UNIT : 9

Reference :-Marty Hall, Larry Brown, "Core Servlets and JavaServer Pages Volume – 2", Pearson Education, 2nd ed.(2004)

Chapter :- 07 : TAG Libraries : The Basics

Dynamic code in JSP

- to generating dynamic content inside the JSP page.
  These options are as follows:
- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility
classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags

Tag Library Component

To use custom JSP tags, you need to define three separate components:

- The tag handler class that defines the tag's behavior
- The TLD file that maps the XML element names to the tag implementations
- The JSP file that uses the tag library

Tag Handler Class

- When defining a new tag, your first task is to define a Java class that tells the system what to do when it sees the tag.
- This class must implement theSimpleTag interface.
- In practice, you extend SimpleTagSupport, which implements the SimpleTag interface and supplies standard implementations for some of its methods.
- Both the SimpleTag interface and the SimpleTagSupport class reside in the javax.servlet.jsp.tagext package.

Tag Handler Class

- Every tag handler must have a no-arg constructor or its instantiation will fail.
- The code that does the actual work of the tag goes inside the doTag method.
- Usually, this code outputs content to the JSP page by invoking the print method of the JspWriter class.
- To obtain an instance of the JstWriter class you call getJsp-Context().getOut() inside the doTag method.
- The doTag method is called at request time.

Tag Handler Class

- A new instance of the tag handler class is created for every tag occurrence on the page.
- You place the compiled tag handler in the same location you would place a regular servlet, inside the WEB-INF/classes directory, keeping the package structure intact.
- For example, if your tag handler class belongs to the mytags package and its class name is MyTag, you would place the MyTag.class file inside the WEB-INF/classes/mytags/ directory.

Tag Handler Class

- A new instance of the tag handler class is created for every tag occurrence on the page.
- You place the compiled tag handler in the same location you would place a regular servlet, inside the WEB-INF/classes directory, keeping the package structure intact.
- For example, if your tag handler class belongs to the mytags package and its class name is MyTag, you would place the MyTag.class file inside the WEB-INF/classes/mytags/ directory.

Tag Library Descriptor File

- Identify this class to the server and to associate it with a particular XML tag name.
- This file contains some fixed information (e.g., XML Schema instance declaration), an arbitrary short name for your library, a short description, and a series of tag descriptions.

Tag Library Descriptor File

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
version="2.0">
<tlib-version>1.0</tlib-version>
<short-name>csajsp-taglib</short-name>
<tag>
<description>Example tag</description>
<name>example</name>
<tag-class>package.TagHandlerClass</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```

Tag Library Descriptor File

- tag element through the following subelements.
- description : This optional element allows the tag developer to document the purpose of the custom tag.
- name: This required element defines the name of the tag as it will be referred to by the JSP page (really tag suffix, as will be seen shortly).
- tag-class : This required element identifies the fully qualified name of the implementing tag handler class.
- body-content : This required element tells the container how to treat the content between the beginning and ending occurrence of the tag, if any. The value that appears here can be either empty, scriptless, tagdependent, or JSP.

Tag Library Descriptor File

The value of empty means that no content is allowed to appear in the body of the tag.

This would mean that the declared tag can only appear in the form:

<prefix:tag/>

or

<prefix:tag></prefix:tag>

(without any spaces between the opening and closing tags). Placing any content inside the tag body would generate a page translation error.

Tag Library Descriptor File

- The value of scriptless means that the tag body is allowed to have JSP content as long as it doesn't contain any scripting elements like <% ... %> or <%= ... %>. If present, the body of the tag would be processed just like any other JSP content.
- The value of tagdependent means that the tag is allowed to have any type of content as its body. However, this content is not processed at all and completely ignored.
- It is up to the developer of the tag handler to get access to that content and do something with it.
- For example, if you wanted to develop a tag that would allow the JSP page developer to execute an SQL statement, providing the SQL in the body of the tag, you would use tagdependent as the value of the body-content element.

Tag Library Descriptor File

- Finally, the value of JSP is provided for backward compatibility with the classic custom tag model. It is not a legal value when used with the SimpleTag API.
- *When using the SimpleTag API, it is illegal to include scripting elements in the body of the tag.*
- The TLD file must be placed inside the WEB-INF directory or any subdirectory thereof.

Tag Library Descriptor File

- Finally, the value of JSP is provided for backward compatibility with the classic custom tag model. It is not a legal value when used with the SimpleTag API.
- *When using the SimpleTag API, it is illegal to include scripting elements in the body of the tag.*
- The TLD file must be placed inside the WEB-INF directory or any subdirectory thereof.

Assigning attribute to files

- <prefix:name attribute1="value1" attribute2="value2"... />
- attributes allow us to pass information to the tag.

Tag Attributes : Tag Handler class

- Use of an attribute called attribute1 simply results in a call to a method called setAttribute1 in your class that extends SimpleTagSupport (or that otherwise implements the Simple-Tag interface).

```
public void setAttribute1(String value1) {
      doSomethingWith(value1);
}
```

- Attribute with the name of attributeName (lowercase a) corresponds to a method called setAttributeName (uppercase A).

Tag Attributes : Tag Handler class

```
private String message = "Default Message";
public void setMessage(String message) {
this.message = message;
}
```

- If the tag handler is accessed from other classes, it is a good idea to provide a getAttributeName method in addition to the setAttributeName method. Only setAttributeName is required, however.

Tag Attributes : Tag Library Descriptor

- Tag attributes must be declared inside the tag element by means of an attribute element.
- The attribute element has three nested elements that can appear between <attribute> and </attribute>.
- 1) name : This is a required element that defines the case-sensitive attribute name.

Tag Attributes : Tag Library Descriptor

- 2) required
- This is an optional element that stipulates whether the attribute must always be supplied, true, or is optional, false (default).
- If required is false and the JSP page omits the attribute, no call is made to the setAttributeName method, so be sure to give default values to the fields that the method sets if the attribute is not declared as required.
- Omitting a tag attribute, which is declared with the required element equal to true, results in an error at page translation time.

Tag Attributes : Tag Library Descriptor

- rtexprvalue
- This is an optional element that indicates whether the attribute value can be either a JSP scripting expression like <%= expression %> or JSP EL like ${bean.value} (true), or whether it must be a fixed string (false).
- The default value is false, so this element is usually omitted except when you want to allow attributes to have values determined at request time.

Tag Attributes : JSP File

- the JSP page has to declare the tag library using the taglib directive.
  <%@ taglib uri="..." prefix="..." %>
- the attribute names are case-sensitive and have to appear in the JSP page exactly as they were declared inside the TLD file.
- the value of an attribute has to be enclosed by either single or double quotes.
- For example: <some-prefix:tag1 attribute1="value" />

Including tag body in tag output

- Up to this point, all of the custom tags you have seen did not allow a body and thuswere always used as standalone tags of the following form:
- <prefix:tagname/>
- <prefix:tagname></prefix:tagname>
- The fact that these tags were not allowed to include a body was a direct result of supplying the element body-content with the value of empty

Tag bodies : Tag Handler Class

- To output the body content of the tag, inside the doTag method you need to acquire the JspFragment instance representing the body of the tag by calling the getJspBody method, then using its invoke method passing it null as its argument.
- getJspBody().invoke(null);
- JspWriter out = getJspContext().getOut();
- out.print("...");
- getJspBody().invoke(null);
- out.print("...");