CHAPTER 11: NEW TAG LIBRARY FEATURES IN JSP 1.2

Topics in This Chapter

- Converting TLD files to the new format
- Bundling life-cycle event listeners with tag libraries
- Checking custom tag syntax with `TagLibraryValidator`
- Using the Simple API for XML (SAX) in validators
- Handling errors with the `TryCatchFinally` interface
- Changing names of method return values
- Looping without creating `BodyContent`
- Declaring scripting variables in the TLD file

Chapter 11

Section 3.7 (Defining Custom JSP Tag Libraries) describes the creation and use of tag libraries in JSP 1.1. You do not need to modify JSP 1.1 tag libraries to make them work in JSP 1.2; they are totally compatible. However, JSP 1.2 has a number of new capabilities that are unavailable in JSP 1.1. Tag libraries that use these new capabilities must use a slightly different format for the TLD file. This chapter first describes that new format and then explains the use of each of the new capabilities. The following list gives a brief summary; details are provided in the following sections.

- **New TLD format.** The `DOCTYPE` definition has changed, some existing elements have been renamed (mostly by the addition of dashes), and several new elements have been introduced.
- **Ability to bundle listeners with tag libraries.** The servlet 2.3 specification introduced application life-cycle listeners. The JSP 1.2 specification added the ability to bundle these listeners with tag libraries.
- **`TagLibraryValidator` for translation-time syntax checking.** JSP 1.2 permits you to create a class that, at page translation time, can read an entire JSP file (represented in XML) and check that the custom tags are used properly.
- **`TryCatchFinally` interface.** JSP 1.2 introduced a new interface with two methods: `doCatch` and `doFinally`. These methods help tags handle uncaught exceptions that occur in any of the tag life-cycle methods or during the processing of the tag body.

- **New return values.** In JSP 1.1, EVAL_BODY_TAG was used in two distinct situations. In JSP 1.2, the constant is replaced with EVAL_BODY_BUFFERED and EVAL_BODY_AGAIN to better differentiate the two cases.
- **Looping without creating BodyContent.** In JSP 1.1, iteration-related tags extended BodyTagSupport and returned EVAL_BODY_TAG from doAfterBody to indicate that the tag body should be reevaluated and made available in a BodyContent object. In JSP 1.2, you can extend TagSupport and return EVAL_BODY_AGAIN to indicate that the body should be reevaluated and sent to the client with no intervening BodyContent. This yields an implementation that is easier to read and more efficient.
- **The variable element for introducing scripting variables.** Instead of declaring variables only in the TagExtraInfo class, JSP 1.2 lets you declare them in the TLD file.

11.1 Using the New Tag Library Descriptor Format

Tag libraries that make use of any new JSP 1.2 capabilities must use a new format for their tag library descriptor (TLD) files. However, libraries that are compatible with JSP 1.1 are allowed to use the JSP 1.1 TLD format.

TLD files in JSP 1.2 differ in the following ways from JSP 1.1 TLD files:

- The DOCTYPE declaration has changed.
- Several elements have been renamed.
- New elements have been added.

Each of these changes is described in one of the following subsections. The final subsection gives a side-by-side comparison of the JSP 1.2 and 1.1 TLD file formats.

New DOCTYPE Declaration

Tag library descriptor files start with an XML header, then have a DOCTYPE declaration and a taglib element. The XML header is unchanged from JSP 1.1 to 1.2, but for the DOCTYPE declaration, you now use:

```
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
```

In JSP 1.1 you used:

```
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
```

Listing 11.1 briefly outlines the resultant file.

Listing 11.1 JSP 1.2 TLD File (Excerpt 1)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <!-- ... -->
</taglib>
```

Renamed Elements

Within the `taglib` element in JSP 1.2, dashes were added to the `tlibversion`, `jspversion`, and `shortname` elements, yielding `tlib-version`, `jsp-version`, and `short-name`, respectively. The `tlib-version` element indicates the version number of the tag library (an arbitrary number), `jsp-version` indicates the JSP version needed (1.2), and `short-name` defines a name that IDEs can use to refer to the library (no spaces are allowed because IDEs might use `short-name` as the default tag prefix). The JSP 1.1 `info` element (used for documentation) was renamed to `description`.

Within the `tag` element, dashes were added to `tagclass`, `teiclass`, and `bodycontent` elements, yielding `tag-class`, `tei-class`, and `body-content`, respectively. The `tag-class` element gives the fully qualified name of the tag implementation class, `tei-class` defines a `TagExtraInfo` class for validation (this is optional; see Section 11.3 for a new and better alternative), and `body-content` provides IDEs a hint as to whether the tag is empty (i.e., uses no separate end tag) or uses a tag body between its start and end tags. The JSP 1.1 `info` element (used for documentation) was renamed to `description`.

Listing 11.2 gives a brief outline of a typical TLD file. Remember that the order of elements within XML files is not arbitrary. You must use the elements in the order shown here.

Listing 11.2 JSP 1.2 TLD File (Excerpt 2)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>some-number</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>some-name</short-name>
  <description>Tag library documentation.</description>

  <tag>
    <name>tag-name</name>
    <tag-class>somePackage.SomeTag</tag-class>
    <body-content>empty, JSP, or tagdependent</body-content>
    <description>Tag documentation.</description>
  </tag>

  <!-- Other tag elements, if any. -->
</taglib>
```

New Elements

Within the `taglib` element, five new elements were added: `display-name`, `small-icon`, `large-icon`, `validator`, and `listener`. All are optional.

The first three of these supply information for IDEs and other authoring tools. If used, they must appear in the order listed here and must be placed immediately before the `description` element within `taglib`. The `display-name` element gives a short name that the IDEs can present to the author; it differs from `short-name` in that it can contain white space (`short-name` cannot) and is used only for identification, not as a preferred prefix in a `taglib` directive (as `short-name` could be). The `small-icon` element gives the location of a 16 × 16 GIF or JPEG image that can be used by the IDE. The `large-icon` element gives the location of a 32 × 32 image, also in GIF or JPEG format.

The `validator` element declares a `TagLibraryValidator` class to be used for page translation-time syntax checking. Its use is discussed in Section 11.3. This element, if used, must appear in the `taglib` element after `description` but before `listener` (if used) and `tag`.

The `listener` element declares an application life-cycle event listener that will be loaded when the Web application is loaded (not when the tag library is first used!). Use of the `listener` element is discussed in Section 11.2; use of life-cycle listeners in general is described in Chapter 10. The `listener` element, if used, must be the last element before `tag`.

There were also five new elements that can appear within the `tag` element: `display-name`, `small-icon`, `large-icon`, `variable`, and `example`. All are optional. The first three (`display-name`, `small-icon`, and `large-icon`) are used for IDE documentation in the same way as just described for the `taglib` element. The elements, if used, must appear after `body-content` but before `description`. The `variable` element is used to introduce scripting variables. It must appear after `description` but before `attribute`; its use is described in Section 11.8. Finally, the `example` element gives a simple textual example of the use of the tag. If used, the `example` element must be the last subelement within `tag`.

Summary

Table 11.1 summarizes the first-, second-, and third-level elements in tag library descriptor files, listed in the order in which they must be used. Items in bold indicate changes from JSP 1.1. Elements marked with an asterisk are optional.

Table 11.1 Tag library descriptor format.

JSP 1.2	*JSP 1.1*
`<?xml version="1.0" encoding="ISO-8859-1" ?>`	`<?xml version="1.0" encoding="ISO-8859-1" ?>`
JSP 1.2 DOCTYPE declaration	JSP 1.1 DOCTYPE declaration
taglib	taglib
tlib-version	tlibversion
jsp-version	jspversion
short-name	shortname
uri*	uri*
display-name*	
small-icon*	
large-icon*	
description*	info*
validator*	
validator-class	
init-param*	
description*	

Table 11.1 Tag library descriptor format. *(continued)*

JSP 1.2	*JSP 1.1*
listener*	
listener-class	
tag	tag
name	name
tag-class	tagclass
tei-class*	teiclass*
body-content*	bodycontent*
display-name*	
small-icon*	
large-icon*	
description*	info*
variable*	
attribute*	attribute*
example*	

11.2 Bundling Listeners with Tag Libraries

Application life-cycle event listeners are described in Chapter 10. They provide a powerful new capability that lets you respond to the creation and deletion of the servlet context and `HttpSession` objects and lets you monitor changes in servlet context and session attributes. In most cases, listeners are declared with the `listener` element of the deployment descriptor (*web.xml*).

However, suppose that the behavior of a tag library depends upon an event listener. In such a case, you would want to be certain that the listeners were available in all Web applications that used the tag library.

If the `listener` and `listener-class` elements of *web.xml* were the only option for declaring listeners, tag library maintenance would be difficult. Normally, the user of a tag library deploys it by simply dropping a JAR file in *WEB-INF/lib* and putting a TLD file in *WEB-INF*. Users of the tag libraries need no knowledge of the individual classes within the library, only of the tags that the library defines. But if the tag libraries used listeners, users of the libraries would need to discover the name of the listener classes and make *web.xml* entries for each one. This would be less flexible and harder to maintain.

Fortunately, the JSP 1.2 specification lets you put the listener declarations in the tag library descriptor file instead of in the deployment descriptor. "Hold on!" you say, "Event listeners need to run when the Web application is first loaded, not just the first time a JSP page that uses a custom library is accessed. I thought TLD files were only loaded the first time a user requests a page that refers to it. How can this work?" Good question. JSP 1.2 introduced a new TLD search mechanism to handle this situation. When a Web application is loaded, the system automatically searches *WEB-INF* and its subdirectories for files with *.tld* extensions and uses all `listener` declarations that it finds in them. This means that your TLD files *must* be in the *WEB-INF* directory or a subdirectory thereof. In fact, although few servers enforce the restriction, the JSP 1.2 specification requires all TLD files to be in *WEB-INF* anyhow. Besides, putting the TLD files in *WEB-INF* is a good strategy to prevent users from retrieving them. So, you should make *WEB-INF* the standard TLD file location, regardless of whether your libraries use event handlers.

Core Approach

Always put your TLD files in the WEB-INF *directory or a subdirectory thereof.*

Unfortunately, there is a problem with this approach: Tomcat 4.0 ignores TLD files at Web application startup time unless there is a `taglib` entry in *web.xml* of the following form:

```
<taglib>
  <taglib-uri>/someName.tld</taglib-uri>
  <taglib-location>/WEB-INF/realName.tld</taglib-location>
</taglib>
```

As discussed in Section 5.13 (Locating Tag Library Descriptors), this entry is a good idea when the name of your tag library changes frequently. However, the JSP 1.2 specification does not require its use, and servers such as ServletExec 4.1 properly handle listener declarations in TLD files when there is no such entry. Nevertheless, Tomcat 4.0 requires it.

Core Warning

Tomcat 4.0 reads listener declarations only from TLD files that have `taglib` *entries in* web.xml.

Tracking Active Sessions

At busy sites, a significant portion of the server's memory can be spent storing `HttpSession` objects. You might like to track session usage so that you can decide if you should lower the session timeout, increase the server's memory allotment, or even use a database instead of the servlet session API.

Listing 11.3 shows a session listener that keeps a running count of the number of sessions in memory. The count is incremented each time a session is created; it is decremented whenever a session is destroyed (regardless of whether the session destruction is from an explicit call to `invalidate` or from timing out).

Listing 11.4 gives a custom tag that simply prints the count of active sessions. Listing 11.5 presents a related tag that prints a large, red warning if the number of sessions in memory exceeds a predefined maximum. Nothing is printed if the number of active sessions is within bounds.

Since it doesn't make sense to use these tags unless the listener is in effect, the TLD file that declares the tags (Listing 11.6) also declares the listener. Finally, an alias for the TLD file is created with the `taglib` element of the *web.xml* deployment descriptor (Listing 11.7) to ensure that Tomcat will read the TLD file when the Web application is loaded and to allow developers to change the name of the TLD file without modifying the JSP pages that use it.

Listing 11.8 shows a JSP page that uses both of the custom tags. Figures 11–1 and 11–2 show the results when the number of sessions is below and above the predefined limit, respectively.

Listing 11.3 *ActiveSessionCounter.java*

```
package moreservlets.listeners;

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

/** Listener that keeps track of the number of sessions
 *  that the Web application is currently using.
 */

public class ActiveSessionCounter
                             implements HttpSessionListener {
  private static int sessionCount = 0;
  private static int sessionLimit = 1000;
  private ServletContext context = null;

  /** Each time a session is created, increment the
   *  running count. If the count exceeds the limit,
   *  print a warning in the log file.
   */
```

Listing 11.3 *ActiveSessionCounter.java (continued)*

```
public void sessionCreated(HttpSessionEvent event) {
  sessionCount++;
  if (context == null) {
    recordServletContext(event);
  }
  String warning = getSessionCountWarning();
  if (warning != null) {
    context.log(warning);
  }
}

/** Each time a session is destroyed, decrement the
 *  running count. A session can be destroyed when a
 *  servlet makes an explicit call to invalidate, but it
 *  is more commonly destroyed by the system when the time
 *  since the last client access exceeds a limit.
 */

public void sessionDestroyed(HttpSessionEvent event) {
  sessionCount--;
}

/** The number of sessions currently in memory. */

public static int getSessionCount() {
  return(sessionCount);
}

/** The limit on the session count. If the number of
 *  sessions in memory exceeds this value, a warning
 *  should be issued.
 */

public static int getSessionLimit() {
  return(sessionLimit);
}

/** If the number of active sessions is over the limit,
 *  this returns a warning string. Otherwise, it returns
 *  null.
 */
```

Listing 11.3 *ActiveSessionCounter.java (continued)*

```
  public static String getSessionCountWarning() {
    String warning = null;
    if (sessionCount > sessionLimit) {
      warning = "WARNING: the number of sessions in memory " +
                "(" + sessionCount + ") exceeds the limit " +
                "(" + sessionLimit + "). Date/time: " +
                new Date();
    }
    return(warning);
  }

  private void recordServletContext(HttpSessionEvent event) {
    HttpSession session = event.getSession();
    context = session.getServletContext();
  }
}
```

Listing 11.4 *SessionCountTag.java*

```
package moreservlets.tags;

import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
import moreservlets.listeners.*;

/** Prints out the number of active sessions. */

public class SessionCountTag extends TagSupport {
  public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print(ActiveSessionCounter.getSessionCount());
    } catch(IOException ioe) {
      System.out.println("Error printing session count.");
    }
    return(SKIP_BODY);
  }
}
```

Listing 11.5 *SessionCountWarningTag.java*

```
package moreservlets.tags;

import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
import moreservlets.listeners.*;

/** If the number of active sessions is above the limit,
 *  this prints a warning. Otherwise, it does nothing.
 */

public class SessionCountWarningTag extends TagSupport {
  public int doStartTag() {
    try {
      String warning =
        ActiveSessionCounter.getSessionCountWarning();
      if (warning != null) {
        JspWriter out = pageContext.getOut();
        out.println("<H1><FONT COLOR=\"RED\">");
        out.println(warning);
        out.println("</FONT></H1>");
      }
    } catch(IOException ioe) {
      System.out.println("Error printing session warning.");
    }
    return(SKIP_BODY);
  }
}
```

Listing 11.6 *session-count-taglib-0.9-beta.tld*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>0.9</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>company-name-tags</short-name>
```

Listing 11.6 *session-count-taglib-0.9-beta.tld (continued)*

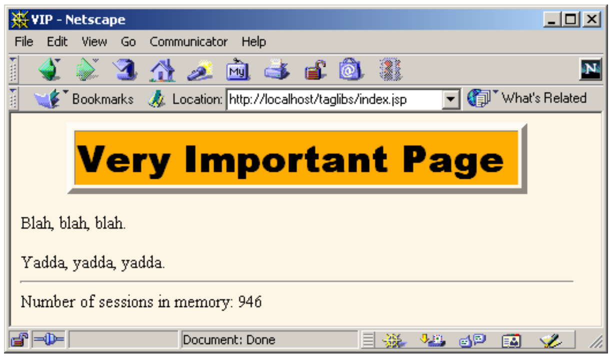
```
  <description>
    A tag library that lets you print out the number of
    sessions currently in memory and/or a warning about
    the session count exceeding the limit.

    The tlib-version number and the TLD filename are intended
    to suggest that this tag library is in development. In
    such a situation, you want to use the web.xml taglib
    element to define an alias for the TLD filename. You
    would want to do so even if you weren't trying
    to accommodate Tomcat 4.0, which only reads listener
    declarations from TLD files that are declared that way.
  </description>

  <!-- Register the listener that records the counts. -->
  <listener>
    <listener-class>
      moreservlets.listeners.ActiveSessionCounter
    </listener-class>
  </listener>

  <!-- Define a tag that prints out the session count. -->
  <tag>
    <name>sessionCount</name>
    <tag-class>
      moreservlets.tags.SessionCountTag
    </tag-class>
    <body-content>empty</body-content>
    <description>The number of active sessions.</description>
  </tag>

  <!-- Define a tag that prints out an optional
       session-count warning. -->
  <tag>
    <name>sessionCountWarning</name>
    <tag-class>
      moreservlets.tags.SessionCountWarningTag
    </tag-class>
    <body-content>empty</body-content>
    <description>
      If the number of sessions exceeds the limit,
      this prints a warning. Otherwise, it does nothing.
    </description>
  </tag>
</taglib>
```

Listing 11.7 *web.xml* (For session-counting tags)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- If URL gives a directory but no filename, try index.jsp
       first and index.html second. If neither is found,
       the result is server specific (e.g., a directory
       listing).  Order of elements in web.xml matters.
       welcome-file-list needs to come after servlet but
       before error-page.
  -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <!-- Register the company-name tag library. Declare an alias
       for the TLD filename since the tag library is under
       development and thus the TLD filename might change.
       You don't want to change all the JSP files each time
       the TLD file changes. Besides, Tomcat 4.0 won't pick
       up listener declarations from TLD files unless they
       are declared this way.
  -->
  <taglib>
    <taglib-uri>
      /session-count-taglib.tld
    </taglib-uri>
    <taglib-location>
      /WEB-INF/session-count-taglib-0.9-beta.tld
    </taglib-location>
  </taglib>
</web-app>
```

Listing 11.8 *index.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>VIP</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Very Important Page
</TABLE>
<P>
Blah, blah, blah.
<P>
Yadda, yadda, yadda.
<HR>
<!-- Note that the uri refers to the location defined by
     the taglib element of web.xml, not to the real
     location of the TLD file. -->
<%@ taglib uri="/session-count-taglib.tld" prefix="counts" %>
Number of sessions in memory:  <counts:sessionCount/>
<counts:sessionCountWarning/>
</BODY>
</HTML>
```


VIP - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/taglibs/index.jsp
What's Related
Document: Done

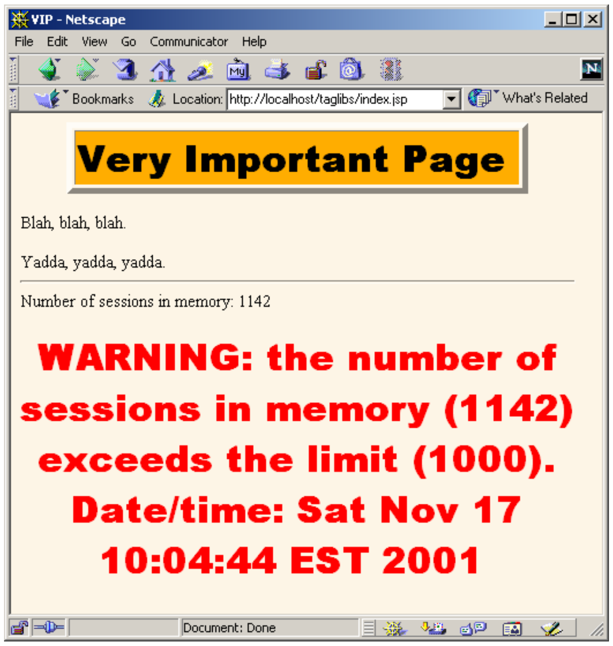
Very Important Page

Blah, blah, blah.

Yadda, yadda, yadda.

Number of sessions in memory: 946

Figure 11–1 When the number of sessions in memory is less than the limit, the `sessionCountWarning` tag does not generate any output.


VIP - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/taglibs/index.jsp
What's Related
Document: Done

Very Important Page

Blah, blah, blah.

Yadda, yadda, yadda.

Number of sessions in memory: 1142

WARNING: the number of sessions in memory (1142) exceeds the limit (1000). Date/time: Sat Nov 17 10:04:44 EST 2001

Figure 11–2 When the number of sessions in memory exceeds the limit, the `sessionCountWarning` tag generates a warning.

Testing Session Counts

On a development system, it is often difficult to test session usage because few (if any) outside users are accessing the server. So, it is helpful to generate sessions manually. The simplest way to do this is to disable cookies in your browser and access a framed page that loads the same JSP page multiple times. For details on the process, see Section 10.7 (Recognizing Session Creation and Destruction).

To test the session-counting page shown earlier in this section, I used the same JSP page as in Section 10.7—a blank page with a randomized background color (Listing 11.9; uses the utility class of Listing 11.10). Listing 11.11 shows a frame-based page that, each time it is loaded, loads 49 copies of the JSP page. Figure 11–3 shows a typical result.

Listing 11.9 *test.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- The purpose of this page is to force the system
     to create a session. -->
<HTML>
<HEAD><TITLE>Test</TITLE></HEAD>

<%@ page import="moreservlets.*" %>
<BODY BGCOLOR="<%= ColorUtils.randomColor() %>">

</BODY></HTML>
```

Listing11.10 *ColorUtils.java*

```
package moreservlets;

/** Small utility to generate random HTML color names. */

public class ColorUtils {
  // The official HTML color names.
  private static String[] htmlColorNames =
    { "AQUA", "BLACK", "BLUE", "FUCHSIA", "GRAY", "GREEN",
      "LIME", "MAROON", "NAVY", "OLIVE", "PURPLE", "RED",
      "SILVER", "TEAL", "WHITE", "YELLOW" };

  public static String randomColor() {
    int index = randomInt(htmlColorNames.length);
    return(htmlColorNames[index]);
  }

  // Returns a random number from 0 to n-1 inclusive.

  private static int randomInt(int n) {
    return((int)(Math.random() * n));
  }
}
```

Listing11.11 *make-sessions.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
  <TITLE>Session Testing...</TITLE>
</HEAD>
<!-- Load the same JSP page 49 times. If cookies are
     disabled and the server is using cookies for session
     tracking (the default), loading this HTML page causes
     the system to create 49 new sessions. -->
<FRAMESET ROWS="*,*,*,*,*,*,*" COLS="*,*,*,*,*,*,*">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp"><FRAME SRC="test.jsp">
  <FRAME SRC="test.jsp">
  <NOFRAMES><BODY>
    This example requires a frame-capable browser.
  </BODY></NOFRAMES>
</FRAMESET>
</HTML>
```

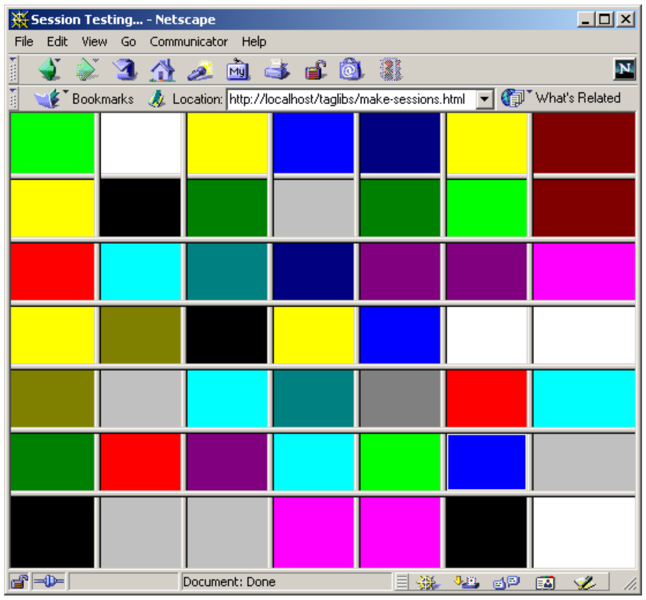

Session Testing... - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/taglibs/make-sessions.html
What's Related
Document: Done

Figure 11–3 The page used to force the system to create new sessions.

11.3 Checking Syntax with TagLibraryValidator

Your tag handlers (i.e., classes that implement the `Tag` interface, usually by extending `TagSupport` or `BodyTagSupport`) can handle some simple usage errors at request time. For instance, if your tag has an attribute that expects an integer, the associated `set`*`Xxx`* method should convert the incoming `String` to an `int` by putting the call to `Integer.parseInt` within a `try/catch` block to handle situations where a malformed value is supplied. In addition to these request time checks, the system will automatically do simple syntax checking at page translation time based on information supplied in the TLD file. For example, if you misspell an attribute or omit one that is listed as required, the system will flag these errors when the JSP page is translated into a servlet.

Both of these approaches are fine for simple cases; neither is sufficient for complicated situations. For example, suppose the TLD file lists two attributes as optional, but it doesn't make sense to use one without the other. How do you enforce that both are supplied or both are omitted? In principle, your tag handler could enforce these restrictions, but doing so would make the code significantly more complicated. Besides, errors of this type should be recognized at page translation time; if you can catch them at page translation time, why slow down request time execution by checking for them? Other syntax errors simply cannot be detected by tag handlers. For example, suppose that a certain tag must always contain one of two possible subelements or cannot contain a certain subelement. The nested tags can discover their enclosing tags with `findAncestorWithClass` (see Section 3.7), but how does the enclosing tag enforce which nested tags it contains?

JSP 1.1 provides a `TagExtraInfo` class that lets you perform some of these checks. However, `TagExtraInfo` is difficult to use and limited in capability. So, JSP 1.2 introduced a new class called `TagLibraryValidator` that lets you perform arbitrary page translation-time syntax checks. This class has a `validate` method indirectly giving you an `InputStream` that lets you read the entire JSP page (represented in XML format), so you have full access to all available translation-time information about the page. In most cases you don't read the input directly from the input stream. Instead, you use an XML-based API like SAX, DOM, or JDOM to look at the various XML elements (start tags, end tags, tag attributes, and tag bodies) of the file. In such a situation, the HTML content is represented as the body of a `jsp:text` element and is usually ignored by the validator.

Using a validator consists of the following steps.

1. **Create a subclass of `TagLibraryValidator`.** Note that `TagLibraryValidator` is in the `javax.servlet.jsp.tagext` package.
2. **Override the `validate` method.** This method takes three arguments: two strings giving the `prefix` and `uri` declared in the `taglib` directive of the JSP page using the tag library, and a `PageData` object. Call `getInputStream` on the `PageData` object to obtain a stream that lets you read the XML representation of the JSP page. This stream is typically passed to an XML parser, and your `validate` method deals with the XML elements, not the raw characters. If the syntax checks out properly, return `null` from `validate`. If there are errors, return an array of `ValidationMessage` objects, each of which is built by calling the `ValidationMessage` constructor with the ID of the tag (usually `null` since servers are not required to give IDs to tags) and a `String` describing the problem. Here is a simplified example:

```
public ValidationMessage[] validate(String prefix,
                                    String uri,
                                    PageData page) {
  InputStream stream = page.getInputStream();
  BookOrder[] orders = findBookOrders(stream);
  for(int i=0; i<orders.length; i++) {
    String title = orders[i].getTitle();
    int numOrdered = orders[i].getNumOrdered();
    if (title.equals("More Servlets and JavaServer Pages") &&
        (numOrdered < 100)) {
      String message = "Too few copies of MSAJSP ordered!";
      ValidationMessage[] errors =
        { new ValidationMessage(null, message) };
      return(errors);
    }
  }
  return(null);
}
```

3. **Optionally override other methods.** You read validator initialization parameters (supplied in the TLD file with the `init-param` subelement of `validator`) with `getInitParameters`. You can set initialization parameters with `setInitParameters`. If you store persistent values in fields of your validator, use `release` to reset the fields—validator instances, like tag instances, can be reused by servers.
4. **Declare the validator in the TLD file.** Use the `validator` element with a `validator-class` subelement and optionally one or more `init-param` subelements and a `description` element. The `validator` element goes after `description` but before `listener` and `tag` in the TLD file. Here is a simplified example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE ...>
<taglib>
  <tlib-version>...</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>...</short-name>
  <description>...</description>
  <validator>
    <validator-class>
      somePackage.SomeValidatorClass
    </validator-class>
  </validator>
  <tag>...</tag>
</taglib>
```

5. **Try JSP pages that use the tag library.** First, deploy your Web application. Second, see if you get errors when the JSP pages are translated. If you use `load-on-startup` (see Section 5.5, “Initializing and Preloading Servlets and JSP Pages”), the JSP page is translated into a servlet when the server is loaded. Otherwise, it is translated the first time it is accessed. If the `validate` method returns `null`, no action is taken. If `validate` returns a nonempty array, the server makes the error messages available in some server-specific manner.

Remember, however, that each JSP page is only translated into a servlet once. Unless the JSP page is modified, it doesn't get retranslated *even if the server is restarted*. This means that, during development, you have to be sure to modify your JSP pages whenever you want to test a change in your validator. It is also possible to delete the servlet that resulted from the JSP page, but that servlet is stored in a server-specific location. I usually just add and then delete a space in the JSP page, then redeploy it.

Core Warning

Tag library validators are only triggered when the associated JSP pages are translated into servlets. So, if you modify a validator, be sure to modify and redeploy the JSP pages that it applies to.

Example: Tracing the Tag Structure

To become acquainted with validators, let's make a validator that simply discovers all of the start and end tags, tag attributes, and tag bodies. It will print a summary to standard output and always return `null` to indicate that no syntax errors were found. I'll show a more interesting validator in the next subsection.

Even this simple task would be a lot of work if we had to parse the JSP page ourselves. Fortunately, since the page is represented in XML, we can use an XML parser and one of the standardized APIs like SAX, DOM, or JDOM. I'll use the Apache Xerces parser and the SAX API in this example. I'll also use the Java API for XML Parsing (JAXP) so that I can switch from the Apache parser to another SAX-compliant parser by changing only a single property value. If you aren't familiar with SAX and JAXP, Section 11.4 summarizes their use.

Accomplishing this task involves the following steps.

1. **Creation of a subclass of `TagLibraryValidator`.** Listing 11.12 shows a class called `SAXValidator` that extends `TagLibraryValidator`.
2. **Overriding of the `validate` method.** I take the third argument to `validate` (the `PageData` object), call `getInputStream`, and pass that to the SAX `InputSource` constructor. I then tell SAX to parse the JSP document using that `InputSource` and a handler called `PrintHandler` (Listing 11.13). This handler simply prints to standard output the start tags (with attributes), the end tags, and the first word of each tag body. Again, see Section 11.4 if you are unfamiliar with the SAX API.

3. **Declaration of the validator in the TLD file.** Listing 11.14 shows an updated version of the TLD file for the session-counting example, with a `validator` entry added.
4. **Try JSP pages that use the tag library.** Listing 11.15 shows the standard output that results when *index.jsp* (shown earlier in Listing 11.8 and Figures 11–1 and 11–2) is accessed.

Listing11.12 *SAXValidator.java*

```
import javax.servlet.jsp.tagext.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/** A "validator" that really just prints out an outline
 *  of the JSP page (in its XML representation). The
 *  validate method always returns null, so the page is
 *  always considered valid.
 */

public class SAXValidator extends TagLibraryValidator {
  /** Print an outline of the XML representation of
   *  the JSP page.
   */
  public ValidationMessage[] validate(String prefix,
                                      String uri,
                                      PageData page) {
    String jaxpPropertyName =
      "javax.xml.parsers.SAXParserFactory";
    // Pass the parser factory in on the command line with
    // -D to override the use of the Apache parser.
    if (System.getProperty(jaxpPropertyName) == null) {
      String apacheXercesPropertyValue =
        "org.apache.xerces.jaxp.SAXParserFactoryImpl";
      System.setProperty(jaxpPropertyName,
                         apacheXercesPropertyValue);
    }
    DefaultHandler handler = new PrintHandler();
    SAXParserFactory factory = SAXParserFactory.newInstance();
    try {
      SAXParser parser = factory.newSAXParser();
      InputSource source =
        new InputSource(page.getInputStream());
      parser.parse(source, handler);
```

Listing11.12 *SAXValidator.java (continued)*

```
    } catch(Exception e) {
      String errorMessage =
        "SAX parse error: " + e;
      System.err.println(errorMessage);
      e.printStackTrace();
    }
    return(null);
  }
}
```

Listing11.13 *PrintHandler.java*

```
package moreservlets;

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.StringTokenizer;

/** A SAX handler that prints out the start tags, end tags,
 *  and first word of tag body. Indents two spaces
 *  for each nesting level.
 */

public class PrintHandler extends DefaultHandler {
  private int indentation = 0;

  /** When you see a start tag, print it out and then increase
   *  indentation by two spaces. If the element has
   *  attributes, place them in parens after the element name.
   */

  public void startElement(String namespaceUri,
                           String localName,
                           String qualifiedName,
                           Attributes attributes)
      throws SAXException {
    indent(indentation);
    System.out.print("<" + qualifiedName);
    int numAttributes = attributes.getLength();
    // For <someTag> just print out "<someTag>". But for
    // <someTag att1="Val1" att2="Val2"> (or variations
    // that have extra white space), print out
    // <someTag att1="Val1" att2="Val2">.
```

Listing11.13 *PrintHandler.java (continued)*

```
    if (numAttributes > 0) {
      for(int i=0; i<numAttributes; i++) {
        System.out.print(" ");
        System.out.print(attributes.getQName(i) + "=\"" +
                         attributes.getValue(i) + "\"");
      }
    }
    System.out.println(">");
    indentation = indentation + 2;
  }

  /** When you see the end tag, print it out and decrease
   *  indentation level by 2.
   */

  public void endElement(String namespaceUri,
                         String localName,
                         String qualifiedName)
      throws SAXException {
    indentation = indentation - 2;
    indent(indentation);
    System.out.println("</" + qualifiedName + ">");
  }

  /** Print out the first word of each tag body. */

  public void characters(char[] chars,
                         int startIndex,
                         int length) {
    String data = new String(chars, startIndex, length);
    // White space makes up default StringTokenizer delimiters
    StringTokenizer tok = new StringTokenizer(data);
    if (tok.hasMoreTokens()) {
      indent(indentation);
      System.out.print(tok.nextToken());
      if (tok.hasMoreTokens()) {
        System.out.println("...");
      } else {
        System.out.println();
      }
    }
  }

  private void indent(int indentation) {
    for(int i=0; i<indentation; i++) {
      System.out.print(" ");
    }
  }
}
```

Listing11.14 *session-count-taglib-0.9-beta.tld* (Updated)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>0.9</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>company-name-tags</short-name>
  <description>
    A tag library that lets you print out the number of
    sessions currently in memory and/or a warning about
    the session count exceeding the limit.

    The tlib-version number and the TLD filename are intended
    to suggest that this tag library is in development. In
    such a situation, you want to use the web.xml taglib
    element to define an alias for the TLD filename. You
    would want to do so even if you weren't trying
    to accommodate Tomcat 4.0, which only reads listener
    declarations from TLD files that are declared that way.
  </description>

  <!-- Declare a validator to do translation-time checking
       of custom tag syntax. -->
  <validator>
    <validator-class>moreservlets.SAXValidator</validator-class>
  </validator>

  <!-- Register the listener that records the counts. -->
  <listener>
    <listener-class>
      moreservlets.listeners.ActiveSessionCounter
    </listener-class>
  </listener>

  <!-- Define a tag that prints out the session count. -->
  <tag>
    <name>sessionCount</name>
    <tag-class>
      moreservlets.tags.SessionCountTag
    </tag-class>
    <body-content>empty</body-content>
    <description>The number of active sessions.</description>
  </tag>
```

Listing11.14 *session-count-taglib-0.9-beta.tld* (Updated) *(continued)*

```
  <!-- Define a tag that prints out an optional
       session-count warning. -->
  <tag>
    <name>sessionCountWarning</name>
    <tag-class>
      moreservlets.tags.SessionCountWarningTag
    </tag-class>
    <body-content>empty</body-content>
    <description>
      If the number of sessions exceeds the limit,
      this prints a warning. Otherwise, it does nothing.
    </description>
  </tag>
</taglib>
```

Listing11.15 Validator output (for *index.jsp*—Listing 11.8)

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
          version="1.2"
          xmlns:counts="/session-count-taglib.tld">
  <jsp:text>
    <!DOCTYPE...
  </jsp:text>
  <jsp:text>
    Number...
  </jsp:text>
  <counts:sessionCount>
  </counts:sessionCount>
  <jsp:text>
  </jsp:text>
  <counts:sessionCountWarning>
  </counts:sessionCountWarning>
  <jsp:text>
    </BODY>...
  </jsp:text>
</jsp:root>
```

Example: Enforcing Tag Nesting Order

The previous subsection showed how to make a simple validator that uses SAX to discover the various tags in the page. Now let's *do* something with those tags. Listings 11.16 and 11.17 show two simple tags: `OuterTag` and `InnerTag`. `OuterTag` should not contain other `OuterTag` instances, and `InnerTag` should only appear within `OuterTag`. Developing a validator to enforce these restrictions involves the following steps.

1. **Creation of a subclass of `TagLibraryValidator`.** Listing 11.18 shows a class called `NestingValidator` that extends `TagLibraryValidator`.
2. **Overriding of the `validate` method.** I take the third argument to `validate` (the `PageData` object), call `getInputStream`, and pass that to the SAX `InputSource` constructor. I then tell SAX to parse the JSP document using that `InputSource` and a handler called `NestingHandler` (Listing 11.19). The `NestingHandler` class throws an exception in two situations: if it finds the outer tag when an existing outer tag instance is open and if it finds the inner tag when an existing outer tag instance is not open. The main validator returns `null` if the handler throws no exceptions. If the handler throws an exception, a 1-element `ValidationMessage` array is returned that contains a `ValidationMessage` describing the error.
3. **Declaration of the validator in the TLD file.** Listing 11.20 shows a TLD file that gives tag names to the two tag handlers and declares the validator.
4. **Try JSP pages that use the tag library.** Listings 11.21 and 11.22 show two JSP pages that correctly follow the rules that the outer tag cannot be nested and that the inner tag must appear directly or indirectly within the outer tag. Figures 11–4 and 11–5 show the results—the validator does not affect the output in any way. Listing 11.23 shows a JSP page that incorrectly attempts to use the inner tag when it is not nested within the outer tag. Figures 11–6 and 11–7 show the results in Tomcat 4.0 and ServletExec 4.1, respectively—the normal output is blocked since the JSP page was not successfully translated into a servlet. Listing 11.24 shows a JSP page that incorrectly attempts to nest the outer tag. Figure 11–8 shows the result in Tomcat 4.0.

Listing11.16 *OuterTag.java*

```
package moreservlets.tags;

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

/** Prints out a simple message. A TagLibraryValidator will
 *  enforce a nesting order for tags associated with this class.
 */

public class OuterTag extends TagSupport {
  public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("OuterTag");
    } catch(IOException ioe) {
      System.out.println("Error printing OuterTag.");
    }
    return(EVAL_BODY_INCLUDE);
  }
}
```

Listing11.17 *InnerTag.java*

```
package moreservlets.tags;

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

/** Prints out a simple message. A TagLibraryValidator will
 *  enforce a nesting order for tags associated with this class.
 */

public class InnerTag extends TagSupport {
  public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("InnerTag");
    } catch(IOException ioe) {
      System.out.println("Error printing InnerTag.");
    }
    return(EVAL_BODY_INCLUDE);
  }
}
```

Listing 11.18 *NestingValidator.java*

```
package moreservlets;

import javax.servlet.jsp.tagext.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/** A validator that verifies that tags follow
 *  proper nesting order.
 */

public class NestingValidator extends TagLibraryValidator {

  public ValidationMessage[] validate(String prefix,
                                      String uri,
                                      PageData page) {
    String jaxpPropertyName =
      "javax.xml.parsers.SAXParserFactory";
    // Pass the parser factory in on the command line with
    // -D to override the use of the Apache parser.
    if (System.getProperty(jaxpPropertyName) == null) {
      String apacheXercesPropertyValue =
        "org.apache.xerces.jaxp.SAXParserFactoryImpl";
      System.setProperty(jaxpPropertyName,
                         apacheXercesPropertyValue);
    }
    DefaultHandler handler = new NestingHandler();
    SAXParserFactory factory = SAXParserFactory.newInstance();
    try {
      SAXParser parser = factory.newSAXParser();
      InputSource source =
        new InputSource(page.getInputStream());
      parser.parse(source, handler);
      return(null);
    } catch(Exception e) {
      String errorMessage = e.getMessage();
      // The first argument to the ValidationMessage
      // constructor can be a tag ID. Since tag IDs
      // are not universally supported, use null for
      // portability. The important part is the second
      // argument: the error message.
      ValidationMessage[] messages =
        { new ValidationMessage(null, errorMessage) };
      return(messages);
    }
  }
}
```

Listing11.19 *NestingHandler.java*

```
package moreservlets;

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.StringTokenizer;

/** A SAX handler that returns an exception if either of
 *  the following two situations occurs:
 *  <UL>
 *    <LI>The designated outer tag is directly or indirectly
 *        nested within the outer tag (i.e., itself).
 *    <LI>The designated inner tag is <I>not</I> directly
 *        or indirectly nested within the outer tag.
 *  </UL>
 */

public class NestingHandler extends DefaultHandler {
  private String outerTagName = "outerTag";
  private String innerTagName = "innerTag";
  private boolean inOuterTag = false;

  public void startElement(String namespaceUri,
                           String localName,
                           String qualifiedName,
                           Attributes attributes)
      throws SAXException {
    String tagName = mainTagName(qualifiedName);
    if (tagName.equals(outerTagName)) {
      if (inOuterTag) {
        throw new SAXException("\nCannot nest " + outerTagName);
      }
      inOuterTag = true;
    } else if (tagName.equals(innerTagName) && !inOuterTag) {
      throw new SAXException("\n" + innerTagName +
                             " can only appear within " +
                             outerTagName);
    }
  }

  public void endElement(String namespaceUri,
                         String localName,
                         String qualifiedName)
      throws SAXException {
    String tagName = mainTagName(qualifiedName);
    if (tagName.equals(outerTagName)) {
      inOuterTag = false;
    }
  }
```

Listing11.19 *NestingHandler.java (continued)*

```
  private String mainTagName(String qualifiedName) {
    StringTokenizer tok =
      new StringTokenizer(qualifiedName, ":");
    tok.nextToken();
    return(tok.nextToken());
  }
}
```

Listing11.20 *nested-tag-taglib.tld*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>nested-tags</short-name>
  <description>
    A tag library that has two tags: outerTag and innerTag.
    A TagLibraryValidator will enforce the following
    nesting rules:
     1) innerTag can only appear inside outerTag. It can
        be nested, however.
     2) outerTag cannot be nested within other outerTag
        instances.
  </description>

  <!-- Declare a validator to do translation-time checking
       of custom tag syntax. -->
  <validator>
    <validator-class>
      moreservlets.NestingValidator
    </validator-class>
  </validator>

  <!-- Define the outerTag tag. -->
  <tag>
    <name>outerTag</name>
    <tag-class>
      moreservlets.tags.OuterTag
    </tag-class>
    <body-content>JSP</body-content>
```

Listing11.20 *nested-tag-taglib.tld (continued)*

```
    <description>
      A simple tag: cannot be nested within other outerTag
      instances.
    </description>
  </tag>

  <!-- Define the innerTag tag. -->
  <tag>
    <name>innerTag</name>
    <tag-class>
      moreservlets.tags.InnerTag
    </tag-class>
    <body-content>JSP</body-content>
    <description>
      A simple tag: can only appear within outerTag.
    </description>
  </tag>
</taglib>
```

Listing11.21 *nesting-test1.jsp* (Proper tag nesting)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Nested Tags: Test 1</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Nested Tags: Test 1
</TABLE>

<%@ taglib uri="/WEB-INF/nested-tag-taglib.tld" prefix="test" %>
```

Listing11.21 *nesting-test1.jsp* (Proper tag nesting) *(continued)*

```
<PRE>
<test:outerTag>
  <test:innerTag/>
  <test:innerTag/>
  <test:innerTag/>
</test:outerTag>
<test:outerTag/>
</PRE>

</BODY>
</HTML>
```

Listing11.22 *nesting-test2.jsp* (Proper tag nesting)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Nested Tags: Test 2</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Nested Tags: Test 2
</TABLE>

<%@ taglib uri="/WEB-INF/nested-tag-taglib.tld" prefix="test" %>

<PRE>
<test:outerTag>
  <test:innerTag>
    <test:innerTag/>
  </test:innerTag>
  <test:innerTag>
    <test:innerTag>
      <test:innerTag/>
    </test:innerTag>
   </test:innerTag>
</test:outerTag>
<test:outerTag/>
</PRE>

</BODY>
</HTML>
```

Listing11.23 *nesting-test3.jsp* (Improper tag nesting)

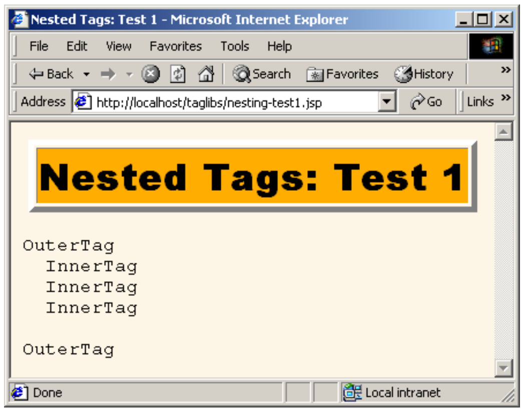
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Nested Tags: Test 3</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Nested Tags: Test 3
</TABLE>

<%@ taglib uri="/WEB-INF/nested-tag-taglib.tld" prefix="test" %>

<PRE>
<test:innerTag/>
</PRE>

</BODY>
</HTML>
```

Listing11.24 *nesting-test4.jsp* (Improper tag nesting)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Nested Tags: Test 4</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Nested Tags: Test 4
</TABLE>

<%@ taglib uri="/WEB-INF/nested-tag-taglib.tld" prefix="test" %>
```

Listing11.24 *nesting-test4.jsp* (Improper tag nesting) *(continued)*

```
<PRE>
<test:outerTag>
  <test:outerTag/>
</test:outerTag>
</PRE>

</BODY>
</HTML>
```

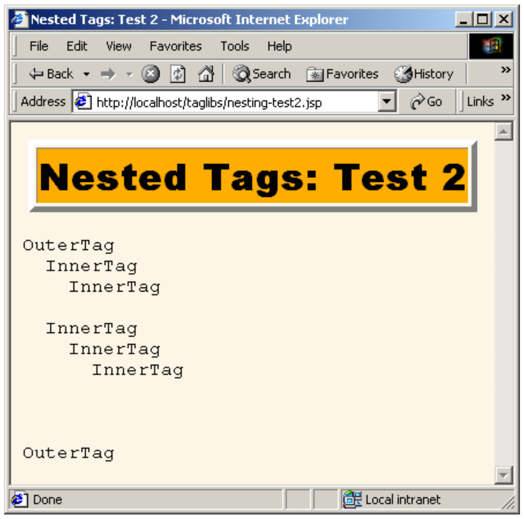


Nested Tags: Test 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/taglibs/nesting-test1.jsp Go Links
Done
Local intranet

Nested Tags: Test 1

```
OuterTag
  InnerTag
  InnerTag
  InnerTag

OuterTag
```

Figure 11–4 Result of *nesting-test1.jsp*. Tags are nested properly, so output is normal.

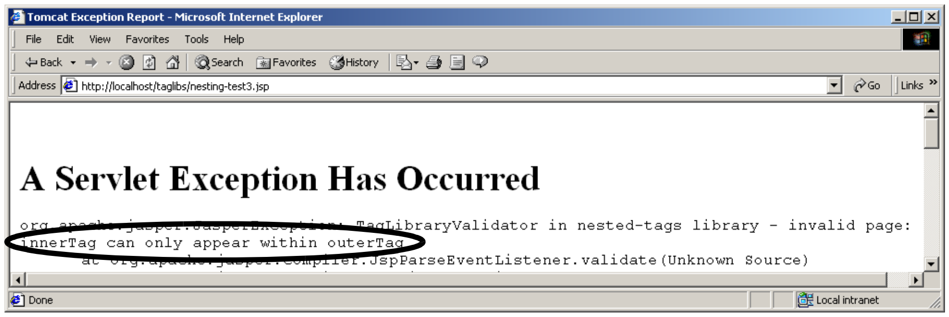


Nested Tags: Test 2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/taglibs/nesting-test2.jsp Go Links
Done
Local intranet

Nested Tags: Test 2

```
OuterTag
  InnerTag
    InnerTag

  InnerTag
    InnerTag
      InnerTag


OuterTag
```

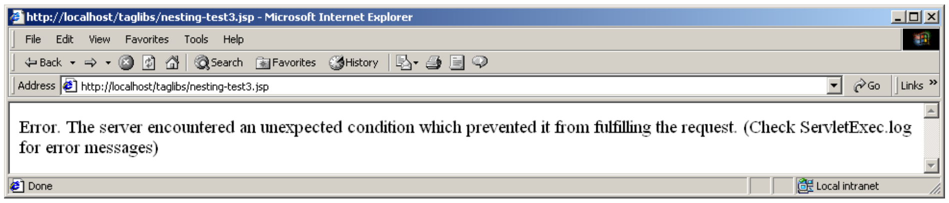
Figure 11–5 Result of *nesting-test2.jsp*. Tags are nested properly, so output is normal.


Tomcat Exception Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/taglibs/nesting-test3.jsp
Go Links
Done
Local intranet

A Servlet Exception Has Occurred

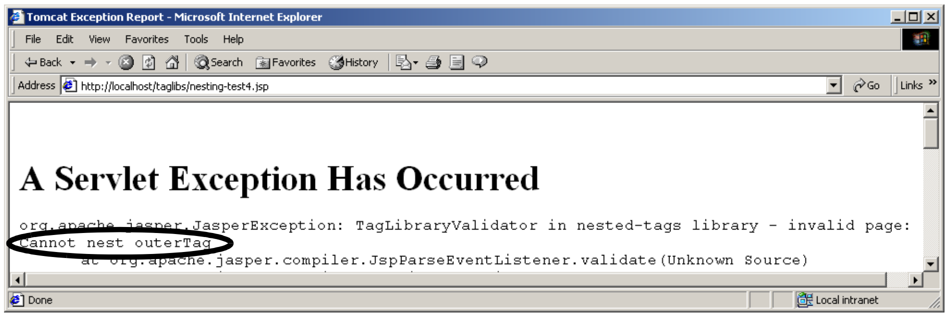
```
org.apache.jasper.JasperException: TagLibraryValidator in nested-tags library - invalid page:
innerTag can only appear within outerTag
        at org.apache.jasper.compiler.JspParseEventListener.validate(Unknown Source)
```

Figure 11–6 Result of *nesting-test3.jsp*. Tags are nested improperly, so normal output is prevented—at page translation time, the normal servlet is replaced by one that gives an error message. This output is from Tomcat 4.0.


http://localhost/taglibs/nesting-test3.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/taglibs/nesting-test3.jsp
Go Links
Done
Local intranet

Error. The server encountered an unexpected condition which prevented it from fulfilling the request. (Check ServletExec.log for error messages)

Figure 11–7 The result of *nesting-test3.jsp* when accessed on ServletExec 4.1.


Tomcat Exception Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/taglibs/nesting-test4.jsp
Go Links
Done
Local intranet

A Servlet Exception Has Occurred

```
org.apache.jasper.JasperException: TagLibraryValidator in nested-tags library - invalid page:
Cannot nest outerTag
        at org.apache.jasper.compiler.JspParseEventListener.validate(Unknown Source)
```

Figure 11–8 Result of *nesting-test4.jsp*. Tags are nested improperly, so normal output is prevented—at page translation time, the normal servlet is replaced by one that gives an error message. This output is from Tomcat 4.0.

11.4 Aside: Parsing XML with SAX 2.0

The `validate` method of `TagLibraryValidator` gives you a `PageData` object from which you can obtain an `InputStream` that is associated with the XML version of the JSP page. You are unlikely to want to read directly from that stream to try to validate proper usage of your custom tags: the input streams and readers in the `java.io` package are too low level to be effective for this usage. Instead, you will probably want to use an XML parser to do the low-level work for you. The validators of Section 11.3 used the SAX 2.0 API, so I'll summarize its use here. More detail on SAX can be found at *http://www.saxproject.org/*. For information on DOM and other XML-related APIs, see *http://www.w3.org/XML/*, one of the many XML and Java texts such as *Java & XML 2nd Edition* (O'Reilly 2001), or the SAX, DOM, and XSLT summary in Chapter 23 of *Core Web Programming 2nd Edition* (Prentice Hall and Sun Microsystems Press 2001).

SAX processing is a lot like writing custom tag handlers: you write methods to handle the start tag, end tag, and tag body. The major difference is that SAX handlers are not associated with specific tags—the same handler fires for all tags. So, with SAX you have to repeatedly check which tag you are working with. Of course, this capability is exactly what makes SAX so useful for checking that different tags are interacting properly.

Installation and Setup

SAX is not a standard part of either Java 2 Standard Edition or the servlet and JSP APIs. So, your first step is to download the appropriate classes and configure them for use in your programs. Here is a summary of what is required.

1. **Download a SAX-compliant parser.** The parser provides the Java classes that follow the SAX 2 API as specified by the WWW Consortium. You can obtain a list of XML parsers in Java at *http://www.xml.com/pub/rg/Java_Parsers*. I use the Apache Xerces-J parser in this book. See *http://xml.apache.org/xerces-j/*. This parser comes with the complete SAX API in Javadoc format.
2. **Download the Java API for XML Processing (JAXP).** This API provides a small layer on top of SAX that lets you plug in different vendor's parsers without making any changes to your basic code. See *http://java.sun.com/xml/*.
3. **Tell your development environment and the server about the SAX classes.** In the case of Apache Xerces, the SAX classes are in *xerces_install_dir/ xerces.jar*. So, for example, to set up your development environment on Windows 98, you would do

Update from Marty: Java now comes with XML parsing already included.
For details on parsing XML in Java, see the XML section of:
• http://courses.coreservlets.com/Course-Materials/java5.html

```
set CLASSPATH=xerces_install_dir\xerces.jar;%CLASSPATH%
```

To tell the server about the SAX classes, you would either copy the JAR file to the Web application's `lib` directory, unpack the JAR file (using `jar -xvf`) into the server's `classes` directory, or put the JAR file in a shared location (if your server supports such a capability—see Section 4.4, "Recording Dependencies on Server Libraries").

4. **Set your `CLASSPATH` to include the JAXP classes.** These classes are in *jaxp_install_dir/jaxp.jar*. For example, to set up your development environment on Unix/Linux with the C shell, you would do

```
setenv CLASSPATH jaxp_install_dir/jaxp.jar:$CLASSPATH
```

To tell the server about the JAXP classes, see the preceding step.

5. **Bookmark the SAX API.** You can browse the official API at *http://www.saxproject.org/apidoc/index.html*, but the API that comes with Apache Xerces is easier to use because it is on your local system and is integrated with the DOM and JAXP APIs. More information on SAX can be found at *http://www.saxproject.org/*.

Parsing

With SAX processing, there are two high-level tasks: creating a content handler and invoking the parser with the designated content handler. The following list summarizes the detailed steps needed to accomplish these tasks.

1. **Tell the system which parser you want to use.** This can be done in a number of ways: through the `javax.xml.parsers.SAXParserFactory` system property, through *jre_dir/lib/jaxp.properties*, through the J2EE Services API and the class specified in *META-INF/services/javax.xml.parsers.SAXParserFactory*, or with a system-dependent default parser. The system property is the easiest method. For example, the following code permits deployers to specify the parser in the server startup script with the `-D` option to `java`, and uses the Apache Xerces parser otherwise.

```
String jaxpPropertyName =
  "javax.xml.parsers.SAXParserFactory";
if (System.getProperty(jaxpPropertyName) == null) {
  String apacheXercesPropertyValue =
    "org.apache.xerces.jaxp.SAXParserFactoryImpl";
  System.setProperty(jaxpPropertyName,
                     apacheXercesPropertyValue);
}
...
```

2. **Create a parser instance.** First make an instance of a parser factory, then use that to create a parser object.

   ```
   SAXParserFactory factory = SAXParserFactory.newInstance();
   SAXParser parser = factory.newSAXParser();
   ```

   Note that you can use the `setNamespaceAware` and `set-Validating` methods on the `SAXParserFactory` to make the parser namespace aware and validating, respectively.
3. **Create a content handler to respond to parsing events.** This handler is typically a subclass of `DefaultHandler`. You override any or all of the following placeholders.
   - **`startDocument`, `endDocument`**
     Use these methods to respond to the start and end of the document; they take no arguments.
   - **`startElement`, `endElement`**
     Use these methods to respond to the start and end tags of an element. The `startElement` method takes four arguments: the namespace URI (a `String`; empty if no namespace), the namespace or prefix (a `String`; empty if no namespace), the fully qualified element name (a `String`; i.e., `"prefix:mainName"` if there is a namespace; `"mainName"` otherwise), and an `Attributes` object representing the attributes of the start tag. The `endElement` method takes the same arguments except for the attributes (since end tags are not permitted attributes).
   - **`characters`, `ignoreableWhitespace`**
     Use these methods to respond to the tag body. They take three arguments: a `char` array, a start index, and an end index. A common approach is to turn the relevant part of the character array into a `String` by passing all three arguments to the `String` constructor. Non-white-space data is always reported to the `characters` method. White space is always reported to the `ignoreable-Whitespace` method when the parser is run in validating mode but can be reported to either method otherwise.
4. **Invoke the parser with the designated content handler.** You invoke the parser by calling the `parse` method, supplying an input stream, URI (represented as a string), or `org.xml.sax.Input-Source` along with the content handler. Note that `InputSource` has a simple constructor that accepts an `InputStream`. Use this constructor to turn the `InputStream` of the `TagLibraryValidator` `validate` method into an `InputSource`.

   ```
   parser.parse(new InputSource(validatorStream), handler);
   ```

   The content handler does the rest.

11.5 Handling Exceptions with the TryCatchFinally Interface

In JSP 1.1, you can trap exceptions that occur in each of your tag handling methods (`doStartTag`, `doEndTag`, etc.). But, what happens if an exception occurs during the processing of the body of the tag? What if you want to respond to exceptions in `doStartTag`, `doEndTag`, and `doAfterBody` the same way and don't want to repeat your code?

JSP 1.2 answers these questions by providing a new interface called `TryCatchFinally` with two methods:

- **`doCatch(Throwable t)`**
- **`doFinally()`**

If your tag implements this interface and an exception occurs during *any* of the tag life-cycle methods *or* during the processing of the tag body, the system calls the `doCatch` method. Regardless of whether an exception occurs, the system calls the `doFinally` method when done executing the tag. Note, however, that this exception-handling behavior applies only to the tag life-cycle methods and the processing of the tag body. It does *not* apply to the `set`*`Xxx`* attribute-setting methods.

Core Warning

The `TryCatchFinally` interface does not apply to the methods that set the tag attributes.

To illustrate this new exception-handling behavior, Listing 11.25 shows a tag that implements `TryCatchFinally`. The tag doesn't actually output anything: it just prints a message to standard output when `doStartTag`, `doEndTag`, `doCatch`, and `doFinally` are called. Listing 11.26 shows the TLD file that declares the tag; no new syntax or entries are needed.

Listing 11.27 shows a JSP page that uses the new tag wrapped around a body that sometimes throws an exception (see Listing 11.28). Figure 11–9 shows the result when none of the tags throws an exception. Listing 11.29 shows the corresponding output—`doStartTag`, `doEndTag`, and `doFinally` are invoked but `doCatch` is not. Figure 11–10 shows the result when the second invocation of the tag throws an exception—the tag body generates no output but the page processing continues normally. Listing 11.30 shows the corresponding output—`doStartTag`, `doEndTag`, and `doFinally` are invoked each time and `doCatch` is invoked only in the case when the tag body threw the exception.

Listing11.25 *CatchTag.java*

```
package moreservlets.tags;

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

/** Tag that traces the life cycle of tags that
 *  implement the TryCatchFinally interface.
 */

public class CatchTag extends TagSupport
                      implements TryCatchFinally {
  public int doStartTag() {
    System.out.println("CatchTag: start");
    return(EVAL_BODY_INCLUDE);
  }

  public int doEndTag() {
    System.out.println("CatchTag: end");
    return(EVAL_PAGE);
  }

  public void doCatch(Throwable throwable) {
    System.out.println("CatchTag: doCatch: " +
                       throwable.getMessage());
  }

  public void doFinally() {
    System.out.print("CatchTag: doFinally\n");
  }
}
```

Listing11.26 *catch-tag-taglib.tld*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>catch-tags</short-name>
  <description>
    A tag library that uses a simple tag to illustrate
    the behavior of the TryCatchFinally interface.

    From More Servlets and JavaServer Pages,
    http://www.moreservlets.com/.
  </description>

  <!-- Define the catchTag tag. -->
  <tag>
    <name>catchTag</name>
    <tag-class>
      moreservlets.tags.CatchTag
    </tag-class>
    <body-content>JSP</body-content>
    <description>
      Implements the TryCatchFinally interface and
      prints simple tag life-cycle information to the
      standard output.
    </description>
  </tag>
</taglib>
```

Listing11.27 *catchTest.jsp*

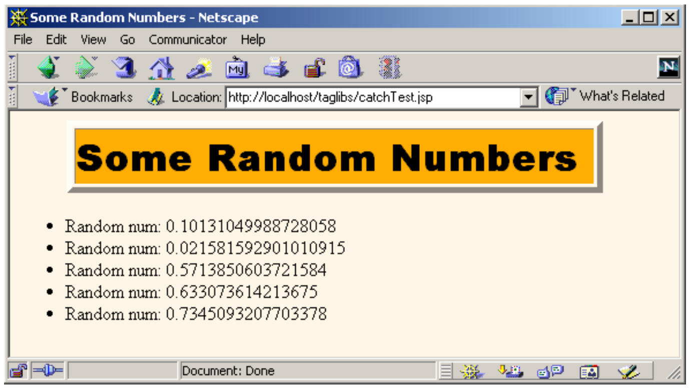
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Some Random Numbers</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>
```

Listing 11.27 *catchTest.jsp (continued)*

```
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Some Random Numbers
</TABLE>

<%@ taglib uri="/WEB-INF/catch-tag-taglib.tld" prefix="msajsp" %>
<UL>
  <LI><msajsp:catchTag>
      <%= moreservlets.Utils.dangerousMethod() %>
      </msajsp:catchTag>
  <LI><msajsp:catchTag>
      <%= moreservlets.Utils.dangerousMethod() %>
      </msajsp:catchTag>
  <LI><msajsp:catchTag>
      <%= moreservlets.Utils.dangerousMethod() %>
      </msajsp:catchTag>
  <LI><msajsp:catchTag>
      <%= moreservlets.Utils.dangerousMethod() %>
      </msajsp:catchTag>
  <LI><msajsp:catchTag>
      <%= moreservlets.Utils.dangerousMethod() %>
      </msajsp:catchTag>
</UL>
</BODY>
</HTML>
```

Listing 11.28 *Utils.java*

```
package moreservlets;

public class Utils {
  public static String dangerousMethod() throws Exception {
    double num = Math.random();
    if (num < 0.8) {
      return("Random num: " + num);
    } else {
      throw(new Exception("No intelligent life here."));
    }
  }
}
```


Some Random Numbers - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/taglibs/catchTest.jsp
What's Related
Document: Done

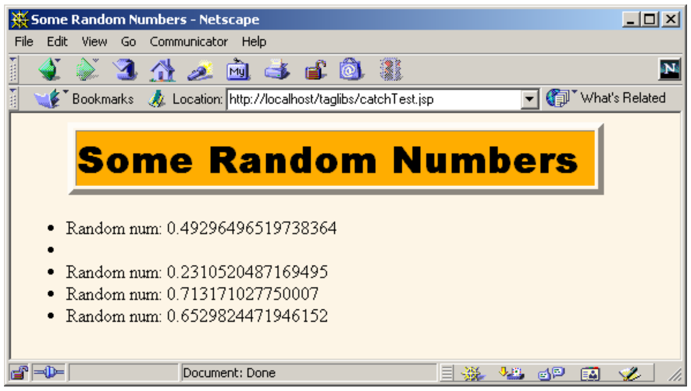
Some Random Numbers

- Random num: 0.10131049988728058
- Random num: 0.021581592901010915
- Random num: 0.5713850603721584
- Random num: 0.633073614213675
- Random num: 0.7345093207703378

Figure 11–9 One possible result of *catchTest.jsp*.

Listing11.29 Output of *catchTest.jsp* corresponding to Figure 11–9

```
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
```


Some Random Numbers - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/taglibs/catchTest.jsp
What's Related
Document: Done

Some Random Numbers

- Random num: 0.492964965197338364
-
- Random num: 0.2310520487169495
- Random num: 0.713171027750007
- Random num: 0.6529824471946152

Figure 11–10 Another possible result of *catchTest.jsp*.

Listing11.30 Output of *catchTest.jsp* corresponding to Figure 11–10

```
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: doCatch: No intelligent life here.
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
CatchTag: start
CatchTag: end
CatchTag: doFinally
```

11.6 New Names for Return Values

In JSP 1.1, the `EVAL_BODY_TAG` constant was used in two totally different situations. First, it was returned from the `doStartTag` method of `TagSupport` or `BodyTagSupport` to indicate that the tag body should be made available in `doAfterBody`. The default `doStartTag` method of `BodyTagSupport` returned this value. Second, it was returned from the doAfterBody method of TagSupport or BodyTagSupport to indicate that the body should be reevaluated.

This dual use of EVAL_BODY_TAG was confusing because it indicated a single evaluation in the first case and multiple evaluations (as in iterative tags) in the second case. To better differentiate the two situations, in JSP 1.2 you return EVAL_BODY_BUFFERED from doStartTag to indicate that the tag body should be made available in doAfterBody. You return EVAL_BODY_AGAIN from doAfterBody when you want to reevaluate the body.

11.7 Looping Without Generating BodyContent

In JSP 1.1, the only way to make an iterative tag is to return EVAL_BODY_TAG from the doAfterBody method, thus instructing the system to reevaluate the tag body. If the doStartTag returns EVAL_BODY_TAG, the system copies the tag body into a BodyContent object and invokes doAfterBody. The BodyTagSupport class overrides doStartTag to automatically return EVAL_BODY_TAG, so you only need to write doAfterBody when using BodyTagSupport. The problem with this approach is that doAfterBody must manually generate the tag body's output by extracting it from the BodyContent object in which the system wraps the body. This is fine if you actually want to manipulate the body. But, if you merely want to iterate, you must use the somewhat clumsy BodyContent object and the system must repeatedly copy the tag body.

JSP 1.2 provides a solution that is both simpler and more efficient. If a tag implements IterationTag, doStartTag can return EVAL_BODY_INCLUDE to instruct the system to output the tag body without first copying it. Then, if the doAfterBody method returns EVAL_BODY_AGAIN, the process is repeated. Since the TagSupport class now implements IterationTag, simple iterative tags in JSP 1.2 need not use BodyTagSupport at all.

The following two subsections present simple examples that contrast the JSP 1.1 and JSP 1.2 approaches.

JSP 1.1 Loop Tag

Listing 11.31 shows a simple iterative tag that uses the JSP 1.1 approach. It extends the BodyTagSupport class, overrides doAfterBody, extracts the tag body from the BodyContent object, and outputs it to the client. The doAfterBody method returns EVAL_BODY_TAG to indicate that looping should continue; it returns SKIP_BODY when done.

Listing 11.32 shows a TLD file that declares the tag; Listing 11.33 shows a JSP page that uses the tag, supplying a request-time form parameter (`repeats`) to dictate how many repetitions should be executed. Figure 11–11 shows the result.

Listing11.31 *RepeatTag1.java*

```
package moreservlets.tags;

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

/** A tag that repeats the body the specified
 *  number of times. JSP 1.1 version.
 */

public class RepeatTag1 extends BodyTagSupport {
  private int reps;

  public void setReps(String repeats) {
    try {
      reps = Integer.parseInt(repeats);
    } catch(NumberFormatException nfe) {
      reps = 1;
    }
  }

  public int doAfterBody() {
    if (reps-- >= 1) {
      BodyContent body = getBodyContent();
      try {
        JspWriter out = body.getEnclosingWriter();
        out.println(body.getString());
        body.clearBody(); // Clear for next evaluation
      } catch(IOException ioe) {
        System.out.println("Error in RepeatTag1: " + ioe);
      }
      return(EVAL_BODY_TAG);
    } else {
      return(SKIP_BODY);
    }
  }
}
```

Listing11.32 *repeat-taglib.tld*

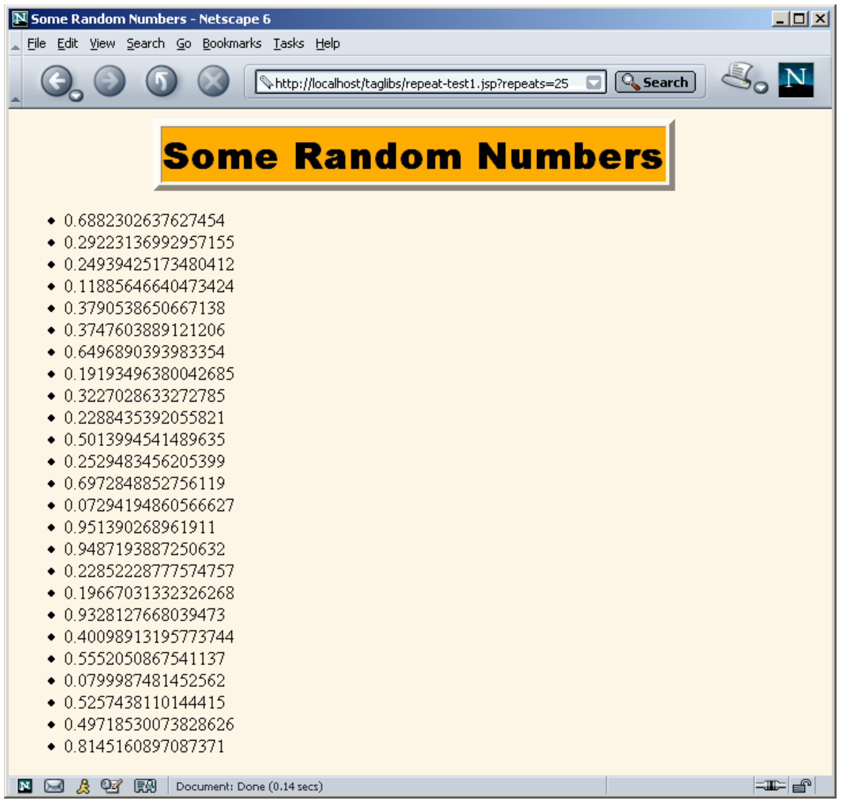
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>repeat-tags</short-name>
  <description>
    A tag library that has two tags: repeatTag1 and repeatTag2.
    These are JSP 1.1 and JSP 1.2 versions of simple
    looping tags.
  </description>

  <!-- An iterative tag. JSP 1.1 version. -->
  <tag>
    <name>repeat1</name>
    <tag-class>moreservlets.tags.RepeatTag1</tag-class>
    <body-content>JSP</body-content>
    <description>
      Repeats body the specified number of times.
    </description>
    <attribute>
      <name>reps</name>
      <required>true</required>
      <!-- rtexprvalue indicates whether attribute
           can be a JSP expression. -->
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>

  <!-- An iterative tag. JSP 1.2 version. -->
  <tag>
    <name>repeat2</name>
    <tag-class>moreservlets.tags.RepeatTag2</tag-class>
    <body-content>JSP</body-content>
    <description>
      Repeats body the specified number of times.
    </description>
```

Listing11.32 *repeat-taglib.tld (continued)*

```
    <attribute>
      <name>reps</name>
      <required>true</required>
      <!-- rtexprvalue indicates whether attribute
           can be a JSP expression. -->
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

Listing11.33 *repeat-test1.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Some Random Numbers</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Some Random Numbers
</TABLE>
<P>
<%@ taglib uri="/WEB-INF/repeat-taglib.tld" prefix="msajsp" %>
<UL>
<msajsp:repeat1 reps='<%= request.getParameter("repeats") %>'>
  <LI><%= Math.random() %>
</msajsp:repeat1>
</UL>
</BODY>
</HTML>
```


Some Random Numbers - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/taglibs/repeat-test1.jsp?repeats=25
Search
Document: Done (0.14 secs)

Some Random Numbers

- 0.6882302637627454
- 0.29223136992957155
- 0.24939425173480412
- 0.11885646640473424
- 0.3790538650667138
- 0.3747603889121206
- 0.6496890393983354
- 0.19193496380042685
- 0.3227028633272785
- 0.2288435392055821
- 0.5013994541489635
- 0.2529483456205399
- 0.6972848852756119
- 0.07294194860566627
- 0.951390268961911
- 0.9487193887250632
- 0.22852228777574757
- 0.19667031332326268
- 0.9328127668039473
- 0.40098913195773744
- 0.5552050867541137
- 0.0799987481452562
- 0.5257438110144415
- 0.49718530073828626
- 0.8145160897087371

Figure 11–11 Result of *repeat-test1.jsp* when the user supplies a `repeats` parameter of 25.

JSP 1.2 Loop Tag

Listing 11.34 shows a simple iterative tag that uses the JSP 1.2 approach. It extends the `TagSupport` class, overrides `doStartTag` to return `EVAL_BODY_INCLUDE`, and overrides `doAfterBody` to either return `EVAL_BODY_AGAIN` (keep looping) or `SKIP_BODY` (done). No `BodyContent`: no need for the programmer to extract it, and no need for the system to waste time and memory copying the tag body into it.

Listing 11.35 shows a JSP page that uses the tag, supplying a request-time form parameter (`repeats`) to dictate how many repetitions should be executed. Figure 11–12 shows the result.

Listing11.34 *RepeatTag2.java*

```
package moreservlets.tags;

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

/** A tag that repeats the body the specified
 *  number of times. JSP 1.2 version.
 */

public class RepeatTag2 extends TagSupport {
  private int reps;

  public void setReps(String repeats) {
    try {
      reps = Integer.parseInt(repeats);
    } catch(NumberFormatException nfe) {
      reps = 1;
    }
  }

  public int doStartTag() {
    if (reps >= 1) {
      return(EVAL_BODY_INCLUDE);
    } else {
      return(SKIP_BODY);
    }
  }

  public int doAfterBody() {
    if (reps-- > 1) {
      return(EVAL_BODY_AGAIN);
    } else {
      return(SKIP_BODY);
    }
  }
}
```

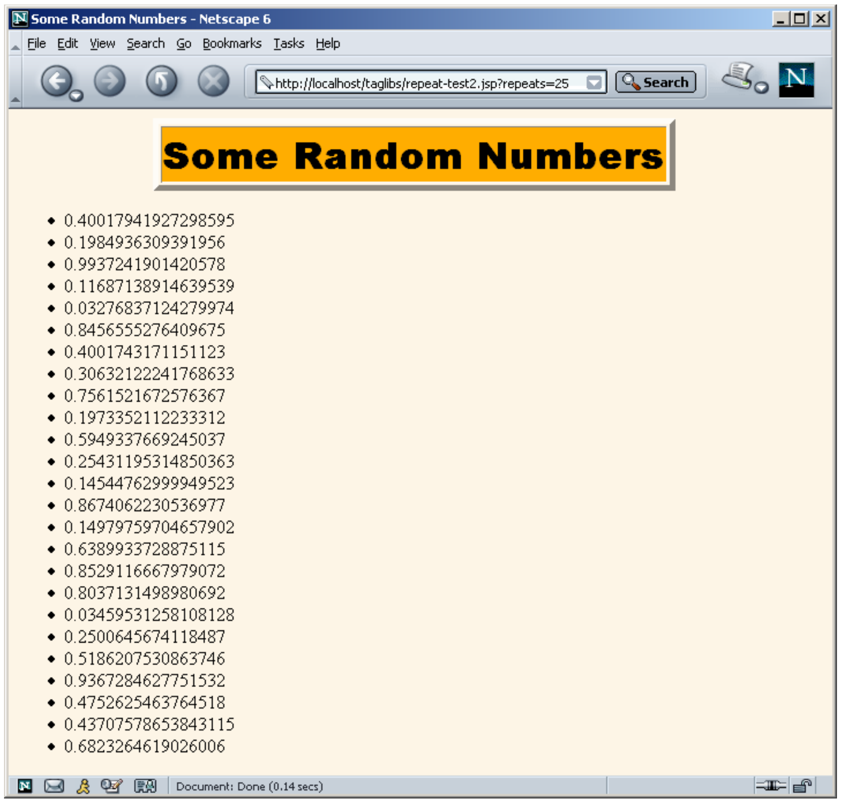
Listing11.35 *repeat-test2.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Some Random Numbers</TITLE>
<LINK REL=STYLESHEET
      HREF="styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Some Random Numbers
</TABLE>
<P>
<%@ taglib uri="/WEB-INF/repeat-taglib.tld" prefix="msajsp" %>
<UL>
<msajsp:repeat2 reps='<%= request.getParameter("repeats") %>'>
  <LI><%= Math.random() %>
</msajsp:repeat2>
</UL>
</BODY>
</HTML>
```

11.8 Introducing Scripting Variables in the TLD File

In JSP 1.1, you override the getVariableInfo method of a TagExtraInfo class to declare scripting variables that your tag will introduce. The getVariableInfo method returns an array of VariableInfo objects. The VariableInfo constructor, in turn, takes four arguments:

- The variable name (a String).
- The variable's type (a String representing a fully qualified class name or a class listed in the current page's import statements).
- A boolean indicating whether the variable should be declared (almost always true; this option supports future scripting in other languages).
- An int indicating the variable's scope (NESTED—available between the tag's start and end tags, AT_BEGIN—available anytime after the tag's start tag, or AT_END—available anytime after the tag's end tag).


Some Random Numbers - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://localhost/taglibs/repeat-test2.jsp?repeats=25
Search
Document: Done (0.14 secs)

Some Random Numbers

- 0.40017941927298595
- 0.1984936309391956
- 0.9937241901420578
- 0.11687138914639539
- 0.03276837124279974
- 0.8456555276409675
- 0.4001743171151123
- 0.30632122241768633
- 0.7561521672576367
- 0.1973352112233312
- 0.5949337669245037
- 0.25431195314850363
- 0.14544762999949523
- 0.8674062230536977
- 0.14979759704657902
- 0.6389933728875115
- 0.8529116667979072
- 0.8037131498980692
- 0.03459531258108128
- 0.2500645674118487
- 0.5186207530863746
- 0.9367284627751532
- 0.4752625463764518
- 0.43707578653843115
- 0.6823264619026006

Figure 11–12 Result of *repeat-test2.jsp* when the user supplies a `repeats` parameter of 25.

Rather than burying this information within the `TagExtraInfo` class, in JSP 1.2 you can declare it directly in the TLD file. To do this, use the `variable` subelement of `tag`. This new element has five possible subelements:

- `name-given`: the variable name.
- `name-from-attribute`: the name of an attribute whose translation-time value will give the variable's name. You must supply either `name-given` or `name-from-attribute`.
- `variable-class`: the variable's type. This element is optional; `java.lang.String` is the default.
- `declare`: whether the variable is declared. This element is optional; `true` is the default.
- `scope`: the variable's scope. This element is optional; `NESTED` is the default.
- `description`: brief documentation on the variable.

The `variable` element appears within `tag` after `description` but before `attribute`. For example, Listing 11.36 shows a TLD file that declares a simple nested `String` variable named `emailAddress`.

Listing11.36 *sample-taglib.tld* (Excerpt)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
 PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>some-name</short-name>
  <description>Tag library documentation.</description>

  <tag>
    <name>tag-name</name>
    <tag-class>somePackage.SomeTagClass</tag-class>
    <tei-class>somePackage.SomeTEIClass</tei-class>
    <body-content>JSP</body-content>
    <description>Tag documentation.</description>
    <variable>
      <name-given>emailAddress</name-given>
    </variable>
  </tag>
</taglib>
```