14

JavaServer Pages Standard Tag Library

THE JavaServer Pages Standard Tag Library (JSTL) encapsulates core functionality common to many JSP applications. Instead of mixing tags from numerous vendors in your JSP applications, JSTL allows you to employ a single, standard set of tags. This standardization allows you to deploy your applications on any JSP container supporting JSTL and makes it more likely that the implementation of the tags is optimized.

JSTL has tags such as iterators and conditionals for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

This chapter demonstrates JSTL through excerpts from the JSP version of the Duke's Bookstore application discussed in the earlier chapters. It assumes that you are familiar with the material in the Using Custom Tags (page 511) section of Chapter 12.

This chapter does not cover every JSTL tag, only the most commonly used ones. Please refer to the reference pages at `http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html` for a complete list of the JSTL tags and their attributes.

The Example JSP Pages

This chapter illustrates JSTL using excerpts from the JSP version of the Duke's Bookstore application discussed in Chapter 12. Here, they are rewritten to replace the JavaBeans component database access object with direct calls to the database via the JSTL SQL tags. For most applications, it is better to encapsulate calls to a database in a bean. JSTL includes SQL tags for situations where a new application is being prototyped and the overhead of creating a bean may not be warranted.

The source for the Duke's Bookstore application is located in the `<INSTALL>/j2eetutorial14/examples/web/bookstore4/` directory created when you unzip the tutorial bundle (see About the Examples, page xxxvi). A sample `bookstore4.war` is provided in `<INSTALL>/j2eetutorial14/examples/web/provided-wars/`. To build the example, follow these steps:

1. Build and package the bookstore common files as described in Duke's Bookstore Examples (page 103).
2. In a terminal window, go to `<INSTALL>/j2eetutorial14/examples/web/bookstore4/`.
3. Run `asant build`. This target will copy files to the `<INSTALL>/j2eetutorial14/examples/web/bookstore4/build/` directory.
4. Start the Application Server.
5. Perform all the operations described in Accessing Databases from Web Applications, page 104.

To package and deploy the example using `asant`, follow these steps:

1. Run `asant create-bookstore-war`.
2. Run `asant deploy-war`.

To learn how to configure the example, use `deploytool` to package and deploy it:

1. Start `deploytool`.
2. Create a web application called `bookstore4` by running the New Web Component wizard. Select File→New→Web Component.
3. In the New Web Component wizard:
   a. Select the Create New Stand-Alone WAR Module radio button.

b. In the WAR File field, enter <*INSTALL*>/j2eetutorial14/examples/web/bookstore4/bookstore4.war. The WAR Display Name field will show bookstore4.

c. In the Context Root field, enter /bookstore4.

d. Click Edit Contents.

e. In the Edit Contents dialog box, navigate to <*INSTALL*>/j2eetutorial14/examples/web/bookstore4/build/. Select the JSP pages bookstore.jsp, bookdetails.jsp, bookcatalog.jsp, bookshowcart.jsp, bookcashier.jsp, and bookreceipt.jsp and the template directory and click Add.

f. Add the shared bookstore library. Navigate to <*INSTALL*>/j2eetutorial14/examples/web/bookstore/dist/. Select bookstore.jar and click Add.

g. Click OK.

h. Click Next.

i. Select the JSP Page radio button.

j. Click Next.

k. Select bookstore.jsp from the JSP Filename combo box.

l. Click Next.

m. Click Add. Enter the alias /bookstore.

n. Click Finish.

4. Add each of the web components listed in Table 14–1. For each component:

a. Select File→New→Web Component.

b. Click the Add to Existing WAR Module radio button. Because the WAR contains all the JSP pages, you do not have to add any more content.

c. Click Next.

d. Select the JSP Page radio button and the Component Aliases checkbox.

e. Click Next.

f. Select the page from the JSP Filename combo box.

g. Click Finish.

Table 14–1 Duke's Bookstore Web Components

Web Component Name	JSP Page	Alias
`bookcatalog`	`bookcatalog.jsp`	`/bookcatalog`
`bookdetails`	`bookdetails.jsp`	`/bookdetails`
`bookshowcart`	`bookshowcart.jsp`	`/bookshowcart`
`bookcashier`	`bookcashier.jsp`	`/bookcashier`
`bookreceipt`	`bookreceipt.jsp`	`/bookreceipt`

5. Set the alias for each web component.
   a. Select the component.
   b. Select the Aliases tab.
   c. Click the Add button.
   d. Enter the alias.
6. Add the context parameter that specifies the JSTL resource bundle base name.
   a. Select the web module.
   b. Select the Context tab.
   c. Click Add.
   d. Enter `javax.servlet.jsp.jstl.fmt.localizationContext` in the Coded Parameter field.
   e. Enter `messages.BookstoreMessages` in the Value field.
7. Set the prelude and coda for all JSP pages.
   a. Select the JSP Properties tab.
   b. Click the Add button next to the Name list.
   c. Enter `bookstore4`.
   d. Click the Add URL button.
   e. Enter `*.jsp`.
   f. Click the Edit Preludes button.

g. Click Add.
h. Enter `/template/prelude.jspf`.
i. Click OK.
j. Click the Edit Codas button.
k. Click Add.
l. Enter `/template/coda.jspf`.
m. Click OK.

8. Add a resource reference for the database.
   a. Select the Resource Ref's tab.
   b. Click Add.
   c. Enter `jdbc/BookDB` in the Coded Name field.
   d. Accept the default type `javax.sql.DataSource`.
   e. Accept the default authorization `Container`.
   f. Accept the default selected `Shareable`.
   g. Enter `jdbc/BookDB` in the JNDI name field of the Sun-specific Settings frame.
9. Select File→Save.
10. Deploy the application.
    a. Select Tools→Deploy.
    b. Click OK.

To run the application, open the bookstore URL `http://localhost:8080/bookstore4/bookstore`.

See Troubleshooting (page 446) for help with diagnosing common problems.

Using JSTL

JSTL includes a wide variety of tags that fit into discrete functional areas. To reflect this, as well as to give each area its own namespace, JSTL is exposed as multiple tag libraries. The URIs for the libraries are as follows:

- *Core*: `http://java.sun.com/jsp/jstl/core`
- *XML*: `http://java.sun.com/jsp/jstl/xml`
- *Internationalization*: `http://java.sun.com/jsp/jstl/fmt`
- *SQL*: `http://java.sun.com/jsp/jstl/sql`

- *Functions*: `http://java.sun.com/jsp/jstl/functions`

Table 14–2 summarizes these functional areas along with the prefixes used in this tutorial.

Table 14–2 JSTL Tags

Area	Subfunction	Prefix
Core	Variable support	`c`
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	`x`
	Flow control	
	Transformation	
I18n	Locale	`fmt`
	Message formatting	
	Number and date formatting	
Database	SQL	`sql`
Functions	Collection length	`fn`
	String manipulation	

Thus, the tutorial references the JSTL core tags in JSP pages by using the following `taglib` directive:

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

In addition to declaring the tag libraries, tutorial examples access the JSTL API and implementation. In the Sun Java System Application Server Platform Edition 8, the JSTL TLDs and libraries are distributed in the archive <*J2EE_HOME*>/

`lib/appserv-jstl.jar`. This library is automatically loaded into the classpath of all web applications running on the Application Server, so you don't need to add it to your web application.

Tag Collaboration

Tags usually collaborate with their environment in implicit and explicit ways. *Implicit* collaboration is done via a well-defined interface that allows nested tags to work seamlessly with the ancestor tag that exposes that interface. The JSTL conditional tags employ this mode of collaboration.

Explicit collaboration happens when a tag exposes information to its environment. JSTL tags expose information as JSP EL variables; the convention followed by JSTL is to use the name `var` for any tag attribute that exports information about the tag. For example, the `forEach` tag exposes the current item of the shopping cart it is iterating over in the following way:

```
<c:forEach var="item" items="${sessionScope.cart.items}">
  ...
</c:forEach>
```

In situations where a tag exposes more than one piece of information, the name `var` is used for the primary piece of information being exported, and an appropriate name is selected for any other secondary piece of information exposed. For example, iteration status information is exported by the `forEach` tag via the attribute `status`.

When you want to use an EL variable exposed by a JSTL tag in an expression in the page's scripting language (see Chapter 16), you use the standard JSP element `jsp:useBean` to declare a scripting variable.

For example, `bookshowcart.jsp` removes a book from a shopping cart using a scriptlet. The ID of the book to be removed is passed as a request parameter. The value of the request parameter is first exposed as an EL variable (to be used later by the JSTL `sql:query` tag) and then is declared as a scripting variable and passed to the `cart.remove` method:

```
<c:set var="bookId" value="${param.Remove}"/>
<jsp:useBean id="bookId" type="java.lang.String" />
<% cart.remove(bookId); %>
<sql:query var="books"
```

```
    dataSource="${applicationScope.bookDS}">
    select * from PUBLIC.books where id = ?
    <sql:param value="${bookId}" />
</sql:query>
```

Core Tag Library

Table 14–3 summarizes the core tags, which include those related to variables and flow control, as well as a generic way to access URL-based resources whose content can then be included or processed within the JSP page.

Table 14–3 Core Tags

Area	Function	Tags	Prefix
Core	Variable support	`remove` `set`	`c`

	Flow control	`choose`
		`  when`
		`  otherwise`
		`forEach`
		`forTokens`
		`if`	

	URL management	`import`
		`  param`
		`redirect`
		`  param`
		`url`
		`  param`	

	Miscellaneous	`catch` `out`	

Variable Support Tags

The `set` tag sets the value of an EL variable or the property of an EL variable in any of the JSP scopes (page, request, session, or application). If the variable does not already exist, it is created.

The JSP EL variable or property can be set either from the attribute `value`:

```
<c:set var="foo" scope="session" value="..."/>
```

or from the body of the tag:

```
<c:set var="foo">
  ...
</c:set>
```

For example, the following sets an EL variable named `bookID` with the value of the request parameter named `Remove`:

```
<c:set var="bookId" value="${param.Remove}"/>
```

To remove an EL variable, you use the `remove` tag. When the bookstore JSP page `bookreceipt.jsp` is invoked, the shopping session is finished, so the `cart` session attribute is removed as follows:

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

Flow Control Tags

To execute flow control logic, a page author must generally resort to using scriptlets. For example, the following scriptlet is used to iterate through a shopping cart:

```
<%
  Iterator i = cart.getItems().iterator();
  while (i.hasNext()) {
     ShoppingCartItem item =
        (ShoppingCartItem)i.next();
     ...
%>
     <tr>
     <td align="right" bgcolor="#ffffff">
     ${item.quantity}
     </td>
     ...
<%
  }
%>
```

Flow control tags eliminate the need for scriptlets. The next two sections have examples that demonstrate the conditional and iterator tags.

Conditional Tags

The `if` tag allows the conditional execution of its body according to the value of the `test` attribute. The following example from `bookcatalog.jsp` tests whether the request parameter `Add` is empty. If the test evaluates to `true`, the page queries the database for the book record identified by the request parameter and adds the book to the shopping cart:

```
<c:if test="${!empty param.Add}">
  <c:set var="bid" value="${param.Add}"/>
  <jsp:useBean id="bid"  type="java.lang.String" />
   <sql:query var="books"
     dataSource="${applicationScope.bookDS}">
     select * from PUBLIC.books where id = ?
     <sql:param value="${bid}" />
  </sql:query>
  <c:forEach var="bookRow" begin="0" items="${books.rows}">
     <jsp:useBean id="bookRow" type="java.util.Map" />
     <jsp:useBean id="addedBook"
        class="database.BookDetails" scope="page" />
  ...
  <% cart.add(bid, addedBook); %>
...
</c:if>
```

The `choose` tag performs conditional block execution by the embedded `when` subtags. It renders the body of the first `when` tag whose test condition evaluates to `true`. If none of the test conditions of nested `when` tags evaluates to `true`, then the body of an `otherwise` tag is evaluated, if present.

For example, the following sample code shows how to render text based on a customer's membership category.

```
<c:choose>
  <c:when test="${customer.category == 'trial'}" >
     ...
  </c:when>
  <c:when test="${customer.category == 'member'}" >
     ...
  </c:when>
     <c:when test="${customer.category == 'preferred'}" >
     ...
```

```
    </c:when>
    <c:otherwise>
      ...
    </c:otherwise>
  </c:choose>
```

The `choose`, `when`, and `otherwise` tags can be used to construct an `if-then-else` statement as follows:

```
<c:choose>
  <c:when test="${count == 0}" >
    No records matched your selection.
  </c:when>
  <c:otherwise>
    ${count} records matched your selection.
  </c:otherwise>
</c:choose>
```

Iterator Tags

The `forEach` tag allows you to iterate over a collection of objects. You specify the collection via the `items` attribute, and the current item is available through a variable named by the `var` attribute.

A large number of collection types are supported by `forEach`, including all implementations of `java.util.Collection` and `java.util.Map`. If the `items` attribute is of type `java.util.Map`, then the current item will be of type `java.util.Map.Entry`, which has the following properties:

- `key`: The key under which the item is stored in the underlying `Map`
- `value`: The value that corresponds to the key

Arrays of objects as well as arrays of primitive types (for example, `int`) are also supported. For arrays of primitive types, the current item for the iteration is automatically wrapped with its standard wrapper class (for example, `Integer` for `int`, `Float` for `float`, and so on).

Implementations of `java.util.Iterator` and `java.util.Enumeration` are supported, but they must be used with caution. `Iterator` and `Enumeration` objects are not resettable, so they should not be used within more than one iteration tag. Finally, `java.lang.String` objects can be iterated over if the string contains a list of comma-separated values (for example: Monday,Tuesday,Wednesday,Thursday,Friday).

Here's the shopping cart iteration from the preceding section, now with the `forEach` tag:

```
<c:forEach var="item" items="${sessionScope.cart.items}">
  ...
  <tr>
    <td align="right" bgcolor="#ffffff">
    ${item.quantity}
  </td>
  ...
</c:forEach>
```

The `forTokens` tag is used to iterate over a collection of tokens separated by a delimiter.

URL Tags

The `jsp:include` element provides for the inclusion of static and dynamic resources in the same context as the current page. However, `jsp:include` cannot access resources that reside outside the web application, and it causes unnecessary buffering when the resource included is used by another element.

In the following example, the `transform` element uses the content of the included resource as the input of its transformation. The `jsp:include` element reads the content of the response and writes it to the body content of the enclosing transform element, which then rereads exactly the same content. It would be more efficient if the `transform` element could access the input source directly and thereby avoid the buffering involved in the body content of the transform tag.

```
<acme:transform>
  <jsp:include page="/exec/employeesList"/>
<acme:transform/>
```

The `import` tag is therefore the simple, generic way to access URL-based resources, whose content can then be included and or processed within the JSP page. For example, in XML Tag Library (page 560), `import` is used to read in the XML document containing book information and assign the content to the scoped variable `xml`:

```
<c:import url="/books.xml" var="xml" />
<x:parse doc="${xml}" var="booklist"
  scope="application" />
```

The `param` tag, analogous to the `jsp:param` tag (see jsp:param Element, page 517), can be used with `import` to specify request parameters.

In Session Tracking (page 474) we discuss how an application must rewrite URLs to enable session tracking whenever the client turns off cookies. You can use the `url` tag to rewrite URLs returned from a JSP page. The tag includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged. Note that this feature requires that the URL be *relative*. The `url` tag takes `param` subtags to include parameters in the returned URL. For example, `bookcatalog.jsp` rewrites the URL used to add a book to the shopping cart as follows:

```
<c:url var="url" value="/catalog" >
  <c:param name="Add" value="${bookId}" />
</c:url>
<p><strong><a href="${url}">
```

The `redirect` tag sends an HTTP redirect to the client. The `redirect` tag takes `param` subtags for including parameters in the returned URL.

Miscellaneous Tags

The `catch` tag provides a complement to the JSP error page mechanism. It allows page authors to recover gracefully from error conditions that they can control. Actions that are of central importance to a page should *not* be encapsulated in a `catch`; in this way their exceptions will propagate instead to an error page. Actions with secondary importance to the page should be wrapped in a `catch` so that they never cause the error page mechanism to be invoked.

The exception thrown is stored in the variable identified by `var`, which always has page scope. If no exception occurred, the scoped variable identified by `var` is removed if it existed. If `var` is missing, the exception is simply caught and not saved.

The `out` tag evaluates an expression and outputs the result of the evaluation to the current `JspWriter` object. The syntax and attributes are as follows:

```
<c:out value="value" [escapeXml="{true|false}"]
  [default="defaultValue"] />
```

If the result of the evaluation is a `java.io.Reader` object, then data is first read from the `Reader` object and then written into the current `JspWriter` object. The

special processing associated with `Reader` objects improves performance when a large amount of data must be read and then written to the response.

If `escapeXml` is true, the character conversions listed in Table 14–4 are applied.

Table 14–4 Character Conversions

Character	Character Entity Code
<	<
>	>
&	&
'	'
"	"

XML Tag Library

The JSTL XML tag set is listed in Table 14–5.

Table 14–5 XML Tags

Area	Function	Tags	Prefix

XML	Core	`out`
		`parse`
		`set`
			x

	Flow control	`choose`
		`  when`
		`  otherwise`
		`forEach`
		`if`	

	Transformation	`transform` `  param`	

A key aspect of dealing with XML documents is to be able to easily access their content. XPath (see How XPath Works, page 255), a W3C recommendation since 1999, provides an easy notation for specifying and selecting parts of an XML document. In the JSTL XML tags, XPath expressions specified using the `select` attribute are used to select portions of XML data streams. Note that XPath is used as a *local* expression language only for the `select` attribute. This means that values specified for `select` attributes are evaluated using the XPath expression language but that values for all other attributes are evaluated using the rules associated with the JSP 2.0 expression language.

In addition to the standard XPath syntax, the JSTL XPath engine supports the following scopes to access web application data within an XPath expression:

- `$foo`
- `$param:`
- `$header:`
- `$cookie:`
- `$initParam:`
- `$pageScope:`
- `$requestScope:`
- `$sessionScope:`
- `$applicationScope:`

These scopes are defined in exactly the same way as their counterparts in the JSP expression language discussed in Implicit Objects (page 500). Table 14–6 shows some examples of using the scopes.

Table 14–6 Example XPath Expressions

XPath Expression	Result
`$sessionScope:profile`	The session-scoped EL variable named `profile`
`$initParam:mycom.productId`	The `String` value of the `mycom.productId` context parameter

The XML tags are illustrated in another version (`bookstore5`) of the Duke's Bookstore application. This version replaces the database with an XML representation of the bookstore database, which is retrieved from another web application. The directions for building and deploying this version of the application

are in The Example JSP Document (page 526). A sample bookstore5.war is provided in <*INSTALL*>/j2eetutorial14/examples/web/provided-wars/.

Core Tags

The core XML tags provide basic functionality to easily parse and access XML data.

The parse tag parses an XML document and saves the resulting object in the EL variable specified by attribute var. In bookstore5, the XML document is parsed and saved to a context attribute in parsebooks.jsp, which is included by all JSP pages that need access to the document:

```
<c:if test="${applicationScope:booklist == null}" >
  <c:import url="${initParam.booksURL}" var="xml" />
  <x:parse doc="${xml}" var="booklist" scope="application" />
</c:if>
```

The set and out tags parallel the behavior described in Variable Support Tags (page 554) and Miscellaneous Tags (page 559) for the XPath local expression language. The set tag evaluates an XPath expression and sets the result into a JSP EL variable specified by attribute var. The out tag evaluates an XPath expression on the current context node and outputs the result of the evaluation to the current JspWriter object.

The JSP page bookdetails.jsp selects a book element whose id attribute matches the request parameter bookId and sets the abook attribute. The out tag then selects the book's title element and outputs the result.

```
<x:set var="abook"
  select="$applicationScope.booklist/
     books/book[@id=$param:bookId]" />
  <h2><x:out select="$abook/title"/></h2>
```

As you have just seen, x:set stores an internal XML representation of a *node* retrieved using an XPath expression; it doesn't convert the selected node into a String and store it. Thus, x:set is primarily useful for storing parts of documents for later retrieval.

If you want to store a String, you must use x:out within c:set. The x:out tag converts the node to a String, and c:set then stores the String as an EL vari-

able. For example, `bookdetails.jsp` stores an EL variable containing a book price, which is later provided as the value of a `fmt` tag, as follows:

```
<c:set var="price">
  <x:out select="$abook/price"/>
</c:set>
<h4><fmt:message key="ItemPrice"/>:
  <fmt:formatNumber value="${price}" type="currency"/>
```

The other option, which is more direct but requires that the user have more knowledge of XPath, is to coerce the node to a `String` manually by using XPath's `string` function.

```
<x:set var="price" select="string($abook/price)"/>
```

Flow Control Tags

The XML flow control tags parallel the behavior described in Flow Control Tags (page 555) for XML data streams.

The JSP page `bookcatalog.jsp` uses the `forEach` tag to display all the books contained in `booklist` as follows:

```
<x:forEach var="book"
  select="$applicationScope:booklist/books/*">
  <tr>
     <c:set var="bookId">
        <x:out select="$book/@id"/>
     </c:set>=
     <td bgcolor="#ffffaa">
        <c:url var="url"
        value="/bookdetails" >
           <c:param name="bookId" value="${bookId}" />
           <c:param name="Clear" value="0" />
        </c:url>
        <a href="${url}">
        <strong><x:out select="$book/title"/> 
        </strong></a></td>
     <td bgcolor="#ffffaa" rowspan=2>
        <c:set var="price">
           <x:out select="$book/price"/>
        </c:set>
        <fmt:formatNumber value="${price}" type="currency"/>
         
     </td>
```

```
        <td bgcolor="#ffffaa" rowspan=2>
        <c:url var="url" value="/catalog" >
          <c:param name="Add" value="${bookId}" />
        </c:url>
        <p><strong><a href="${url}"> 
          <fmt:message key="CartAdd"/> </a>
        </td>
      </tr>
      <tr>
        <td bgcolor="#ffffff">
          <fmt:message key="By"/> <em>
          <x:out select="$book/firstname"/> 
          <x:out select="$book/surname"/></em></td></tr>
    </x:forEach>
```

Transformation Tags

The `transform` tag applies a transformation, specified by an XSLT stylesheet set by the attribute `xslt`, to an XML document, specified by the attribute `doc`. If the `doc` attribute is not specified, the input XML document is read from the tag's body content.

The `param` subtag can be used along with `transform` to set transformation parameters. The attributes `name` and `value` are used to specify the parameter. The `value` attribute is optional. If it is not specified, the value is retrieved from the tag's body.

Internationalization Tag Library

Chapter 22 covers how to design web applications so that they conform to the language and formatting conventions of client locales. This section describes tags that support the internationalization of JSP pages.

JSTL defines tags for setting the locale for a page, creating locale-sensitive messages, and formatting and parsing data elements such as numbers, currencies,

dates, and times in a locale-sensitive or customized manner. Table 14–7 lists the tags.

Table 14–7 Internationalization Tags

Area	Function	Tags	Prefix
I18n	Setting Locale	`setLocale` `requestEncoding`	`fmt`

	Messaging	`bundle`
		`message`
		`  param`
		`setBundle`	

	Number and Date Formatting	`formatNumber`
		`formatDate`
		`parseDate`
		`parseNumber`
		`setTimeZone`
		`timeZone`	

JSTL i18n tags use a localization context to localize their data. A *localization context* contains a locale and a resource bundle instance. To specify the localization context at deployment time, you define the context parameter `javax.servlet.jsp.jstl.fmt.localizationContext`, whose value can be a `javax.servlet.jsp.jstl.fmt.LocalizationContext` or a `String`. A `String` context parameter is interpreted as a resource bundle base name. For the Duke's Bookstore application, the context parameter is the `String messages.BookstoreMessages`. When a request is received, JSTL automatically sets the locale based on the value retrieved from the request header and chooses the correct resource bundle using the base name specified in the context parameter.

Setting the Locale

The `setLocale` tag is used to override the client-specified locale for a page. The `requestEncoding` tag is used to set the request's character encoding, in order to be able to correctly decode request parameter values whose encoding is different from `ISO-8859-1`.

Messaging Tags

By default, the capability to sense the browser locale setting is enabled in JSTL. This means that the client determines (via its browser setting) which locale to use, and allows page authors to cater to the language preferences of their clients.

The setBundle and bundle Tags

You can set the resource bundle at runtime with the JSTL `fmt:setBundle` and `fmt:bundle` tags. `fmt:setBundle` is used to set the localization context in a variable or configuration variable for a specified scope. `fmt:bundle` is used to set the resource bundle for a given tag body.

The message Tag

The `message` tag is used to output localized strings. The following tag from `bookcatalog.jsp` is used to output a string inviting customers to choose a book from the catalog.

```
<h3><fmt:message key="Choose"/></h3>
```

The `param` subtag provides a single argument (for parametric replacement) to the compound message or pattern in its parent `message` tag. One `param` tag must be specified for each variable in the compound message or pattern. Parametric replacement takes place in the order of the `param` tags.

Formatting Tags

JSTL provides a set of tags for parsing and formatting locale-sensitive numbers and dates.

The `formatNumber` tag is used to output localized numbers. The following tag from `bookshowcart.jsp` is used to display a localized price for a book.

```
<fmt:formatNumber value="${book.price}" type="currency"/>
```

Note that because the price is maintained in the database in dollars, the localization is somewhat simplistic, because the `formatNumber` tag is unaware of exchange rates. The tag formats currencies but does not convert them.

Analogous tags for formatting dates (`formatDate`) and for parsing numbers and dates (`parseNumber`, `parseDate`) are also available. The `timeZone` tag establishes the time zone (specified via the `value` attribute) to be used by any nested `formatDate` tags.

In `bookreceipt.jsp`, a "pretend" ship date is created and then formatted with the `formatDate` tag:

```
<jsp:useBean id="now" class="java.util.Date" />
<jsp:setProperty name="now" property="time"
  value="${now.time + 432000000}" />
<fmt:message key="ShipDate"/>
<fmt:formatDate value="${now}" type="date"
  dateStyle="full"/>.
```

SQL Tag Library

The JSTL SQL tags for accessing databases listed in Table 14–8 are designed for quick prototyping and simple applications. For production applications, database operations are normally encapsulated in JavaBeans components.

Table 14–8 SQL Tags

Area	Function	Tags	Prefix
Database		`setDataSource`	`sql`

	SQL	`query`
		`  dateParam`
		`  param`
		`transaction`
		`update`
		`  dateParam`
		`  param`	

The `setDataSource` tag allows you to set data source information for the database. You can provide a JNDI name or `DriverManager` parameters to set the data source information. All of the Duke's Bookstore pages that have more than one SQL tag use the following statement to set the data source:

```
<sql:setDataSource dataSource="jdbc/BookDB" />
```

The query tag performs an SQL query that returns a result set. For parameterized SQL queries, you use a nested param tag inside the query tag.

In bookcatalog.jsp, the value of the Add request parameter determines which book information should be retrieved from the database. This parameter is saved as the attribute name bid and is passed to the param tag.

```
<c:set var="bid" value="${param.Add}"/>
<sql:query var="books" >
  select * from PUBLIC.books where id = ?
  <sql:param value="${bid}" />
</sql:query>
```

The update tag is used to update a database row. The transaction tag is used to perform a series of SQL statements atomically.

The JSP page bookreceipt.jsp page uses both tags to update the database inventory for each purchase. Because a shopping cart can contain more than one book, the transaction tag is used to wrap multiple queries and updates. First, the page establishes that there is sufficient inventory; then the updates are performed.

```
<c:set var="sufficientInventory" value="true" />
<sql:transaction>
  <c:forEach var="item" items="${sessionScope.cart.items}">
     <c:set var="book" value="${item.item}" />
     <c:set var="bookId" value="${book.bookId}" />

     <sql:query var="books"
        sql="select * from PUBLIC.books where id = ?" >
        <sql:param value="${bookId}" />
     </sql:query>
     <jsp:useBean id="inventory"
        class="database.BookInventory" />
     <c:forEach var="bookRow" begin="0"
        items="${books.rowsByIndex}">
        <jsp:useBean id="bookRow"  type="java.lang.Object[]" />
        <jsp:setProperty name="inventory" property="quantity"
           value="${bookRow[7]}" />

        <c:if test="${item.quantity > inventory.quantity}">
           <c:set var="sufficientInventory" value="false" />
           <h3><font color="red" size="+2">
           <fmt:message key="OrderError"/>
           There is insufficient inventory for
           <i>${bookRow[3]}</i>.</font></h3>
        </c:if>
```

```
      </c:forEach>
    </c:forEach>

    <c:if test="${sufficientInventory == 'true'}" />
      <c:forEach var="item" items="${sessionScope.cart.items}">
        <c:set var="book" value="${item.item}" />
        <c:set var="bookId" value="${book.bookId}" />

        <sql:query var="books"
          sql="select * from PUBLIC.books where id = ?" >
          <sql:param value="${bookId}" />
        </sql:query>

        <c:forEach var="bookRow" begin="0"
          items="${books.rows}">
          <sql:update var="books" sql="update PUBLIC.books set
            inventory = inventory - ? where id = ?" >
            <sql:param value="${item.quantity}" />
            <sql:param value="${bookId}" />
          </sql:update>
        </c:forEach>
      </c:forEach>
      <h3><fmt:message key="ThankYou"/>
        ${param.cardname}.</h3><br>
    </c:if>
  </sql:transaction>
```

query Tag Result Interface

The `Result` interface is used to retrieve information from objects returned from a `query` tag.

```
public interface Result
  public String[] getColumnNames();
  public int getRowCount()
  public Map[] getRows();
  public Object[][] getRowsByIndex();
  public boolean isLimitedByMaxRows();
```

For complete information about this interface, see the API documentation for the JSTL packages.

The `var` attribute set by a `query` tag is of type `Result`. The `getRows` method returns an array of maps that can be supplied to the `items` attribute of a `forEach` tag. The JSTL expression language converts the syntax `${result.rows}` to a

call to *result*.getRows. The expression ${books.rows} in the following example returns an array of maps.

When you provide an array of maps to the forEach tag, the var attribute set by the tag is of type Map. To retrieve information from a row, use the get("*colname*") method to get a column value. The JSP expression language converts the syntax ${*map.colname*} to a call to *map*.get("*colname*"). For example, the expression ${book.title} returns the value of the title entry of a book map.

The Duke's Bookstore page bookdetails.jsp retrieves the column values from the book map as follows.

```
<c:forEach var="book" begin="0" items="${books.rows}">
  <h2>${book.title}</h2>
   <fmt:message key="By"/> <em>${book.firstname}
  ${book.surname}</em>  
  (${book.year})<br>   <br>
  <h4><fmt:message key="Critics"/></h4>
  <blockquote>${book.description}</blockquote>
  <h4><fmt:message key="ItemPrice"/>:
  <fmt:formatNumber value="${book.price}" type="currency"/>
  </h4>
</c:forEach>
```

The following excerpt from bookcatalog.jsp uses the Row interface to retrieve values from the columns of a book row using scripting language expressions. First, the book row that matches a request parameter (bid) is retrieved from the database. Because the bid and bookRow objects are later used by tags that use scripting language expressions to set attribute values and by a scriptlet that adds a book to the shopping cart, both objects are declared as scripting variables using the jsp:useBean tag. The page creates a bean that describes the book, and scripting language expressions are used to set the book properties from book row column values. Then the book is added to the shopping cart.

You might want to compare this version of bookcatalog.jsp to the versions in JavaServer Pages Technology (page 479) and Custom Tags in JSP Pages (page 575) that use a book database JavaBeans component.

```
<sql:query var="books"
  dataSource="${applicationScope.bookDS}">
  select * from PUBLIC.books where id = ?
  <sql:param value="${bid}" />
</sql:query>
<c:forEach var="bookRow" begin="0"
```

```
      items="${books.rowsByIndex}">
  <jsp:useBean id="bid"  type="java.lang.String" />
  <jsp:useBean id="bookRow" type="java.lang.Object[]" />
  <jsp:useBean id="addedBook" class="database.BookDetails"
    scope="page" >
    <jsp:setProperty name="addedBook" property="bookId"
      value="${bookRow[0]}" />
    <jsp:setProperty name="addedBook" property="surname"
      value="${bookRow[1]}" />
    <jsp:setProperty name="addedBook" property="firstName"
      value="${bookRow[2]}" />
    <jsp:setProperty name="addedBook" property="title"
      value="${bookRow[3]}" />
    <jsp:setProperty name="addedBook" property="price"
      value="${bookRow[4])}" />
    <jsp:setProperty name="addedBook" property="year"
      value="${bookRow[6]}" />
    <jsp:setProperty name="addedBook"
      property="description"
      value="${bookRow[7]}" />
    <jsp:setProperty name="addedBook" property="inventory"
      value="${bookRow[8]}" />
  </jsp:useBean>
  <% cart.add(bid, addedBook); %>
  ...
</c:forEach>
```

Functions

Table 14–9 lists the JSTL functions.

Table 14–9 Functions

Area	Function	Tags	Prefix
Functions	Collection length	`length`	`fn`

	String manipulation	`toUpperCase, toLowerCase`
		`substring, substringAfter, substringBefore`
		`trim`
		`replace`
		`indexOf, startsWith, endsWith, contains, containsIgnoreCase`
		`split, join`
		`escapeXml`	

Although the `java.util.Collection` interface defines a `size` method, it does not conform to the JavaBeans component design pattern for properties and so cannot be accessed via the JSP expression language. The `length` function can be applied to any collection supported by the `c:forEach` and returns the length of the collection. When applied to a `String`, it returns the number of characters in the string.

For example, the `index.jsp` page of the `hello1` application introduced in Chapter 3 uses the `fn:length` function and the `c:if` tag to determine whether to include a response page:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
  prefix="fn" %>
<html>
<head><title>Hello</title></head>
...
<input type="text" name="username" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
```

```
</form>

<c:if test="${fn:length(param.username) > 0}" >
  <%@include file="response.jsp" %>
</c:if>
</body>
</html>
```

The rest of the JSTL functions are concerned with string manipulation:

- `toUpperCase`, `toLowerCase`: Changes the capitalization of a string
- `substring`, `substringBefore`, `substringAfter`: Gets a subset of a string
- `trim`: Trims whitespace from a string
- `replace`: Replaces characters in a string
- `indexOf`, `startsWith`, `endsWith`, `contains`, `containsIgnoreCase`: Checks whether a string contains another string
- `split`: Splits a string into an array
- `join`: Joins a collection into a string
- `escapeXml`: Escapes XML characters in a string

Further Information

For further information on JSTL, see the following:

- The tag reference documentation:
  `http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html`
- The API reference documentation:
  `http://java.sun.com/products/jsp/jstl/1.1/docs/api/index.html`
- The JSTL 1.1 specification:
  `http://java.sun.com/products/jsp/jstl/downloads/index.html#specs`
- The JSTL web site:
  `http://java.sun.com/products/jsp/jstl`