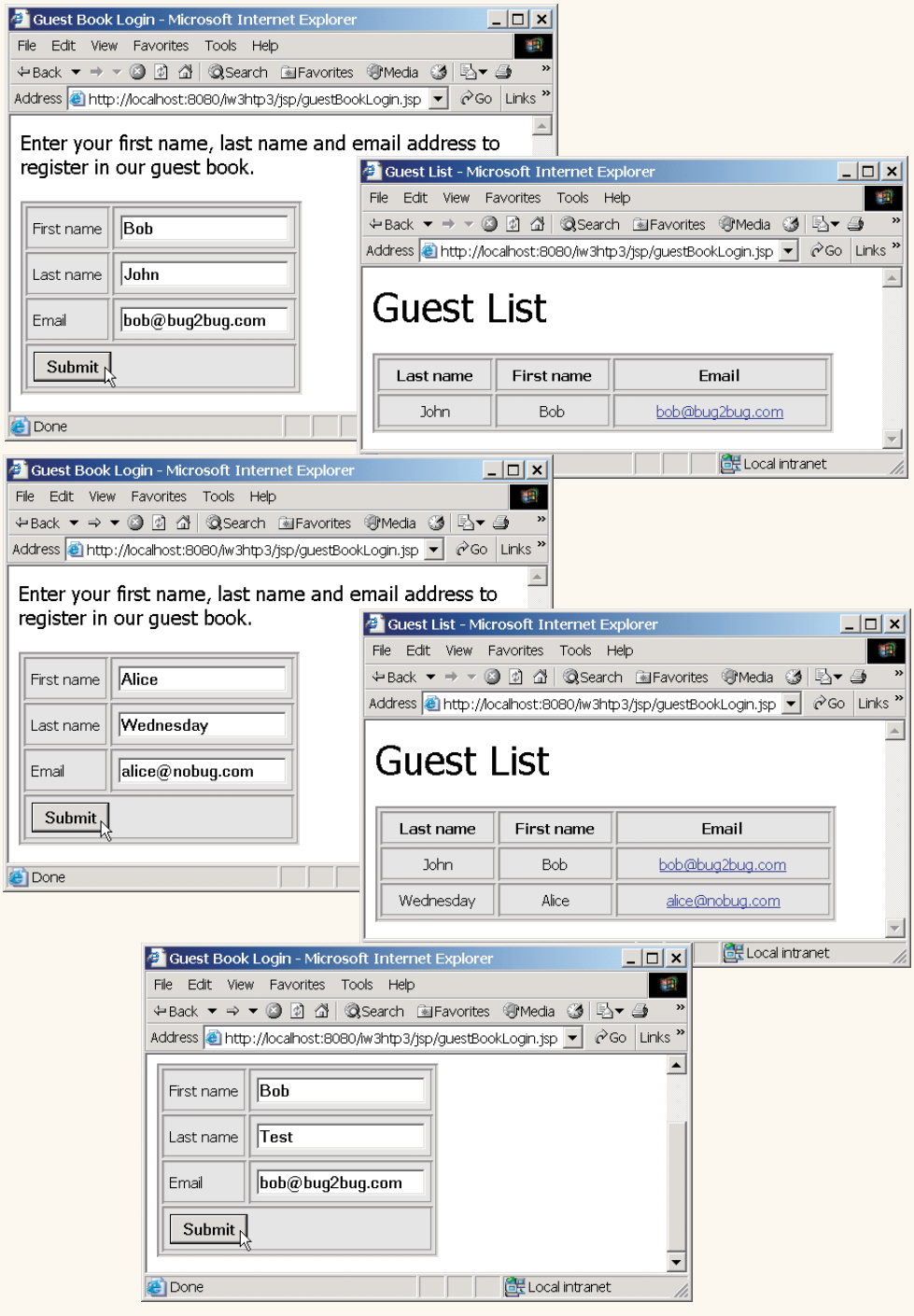


Fig. 37.25 JSP guest book sample output windows (Part 1 of 2.).

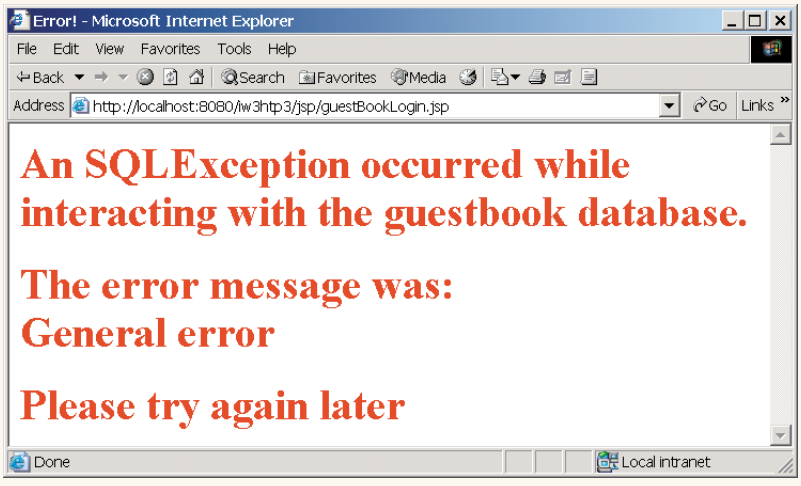


Fig. 37.25 JSP guest book sample output windows (Part 2 of 2.).

37.9 Web Resources

`java.sun.com/products/jsp`
Home page for information about JavaServer Pages at the Sun Microsystems Java site.

`java.sun.com/products/servlet`
Home page for information about servlets at the Sun Microsystems Java site.

`java.sun.com/j2ee`
Home page for the Java 2 Enterprise Edition at the Sun Microsystems Java site.

`www.w3.org`
The World Wide Web Consortium home page. This site provides information about current and developing Internet and Web standards, such as XHTML, XML and CSS.

`jsptags.com`
Tutorials, tag libraries, software and other resources for JSP programmers.

`jspinsider.com`
This Web programming site concentrates on resources for JSP programmers. It includes software, tutorials, articles, sample code, references and links to other JSP and Web programming resources.

SUMMARY

- JavaServer Pages (JSPs) are an extension of servlet technology.
- JavaServer Pages enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting.
- JSP programmers can create custom tag libraries that enable Web-page designers who are not familiar with Java programming to enhance their Web pages with powerful dynamic content and processing capabilities.
- Classes and interfaces specific to JavaServer Pages programming are located in packages `javax.servlet.jsp` and `javax.servlet.jsp.tagext`.
- The JavaServer Pages 1.1 specification can be downloaded from `java.sun.com/products/jsp/download.html`.

- There are four key components to JSPs—directives, actions, scriptlets and tag libraries.
- Directives specify global information that is not associated with a particular JSP request.
- Actions encapsulate functionality in predefined tags that programmers can embed in a JSP.
- Scriptlets, or scripting elements, enable programmers to insert Java code that interacts with components in a JSP (and possibly other Web application components) to perform request processing.
- Tag libraries are part of the tag extension mechanism that enables programmers to create new tags that encapsulate complex Java functionality.
- JSPs normally include XHTML or XML markup. Such markup is known as fixed template data or fixed template text.
- Programmers tend to use JSPs when most of the content sent to the client is fixed template data and only a small portion of the content is generated dynamically with Java code.
- Programmers use servlets when a small portion of the content is fixed template data.
- JSPs normally execute as part of a Web server. The server often is referred to as the JSP container.
- When a JSP-enabled server receives the first request for a JSP, the JSP container translates that JSP into a Java servlet that handles the current request and future requests to the JSP.
- The JSP container places the Java statements that implement a JSP's response in method `_jspService` at translation time.
- The request/response mechanism and life cycle of a JSP are the same as those of a servlet.
- JSPs can define methods `jspInit` and `jspDestroy` that are invoked when the container initializes a JSP and when the container terminates a JSP, respectively.
- JSP expressions are delimited by `<%=` and `%>`. Such expressions are converted to `String`s by the JSP container and are output as part of the response.
- The XHTML `meta` element can set a refresh interval for a document that is loaded into a browser. This causes the browser to request the document repeatedly at the specified interval in seconds.
- When you first invoke a JSP in Tomcat, there is a delay as Tomcat translates the JSP into a servlet and invokes the servlet to respond to your request.
- Implicit objects provide programmers with servlet capabilities in the context of a JavaServer Page.
- Implicit objects have four scopes—application, page, request and session.
- Objects with application scope are part of the JSP and servlet container application. Objects with page scope exist only as part of the page in which they are used. Each page has its own instances of the page-scope implicit objects. Objects in request scope exist for the duration of the request. Request-scope objects go out of scope when request processing completes with a response to the client. Objects in session scope exist for the client's entire browsing session.
- JSP scripting components include scriptlets, comments, expressions, declarations and escape sequences.
- Scriptlets are blocks of code delimited by `<%` and `%>`. They contain Java statements that are placed in method `_jspService` when the container translates a JSP into a servlet.
- JSP comments are delimited by `<%--` and `--%>`. XHTML comments are delimited by `<!--` and `-->`. Java's end-of-line comments (`//`) and traditional comments (delimited by `/*` and `*/`) can be used inside scriptlets.
- JSP comments and scripting language comments are ignored and do not appear in the response.
- A JSP expression, delimited by `<%=` and `%>`, contains a Java expression that is evaluated when a client requests the JSP containing the expression. The container converts the result of a JSP expression to a `String` object, then outputs the `String` as part of the response to the client.

- Declarations, delimited by `<%!` and `%>`, enable a JSP programmer to define variables and methods. Variables become instance variables of the class that represents the translated JSP. Similarly, methods become members of the class that represents the translated JSP.
- Special characters or character sequences that the JSP container normally uses to delimit JSP code can be included in a JSP as literal characters in scripting elements, fixed template data and attribute values by using escape sequences.
- JSP standard actions provide JSP implementors with access to several of the most common tasks performed in a JSP. JSP containers process actions at request time.
- JavaServer Pages support two include mechanisms—the `<jsp:include>` action and the `include` directive.
- Action `<jsp:include>` enables dynamic content to be included in a JavaServer Page. If the included resource changes between requests, the next request to the JSP containing the `<jsp:include>` action includes the new content of the resource.
- The `include` directive is processed once, at JSP translation time, and causes the content to be copied into the JSP. If the included resource changes, the new content will not be reflected in the JSP that used the include directive unless that JSP is recompiled.
- Action `<jsp:forward>` enables a JSP to forward the processing of a request to a different resource. Processing of the request by the original JSP terminates as soon as the request is forwarded.
- Action `<jsp:param>` specifies name/value pairs of information that are passed to the `include`, `forward` and `plugin` actions. Every `<jsp:param>` action has two required attributes—`name` and `value`. If a `param` action specifies a parameter that already exists in the request, the new value for the parameter takes precedence over the original value. All values for that parameter can be obtained with the JSP implicit object `request`'s `getParameterValues` method, which returns an array of `String`s.
- Action `<jsp:useBean>` enables a JSP to manipulate a Java object. This action can be used to create a Java object for use in the JSP or to locate an existing object.
- Like JSP implicit objects, objects specified with action `<jsp:useBean>` have `page`, `request`, `session` or `application` scope that indicates where they can be used in a Web application.
- Action `<jsp:getProperty>` obtains the value of a JavaBean's property. Action `<jsp:getProperty>` has two attributes—`name` and `property`—that specify the bean object to manipulate and the property to get.
- JavaBean property values can be set with action `<jsp:setProperty>`, which is particularly useful for mapping request parameter values to JavaBean properties. Request parameters can be used to set properties of primitive types `boolean`, `byte`, `char`, `int`, `long`, `float` and `double` and `java.lang` types `String`, `Boolean`, `Byte`, `Character`, `Integer`, `Long`, `Float` and `Double`.
- The `page` directive defines information that is globally available in a JSP. Directives are delimited by `<%@` and `%>`. The `page` directive's `errorPage` attribute indicates where all uncaught exceptions are forwarded for processing.
- Action `<jsp:setProperty>` has the ability to match request parameters to properties of the same name in a bean by specifying `"*"` for attribute `property`.
- Attribute `import` of the `page` directive enables programmers to specify Java classes and packages that are used in the context of a JSP.

- If the attribute `isErrorPage` of the `page` directive is set to `true`, the JSP is an error page. This condition enables access to the JSP implicit object `exception` that refers to an exception object indicating the problem that occurred.
- Directives are messages to the JSP container that enable the programmer to specify page settings (such as the error page), to include content from other resources and to specify custom tag libraries that can be used in a JSP. Directives are processed at the time a JSP is translated into a servlet and compiled. Thus, directives do not produce any immediate output.
- The `page` directive specifies global settings for a JSP in the JSP container. There can be many `page` directives, provided that there is only one occurrence of each attribute. The exception to this rule is the `import` attribute, which can be used repeatedly to import Java packages.

TERMINOLOGY

%\> escape sequence for %>
<!-- and --> XHTML comment delimiters
<%-- and --%> JSP comment delimiters
<% and %> scriptlet delimiters
<%! and %> declaration delimiters
<%= and %> JSP expression delimiters
<%@ and %> directive delimiters
<\% escape sequence for <%
action
`autoFlush` attribute of `page` directive
`beanName` attribute of `<jsp:useBean>` action
`buffer` attribute of `page` directive
`class` attribute of `<jsp:useBean>` action
client-server networking
comment
`config` implicit object
container
`contentType` attribute of `page` directive
declaration
directive
dynamic content
error page
`errorPage` attribute of `page` directive
escape sequence
expression
`extends` attribute of `page` directive
`file` attribute of `include` directive
fixed template data
fixed template text
`flush` attribute of `<jsp:include>` action
forward a request
`getParameterValues` method of `request` object
`id` attribute of `<jsp:useBean>` action
implicit object
implicit object scopes
`import` attribute of `page` directive
`isErrorPage` attribute of `page` directive
`isThreadSafe` attribute of `page` directive
JavaServer Pages (JSPs)
JavaServer Pages 1.1 specification
`javax.servlet.jsp` package
`javax.servlet.jsp.tagext` package
`<jsp:forward>` action
`<jsp:getProperty>` action
`<jsp:include>` action
`<jsp:param>` action
`<jsp:setProperty>` action
`<jsp:useBean>` action
`jspDestroy` method
`jspInit` method
`_jspService` method
`JspWriter` (package `javax.servlet.jsp`)
`language` attribute of `page` directive
match request parameters
`meta` element
`name` attribute of `<jsp:param>`
`name` attribute of `<jsp:setProperty>`
name/value pair
`out` implicit object
`page` attribute of `<jsp:forward>`
`page` attribute of `<jsp:include>`
`page` directive
`page` implicit object
page scope
`param` attribute of `<jsp:setProperty>`
`prefix` attribute of `taglib` directive
`property` attribute of `<jsp:setProperty>`
refresh interval
`request` implicit object
request scope
request-time error
`response` implicit object
`scope` attribute of `<jsp:useBean>`

include a resource
`include` directive
`info` attribute of `page` directive
specify attributes of a custom tag
standard actions
translation-time error
translation-time include
scope of a bean
scripting element
scriptlet
`type` attribute of `<jsp:plugin>`
`type` attribute of `<jsp:useBean>`
`value` attribute of `<jsp:param>`
`value` attribute of `<jsp:setProperty>`

SELF-REVIEW EXERCISES

37.1 Fill in the blanks in each of the following statements:

a) Action _____ has the ability to match request parameters to properties of the same name in a bean by specifying `"*"` for attribute `property`.
b) There are four key components to JSPs: _____, _____, _____ and _____.
c) The implicit objects have four scopes: _____, _____, _____ and _____.
d) The _____ directive is processed once, at JSP translation time and causes content to be copied into the JSP.
e) Classes and interfaces specific to JavaServer Pages programming are located in packages _____ and _____.
f) JSPs normally execute as part of a Web server that is referred to as the _____.
g) JSP scripting components include _____, _____, _____, _____ and _____.

37.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) An object in page scope exists in every JSP of a particular Web application.
b) Directives specify global information that is not associated with a particular JSP request.
c) Action `<jsp:include>` is evaluated once at page translation time.
d) Like XHTML comments, JSP comments and script language comments appear in the response to the client.
e) Objects in application scope are part of a particular Web application.
f) Each page has its own instances of the page-scope implicit objects.
g) Action `<jsp:setProperty>` has the ability to match request parameters to properties of the same name in a bean by specifying `"*"` for attribute `property`.
h) Objects in session scope exist for the client's entire browsing session.

ANSWERS TO SELF-REVIEW EXERCISES

37.1 a) `<jsp:setProperty>`. b) directives, actions, scriptlets, tag libraries. c) application, page, request and session. d) `include`. e) `javax.servlet.jsp`, `javax.servlet.jsp.tagext`. f) JSP container. g) scriptlets, comments, expressions, declarations, escape sequences.

37.2 a) False. Objects in page scope exist only as part of the page in which they are used. b) True. c) False. Action `<jsp:include>` enables dynamic content to be included in a JavaServer Page. d) False. JSP comments and script language comments are ignored and do not appear in the response. e) False. Objects in application scope are part of the JSP container application. f) True. g) True. h) True.

EXERCISES

37.3 Write a JSP page to output the string `"Hello world!"` ten times.

37.4 Modify Exercise 36.4 to run as a JSP page.

37.5 Rewrite Figure 37.15 to allow users to select the image. Use a JSP expression instead of the `getProperty` JSP tag.

37.6 Create a JSP and JDBC-based address book. Use the guest book example of Fig. 37.20 through Fig. 37.24 as a guide. Your address book should allow one to insert entries, delete entries and search for entries.

37.7 Reimplement the Web application of Fig. 36.20 (favorite animal survey) using JSPs.

37.8 Modify your solution to Exercise 37.7 to allow the user to see the survey results without responding to the survey.