TRAINING & REFERENCE

# murach's Java servlets and JSP

## 2ND EDITION

## (Chapter 11)

Thanks for downloading this chapter from ***Murach's Java Servlets and JSP (2nd Edition)***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **web site**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books for professional developers.

Thanks for your interest in our books!

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
**murachbooks@murach.com** • **www.murach.com**

# Contents

# 11

# How to use the JSP Standard Tag Library (JSTL)

In chapter 10, you learned how to use the Expression Language (EL) that was introduced with JSP 2.0 to reduce the amount of scripting in your applications. Now, in this chapter, you'll learn how to use the JSP Standard Tag Library (JSTL) to further reduce the amount of scripting in your applications. In fact, for most applications, using JSTL and EL together makes it possible to remove all scripting.

# An introduction to JSTL

The *JSP Standard Tag Library* (*JSTL*) provides tags for common tasks that need to be performed in JSPs.

## The JSTL libraries

Figure 11-1 shows the five tag libraries that are included with JSTL 1.1. In this chapter, you'll learn the details for working with the common tags in the core library. This library contains tags that you can use to encode URLs, loop through collections, and code if/else statements. If you use the MVC pattern, the tags in the core library are often the only JSTL tags you'll need as you develop your JSPs. If necessary, though, you can use the other four libraries to work with internationalization, databases, XML, or strings.

## How to make the JSTL JAR files available to your application

Before you can use JSTL tags within an application, you must make the jstl.jar and standard.jar files available to the application. With the NetBeans IDE, for example, you can add the JSTL 1.1 library to the application as shown in figure 3-17 in chapter 3. Then, the jstl.jar and standard.jar files will be shown beneath the Libraries folder in the Projects window.

## How to code the taglib directive

Before you can use JSTL tags within a JSP, you must code a taglib directive to specify the URI and prefix for the JSTL library. In this figure, for example, the taglib directive specifies the URI for the JSTL core library with a prefix of c, which is the prefix that's typically used for this library. In fact, all of the examples in this chapter assume that the page includes a taglib directive like this one before the JSTL tags are used. Although you can use different prefixes than the ones in this figure, we recommend using the standard prefixes.

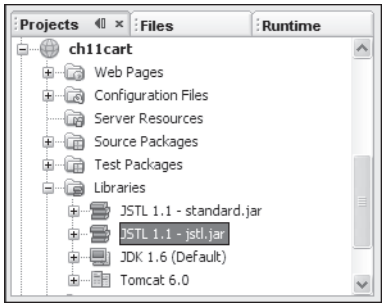
## How to code a JSTL tag

Once you've added the appropriate JAR files to your application and used the taglib directive to identify a library, you can code a JSTL tag. In this figure, for example, the url tag is used to encode a URL that refers to the index.jsp file in the web applications root directory. Note how the prefix for this tag is c. Also note how this tag looks more like an HTML tag, which makes it easier to code and read than the equivalent JSP script, especially for web designers and other nonprogrammers who are used to HTML syntax.

### The primary JSTL libraries

| Name | Prefix | URI | Description |
|---|---|---|---|
| Core | c | http://java.sun.com/jsp/jstl/core | Contains the core tags for common tasks such as looping and if/else statements. |
| Formatting | fmt | http://java.sun.com/jsp/jstl/fmt | Provides tags for formatting numbers, times, and dates so they work correctly with internationalization (i18n). |
| SQL | sql | http://java.sun.com/jsp/jstl/sql | Provides tags for working with SQL queries and data sources. |
| XML | x | http://java.sun.com/jsp/jstl/xml | Provides tags for manipulating XML documents. |
| Functions | fn | http://java.sun.com/jsp/jstl/functions | Provides functions that can be used to manipulate strings. |

### The NetBeans IDE after the JSTL 1.1 library has been added




### The taglib directive that specifies the JSTL core library

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

### An example that uses JSTL to encode a URL

#### JSP code with JSTL

```
<a href="<c:url value='/index.jsp' />">Continue Shopping</a>
```

#### Equivalent script

```
<a href="<%=response.encodeURL("index.jsp")%>">Continue Shopping</a>
```

### Description

- The *JSP Standard Tag Library* (*JSTL*) provides tags for common JSP tasks.
- Before you can use JSTL tags within an application, you must make the jstl.jar and standard.jar files available to the application. To do that for NetBeans, you can add the JSTL 1.1 class library to your project as in figure 3-17 in chapter 3. Otherwise, you can consult the documentation for your IDE.
- Before you can use JSTL tags within a JSP, you must code a taglib directive that identifies the JSTL library and its prefix.

Figure 11-1 An introduction to JSTL

## How to view the documentation for a library

As you progress through this chapter, you'll learn how to code the tags in the JSTL core library that you'll use most of the time. If necessary, though, you can view the documentation for any of the tags in this library as shown in figure 11-2.

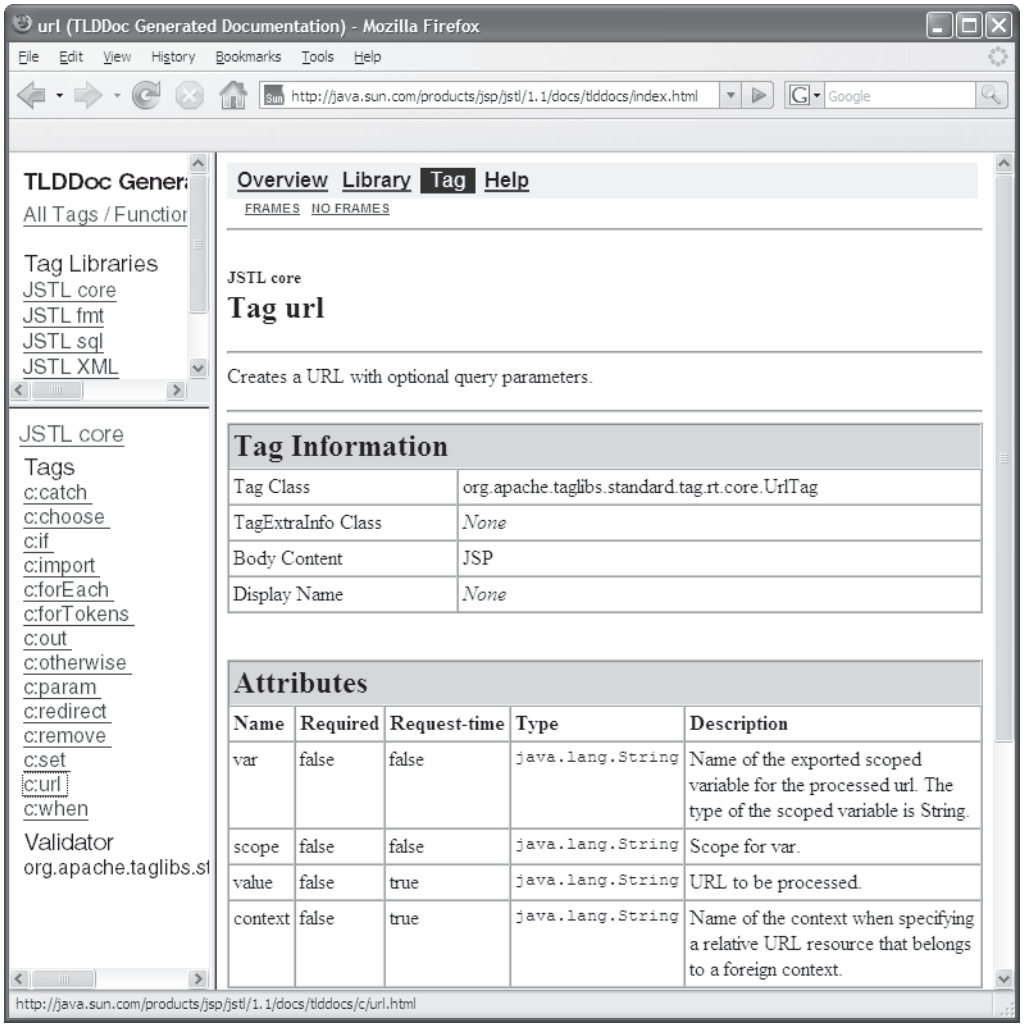
If, for example, you want to learn more about the url tag in the core library, you can click on the "JSTL core" link in the upper left window. Then, you can click on the "c:url" link in the lower left window to display the documentation for this tag in the window on the right. This documentation provides a general description of the tag, a list of all available attributes for the tag, and detailed information about each of these attributes.

You can also use this documentation to learn more about the JSTL libraries that aren't covered in this chapter. If, for example, you want to learn more about the formatting library for working with internationalization, you can click on the "JSTL fmt" link in the upper left window. Then, you can click on the tags in the lower left window to display information in the window on the right. Incidentally, *i18n* is sometimes used as an abbreviation for *internationalization* because *internationalization* begins with an *i*, followed by *18* letters, followed by an *n*.

## The URL for the JSTL 1.1 documentation

```
http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
```

## A browser that displays the JSTL documentation




## Description

- To view the documentation for the JSTL 1.1 library, use your browser to visit the URL shown above. Then, you can use the upper left window to select the JSTL library, the lower left window to select the tag, and the window on the right to get information about the tag.

Figure 11-2 How to view the documentation for a library

# How to work with the JSTL core library

Now that you have a general idea of how JSTL works, you're ready to learn the details for coding the most commonly used JSTL tags. All of these tags are available from the JSTL core library.

## How to use the url tag

In chapter 8, you learned how to encode the URLs that are returned to the client so your application can track sessions even if the client doesn't support cookies. Since you usually want your application to do that, you typically encode the URLs in your applications. Without JSTL, though, this requires calling the encodeURL method of the response object from a script within a JSP. With JSTL, you can use the url tag to encode URLs without using scripting.

Figure 11-3 shows how to use the url tag. Here, the first example shows the same url tag that's presented in figure 11-1. This url tag encodes a relative URL that refers to the index.jsp file in the root directory for the web application. Its value attribute is used to specify the URL.

When you specify JSTL tags, you need to be aware that they use XML syntax, not HTML syntax. As a result, you must use the exact capitalization shown in this example for the name of the tag and its attributes. In addition, attributes must be enclosed in either single quotes or double quotes. In this figure, I have used both single and double quotes to differentiate between the href attribute of the A tag (which uses double quotes), and the value attribute of the url tag (which uses single quotes). I think this improves the readability of this code.

The second example shows how to use the url tag to encode a URL that includes a parameter named productCode with a hard-coded value of 8601. Then, the third example shows how to use the url tag to encode a URL that includes a parameter named productCode with a value that's supplied by an EL expression. Here, the EL expression gets the code property of a Product object named product.

The third example also shows how you can code a JSTL param tag within a url tag to specify the name and value for a parameter. The benefit of using this tag is that it automatically encodes any unsafe characters in the URL, such as spaces, with special characters, such as plus signs.

If you compare the url tags in these examples with the equivalent scripting, I think you'll agree that the JSTL tags are easier to code, read, and maintain. In addition, the syntax is closer to HTML than scripting, which makes it easier for web designers and other nonprogrammers to use.

## An example that encodes a URL

### JSP code with JSTL

```
<a href="<c:url value='/index.jsp' />">Continue Shopping</a>
```

### Equivalent scripting

```
<a href="<%=response.encodeURL("index.jsp")%>">Continue Shopping</a>
```

## An example that adds a parameter to the URL

### JSP code with JSTL

```
<a href="<c:url value='/cart?productCode=8601' />">
   Add To Cart
</a>
```

### Equivalent scripting

```
<a href="<%=response.encodeURL("cart?productCode=8601")%>">
    Add To Cart
</a>
```

## An example that uses EL to specify the value of a parameter value

### JSP code with JSTL

```
<a href="<c:url value='/cart?productCode=${product.code}' />">
    Add To Cart
</a>
```

### The same code with the JSTL param tag

```
<a href="
     <c:url value='/cart'>
         <c:param name='productCode' value='${product.code}' />
     </c:url>
">Add To Cart</a>
```

### Equivalent scripting

```
<%@ page import="business.Product" %>
<%
    Product product = (Product) session.getAttribute("product");
    String cartUrl = "cart?productCode=" + product.getCode();
%>
<a href="<%=response.encodeURL(cartUrl)%>">Add To Cart</a>
```

## Description

- You can use the url tag to encode URLs within your web application. This tag will automatically rewrite the URL to include a unique session ID whenever the client doesn't support cookies.
- You can use the JSTL param tag if you want to automatically encode unsafe characters such as spaces with special characters such as plus signs.

Figure 11-3 How to use the url tag

## How to use the forEach tag

You can use the forEach tag to loop through items that are stored in most collections, including arrays. For example, figure 11-4 shows how to use the forEach tag to loop through the LineItem objects that are available from the items property of the cart attribute. Here, the var attribute specifies a variable name of item to access each item within the collection. Then, the items attribute uses EL to specify the collection that stores the data. In this case, the collection is the ArrayList<LineItem> object that's returned by the getItems method of the Cart object for the current session. This Cart object has been stored as an attribute with a name of cart.

Within the forEach loop, the JSP code creates one row with four columns for each item in the cart. Here, each column uses EL to display the data that's available from the LineItem object. In particular, the first column displays the quantity, the second column displays the product description, the third column displays the price per item, and the fourth column displays the total amount (quantity multiplied by price). Note that the LineItem object includes code that applies currency formatting to the price and amount.

If you have trouble understanding the examples in this figure, you may want to study the code for the Cart, LineItem, and Product objects that are presented in figure 11-12. In particular, note how a Cart object can contain multiple LineItem objects and how a LineItem object must contain one Product object. Also, note how the appropriate get methods are provided for all of the properties that are accessed by EL. For example, the Cart class provides a method named getItems that returns an ArrayList of LineItem objects. As a result, with EL, you can use the items property of the cart attribute to get this ArrayList object.

If necessary, you can nest one forEach tag within another. For example, if you wanted to display several Invoice objects on a single web page, you could use an outer forEach tag to loop through the Invoice objects. Then, you could use an inner forEach tag to loop through the LineItem objects within each invoice. However, for most JSPs, you won't need to nest forEach statements.

If you compare the JSTL tags shown in this figure with the equivalent scripting, I think you'll agree that the benefits of the JSTL tags are even more apparent in this figure than in the last one. In particular, the JSP code that uses JSTL is much shorter and easier to read than the equivalent scripting. As a result, it's easier for web designers and other nonprogrammers to work with this code.

## An example that uses JSTL to loop through a collection

### JSP code with JSTL

```
<c:forEach var="item" items="${cart.items}">
<tr valign="top">
  <td>${item.quantity}</td>
  <td>${item.product.description}</td>
  <td>${item.product.priceCurrencyFormat}</td>
  <td>${item.totalCurrencyFormat}</td>
</tr>
</c:forEach>
```

### The result that's displayed in the browser for a cart that has two items

**Your cart**

| Quantity | Description | Price | Amount |
|---|---|---|---|
| 1 | 86 (the band) - True Life Songs and Pictures | $14.95 | $14.95 |
| 1 | Paddlefoot - The first CD | $12.95 | $12.95 |

### Equivalent scripting

```
<%@ page import="business.*, java.util.ArrayList" %>
<%
  Cart cart = (Cart) session.getAttribute("cart");
  ArrayList<LineItem> items = cart.getItems();
  for (LineItem item : items)
  {
%>
  <tr valign="top">
    <td><%=item.getQuantity()%></td>
    <td><%=item.getProduct().getDescription()%></td>
    <td><%=item.getProduct().getPriceCurrencyFormat()%></td>
    <td><%=item.getTotalCurrencyFormat()%></td>
  </tr>
<% } %>
```

## Description

- You can use the forEach tag to loop through most types of collections, including arrays.
- You can use the var attribute to specify the variable name that will be used to access each item within the collection.
- You can use the items attribute to specify the collection that stores the data.
- If necessary, you can nest one forEach tag within another.

Figure 11-4 How to use the forEach tag

## How to use the forTokens tag

You can use the forTokens tag to loop through items that are stored in a string as long as the items in the string are separated by one or more delimiters, which are characters that are used to separate the items. For instance, the string in the first example in figure 11-5 uses a comma as the delimiter. As a result, this string can be referred to as a *comma-delimited string*.

The first example in this figure also shows how to use the forTokens tag to loop through the four product codes that are stored in the string. Here, the var attribute specifies a variable name of productCode to identify each product code in the list. Then, the items attribute uses EL to specify the productCodes attribute as the string that stores the items. Finally, the delims attribute specifies the comma as the delimiter.

To keep this example simple, the servlet code creates the productCodes attribute by storing a hard-coded list of four product codes that are separated by commas. In a more realistic example, of course, the servlet code would dynamically generate this list.

The second example works similarly to the first example, but it uses two delimiters instead of one. In particular, the delims attribute specifies the at symbol (@) as the first delimiter and the period (.) as the second delimiter. As a result, the loop processes three items, one for each part of the email address.

If necessary, you can nest one forTokens tag within another. Or, you can nest a forTokens tag within a forEach tag. However, since you'll rarely need to nest forTokens tags, this technique isn't illustrated in this figure.

## An example that uses JSTL to loop through a comma-delimited string

### Servlet code

```
session.setAttribute("productCodes", "8601,pf01,pf02,jr01");
```

### JSP code

```
<p>Product codes<br>
<c:forTokens var="productCode" items="${productCodes}" delims="," >
    <li>${productCode}</li>
</c:forTokens>
</p>
```

### The result that's displayed in the browser

Product codes
- 8601
- pf01
- pf02
- jr01

## An example that uses JSTL to parse a string

### Servlet code

```
session.setAttribute("emailAddress", "jsmith@gmail.com");
```

### JSP code

```
<p>Email parts<br>
<c:forTokens var="part" items="${emailAddress}" delims="@.">
    <li>${part}</li>
</c:forTokens>
</p>
```

### The result that's displayed in the browser

Email parts
- jsmith
- gmail
- com

## Description

- You can use the forTokens tag to loop through delimited values that are stored in a string.
- You can use the var attribute to specify the variable name that will be used to access each delimited string.
- You can use the items attribute to specify the string that stores the data.
- You can use the delims attribute to specify the character or characters that are used as the delimiters for the string.
- If necessary, you can nest one forTokens tag within another.

Figure 11-5 How to use the forTokens tag

## Four more attributes for looping

When working with collections, the servlet code typically creates a collection and passes it to the JSP so the collection can be displayed to the user. Then, the JSP uses the forEach tag to loop through the collection and display it to the user as shown in figure 11-4.

However, there may be times when the JSP will need to do some additional processing. For example, the JSP may need to know whether the item is the first or last item, so it can apply special formatting to that item. Or, the JSP may need to know the item number, so it can apply shading to alternating items. In that case, you can use the attributes described in figure 11-6. These attributes work the same for the forEach and the forTokens tags.

The example in this figure shows how to work with the begin, end, and step attributes that are available for the forEach and forTokens tags. Here, the begin attribute specifies the starting index for the loop; the end attribute specifies the last index for the loop; and the step attribute specifies the amount to increment the index each time through the loop. If you understand how a for loop works in Java, you shouldn't have much trouble understanding these attributes. In this example, these attributes are used to print the first 10 numbers that are stored in an array of 30 int values.

This example also shows how to use the varStatus attribute. This attribute specifies the name of a variable that can be used to get information about the status of the loop. In particular, this variable provides four properties named first, last, index, and count that you can use within the body of a loop. For example, you can use the first and last properties to return a Boolean value that indicates whether the item is the first or last item in the collection. Or, you can use the index and count properties to return an integer value for the item. Note, however, that the index property returns an integer value that's one less than the count value. That's because the index property starts at 0 while the count property starts at 1.

## Attributes that you can use for advanced loops

| Attribute | Description |
|---|---|
| begin | Specifies the first index for the loop. |
| end | Specifies the last index for the loop. |
| step | Specifies the amount to increment the index each time through the loop. |
| varStatus | Specifies the name of a variable that can be used to get information about the status of the loop. This variable provides the first, last, index, and count properties. |

## An example that uses all four attributes

### Servlet code

```
int[] numbers = new int[30];
for (int i = 0; i < 30; i++)
{
    numbers[i] = i+1;
}
session.setAttribute("numbers", numbers);
```

### JSP code

```
<p>Numbers<br>
<c:forEach items="${numbers}" var="number"
           begin="0" end="9" step="1"
           varStatus="status">
    <li>${number} | First: ${status.first} | Last: ${status.last} |
        Index: ${status.index} | Count: ${status.count} </li>
</c:forEach>
</p>
```

### The result that's displayed in the browser

Numbers

- 1 | First: true | Last: false | Index: 0 | Count: 1
- 2 | First: false | Last: false | Index: 1 | Count: 2
- 3 | First: false | Last: false | Index: 2 | Count: 3
- 4 | First: false | Last: false | Index: 3 | Count: 4
- 5 | First: false | Last: false | Index: 4 | Count: 5
- 6 | First: false | Last: false | Index: 5 | Count: 6
- 7 | First: false | Last: false | Index: 6 | Count: 7
- 8 | First: false | Last: false | Index: 7 | Count: 8
- 9 | First: false | Last: false | Index: 8 | Count: 9
- 10 | First: false | Last: true | Index: 9 | Count: 10

## Description

- The begin, end, step, and varStatus attributes work for both the forEach and forTokens tags.

Figure 11-6 Four more attributes for looping

## How to use the if tag

When coding a JSP, you may need to perform conditional processing to change the appearance of the page depending on the values of the attributes that are available to the page. To do that, you can use the if tag as shown in figure 11-7.

To start, you code an opening if tag that includes the test attribute. In the example in this figure, this test attribute uses EL to get the count property of the cart attribute, which indicates the number of items that are in the cart. Then, the code within the opening and closing if tags displays a message that's appropriate for the number of items in the cart. In particular, the first if tag displays a message if the cart contains 1 item, and the second if tag displays a message if the cart contains more than one item. The main difference between the two messages is that the second message uses the plural (items) while the first uses the singular (item).

If necessary, you can use the var and scope attributes to expose the Boolean condition in the test attribute as a variable with the specified scope. Then, you can reuse the Boolean condition in other if statements. This works similarly to the set tag that's briefly described later in this chapter. However, since you'll rarely need to use these attributes, they aren't illustrated in this figure.

As with the forEach and forTokens tags, you can nest one if tag within another. Or, you can nest an if tag within a forEach or forTokens tag. In short, as you might expect by now, you can usually nest JSTL tags within one another whenever that's necessary.

## An example that uses JSTL to code an if statement

### JSP code with JSTL

```
<c:if test="${cart.count == 1}">
    <p>You have 1 item in your cart.</p>
</c:if>
<c:if test="${cart.count > 1}">
    <p>You have ${cart.count} items in your cart.</p>
</c:if>
```

### The result that's displayed in the browser for a cart that has two items


### Equivalent scripting

```
<%@ page import="business.Cart, java.util.ArrayList" %>
<%
  Cart cart = (Cart) session.getAttribute("cart");
  if (cart.getCount() == 1)
    out.println("<p>You have 1 item in your cart.</p>");
  if (cart.getCount() > 1)
    out.println("<p>You have " + cart.getCount() +
                " items in your cart.</p>");
%>
```

## Description

- You can use the if tag to perform conditional processing that's similar to an if statement in Java.
- You can use the test attribute to specify the Boolean condition for the if statement.
- If necessary, you can nest one if tag within another.

Figure 11-7 How to use the if tag

## How to use the choose tag

In the last figure, you learned how to code multiple if tags. This is the equivalent of coding multiple if statements in Java. However, there are times when you will need to code the equivalent of an if/else statement. Then, you can use the choose tag as described in figure 11-8.

To start, you code the opening and closing choose tags. Within those tags, you can code one or more when tags. For instance, in the example in this figure, the first when tag uses the test attribute to check if the cart contains zero items. Then, the second tag uses the test attribute to check if the cart contains one item. In either case, the when tag displays an appropriate message.

After the when tags but before the closing choose tag, you can code a single otherwise tag that's executed if none of the conditions in the when tags evaluate to true. In this example, the otherwise tag displays an appropriate message if the cart doesn't contain zero or one items. Since the number of items in a cart can't be negative, this means that the otherwise tag uses EL to display an appropriate message whenever the cart contains two or more items.

## An example that uses JSTL to code an if/else statement

### JSP code with JSTL

```
<c:choose>
    <c:when test="${cart.count == 0}">
        <p>Your cart is empty.</p>
    </c:when>
    <c:when test="${cart.count == 1}">
        <p>You have 1 item in your cart.</p>
    </c:when>
    <c:otherwise>
        <p>You have ${cart.count} items in your cart.</p>
    </c:otherwise>
</c:choose>
```

### The result that's displayed in the browser for a cart that has two items

**Your cart**

| Quantity | Description | Price | Amount |
|---|---|---|---|
| 1 | 86 (the band) - True Life Songs and Pictures | $14.95 | $14.95 |
| 1 | Paddlefoot - The first CD | $12.95 | $12.95 |

You have 2 items in your cart.

### Equivalent scripting

```
<%@ page import="business.Cart, java.util.ArrayList" %>
<%
  Cart cart = (Cart) session.getAttribute("cart");
  if (cart.getCount() == 0)
    out.println("<p>Your cart is empty.</p>");
  else if (cart.getCount() == 1)
    out.println("<p>You have 1 item in your cart.</p>");
  else
    out.println("<p>You have " + cart.getCount() +
                " items in your cart.</p>");
%>
```

## Description

- You can use the choose tag to perform conditional processing similar to an if/else statement in Java. To do that, you can code multiple when tags and a single otherwise tag within the choose tag.
- You can use the test attribute to specify the Boolean condition for a when tag.
- If necessary, you can nest one choose tag within another.

Figure 11-8 How to use the choose tag

## How to use the import tag

In chapter 7, you learned two ways to work with includes. The import tag shown in figure 11-9 provides another way to work with includes, and it works like the standard JSP include tag. In other words, it includes the file at *runtime*, not at compile-time.

Neither the standard JSP include tag or the JSTL import tag uses scripting. As a result, it usually doesn't matter which tag you use. However, the JSTL import tag does provide one advantage: it lets you include files from other applications and web servers.

For instance, the second last example in this figure shows how to use the import tag to include the footer.jsp file that's available from the musicStore application that's running on the same local server as the current web application. Then, the last example shows how to use the import tag to include the footer.jsp file that's available from the remote server for the www.murach.com web site.

### An example that imports a header file

#### JSP code with JSTL

```
<c:import url="/includes/header.html" />
```

#### Equivalent standard JSP tag

```
<jsp:include page="/includes/header.html" />
```

### An example that imports a footer file

#### JSP code with JSTL

```
<c:import url="/includes/footer.jsp" />
```

#### Equivalent standard JSP tag

```
<jsp:include page="/includes/footer.jsp" />
```

### An example that imports a file from another application

```
<c:import url="http://localhost:8080/musicStore/includes/footer.jsp" />
```

### An example that imports a file from another web server

```
<c:import url="www.murach.com/includes/footer.jsp" />
```

### Description

- The import tag includes the file at *runtime*, not at compile-time, much like the standard JSP include tag described in chapter 7.
- One advantage of the import tag over the standard JSP include tag is that it lets you include files from other applications and web servers.

Figure 11-9 How to use the import tag

## Other tags in the JSTL core library

Figure 11-10 shows six more tags in the JSTL core library. However, if you use the MVC pattern, you probably won't need to use these tags. As a result, I've only provided brief examples to give you an idea of how these tags work. If you do need to use them, though, you can look them up in the documentation for the JSTL core library as described in figure 11-2.

If you need to be able to display special characters in your JSPs, you can use the out tag as illustrated by the first example in this figure. Then, this tag automatically handles any special characters before they are displayed on the JSP. If, for example, you try to use EL by itself to display a string that contains the left and right angle brackets (< >), the JSP interprets those brackets as an HTML tag and the string isn't displayed correctly. However, if you use the out tag, these characters display correctly on the JSP.

If you need to set the value of an attribute in a scope, you can use the set tag. For instance, the second example in this figure shows how to set an attribute named message with a value of "Test message" in session scope.

You can also use the set tag if you need to set the value of a property of an attribute within a specified scope. However, instead of using the var attribute to specify the name of the attribute, you use the target attribute to specify the attribute that contains the property. To do that, you use EL within the target attribute to specify a reference to the attribute. This is illustrated by the third example.

The fourth example shows how to use the remove tag to remove an attribute from a scope. When you use this tag, you use the var attribute to specify the name of the attribute that you want to remove, and you use the scope attribute to specify the scope that contains the attribute.

If your JSP includes code that may cause an exception to be thrown, you can use the catch tag to catch the exceptions. This is illustrated by the fifth example. Here, the opening and closing catch tags are coded around a Java scriptlet that causes an ArithmeticException to be thrown due to a divide by zero error. Then, when the exception is thrown, execution jumps over the Java expression that displays the result of the calculation. However, the catch tag also exposes the exception as a variable named e. As a result, the if tag that follows the catch tag is able to display an appropriate error message.

Of course, if you edit the Java scriptlet that's in the catch tag so it performs a legal calculation, no exception will be thrown. In that case, the result of the calculation will be displayed and the error message won't be displayed.

The sixth example shows how to use the redirect tag to redirect a client to a new URL. In this case, the redirect tag is coded within an if tag so the client isn't redirected unless the condition in the if statement is true.

Although this figure doesn't include an example of the param tag, figure 11-3 does illustrate the use of this tag within the url tag. If you read through the documentation for the param tag, you'll find that you can also use it with other tags such as the import tag.

## Other tags in the JSTL core library

| Tag name | Description |
|---|---|
| out | Uses EL to display a value, automatically handling most special characters such as the left angle bracket (<) and right angle bracket (>). |
| set | Sets the value of an attribute in a scope. |
| remove | Removes an attribute from a scope. |
| catch | Catches any exception that occurs in its body and optionally creates an EL variable that refers to the Throwable object for the exception. |
| redirect | Redirects the client browser to a new URL. |
| param | Adds a parameter to the parent tag. |

## An out tag that displays a message

### Using the Value attribute

```
<c:out value="${message}" default="No message" />
```

### Using the tag's body

```
<c:out value="${message}">
    No message
</c:out>
```

## A set tag that sets a value in an attribute

```
<c:set var="message" scope="session" value="Test message" />
```

## A set tag that sets a value in a JavaBean

### JSP code with JSTL

```
<c:set target="${user}" property="firstName" value="John" />
```

### Equivalent standard JSP tag

```
<jsp:setProperty name="user" property="firstName" value="John"/>
```

## A remove tag that removes an attribute

```
<c:remove var="message" scope="session" />
```

## A catch tag that catches an exception

```
<c:catch var="e">
    <%  // this scriptlet statement will throw an exception
        int i = 1/0;
    %>
    <p>Result: <%= i %></p>
</c:catch>
<c:if test="${e != null}">
    <p>An exception occurred. Message: ${e.message}</p>
</c:if>
```

## A redirect tag that redirects to another page

```
<c:if test="${e != null}">
    <c:redirect url="/error_java.jsp" />
</c:if>
```

Figure 11-10 Other tags in the JSTL core library

# The Cart application

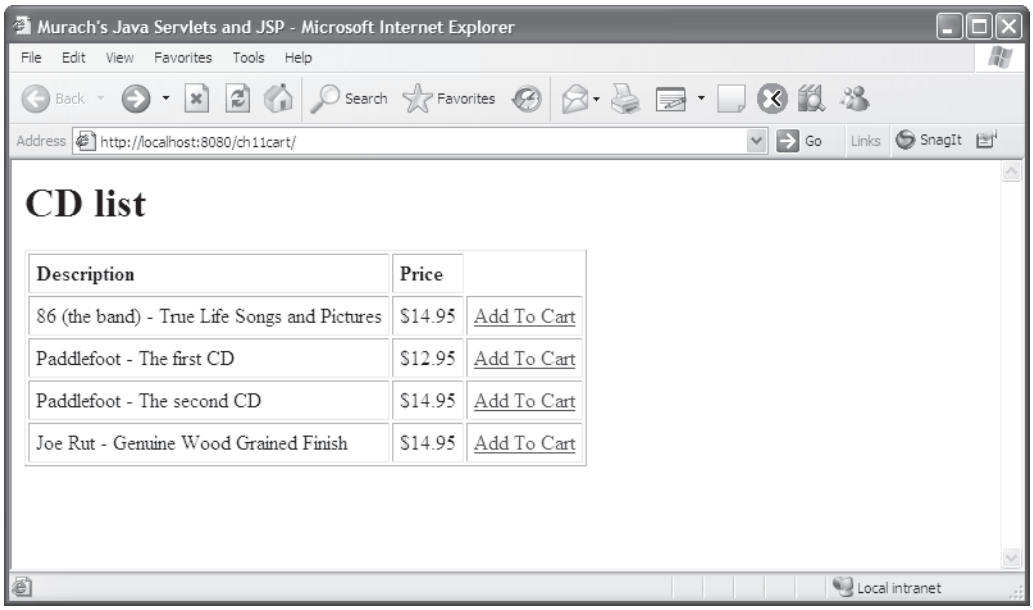
Now that you've learned the details for coding JSTL tags, you're ready to see how they're used within the context of an application. To show that, this chapter finishes by showing a Cart application that maintains a simple shopping cart for a user. Since this application uses the MVC pattern, the JSPs don't require extensive use of JSTL tags. However, the url tag is needed to encode URLs, and the forEach tag is needed to display the items in the user's cart.

## The user interface

Figure 11-11 shows the user interface for the Cart application. From the Index page, you can click on the Add To Cart link for any of the four CDs to add the CD to your cart. Then, the Cart page will display all of the items that have been added to your cart.

On the Cart page, you can update the quantity for an item by entering a new quantity in the Quantity column and clicking on the Update button. Or, you can remove an item from the cart by clicking on its Remove Item button. Finally, you can return to the Index page by clicking on the Continue Shopping button, or you can begin the checkout process by clicking on the Checkout button.

### The Index page




### The Cart page

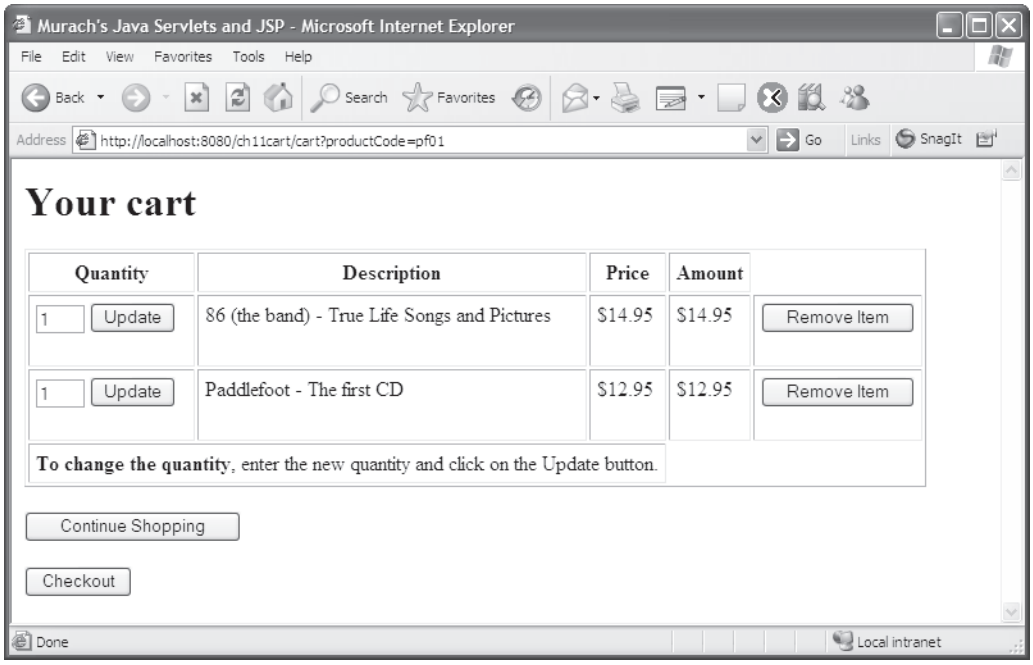



Figure 11-11 The user interface for the Cart application

## The code for the business classes

Figure 11-12 shows the three business classes for the Cart application. These classes are the Model in the MVC pattern. All of these classes follow the rules for creating a JavaBean and implement the Serializable interface as described in chapter 9.

Part 1 shows the Product class. This class stores information about each product that's available from the web site. In particular, it provides get and set methods for the code, description, and price fields for the product. In addition, this class provides the getPriceCurrencyFormat method, which gets a string for the price after the currency format has been applied to the price. For example, for a double value of 11.5, this method returns a string of "$11.50", which is usually the format that you want to display on a JSP.

Part 2 shows the LineItem class. This class stores information about each line item that's stored in the cart. To do that, this class uses a Product object as one of its instance variables to store the product information for the line item. In addition, this class always calculates the value of the total field by multiplying the product price by the quantity. As a result, there's no need to provide a set method for this field. Finally, this class provides a getTotalCurrencyFormat method that applies currency formatting to the double value that's returned by the getTotal method.

Part 3 shows the Cart class. This class stores each line item that has been added to the cart. To do that, the Cart class uses an ArrayList to store zero or more LineItem objects. When you use the constructor to create a Cart object, the constructor initializes the ArrayList object. Then, you can use the addItem method to add an item, or you can use the removeItem method to remove an item. In addition, you can use the getItems method to return the ArrayList object, or you can use the getCount method to get the number of items that are stored in the cart.

### The code for the Product class

```
package business;

import java.io.Serializable;
import java.text.NumberFormat;

public class Product implements Serializable
{
    private String code;
    private String description;
    private double price;

    public Product()
    {
        code = "";
        description = "";
        price = 0;
    }

    public void setCode(String code)
    {
        this.code = code;
    }

    public String getCode()
    {
        return code;
    }

    public void setDescription(String description)
    {
        this.description = description;
    }

    public String getDescription()
    {
        return description;
    }

    public void setPrice(double price)
    {
        this.price = price;
    }

    public double getPrice()
    {
        return price;
    }

    public String getPriceCurrencyFormat()
    {
        NumberFormat currency = NumberFormat.getCurrencyInstance();
        return currency.format(price);
    }
}
```

Figure 11-12 The code for the business classes (part 1 of 3)

## The code for the LineItem class

```
package business;

import java.io.Serializable;
import java.text.NumberFormat;

public class LineItem implements Serializable
{
    private Product product;
    private int quantity;

    public LineItem() {}

    public void setProduct(Product p)
    {
        product = p;
    }

    public Product getProduct()
    {
        return product;
    }

    public void setQuantity(int quantity)
    {
        this.quantity = quantity;
    }

    public int getQuantity()
    {
        return quantity;
    }

    public double getTotal()
    {
        double total = product.getPrice() * quantity;
        return total;
    }

    public String getTotalCurrencyFormat()
    {
        NumberFormat currency = NumberFormat.getCurrencyInstance();
        return currency.format(this.getTotal());
    }
}
```

Figure 11-12 The code for the business classes (part 2 of 3)

## The code for the Cart class

```
package business;

import java.io.Serializable;
import java.util.ArrayList;

public class Cart implements Serializable
{
    private ArrayList<LineItem> items;

    public Cart()
    {
        items = new ArrayList<LineItem>();
    }

    public ArrayList<LineItem> getItems()
    {
        return items;
    }

    public int getCount()
    {
        return items.size();
    }

    public void addItem(LineItem item)
    {
        String code = item.getProduct().getCode();
        int quantity = item.getQuantity();
        for (int i = 0; i < items.size(); i++)
        {
            LineItem lineItem = items.get(i);
            if (lineItem.getProduct().getCode().equals(code))
            {
                lineItem.setQuantity(quantity);
                return;
            }
        }
        items.add(item);
    }

    public void removeItem(LineItem item)
    {
        String code = item.getProduct().getCode();
        for (int i = 0; i < items.size(); i++)
        {
            LineItem lineItem = items.get(i);
            if (lineItem.getProduct().getCode().equals(code))
            {
                items.remove(i);
                return;
            }
        }
    }
}
```

Figure 11-12 The code for the business classes (part 3 of 3)

## The code for the servlets and JSPs

Figure 11-13 shows the one servlet and two JSPs for the Cart application. Here, the servlet is the Controller and the two JSPs are the View in the MVC pattern.

Part 1 shows the JSP code for the Index page that's displayed when the Cart application first starts. This page includes a taglib directive that imports the JSTL core library. Then, this page displays a table where there is one row for each product. Here, each product row includes an Add To Cart link that uses the JSTL url tag to encode the URL that's used to add each product to the cart. This code works because the CartServlet shown in part 2 of this figure has been mapped to the "/cart" URL.

Although these four rows are hard-coded for this page, the product data could also be read from a database and stored in an ArrayList. Then, you could use a forEach tag to display each product in the ArrayList. The technique for doing this is similar to the technique for displaying each line item in the cart as shown in figure 11-4.

## The code for the index.jsp file

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>
<body>

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

<h1>CD list</h1>

<table cellpadding="5" border=1>

  <tr valign="bottom">
    <td align="left"><b>Description</b></td>
    <td align="left"><b>Price</b></td>
    <td align="left"></td>
  </tr>

  <tr valign="top">
    <td>86 (the band) - True Life Songs and Pictures</td>
    <td>$14.95</td>
    <td><a href="<c:url value='/cart?productCode=8601' />">
        Add To Cart</a></td>
  </tr>

  <tr valign="top">
    <td>Paddlefoot - The first CD</td>
    <td>$12.95</td>
    <td><a href="<c:url value='/cart?productCode=pf01' />">
        Add To Cart</a></td>
  </tr>

  <tr valign="top">
    <td>Paddlefoot - The second CD</td>
    <td>$14.95</td>
    <td><a href="<c:url value='/cart?productCode=pf02' />">
        Add To Cart</a></td>
  </tr>

  <tr valign="top">
    <td>Joe Rut - Genuine Wood Grained Finish</td>
    <td>$14.95</td>
    <td><a href="<c:url value='/cart?productCode=jr01' />">
        Add To Cart</a></td>
  </tr>

</table>

</body>
</html>
```

Figure 11-13 The code for the servlets and JSPs (page 1 of 4)

Part 2 shows the servlet code for the CartServlet. To start, this code gets the value of the productCode parameter from the request object. This parameter uniquely identifies the Product object. Then, this code gets the value of the quantity parameter if there is one. However, unless the user clicked on the Update button from the Cart page, this parameter will be equal to a null value.

After getting the parameter values from the request, this servlet uses the getAttribute method to get the Cart object from a session attribute named cart. If this method returns a null value, this servlet creates a new Cart object.

After the Cart object has been retrieved or created, this servlet sets the value of the quantity variable. To do that, it starts by setting the quantity variable to a default value of 1. Then, if the quantityString variable contains an invalid integer value, such as a null value, the parseInt method of the Integer class will throw an exception. This also causes the quantity to be set to 1. However, if the user enters a valid integer such as 0 or 2 or -2, the quantity will be set to that value. Finally, if the quantity is a negative number, the quantity will be set to 1.

After the quantity variable has been set, this servlet uses the getProduct method of the ProductIO class to read the Product object that corresponds with the productCode variable from a text file named products.txt that's stored in the application's WEB-INF directory. To do that, this code specifies the productCode variable as the first argument of the getProduct method. Although this application stores data in a text file to keep things simple, a more realistic application would probably read this data from a database as described in section 3 of this book.

After the Product object has been read from the text file, this servlet creates a LineItem object and sets its Product object and quantity. Then, if the quantity is greater than 0, this code adds the LineItem object to the Cart object. However, if the quantity is 0, this code removes the item from the Cart object.

Finally, this servlet sets the Cart object as a session attribute named cart. Then, it forwards the request and response to the Cart page.

As you review this code, you may notice that the CartServlet only provides an HTTP Get method. As a result, you can't use the HTTP Post method to call this servlet. However, this servlet doesn't write any data to the server, and a user can request this servlet multiple times in a row without causing any problems. As a result, you don't need to implement the HTTP Post method for this servlet.

## The code for the CartServlet class

```
package cart;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import business.*;
import data.*;

public class CartServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException
    {
        String productCode = request.getParameter("productCode");
        String quantityString = request.getParameter("quantity");

        HttpSession session = request.getSession();
        Cart cart = (Cart) session.getAttribute("cart");
        if (cart == null)
            cart = new Cart();

        int quantity = 1;
        try
        {
            quantity = Integer.parseInt(quantityString);
            if (quantity < 0)
                quantity = 1;
        }
        catch(NumberFormatException nfe)
        {
            quantity = 1;
        }

        ServletContext sc = getServletContext();
        String path = sc.getRealPath("WEB-INF/products.txt");
        Product product = ProductIO.getProduct(productCode, path);

        LineItem lineItem = new LineItem();
        lineItem.setProduct(product);
        lineItem.setQuantity(quantity);
        if (quantity > 0)
            cart.addItem(lineItem);
        else if (quantity == 0)
            cart.removeItem(lineItem);

        session.setAttribute("cart", cart);
        String url = "/cart.jsp";
        RequestDispatcher dispatcher =
                getServletContext().getRequestDispatcher(url);
        dispatcher.forward(request, response);
    }
}
```

Figure 11-13 The code for the servlets and JSPs (page 2 of 4)

Part 3 shows the JSP code for the Cart page. Like the Index page, this page uses the taglib directive to import the JSTL core library. Then, it uses a table to display one row for each item in the cart. To do that, it uses a forEach tag to loop through each LineItem object in the ArrayList that's returned by the items property of the cart attribute, and it uses EL to display the data for each line item.

At first glance, the code for this row seems complicated because the first and last columns contain HTML forms that include text boxes, hidden text boxes, and buttons. For example, the first column contains a form that includes a hidden text box that sets the productCode parameter for the form, a text box that allows the user to enter a quantity for the form, and a button that submits the form to the CartServlet. Similarly, the last column contains a hidden text box that sets the productCode parameter for the form, another hidden text box that sets the quantity parameter to 0 (which causes the item to be removed from the cart), and a button that submits the form to the CartServlet. However, if you study this code, you shouldn't have much trouble understanding how it works.

## The code for the cart.jsp file Page 1

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>
<body>

<h1>Your cart</h1>

<table border="1" cellpadding="5">
  <tr>
    <th>Quantity</th>
    <th>Description</th>
    <th>Price</th>
    <th>Amount</th>
  </tr>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach var="item" items="${cart.items}">

  <tr valign="top">
    <td>
      <form action="<c:url value='/cart' />">
        <input type="hidden" name="productCode"
               value="${item.product.code}">
        <input type=text size=2 name="quantity"
               value="${item.quantity}">
        <input type="submit" value="Update">
      </form>
    </td>
    <td>${item.product.description}</td>
    <td>${item.product.priceCurrencyFormat}</td>
    <td>${item.totalCurrencyFormat}</td>
    <td>
      <form action="<c:url value='/cart' />">
        <input type="hidden" name="productCode"
               value="${item.product.code}">
        <input type="hidden" name="quantity"
               value="0">
        <input type="submit" value="Remove Item">
      </form>
    </td>
  </tr>

</c:forEach>

  <tr>
    <td colspan="3">
      <p><b>To change the quantity</b>, enter the new quantity
            and click on the Update button.</p>
    </td>
  </tr>

</table>
```

Figure 11-13 The code for the servlets and JSPs (page 3 of 4)

Part 4 shows the rest of the JSP code for the Cart page. This code contains two forms where each form contains a single button. The button on the first form displays the Index page, and the button on the second form displays the Checkout page (which isn't shown or described in this chapter).

If you review the use of the JSTL and EL code in the Index and Cart pages, you'll see that the url tag is used to encode all of the URLs. As a result, the Cart application will be able to track sessions even if the user has disabled cookies. You'll also see that the only other JSTL tag that's used is the forEach tag in the Cart page. Finally, you'll see that EL is used to display the nested properties that are available from the Product, LineItem, and Cart objects. This is a typical JSTL and EL usage for applications that use the MVC pattern.

### The code for the cart.jsp file — Page 2

```
<br>

<form action="<c:url value='/index.jsp' />" method="post">
  <input type="submit" value="Continue Shopping">
</form>

<form action="<c:url value='/checkout.jsp' />" method="post">
  <input type="submit" value="Checkout">
</form>

</body>
</html>
```

### Note

- In the web.xml file, the CartServlet class is mapped to the "/cart" URL.

Figure 11-13 The code for the servlets and JSPs (page 4 of 4)

# Perspective

The goal of this chapter has been to show you how to use JSTL with EL to eliminate or reduce scripting from your JSPs. However, it isn't always possible to remove all scripting from your applications by using JSTL. In that case, you may occasionally want to use scripting. Another option, though, is to create and use custom tags that are stored in a custom tag library as described in the next chapter.

## Summary

- The *JSP Standard Tag Library* (*JSTL*) provides tags for common tasks that need to be performed in JSPs.
- Before you can use JSTL tags, you must make the jstl.jar and standard.jar files available to the application.
- Before you can use JSTL tags in a JSP, you must code a taglib directive for the library that you want to use.
- You can use a web browser to view the documentation for JSTL.
- You can use the url tag to encode URLs so the application can track sessions even if the client browser has cookies disabled.
- You can use the forEach tag to loop through most types of collections, including regular arrays.
- You can use the forTokens tag to loop through items in a delimited string.
- You can use the if tag to code the equivalent of a Java if statement.
- You can use the choose tag to code the equivalent of a Java if/else statement.
- You can use the import tag to include files at runtime. This works like the standard JSP include tag, but it can be used to include files from other web applications even when they're running on remote web servers.

## Exercise 11-1 Use JSTL in the Download application

In this exercise, you'll enhance the Download application that you used in exercise 10-2 of the last chapter.

1. Open the ch11download project in the ex_starts directory. Then, run the application to refresh your memory about how it works.
2. Use your IDE to add the JSTL library to this project. With NetBeans, you can do that by right-clicking on the Libraries folder for the project and selecting the Add Libraries command from the resulting menu.
3. Open the JSPs for this project. Then, add the taglib directive for the core JSTL library to the beginning of these pages. Finally, use the url tag to encode all the URLs in this application.
4. Test the application to make sure it works correctly.
5. Open the index.jsp file. Then, modify it so it uses the if tag to only display the welcome message if the cookie for the first name doesn't contain a null value.
6. Test the application to make sure it works correctly.

## Exercise 11-2 Use JSTL in the Cart application

In this exercise, you'll use JSTL to loop through an array list of Product objects.

1. Open the ch11cart project in the ex_starts directory.
2. Open the web.xml file. Note that the ProductsServlet class is called when this application starts. This means that the browser will issue an HTTP Get request for the ProductsServlet class so its doGet method will be called.
3. Open the ProductsServlet.java file. Note how this servlet uses the processRequest method to read an ArrayList of Product objects from the projects.txt file and store them as an attribute of the session object. Note too that this method is called from both the doGet and doPost methods.
4. Test the application to make sure it works correctly.
5. Add the JSTL library to this project. Then, open the index.jsp file, and add the taglib directive that imports the core JSTL library.
6. In the index.jsp file, add a forEach tag that loops through the ArrayList of Product objects and displays one row for each product. To do that, you can use EL to display the properties of each Product object. (Be sure to delete any old code that you no longer need.)
7. Test the application to make sure that it works correctly.