Software Development

An Introduction to JSP Standard Template Library (JSTL)

By Jeff Heaton

Introduction

The JSP Standard Template Library (JSTL) is a very new component released by Sun fc JSP programming. JSTL allows you to program your JSP pages using tags, rather than the scriptlet code that most JSP programmers are already accustomed to. JSTL can do nearly everything that regular JSP scriptlet code can do. You may be wondering why we need yet another HTML generation programming language.

JSTL was introduced was to allow JSP programmers to program using tags rather than Java code. To show why this is preferable, a quick example is in order. We will examine a very simple JSP page that counts to ten. We will examine this page both as regular scriptlet-based JSP, and then as JSTL. When the count to ten example is programmed using scriptlet based JSP, the JSP page appears as follows.

```
<html>
  <head>
    <title>Count to 10 in JSP scriptlet</title>
  </head>
  <body>
<%
  for(int i=1;i<=10;i++)
{%>
<%=i%><br/>
<%
}
%>
  </body>
</html>
```

As you can see from the preceding example, using scriptlet code produces page sourc code the contains a mix of HTML tags and Java statements. There are several reasons why this mixing of programming styles is not optimal.

The primary reason that it is not optimal to mix scriptlet and tag-based code is readability. This readability apples both to humans and computers. JSTL allows the human programmer to look at a program that consists entirely of HTML and HTML-like tags.

The readability of JSP scriptlet code does not just apply to human beings. The mixing c scriptlet and HTML code is also hard for computers to read. This is particularly true of

HTML authoring tools such as Someone's Dreamweaver and Microsoft FrontPage. Currently, most HTML authoring tools will segregate JSP scriptlet code as non-editable blocks. The HTML authoring tools usually do not modify the JSP scriptlet code directly.

The following code shows how the count to ten example would be written using JSTL. As you can see, this code listing is much more constant, as only tags are used. HTML and JSTL tags are mixed to produce the example.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Count to 10 Example (using JSTL)</title>
  </head>

  <body>
    <c:forEach var="i" begin="1" end="10" step="1">
      <c:out value="${i}" />

      <br />
    </c:forEach>
  </body>
</html>
```

When you examine the preceding source code, you can see that the JSP page consists entirely of tags. The above code makes use of HTML tags such <head> and
. The use of tags is not confined just to HTML tags. This code also makes use of JSTL tags such as <c:forEach> and <c:out>. In this article you will be introduced to some of the basics of JSTL.

Installing JSTL

To use JSTL, you must have a JSP 1.2 (or higher) container installed. One of the most commonly used JSP containers is the Apache Tomcat Web server. You can obtain a copy of Tomcat from http://jakarta.apache.org/tomcat/. Using Tomcat alone will allow you to use regular JSP scriptlet code. To use JSTL, you must install JSTL into Tomcat. JSTL can be obtained from the same source as Tomcat. The main URL for JSTL is http://java.sun.com/products/jsp/jstl/. To use JSTL, you must unzip the distribution file and install them into the correct locations within Tomcat.

To properly install JSTL for use with Tomcat, follow these three steps:

1. **Copy the JSTL JAR files to Tomcat's lib directory.**

   If you are using Windows, the likely location of your lib directory is C: \ Program Files \ Apache Tomcat 4.0 \ webapps \ ROOT \ WEB-INF \ lib. There are a number of JAR files included with the JSTL release. You should copy each of these JAR files to the Tomcat JAR directory.

2. **Copy the JSTL TLD files to Tomcat's web-inf directory.**

   The web-inf directory is likely at this location: C: \ Program Files \ Apache Tomcat 4.0 \ webapps \ ROOT \ WEB-INF. If you examine the JSTL distribution files, you should notice eight files that end with the TLD extension. All eight files should be copied to your web-inf directory.

3. **Modify the web.xml file to include the TLD files.**

   Finally, you must modify your web.xml, and add an entry for all eight of the tag libraries that you added. This consists of adding <taglib> directives inside the mai <web-app> directive. The entries you should add are listed here.

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
  <taglib-location>/WEB-INF/fmt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/fmt-rt</taglib-uri>
  <taglib-location>/WEB-INF/fmt-rt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
  <taglib-location>/WEB-INF/c-rt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
  <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/sql-rt</taglib-uri>
  <taglib-location>/WEB-INF/sql-rt.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
  <taglib-location>/WEB-INF/x.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>http://java.sun.com/jstl/x-rt</taglib-uri>
  <taglib-location>/WEB-INF/x-rt.tld</taglib-location>
</taglib>
```

After completing the preceding three steps, you are now ready to test your installation JSTL. This can be done by creating a JSP page that uses JSTL. A very simple example would be the "count to ten" example shown before. You should place your JSP file insi the Webroot directory (C: \ Program Files \ Apache Tomcat 4.0 \ webapps \ ROOT). Onc the Tomcat server is started, you should now be able to browse to http://127.0.0.1:8080/count.jsp to see your page.

If you do not have JSTL installed correctly, there will likely be no error message. If JSTI is not interpreting your tags, they will be passed through directly to the Web browser. The Web browser will interpret these tags as unknown HTML tags. Most browsers simpl ignore the unknown HTML tags.

The JSTL Tag Libraries

JSTL is often spoken of as a single-tag library. JSTL is actually four tag libraries. These tag libraries are summarized as follows.

- **Core Tag Library**—Contains tags that are essential to nearly any Web application. Examples of core tag libraries include looping, expression evaluation, and basic input and output.
- **Formatting/Internationalization Tag Library**—Contains tags that are used to and parse data. Some of these tags will parse data, such as dates, differently based o the current locale.
- **Database Tag Library**—Contains tags that can be used to access SQL databases. These tags are normally used only to create prototype programs. This is because most programs will not handle database access directly from JSP pages. Databas access should be embedded in EJBs that are accessed by the JSP pages.
- **XML Tag Library**—Contains tags that can be used to access XML elements. Because XML is used in many Web applications, XML processing is an important feature of JSTL.

In this article, we will only take a brief look at a few of the core tags. We will examine a simple example that shows how to process data that a user enters into a form. Before we examine this program, we must first see how JSTL handles expressions. Expression handling in JSTL is accomplished by using the EL expression language, just as it is don in JSP 2.0. In the next section, we will examine the EL expression language.

The EL Expression Language

One major component of JSP 2.0 is the new expression language named EL. EL is use extensively in JSTL. However, it is important to remember that EL is a feature of JSP an not of JSTL. JSP scriptlet code used with JSP 2.0 can contain EL expressions. The following lines of code demonstrate using EL inside of JSP scriptlet code.

```
<p>
  Your total, including shipping is ${total+shipping}
</p>
```

As you can see form the preceding code, the values "total" and "shipping" are added a displayed as the HTML is generated. These expressions can be used inside of JSTL tag as well. One important requirement of JSTL 1.0 was that JSTL could be used with JSP 1.2. Because JSP 1.2 does not support EL, it is necessary to provide a few additional JSTL tags that facilitate the use of EL. For example, if you wanted to use JSTL to displa the above expression, you would use the following code.

```
<p>
  Your total, including shipping is <c:out var="${total+shipping"/>
</p>
```

One of the requirements of JSTL was that it not require JSP 2.0 to run. By providing a t that is capable of displaying EL expressions, this requirement is met.

JSTL Example

We will now examine a simple example that uses JSTL. For this example, we will examine a common procedure that is done by many Web applications. We will see how to POST a form and process the results from that POST. A simple program that is capable of doing this is shown below.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>If with Body</title>
  </head>

  <body>
    <c:if test="${pageContext.request.method=='POST'}">
      <c:if test="${param.guess=='Java'}">You guessed it!
      <br />

      <br />

      <br />
      </c:if>

      <c:if test="${param.guess!='Java'}">You are wrong
      <br />

      <br />

      <br />
      </c:if>
    </c:if>

    <form method="post">Guess what computer language
                        I am thinking of?
    <input type="text" name="guess" />

    <input type="submit" value="Try!" />

    <br />
    </form>
  </body>
</html>
```

This simple Web page will display a form and ask the user to guess what computer language the program is thinking of. Of course, the computer is thinking of "Java." This page begins by checking to see if a POST was done. This allows both the form, and the code that handles the form, to be placed on one single page. This is done with the following JSTL if statement.

```
    <c:if test="${pageContext.request.method=='POST'}">
```

Here you can see that the <c:if> tag uses an EL expression to evaluate whether the request method is POST. If data was posted to the page, the value that the user entered for their guess is stored in a parameter named "guess". This is because "guess" was specified as the name of the form input item. We must now check to see whether this parameter is equal to the word "Java". This is done with the following <c:if> tag.

```
      <c:if test="${param.guess=='Java'}">
```

```
        You guessed it!
    </c:if>
```

As you can see, the body of the <c:if> tag is executed if the statement evaluates to tru
In this article, we began to examine the basics of how JSTL is installed and how it work
There is much more to JSTL than the small example we examined in this article.

The core tags of JSTL also include tags for looping, iteration, and variable handling. By using these tags, you can iterate through collections, access user session data, and perform other core tasks that all Web applications perform. In addition to the core tag library, the XML, database, and formatting tag libraries are also provided for more advanced uses.

Conclusions

This article showed you some of the differences between the JSTL and standard JSP scriptlet programming. As you can see, JSTL allows a more consistent programming environment by allowing both HTML and procedural code to be expressed as tags. JST and tag libraries represent a new method of programming Web pages.

JSTL does not provide everything that a programmer would need to create a full-featur Web application. Further, some procedures that could be programmed in JSTL is often best not programmed in JSTL. One perfect example of this is the database JSTL tags. Except for very small Web applications, it is generally considered bad programming practice to embed actual database commands into a JSP page. The proper location for such program code is in Java beans and EJBs that will be used by your Web applicatio For such routines, you may consider creating your own tag libraries. This way, your JSI pages can use JSTL to perform basic programming procedures that are not unique to your business. You should implement your own tag libraries to implement components, which are unique to your business, which will be used by your Web application.

JSTL allows you to create a very consistent programming JSP-based application. This done not just through JSTL, but through a combination of JSTL, your own custom tag libraries, and an underlying database. Understanding each of these components allows you to deploy an effective Web application.

Author Bio

Jeff Heaton is the author of the upcoming *JSTL: JSP Standard Tag Library* (Sams, 2002 Jeff works as a software designer for Reinsurance Group of America. Jeff has written three books and numerous magazine articles about computer programming. Jeff may b contacted through his Web site http://www.jeffheaton.com.

August 15, 2002