URL Actions

Topics in This Chapter

- Overview
- The <c:import> Action
- The <c:redirect> Action
- The <c:url> Action
- The <c:param> Action
- Accessing External Resources
- Accessing Resources in Foreign Contexts
- Redirecting a Response

Chapter 5

If you've developed Web applications with JavaServer Pages (JSP), you have probably found many uses for <jsp:include> and <jsp:forward>. The former includes the contents of a resource and the latter forwards control to a Web component, such as a servlet or another JSP page. On the other hand, you may have found that those actions have limited capabilities; for example, the URLs that you specify for those actions must be relative URLs, so you cannot use them to access URLs outside your Web application. JSTL provides a set of URL actions that augment the capabilities provided by <jsp:include> and <jsp:forward>; those actions are the subject of this chapter.

Before we discuss the JSTL URL actions, let's review some Web application basics and define a few terms used throughout this chapter. Java-based Web applications are stored in a directory on your filesystem; for example, Figure 5–1 illustrates a Web application that resides under the `C:\core-jstl\webapp` directory.

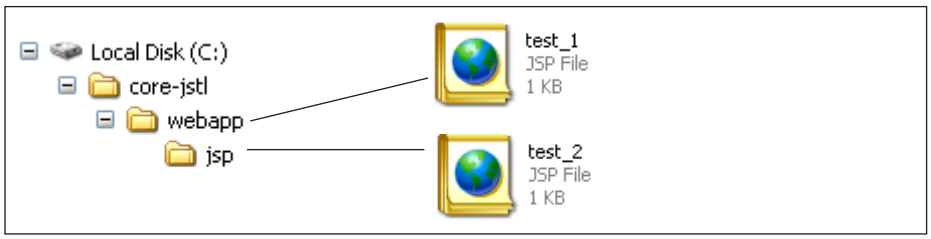

Figure 5–1 A Simple Java-Based Web Application

Java-based Web applications *reside* in a *directory*, but they are *defined* by a *context*; for example, the Web application depicted in Figure 5–1 could be defined in Tomcat's configuration file with a `Context` element, like this:[1]

```
<Context path="/core-jstl"
      docBase="C:/core-jstl/webapp"/>
```

The `path` attribute of the `Context` element defines a URL that you use to access a Web application that resides in a directory specified by the `docBase` attribute; for example, to access the Web application shown in Figure 5–1 you would use the URL `$SCHEME$HOSTNAME/core-jstl`, where `$SCHEME$HOSTNAME` represents a scheme and a host name. For example, if the scheme is `http://` and the host name is `localhost`, the URL for the Web application defined above would be `http://localhost/core-jstl`.

As websites grow, it is not uncommon for them to contain more than one Web application. From the perspective of a single Web application, the other Web applications in the same website are referred to as *foreign contexts*. For example, if your website has a registration Web application and a shopping application, the registration application is a foreign context relative to the shopping application, and vice versa.

When you access resources with <jsp:include>, you can specify either a *context-relative path* or a *page-relative path*; the former is relative to the top-level directory in a context (a Web application), and the latter is relative to the JSP page in which the <jsp:include> action resides.

Context-relative paths always start with a forward slash, whereas page-relative paths do not. For example, for the application shown in Figure 5–1, you can:

- Access `test_2.jsp` from `test_1.jsp`
  - with a context-relative path, like this:
    `<jsp:include page='/jsp/test_2.jsp'/>`
  - or with a page-relative path, like this:
    `<jsp:include page='jsp/test_2.jsp'/>`.
- Access `test_1.jsp` from `test_2.jsp`
  - with a context-relative path, like this:
    `<jsp:include page='/test_1.jsp'/>`
  - or with a page-relative path, like this:
    `<jsp:include page='../test_1.jsp'/>`.

1. Different JSP containers use different terms for context paths; for example, Resin calls them web-app ids.

Now that we have established some common vocabulary, let's take a look at the JSTL URL actions.

5.1 Overview

JSTL provides four URL actions that let you do the following:

- Import page-relative resources, context-relative resources, resources that reside in a foreign context, and external resources
- Redirect HTTP responses
- Create URLs with automatic URL rewriting and encoded request parameters

The JSTL URL actions are listed in Table 5.1.

Table 5.1 JSTL URL Actions

Action	Description
<c:import>	Imports the content of a URL-based resource
<c:redirect>	Redirects an HTTP response
<c:url>	Creates a URL, applying URL rewriting as necessary
<c:param>	Encodes a request parameter for <c:import>, <c:redirect>, or <c:url>

The actions listed in Table 5.1 are discussed—in the order in which they are listed—in the following sections. After we discuss those actions, we examine how they can be used in several real-world scenarios, such as scraping book information from Amazon.com, importing JSP pages from foreign contexts, and redirecting HTTP responses for logging access to external resources.

5.2 The <c:import> Action

The <jsp:include> action lets you encapsulate functionality in one JSP page and include it in another; for example, you could include company headers and footers, like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   ...
   <body>
      <jsp:include page='/WEB-INF/jsp/company/companyHeader.jsp'/>

      <%-- Page content goes here--%>

      <jsp:include page='/WEB-INF/jsp/company/companyFooter.jsp'/>
   </body>
</html>
```

The preceding JSP page includes JSP files specified with context-relative URLs that reside in a `/WEB-INF/jsp/company` directory.[2] You can also specify request parameters for included files with the <jsp:param> action, like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   ...
   <body>
      <jsp:include page='/WEB-INF/jsp/company/companyHeader.jsp'>
         <jsp:param name='user'
                    value='<%=session.getAttribute("userName")%>'/>
      </jsp:include>

      <%-- Page content goes here--%>

      <jsp:include page='/WEB-INF/jsp/company/companyFooter.jsp'/>
   </body>
</html>
```

In the preceding code fragment, `companyHeader.jsp` is passed a request parameter named `user` that references a user name stored in session scope.

As handy as the <jsp:include> action is, its capabilities are limited; for example, it cannot import external resources or resources stored in foreign contexts. The JSTL

2. Files stored under `WEB-INF` cannot be directly accessed by users.

<c:import> action can do all those things and more. Table 5.2 lists the features supported by <jsp:include> and <c:import>.

Table 5.2 <jsp:include> vs. <c:import>

Feature	<jsp:include>	<c:import>
Access resources in the same Web application	Yes	Yes
Access resources in a foreign context[a]	No	Yes
Access external resources	No	Yes
Provide a performance boost option	No	Yes
Store imported content in a scoped variable	No	Yes
Specify a character encoding for the imported resource	No	Yes
Support the JSTL Expression Language	No	Yes

a. This feature is not supported by all JSP containers.

You can use <c:import> instead of <jsp:include> to import resources in the same Web application; for example, you could import company headers and footers like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   ...
   <body>
      <c:import url='/WEB-INF/jsp/company/companyHeader.jsp'/>

      <%-- Page content goes here--%>

      <c:import url='/WEB-INF/jsp/company/companyFooter.jsp'/>
   </body>
</html>
```

JSTL also provides a <c:param> action that you can use just like <jsp:param>; for example, the code fragment listed on page 202 could be rewritten like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   ...
   <body>
      <c:import url='/WEB-INF/jsp/company/companyHeader.jsp'>
         <c:param name='user'
```

```
                value='${sessionScope.userName}'/>
        </c:import>

        <%-- Page content goes here--%>

        <c:import url='/WEB-INF/jsp/company/companyFooter.jsp'/>
    </body>
</html>
```

The <c:import> action applies URL rewriting as necessary to maintain sessions if cookies are disabled on the client. The <c:import> action has two syntaxes; here's one of them:[3]

<c:import url [context] [charEncoding] [var] [scope]>
* <c:param> actions*
</c:import>

The `url` attribute is similar to the <jsp:include> action's `page` attribute—both attributes specify a URL, either context-relative or page-relative—whose content is included in the JSP page in which the respective actions reside. But the URL specified with the <c:import> `url` attribute can also represent an external resource or a resource that resides in a foreign context.

To access an external resource, you simply specify an absolute URL for the `url` attribute. To access a resource in a foreign context, you must specify a value for the `context` attribute that represents a context path for the foreign context in conjunction with the `url` attribute, which represents a context-relative path to the resource. For example, from another Web application in the same website, you could import `test_2.jsp` shown in Figure 5–1 on page 199 like this:

```
<c:import url='/jsp/test_2.jsp' context='/core-jstl'/>
```

See "Accessing External Resources" on page 210 for an example of importing external resources and "Accessing Resources in Foreign Contexts" on page 215 for an example of importing resources from a foreign context.

The `charEncoding` attribute specifies a character encoding, such as `UTF-8`, that <c:import> uses to decode characters that it imports;[4] for example, you could specify a character encoding of `Shift_JIS` to import a URL whose content is in Japanese like this:

3. Items in brackets are optional. See "<c:import>" on page 486 for a more complete description of <c:import> syntax.
4. See "Unicode and Charsets" on page 260 for more information about character encodings.

```
<c:import url='http://www.tcvb.or.jp/jp/index-j.htm'
 charEncoding='Shift_JIS'/>
```

By default, the <c:import> action writes the content of the URL that it imports to the current `JspWriter`; however, if you specify the `var` attribute, <c:import> will create a scoped variable whose name is specified with that attribute. That scoped variable references a string that contains the content that <c:import> imports. By default, <c:import> stores that scoped variable in page scope, but you can specify the `scope` attribute to store it in page, request, session, or application scope.

You can also use <c:import> with this syntax:

<c:import url [context] [charEncoding] varReader>
body content that uses the varReader scoped variable:
<c:param> actions not allowed
</c:import>

The preceding syntax is the same as the first syntax, except that the `var` and `scope` attributes are replaced by a `varReader` attribute and <c:param> actions are disallowed in the body of the <c:import> action. The `varReader` attribute specifies the name of a reader that references the imported content. That reader is only accessible in the body of the <c:import> action, and because it must be available immediately after the <c:import> start tag, <c:param> actions are not allowed in the body of the <c:import> action. This syntax is provided for efficiency because the imported content is not stored in a string but instead is accessed directly with a reader. Figure 5–2 shows a JSP page that uses <c:import> with the preceding syntax to display the content of a URL.

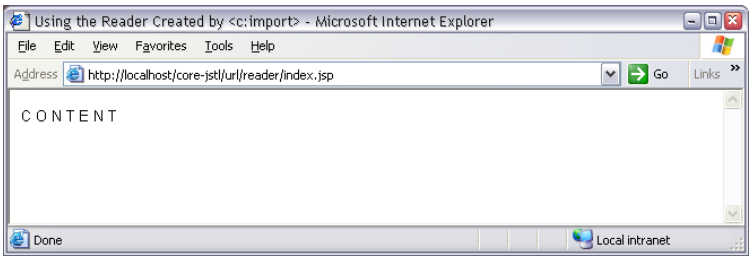

Figure 5–2 Using <c:import> with a Reader

The JSP page shown in Figure 5–2 uses a custom action nested in the body of a <c:import> action. That custom action uses a reader created by <c:import> to directly access the content of a URL. That JSP page is listed in Listing 5.1.

The preceding JSP page uses <c:import> to read the content of another JSP page named `someContent.jsp`, which resides in the same directory as the preceding

Listing 5.1 *Using a Custom Action That Uses the Optional Reader Created by <c:import>*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>
        Using the Reader Created by &lt;c:import&gt;
      </title>
   </head>

   <body>
      <%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>
      <%@ taglib uri='WEB-INF/core-jstl.tld' prefix='core-jstl' %>

      <c:import url='someContent.jsp' varReader='reader'>
         <core-jstl:displayUrlContent reader='reader'/>
      </c:import>
   </body>
</html>
```

JSP page. The `varReader` attribute is specified so that <c:import> will create a reader and store it in page scope. The custom action—<core-jstl:displayUrlContent>—uses the reader to display the imported content. Notice that the custom action has a `reader` attribute that specifies the name of the reader. That attribute's value must be the same as the value specified for the enclosing <c:import> action's `varReader` attribute.

The JSP page imported by the preceding JSP page is listed in Listing 5.2.

Listing 5.2 *someContent.jsp*

```
CONTENT
```

The tag handler for the <core-jstl:displayUrlContent> custom action is listed in Listing 5.3.

The preceding tag handler's `doStartTag` method invokes the `PageContext.findAttribute` method to locate the reader created by an enclosing <c:import> action to read each character and write it to the current `JspWriter`.

Note: Unlike other JSTL actions, such as the iteration, SQL, and internationalization actions, the URL actions do not expose any classes or interfaces.

Listing 5.3 *WEB-INF/classes/tags/DisplayUrlAction.java*

```
package tags;

import java.io.Reader;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.*;

public class DisplayUrlAction extends TagSupport {
   private String readerName;

   public void setReader(String readerName) {
      this.readerName = readerName;
   }
   public int doStartTag() throws JspException {
      int character;

      Reader reader = (Reader)
                      pageContext.findAttribute(readerName);

      if(reader == null) {
         throw new JspException("You can only use this action " +
                                "in the body of a "  +
                                "&lt;c:import&gt; "  +
                                "action that exposes a reader ");
      }
      try {
         while((character = reader.read()) != -1)
            pageContext.getOut().print((char)character);
      }
      catch(java.io.IOException ex) {
         throw new JspException(ex.getMessage());
      }
      return SKIP_BODY;
   }
}
```

If the <c:import> tag handler's class had been exposed, the preceding tag handler could check to make sure that it had an <c:import> ancestor action and could also obtain a reference to the reader that the enclosing <c:import> action created. However, because the URL actions do not expose any classes or interfaces, you must explicitly pass the tag handler the name of the reader created by its enclosing <c:import> action and the tag handler must also check to make sure that the reader is not `null`.

5.3 The <c:redirect> Action

The <c:redirect> action sends an HTTP redirect response to the client and aborts the processing of the JSP page in which the action resides. You can use <c:redirect> to redirect to an external resource or a resource in a foreign context with the following syntax:[5]

<c:redirect url [context]/>

As is the case for <c:import>, if you specify the `context` attribute for <c:redirect>, you must also specify a context-relative URL with the `url` attribute that points to a resource contained in that foreign context. You can also use <c:redirect> with <c:param> actions with this syntax:

<c:redirect url [context]>
* <c:param> actions*
</c:redirect>

Like <c:import>, the <c:redirect> action applies URL rewriting as necessary. See "Redirecting a Response" on page 225 for an example of how you can use <c:redirect>.

5.4 The <c:url> Action

The <c:url> action processes a URL, applying URL rewriting—for relative URLs only—as necessary. The <c:url> action has two syntaxes; here's one of them:[6]

<c:url value [context] [var] [scope]/>

The mandatory `value` attribute specifies the URL that's processed by the <c:url> action. The `context` attribute lets you specify a foreign context. Like <c:import> and <c:redirect>, if you specify the `context` attribute for <c:url>, you must also specify a context-relative URL, with the `value` attribute, that points to a resource in that foreign context. By default, <c:url> sends the processed URL to the current

5. Items in brackets are optional. See "<c:redirect>" on page 489 for a more complete description of <c:redirect> syntax.
6. Items in brackets are optional. See "<c:url>" on page 490 for a more complete description of <c:url> syntax.

JspWriter, but you can store that URL in a scoped variable instead if you specify the var attribute and, optionally, the scope attribute.

Like <c:import> and <c:redirect>, you can specify request parameters that are encoded in the URL that <c:url> processes with nested <c:param> actions. You can do that with the following syntax:

<c:url value [context] [var] [scope]>
* <c:param> actions*
</c:url>

If you specify a context-relative or page-relative URL for the value attribute, <c:url> will prepend the context path of the Web application to the URL; for example, consider the following use of <c:url>:

```
<c:url value='/test_1.jsp'/>
```

If the context path of the Web application is /core-jstl/webapp, <c:url> will produce the following URL: /core-jstl/webapp/test_1.jsp, not taking into account possible URL rewriting. Because of this feature, you must not use <c:url> in conjunction with <c:import> or <c:redirect> for relative URLs because those actions also prepend the context path to relative URLs before passing the URL to the request dispatcher. For example, consider the following code:

```
<c:import url='/test_1.jsp'/>
```

The preceding code fragment is *not equivalent* to the following code fragment:

```
<%-- WARNING: this code fragment will throw an exception --%>

<c:url value='/test_1.jsp' var='processedURL'/>
<c:import url='${processedURL}'/>
```

The preceding code fragment will throw an exception because both <c:url> and <c:import> will try to prepend the context path to the relative URL. URLs processed by <c:url> actions are meant to be *sent directly to the browser*; for example:

```
<c:url value='/test_1.jsp' var='processedURL'/>
<a href='<c:out value="${processedURL}"/>'>Click Here</a>
```

The preceding code fragment creates a URL with <c:url> and uses the resulting URL with the HTML anchor element, which is how <c:url> is meant to be used.

The examples discussed in "Accessing External Resources" on page 210 and "Accessing Resources in Foreign Contexts" on page 215 both use <c:url> to process URLs that are sent directly to the browser.

Core Warning

Don't use <c:url> to encode relative URLs used by <c:import> or <c:redirect>.

5.5 The <c:param> Action

The <c:param> action specifies a request parameter that is used by enclosing <c:import>, <c:redirect>, or <c:url> actions. The <c:param> action can be used with the following syntax:[7]

<c:param name value/>

The <c:param> action encodes the values specified for its `name` and `value` attributes. Instead of specifying the value for a request parameter with the `value` attribute, you can also specify that value in the body of a <c:param> action with this syntax:

```
<c:param name>
   value
</c:param>
```

Now that we have a basic understanding of the JSTL URL actions, let's see how to put them to use with three real-world examples, as discussed in the following sections.

5.6 Accessing External Resources

This section discusses a Web application, shown in Figure 5–3, that illustrates how you can use JSTL URL actions to access external resources by scraping book information from Amazon.com. The application consists of two JSP pages that use the <c:import>, <c:url>, and <c:param> actions.

The top picture in Figure 5–3 shows the JSP page that serves as the application's welcome page. That page creates four links that are created by HTML `anchor` elements. The corresponding URLs for those links are created by <c:url> actions

7. See "<c:param>" on page 491 for a more complete description of <c:param> syntax.

with nested <c:param> actions. The rest of the pictures in Figure 5–3 show information for each of the books listed in the welcome page. That information is scraped from Amazon.com with a combination of <c:import> actions and the <str:nestedString> action from the Jakarta String Tag Library.[8]

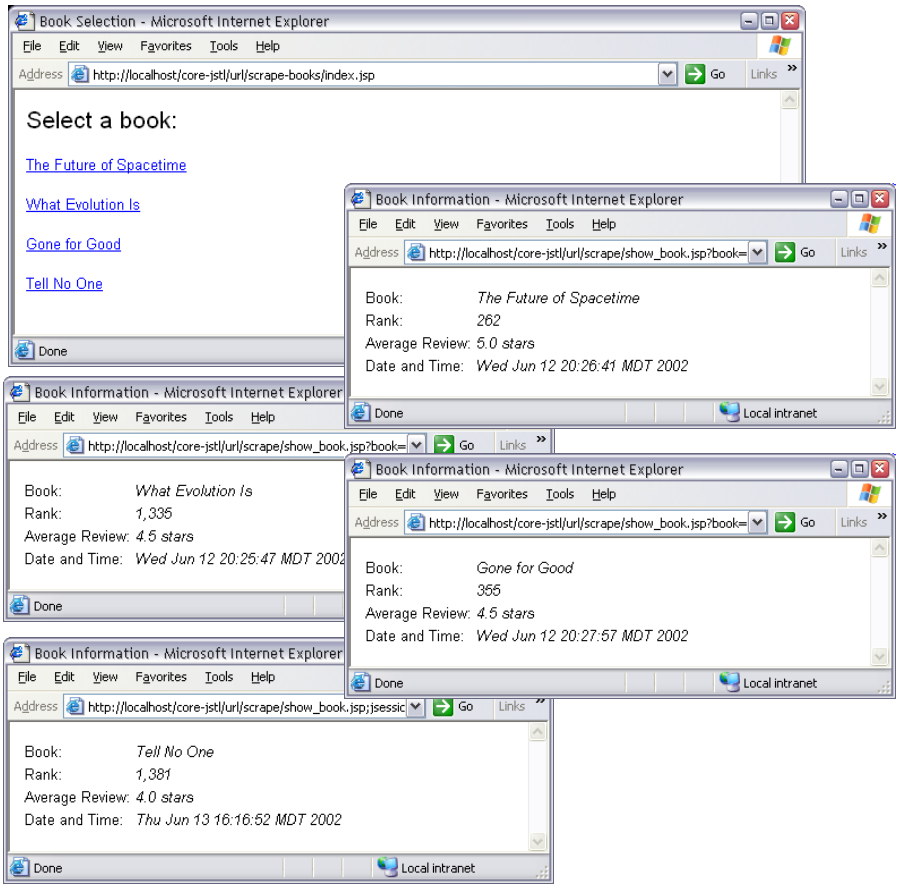

Figure 5–3 Scraping Book Information from Amazon.com

The JSP page shown in the top picture in Figure 5–3 is listed in Listing 5.4.

The preceding JSP page uses four <c:url> actions with nested <c:param> actions to create four URLs that all point to `show_book.jsp`. That JSP page is specified

8. You can download the Jakarta String Tag Library from `http://jakarta.apache.org/builds/jakarta-taglibs/nightly/projects/string`.

Listing 5.4 *Creating the Book URLs*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>Book Selection</title>
   </head>

   <body>
      <%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>

      <font size='5'>
         Select a book:
      </font><p>

      <%-- Create URLs for each book with a page-relative path
           to show_book.jsp and a request parameter named bookUrl
           whose value represents the book's URL on Amazon.com.
           Those URLs are stored in page-scoped variables that
           are used to create HTML links below --%>

      <c:url var='theFutureOfSpacetime' value='show_book.jsp'>
        <c:param name='bookUrl' value='http://www.amazon.com/exec/
obidos/ASIN/0393020223/ref=pd_sim_books_1/102-5303437-2118551'/>
      </c:url>

      <c:url var='whatEvolutionIs' value='show_book.jsp'>
        <c:param name='bookUrl' value='http://www.amazon.com/exec/
obidos/ASIN/0465044255/ref=pd_sim_books_4/102-5303437-2118551'/>
      </c:url>

      <c:url var='goneForGood' value='show_book.jsp'>
        <c:param name='bookUrl' value='http://www.amazon.com/exec/
obidos/ASIN/038533558X/ref=pd_sim_books_3/102-5303437-2118551'/>
      </c:url>

      <c:url var='tellNoOne' value='show_book.jsp'>
        <c:param name='bookUrl' value='http://www.amazon.com/exec/
obidos/ASIN/0440236703/qid=1023935482/sr=8-1/ref=sr_8_1/
104-6556245-7867920'/>
      </c:url>

      <%-- Create HTML links for each book using the URLs stored
           in page-scoped variables that were created above by
           <c:url> --%>
```

Listing 5.4 *Creating the Book URLs (cont.)*

```
      <a href='<c:out value="${theFutureOfSpacetime}"/>'>
         The Future of Spacetime
      </a><p>

      <a href='<c:out value="${whatEvolutionIs}"/>'>
         What Evolution Is
      </a><p>

      <a href='<c:out value="${goneForGood}"/>'>
         Gone for Good
      </a><p>

      <a href='<c:out value="${tellNoOne}"/>'>
         Tell No One
      </a><p>

   </body>
</html>
```

with the <c:url> action's `value` attribute as a page-relative URL. Each of the four URLs created by the <c:url> actions also has a request parameter named `bookUrl` whose value represents an external URL that points to the respective book's page on Amazon.com. Each of the four <c:url> actions stores its processed URLs in page-scoped variables whose names correspond to the books that they represent. Subsequently, four HTML `anchor` elements are created to reference the values stored in those scoped variables. When a user clicks on one of those anchors, control is transferred to `show_book.jsp`, which is listed in Listing 5.5.

Listing 5.5 *show_book.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>Book Information</title>
   </head>

   <body>
      <%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>

      <%-- Declare the Jakarta Strings tag library --%>
      <%@ taglib uri='WEB-INF/string.tld' prefix='str'%>
      <%-- Import the page from Amazon.com using the bookUrl
           request parameter --%>
```

Listing 5.5 *show_book.jsp (cont.)*

```
<c:import var='book' url='${param.bookUrl}'/>

<%-- Store today's date and time in page scope --%>
<jsp:useBean id='now' class='java.util.Date'/>

<table>
   <tr>
      <td>Book:</td>
      <td><i>
         <%-- Show the book title --%>
         <str:nestedString open='buying info: '
                           close='&lt;/title&gt;'>
            <c:out value='${book}'/>
         </str:nestedString>
      </td></i>
   </tr>
   <tr>
      <td>Rank:</td>
      <td><i>
         <%-- Show the book rank --%>
         <str:nestedString open='Sales Rank: &lt;/b&gt; '
                           close='&lt;/span&gt;'>
            <c:out value='${book}'/>
         </str:nestedString>
      </td></i>
   </tr>
   <tr>
      <td>Average Review:</td>
      <td><i>
         <%-- Show the average review --%>
         <str:replace replace='-' with='.'>
            <str:nestedString open='stars-' close='.gif'>
               <c:out value='${book}'/>
            </str:nestedString> stars
         </str:replace>
      </td></i>
   </tr>
   <tr>
      <td>Date and Time:</td>
      <td><i>
         <c:out value='${now}'/>
      </td></i>
   </tr>
</table>
</body>
</html>
```

The preceding JSP page uses <c:import> to import content from Amazon.com with the URL specified by the `bookUrl` request parameter. The `var` attribute is specified for the <c:import> actions so that the imported content is stored in a string that is referenced by a page-scoped variable named `book`. Subsequently, the preceding JSP page uses <jsp:useBean> to create a date representing the current date and time. Finally, the JSP page uses the <str:nestedString> action from the Jakarta String Tag Library—which extracts a substring specified with strings that precede and follow the substring—to extract the book's title, sales rank, and average review from the string stored in the `book` page-scoped variable. The preceding JSP page also displays the current date and time with the scoped variable created by the <jsp:useBean> action at the top of the page.

Disclaimer: *Scraping information from webpages is inherently risky business, because it relies on the absolute position of static text in a webpage's HTML; if the HTML is modified, you may have to change the code that scrapes information. As this book went to press, the example discussed in this section worked as advertised, but if Amazon.com modifies their webpage format, it may break that example.*

5.7 Accessing Resources in Foreign Contexts

As websites grow, it's often convenient to encapsulate distinct functionality in separate Web applications. For example, if you develop open-source software, you may find it convenient to implement a Web application that documents your product and another Web application that provides examples that potential customers can try. From the perspective of a single Web application, other Web applications in the same website are known as foreign contexts.

Websites that have multiple Web applications often need to share resources between those applications. This section shows you how to use <c:import> to access resources in a foreign context. Before we proceed with the example, however, you should know that not all JSP containers support accessing resources that reside in foreign contexts. The example discussed in this section was tested with Tomcat 4.1, which lets you enable cross-context access with a special attribute that you specify when you declare your Web applications. Other JSP containers, such as Resin, do not support cross-context access.

Core Warning

Not all JSP containers support accessing resources in foreign contexts .

Two Web applications are used in the following example. Those Web applications and their pertinent JSP files are depicted in Figure 5–4.

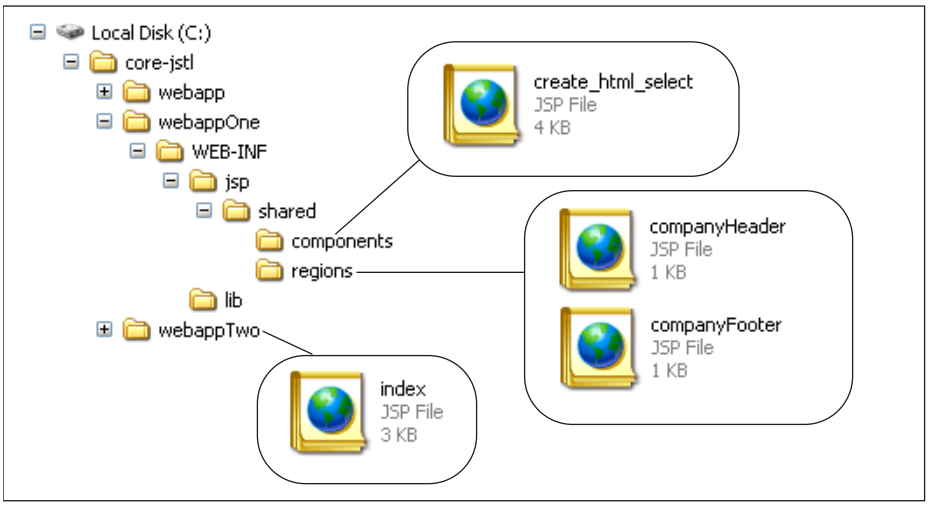

Figure 5–4 Two Web Applications and Their Contents

In the following example, `index.jsp` from the `webappTwo` application accesses `companyHeader.jsp`, `companyFooter.jsp`, and `create_html_select.jsp` from the `webappOne` application. Tomcat 4.1 requires you to specify those contexts with a `crossContext` attribute in the `Context` element in `server.xml`. Here is an excerpt from `server.xml` for the Web applications shown in Figure 5–4:

```
...
<Context path="/core-jstl/url/webappOne"
      docBase="C:/core-jstl/webappOne"
 crossContext="true"/>

<Context path="/core-jstl/url/webappTwo"
      docBase="C:/core-jstl/webappTwo"
 crossContext="true"/>
...
```

The `webappTwo` application, which consists of a single JSP page—`index.jsp`—is shown in Figure 5–5. That application lets you make a donation by filling out a form, as shown in the top picture in Figure 5–5. If you specify less than $1000 for your donation, the JSP page is redisplayed with the original information that you entered and you are asked to increase your donation, as you can see from the bottom picture in Figure 5–5.

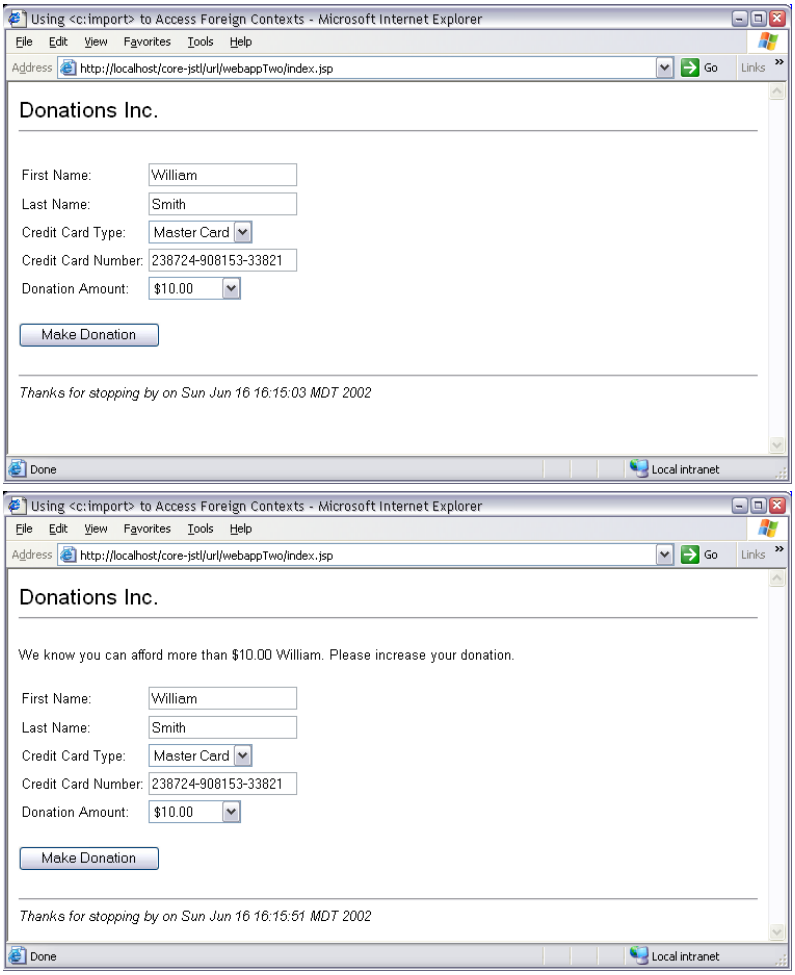

Figure 5–5 Accessing Resources in a Foreign Context with <c:import> and <c:param>

The JSP page shown in Figure 5–5 is listed in Listing 5.6.

There are three points of interest in the preceding JSP page. First, that JSP page uses <c:import> to import a header and a footer from the `webappOne` application. Second, the JSP page also uses <c:import> to import a JSP page from `webappOne` that creates HTML `select` elements. Finally, the JSP page uses <fmt:formatNumber> to format the donation amount as currency; we discuss formatting actions in "Formatting Actions" on page 308.

The header and footer imported from `webappOne` are simple JSP pages that are listed in Listing 5.7 and Listing 5.8, respectively.

Listing 5.6 *Accessing Foreign Contexts*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>
         Using &lt;c:import&gt; to Access Foreign Contexts
      </title>
   </head>

   <body>
      <%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>
      <%@ taglib uri='http://java.sun.com/jstl/fmt'  pre-
fix='fmt'%>

      <%-- Import a header shared among Web applications for this
           Web site --%>
      <c:import url='/WEB-INF/jsp/shared/regions/company-
Header.jsp'
            context='/core-jstl/url/webappOne'/>
      <p>

      <%-- If the user didn't give enough, ask for more --%>
      <c:if test='${param.amount < 1000}'>
         We know you can afford more than
         <fmt:formatNumber value='${param.amount}'
                           type='currency'/>
         <c:out value='${param.first}'/>. Please increase
         your donation.
      </c:if>

     <%-- Create a page-scoped variable -- for readability only --
           that represents the context-relative URL for
           create_html_select.jsp, which resides in the
           /core-jstl/url/webappOne context --%>
      <c:set var='create_html_component'>
         /WEB-INF/jsp/shared/components/create_html_select.jsp
      </c:set>

      <%-- This form does not specify an action, so this JSP page
           is reloaded when the submit button is activated --%>
      <form method='post'>
         <table>
            <tr>
               <td>First Name:</td>
               <td><input type='text' name='first'
                          value='<c:out value="${param.first}"/>'/>
               </td>
            </tr>
```

Listing 5.6 *Accessing Foreign Contexts (cont.)*

```
            <tr>
               <td>Last Name:</td>
               <td><input type='text' name='last'
                         value='<c:out value="${param.last}"/>'/>
               </td>
            </tr>
            <tr>
               <td>Credit Card Type:</td>
               <td>
                  <c:import url='${create_html_component}'
                        context='/core-jstl/url/webappOne'>
                     <c:param name='selectName' value='cardType'/>
                     <c:param name='items'
                              value='Visa, Master Card, Discover'/>
                  </c:import>
               </td>
            </tr>
            <tr>
               <td>Credit Card Number:</td>
               <td><input type='text' name='cardNumber'
                    value='<c:out value="${param.cardNumber}"/>'/>
               </td>
            </tr>
            <tr>
               <td>Donation Amount:</td>
               <td>
                  <c:import url='${create_html_component}'
                        context='/core-jstl/url/webappOne'>
                     <c:param name='selectName' value='amount'/>
                     <c:param name='items'
                              value='10,100,1000,10000'/>

                     <c:param name='formatDisplay'
                              value='currency'/>
                  </c:import>
               </td>
            </tr>
         </table>
         <p><input type='submit' value='Make Donation'/>
      </form>

      <%-- Import a footer shared among Web applications for this
           Web site --%>
      <c:import url='/WEB-INF/jsp/shared/regions/company-
Footer.jsp'
           context='/core-jstl/url/webappOne'/>
   </body>
</html>
```

Listing 5.7 */WEB-INF/jsp/shared/regions/companyHeader.jsp (from /core-jstl/url/webappOne context)*

```
<font size='5'>
   Donations Inc.
</font>
<hr>
```

Listing 5.8 */WEB-INF/jsp/shared/regions/companyFooter.jsp (from /core-jstl/url/webappOne context)*

```
<%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>

<hr>
<jsp:useBean id='now' class='java.util.Date'/>
<i>Thanks for stopping by on <c:out value='${now}'/></i>
```

As you can see from Figure 5–5 on page 217, the HTML `select` elements created by that JSP page retain their values when the page is reloaded. As discussed in "Retaining Values for HTML Option Elements" on page 129, that behavior can be implemented with the <c:if> and <c:out> actions; for example, you could implement the donation amount element like this:

```
<%-- Create the HTML select element for the donation amount --%>
<select name ='amount'>

   <%-- For each item displayed by this select element, ... --%>
   <c:forEach var='item' items='10,100,1000,10000'>

      <%-- Create the starting option element--%>
      <option

      <%-- If the current item is the same as the last amount
           specified, generate a "select" string contained in the
           option start tag --%>
      <c:if test='${param.amount == item}'>
         selected
      </c:if>

      <%-- Generate the value for the option element --%>
      value='<c:out value="${item}"/>'

      <%-- Close off the option element start tag --%>
      >
```

```
      <%-- Generate the display value as currency for the
           option element body --%>
      <fmt:formatNumber value='${item}' type='currency'/>

      <%-- Generate the option element end tag --%>
      </option>
   </c:forEach>
</select>
```

The preceding code fragment creates an HTML `select` element for donation amounts. That `select` element retains the previously entered donation amount and formats the displayed amount as currency. It's not overly difficult to create that HTML `select` element with the preceding code, but nonetheless, you must remember the algorithm and know how to use the <c:forEach>, <c:out>, <c:if> and <fmt:formatNumber> actions. It would be beneficial if you could encapsulate that algorithm in a JSP page so you don't have to recall the algorithm every time you need to create an HTML `select` element that retains its values. The JSP page listed in Listing 5.6 on page 218 uses <c:import> and <c:param> to import a JSP page that creates HTML `select` elements. Here's how the JSP page listed in Listing 5.6 on page 218 imports that JSP page:

```
<html>
   ...
   <body>
      ...
      <%-- Create a page-scoped variable -- for readability only --
           that represents the context-relative URL for
           create_html_select.jsp, which resides in the
           /core-jstl/url/webappOne context --%>

      <c:set var='create_html_component'>
         /WEB-INF/jsp/shared/components/create_html_select.jsp
      </c:set>

      <%-- This form does not specify an action, so this JSP page
           is reloaded when the submit button is activated --%>
      <form method='post'>
         <table>
            ...
            <tr>
               <td>Donation Amount:</td>
               <td>
                  <c:import url='${create_html_component}'
                        context='/core-jstl/url/webappOne'>
                     <c:param name='selectName' value='amount'/>
                     <c:param name='items'
```

```
                                    value='10,100,1000,10000'/>
                        </c:import>
                    </td>
                </tr>
                ...
            </table>
            ...
        </form>
        ...
    </body>
</html>
```

The preceding code fragment creates a scoped variable, solely for readability, that contains a string representing the URL of the JSP page—`create_html_select.jsp`—that creates HTML `select` elements that retain their values. That URL is a context-relative path, out of necessity because the JSP page that it references resides in a foreign context. The scoped variable is subsequently used to specify the `url` attribute for the <c:import> action, which also specifies the foreign context with its `context` attribute. The <c:import> action contains two <c:param> actions that specify a `selectName` request parameter whose value is `amount` and an `items` request parameter whose value is the comma-separated string `10,100,1000,10000`.

The `create_html_select.jsp` JSP page is listed in Listing 5.9.

Listing 5.9 */WEB-INF/jsp/shared/components/create_html_select.jsp (from /core-jstl/url/webappOne context)*

```
<%--  This handy JSTL component creates an HTML select
      element that retains its value if its page is reloaded.

      This component can be passed the following request
      parameters:

      selectName:    The name of the select element
      items:         The option values
      formatValue:   How option values should be formatted
      formatDisplay: How option display values should be formatted

      The items parameter must be a comma-separated string, and
      the formatValue and formatDisplay parameters must be one
      of the following: number, percent, or currency.

      Here's an example of how you'd use this component:
```

Listing 5.9 */WEB-INF/jsp/shared/components/create_html_select.jsp (from /core-jstl/url/webappOne context) (cont.)*

```
      <form>
         <table>
            <tr>
               <td>Select a value:</td>
               <td>
                  <c:import url='create_html_select.jsp'>
                     <c:param name='selectName' value='amount'/>
                    <c:param name='items'      value='1,2,3,4,5'/>

                     <c:param name='formatValue'
                              value='currency'/>

                     <c:param name='formatDisplay'
                              value='currency'/>
                  </c:import>
               </td>
            </tr>
         </table>
         <p><input type='submit'/>
      </form>

      The preceding code creates an HTML select element with
      the values 1-5. Those values are formatted as currency, so
      they look like this for the US English locale: $1.00,
      $2.00, ... $5.00.
--%>

<%@ taglib uri='http://java.sun.com/jstl/core' prefix='c'  %>
<%@ taglib uri='http://java.sun.com/jstl/fmt'  prefix='fmt'%>

<%-- Set a page-scoped variable representing the last value
     selected for the HTML select element generated by this
     component --%>
<c:set var='lastValue' value='${param[param.selectName]}'/>

<%-- Create the HTML select element --%>
<select name='<c:out value="${param.selectName}"/>'>
   <c:forEach var='item' items='${param.items}'>
      <%-- Start the option starting tag --%>
      <option

      <%-- If this item was the last item selected, add the
           string selected to the option starting tag --%>
      <c:if test='${item == lastValue}'>
         selected
```

Listing 5.9 *(/WEB-INF/jsp/shared/components/create_html_select.jsp (from /core-jstl/url/webappOne context) (cont.)*

```
      </c:if>

      <%-- If the option's value should be formatted, format
           it with the formatting type (number, percent, or
           currency) specified with the formatValue parameter --%>
      <c:choose>
         <c:when test='${not empty param.formatValue}'>
            <fmt:formatNumber var='value' value='${item}'
                              type='${param.formatValue}'/>
               <c:out escapeXml='false' value='value=${value}'/>
         </c:when>

         <c:otherwise>
            value='<c:out value="${item}"/>'
         </c:otherwise>
      </c:choose>

      <%-- Close off the <option> starting tag --%>
      >

      <%-- If the option's display value should be formatted,
           format it with the formatting type (number, percent,
           or currency) specified with the formatDisplay
           parameter --%>
      <c:choose>
         <c:when test='${not empty param.formatDisplay}'>
            <fmt:formatNumber var='value' value='${item}'
                              type='${param.formatDisplay}'/>
            <c:out value='${value}'/>
         </c:when>

         <c:otherwise>
            <c:out value='${item}'/>
         </c:otherwise>
      </c:choose>

      <%-- Add the <option> end tag --%>
      </option>
   </c:forEach>
</select>
```

The preceding JSP page encapsulates the creation of HTML `select` elements that retain their values. That JSP page can be passed four request parameters that represent the name of the `select` element, the items it displays, and how the

element's values and display values should be formatted. *That JSP page represents a reusable component that can be used by multiple Web applications*; therefore, the example in this section shows how to access that component from a foreign context. If your JSP container does not support accessing resources in foreign contexts, you can still take advantage of components like the preceding JSP page by accessing them with an absolute URL.

5.8 Redirecting a Response

Before JSTL, the only way to redirect an HTTP response in a Java-based Web application was to use the `HttpServletResponse.sendRedirect` method. JSTL makes redirecting HTTP responses much easier with the <c:redirect> action, as illustrated by the application shown in Figure 5–6.

The application shown in Figure 5–6 logs access to external resources, which are JavaWorld articles that discuss Java design patterns. The application consists of two JSP pages. One of those JSP pages, shown in the top picture in Figure 5–6, uses the <c:url> and <c:param> JSTL actions, in conjunction with HTML `anchor` element, to provide links to five JavaWorld articles. Instead of pointing directly to the articles, those links point to a second JSP page that's passed the article's URL as a request parameter. The second JSP page, which is not shown in Figure 5–6, logs information about the links that were selected in the first JSP page and redirects the HTTP response to the JavaWorld article in question. The bottom pictures in Figure 5–6 show two of those articles.

The second JSP page sends information to the standard servlet log; that information looks like this:

```
Remote host 127.0.0.1 accessed Decorator Design Pattern article on
Wed Jun 12 13:31:09 MDT 2002
...
Remote host 127.0.0.1 accessed Composite Design Pattern article on
Wed Jun 12 13:31:44 MDT 2002
```

The information stored in the log file provides information about the remote host that accessed the article, the name of the article, and the date and time when the access occurred.

The JSP page shown in the top picture in Figure 5–6 is listed in Listing 5.10.

For readability, the preceding JSP page uses <c:set> actions to create scoped variables that reference the JavaWorld article URLs. Subsequently, the JSP page uses <c:url> with enclosed <c:param> actions to create five URLs that all point to a JSP page named `log_access.jsp`. Those URLs all contain a request parameter named `page` whose value represents the JavaWorld article's URL and a request

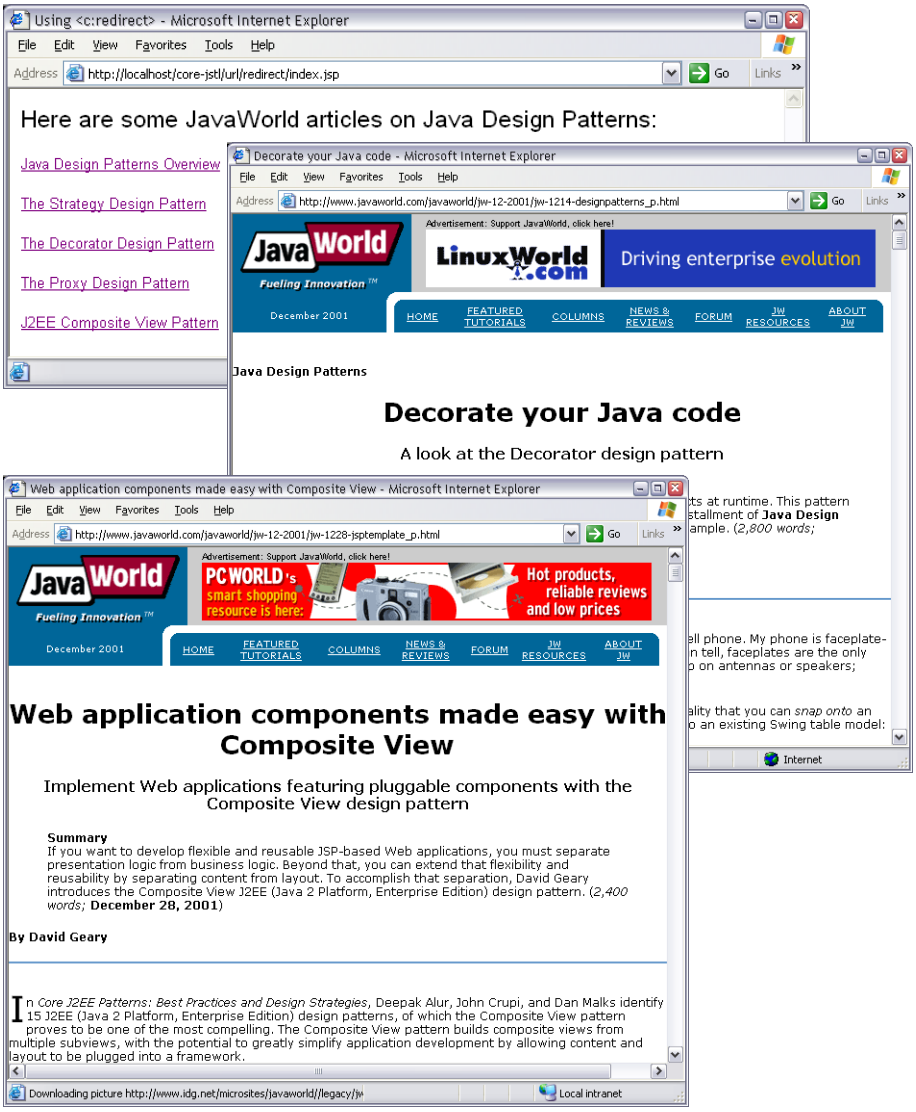

Figure 5–6 Use <c:redirect> to Log Access to External Resources

Listing 5.10 *Creating Article URLs*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>Using &lt;c:redirect&gt;</title>
   </head>

   <body>
      <%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>

      <%-- Define a prefix variable for readability --%>
      <c:set var='prefix'
         value='http://www.javaworld.com/javaworld/'/>

      <%-- The base URL for the overview article --%>
      <c:set var='overview'
       value='${prefix}jw-10-2001/jw-1012-designpatterns_p.html'/>

      <%-- The base URL for the strategy article --%>
      <c:set var='strategy'
       value='${prefix}jw-04-2002/jw-0426-designpatterns_p.html'/>

      <%-- The base URL for the decorator article --%>
      <c:set var='decorator'
       value='${prefix}jw-12-2001/jw-1214-designpatterns_p.html'/>

      <%-- The base URL for the proxy article --%>
      <c:set var='proxy'
       value='${prefix}jw-02-2002/jw-0222-designpatterns_p.html'/>

      <%-- The base URL for the composite article --%>
      <c:set var='composite'
       value='${prefix}jw-12-2001/jw-1228-jsptemplate_p.html'/>

      <%-- The encoded URL for the overview article --%>
      <c:url var='overviewUrl' value='log_access.jsp'>
         <c:param name='page' value='${overview}'/>
         <c:param name='name' value='Design Patterns Overview'/>
      </c:url>

      <%-- The encoded URL for the strategy article --%>
      <c:url var='strategyUrl' value='log_access.jsp'>
         <c:param name='page' value='${strategy}'/>
         <c:param name='name' value='Strategy Design Pattern'/>
      </c:url>
```

Listing 5.10 *Creating Article URLs (cont.)*

```
<%-- The encoded URL for the decorator article --%>
<c:url var='decoratorUrl' value='log_access.jsp'>
   <c:param name='page' value='${decorator}'/>
   <c:param name='name' value='Decorator Design Pattern'/>
</c:url>

<%-- The encoded URL for the proxy article --%>
<c:url var='proxyUrl' value='log_access.jsp'>
   <c:param name='page' value='${proxy}'/>
   <c:param name='name' value='Proxy Design Pattern'/>
</c:url>

<%-- The encoded URL for the composite article --%>
<c:url var='compositeUrl' value='log_access.jsp'>
   <c:param name='page' value='${composite}'/>
   <c:param name='name' value='Composite Design Pattern'/>
</c:url>

<font size='5'>
   Here are some JavaWorld articles on Java Design Patterns:
</font><p>

<%-- Links to articles with URLs created above --%>
<a href='<c:out value="${overviewUrl}"/>'>
   Java Design Patterns Overview
</a><p>
<a href='<c:out value="${strategyUrl}"/>'>
   The Strategy Design Pattern
</a><p>
<a href='<c:out value="${decoratorUrl}"/>'>
   The Decorator Design Pattern
</a><p>
<a href='<c:out value="${proxyUrl}"/>'>
   The Proxy Design Pattern
</a><p>
<a href='<c:out value="${compositeUrl}"/>'>
   J2EE Composite View Pattern
</a>
</body>
</html>
```

parameter named `name` that represents the name of the article. Finally, the JSP page creates five HTML `anchor` elements that reference the URLs that point to `log_access.jsp`. That JSP page is listed in Listing 5.11.

Listing 5.11 *log_access.jsp*

```
<%@ taglib uri='http://java.sun.com/jstl/core' prefix='c' %>

<%-- Log a message in the log file --%>
<% application.log("Remote host " + request.getRemoteHost() +
                   " accessed " + request.getParameter("name") +
                   " article on " + new java.util.Date()); %>

<%-- Redirect the response to the specified page --%>
<c:redirect url='${param.page}'/>
```

The preceding JSP page uses a scriptlet to write a message to the application log file and subsequently uses <c:redirect> to redirect the response to the JavaWorld article.