JSP 2.0 Tag Files

Noel J. Bergman

DevTech®
Session Overview

It is one thing to preach that developers should keep scripting out of JSP pages, and use custom tags, but it is another thing in practice. Custom tag development has lagged need due to the complexity and learning curve in developing custom tags. JSP 2.0 introduces a new way to write custom tags. This session show you how to rapidly develop custom tags using the new Tag File mechanism.

Developers: this is your chance to have your cake and eat it, too. You get to use scripting and still deliver clean pages to page designers.

This session presumes knowledge of Java and JavaServer Pages.

PLEASE ASK QUESTIONS! 😊
The Problem

- We use JSP pages to add dynamic content into our presentation.

```html
<html>
<head><title>Date Demo</title></head>
<body>
  The date is: <%= new java.util.Date() %>
</body>
</html>
```

- Scripting elements are not suitable for non-Java programmers, nor well suited for page design tools.
The Solution

- Custom tags are the key to removing scripting elements from presentation pages.
- We use custom tags to add dynamic content to our presentation.

```html
<html>
<head><title>Date Demo</title></head>
<body>
  The date is: <my:Date/>
</body>
</html>
```

This page is incomplete and is for illustration only

- Custom tags are well suited for non-Java programmers and page design tools.
Anatomy of a Tag

- A tag has the following structure
  
  ```xml
  <prefix:name [attribute=value]*/>
  or
  <prefix:name [attribute=value]*>
  body
  <prefix:name>
  ```

- **Items:**
  - `prefix` – user assigned namespace
  - `name` – name of the tag
  - `attribute` – attribute name
  - `value` – attribute value
  - `body` – any body element nested within the tag
Using a Custom Tag

- Use the taglib directive to inform the page translator about tag libraries that you want to use:
  
  ```jsp
  <%@ taglib (uri="tagLibraryURI" | tagdir="tagDir")
  prefix="tagPrefix" %>
  ```

  **Example:**
  
  ```jsp
  <%@ taglib tagdir="/WEB-INF/tags/my" prefix="my" %>
  ```

- The prefix can be whatever you want.
Tag Library Location

- The location of a tag library is specified with either a **uri** or **tagdir** attribute.

- The URI attribute is mapped by:
  - **Explicit mapping**
    - A `<taglib>` entry in `web.xml`
  - **Implicit mapping**
    - Container scans JAR files under `/WEB-INF/lib/` looking for `.tld` files under `/META-INF/`
    - Container scans for `.tld` files under `/WEB-INF/`
    - Container-provided TLD.
    - Container treats `uri` as a relative path to a TLD file. *This is a convenience for developers, and is *not* recommended.*
Taglib URI Examples

- Either approach:
  - Explicit mapping *via* web.xml:
    
    ```xml
    <taglib>
    <taglib-uri>/MyTags</taglib-uri>
    <taglib-location>/WEB-INF/tlds/MyTags.tld</taglib-location>
    </taglib>
    ```
  
  - Implicit mapping *via*.tld:
    
    ```
    /WEB-INF/lib/mytags.jar#META-INF/taglib.tld, with:
    <uri>/MyTags</uri>
    ```

    would allow:

    ```
    <%@ taglib uri="/MyTags" prefix="my" %>
    ```

- Direct path (for developers):

  ```
  <%@ taglib uri="/WEB-INF/tlds/MyTags.tld" prefix="my" %>
  ```
The Corrected Solution

Now we can complete the earlier example:

```jsp
<%@ taglib uri="/MyTags" prefix="my" %>
<html>
<head><title>Date Demo</title></head>
<body>
  The date is: <my:Date/>
</body>
</html>
```
Problems Developing Custom Tags

- Traditional custom tags require Java programming skills.
- All but the simplest custom tags are not easy to write.
- The purpose of JSP, in contrast to servlets, is to use markup language to manage layout with embedded dynamic content. Having to write complex Java code in custom tags that focus on markup language is going backwards.
- We might want to use the JSP expression language or other custom tags when implementing a new custom tag.
The Solution

JSP 2.0 Tag Files
JSP 2.0 Tag Files

- Tag files are one of the important new concepts introduced with JSP 2.0.
- Tag files permit easier and more rapid development of custom tags.
- Tag files are developed using normal JSP syntax, including scripting elements, and used just like any other custom tag.
- Each custom tag is a separate tag file.
How Do Tag Files Differ?

- Written using JSP syntax.
  - Not all JSP directives are permitted in a tag file.
  - Supported by new tag file specific directives, actions and implicit objects.
- Identified by either a `.tag` or `.tagx` suffix.
- Intended to provide custom tag developers ease of development without loss of functionality.
  - One of the lead developers of JSTL has commented that if she had the time, she would rewrite all of JSTL using tag files.
Where Do We Put Tag Files?

- A Tag Library can be deployed
  - Under `/WEB-INF/tags/`
  - Under `/META-INF/tags/` within a JAR file under `/WEB-INF/lib/`.
  - Tag files will not be recognized as such in any other location.
- The tag library location can be specified either by the `tagdir` attribute of `<%@ taglib %>` or by a `<tag-file>` element in an optional TLD.
  - A TLD is not necessary when using `tagdir`, although they can be desirable in production.
- A TLD is mandatory to recognize tag files packaged in a JAR file.
  - The `<tag-file>` element is used in a TLD to specify a tag file.
    - Example:
      ```xml
      <tag-file>
        <name>date</name>
        <path>/META-INF/tags/MyTags/date.tag</path>
      </tag-file>
      ```

The tagdir Attribute

- A Tag Library using tag files can be deployed under /WEB-INF/tags/ by using the `tagdir` attribute of `<%@ taglib %>`.
- Each directory is a separate tag library
- We don’t need a TLD when the location is specified by the `tagdir` attribute, but certain assumptions will be made by the container, including:
  - Each `.tag` or `.tagx` file in that library is a separate tag
  - The tag name is the filename minus the suffix `.tag` or `.tagx` suffix
- Example:

```
<%@ taglib prefix="my" tagdir="/WEB-INF/tags/m" %>
<my:sample/>
```

refers to `/WEB-INF/tags/m/sample.[tag|tagx]`
Tag Files and JSP Scopes

- The request, session, and application scopes are the same as for the invoking JSP page.
- The fresh, clean, page scope is setup for each invocation of the tag, and does **not** affect the invoking JSP page.
New Directives

- New Directives
  - `<%@ tag %>`
  - `<%@ attribute %>`
  - `<%@ variable %>`

- New Actions
  - `<jsp:doBody />`
  - `<jsp:invoke />`
<%@ tag %>

- Declares that this is a Tag File.
- Used instead of `<%@ page %>`
- `<%@ tag [attributes]* %>`
- Takes a number of optional attributes, of which `import`, `body-content` and `dynamic-attributes` are probably the most commonly used.
The **body-content** attribute

- Determines what type of body is permitted for the tag. Allowed values are:
  - **empty** – This tag does not have a body.
  - **scriptless** – the body is interpreted as JSP content without scripting elements. This is the default for tag files.
  - **tagdependent** – the body is interpreted by the tag, itself.
Defining a *date* Tag

- We can replace the JSP expression we used in the “bad” example with a custom tag by moving the JSP scripting element into a tag file. For example:

  ```jsp
  <%@ tag body-content="empty" %>
  <%= new java.util.Date() %>
  ```

- Pretty easy, huh? 😊
Using Our `date` Tag

- If we place our code in
  
  `/WEB-INF/tags/css/date.tag`

  we can use it as:

```html
<%@ taglib tagdir="/WEB-INF/tags/css" prefix="css" %>
<html>
<head><title>Date Demo</title></head>
<body>The date is: <css:date/>
</body>
</html>
```
<%@ attribute %>

- Declares attributes for the tag.
- Analogous to the `<attribute>` element in a TLD.
- Syntax:

```
<%@ attribute
    name="attribute-name"
    [required="true|false"]
    [fragment="true|false"]
    [rtexprvalue="true|false"]
    [type="type"]
    [description="description"] %>
```

- Each attribute is created as a page-scoped variable of the specified type (defaults to `String`).
We can use an attribute to pass an optional format string to our tag. For example:

```<%@ tag body-content="empty" import="java.text.SimpleDateFormat"%>
<%@ attribute name="format" %>
<%
    SimpleDateFormat formatter = null;
    if (format == null) {
        formatter = new SimpleDateFormat();
    } else {
        formatter = new SimpleDateFormat(format);
    }
%
<%=
    formatter.format(new java.util.Date())
%>```
Using Our formattedDate Tag

- If we place our code in
  
  `/WEB-INF/tags/css/formattedDate.tag`

  we can use it as:

  ```
  <%@ taglib tagdir="/WEB-INF/tags/css" prefix="css" %>
  <html>
  <head><title>Date Demo</title></head>
  <body>The date is: <css:formattedDate/>
  </body>
  </html>
  ```

- Since we did not use the `format` attribute,
  `SimpleDateFormat`’s default would apply.
Using Our formattedDate Tag

- Or we can specify the date format, such as:

```html
<%@ taglib tagdir="/WEB-INF/tags/css" prefix="css" %>
<html>
<head><title>Date Demo</title></head>
<body>The date is: <css:formattedDate format="d MMM yyyy"/>
</body>
</html>
```

- This time, the scripting variable has a value – our format string – which will be used to format the date.
The `<jsp:attribute>` tag allows a JSP page to define an element’s attribute in the element body.

This allows us to easily pass JSP fragments to a custom tag, such as:

```xml
<jsp:attribute name="item" >
attr["${aKey}" ] = ${aValue} <br/>
</jsp:attribute >
```
If we use the `<jsp:attribute>` tag, we must use the `<jsp:body>` tag to denote the tag’s body content.

```xml
<jsp:attribute name="name" >
    attribute’s content
</jsp:attribute >
<jsp:body>
    body content
</jsp:body>
```
Causes the named fragment provided to the tag by a `<jsp:attribute>` element to be executed.

For example:

```xml
<c:set var="aKey" value="foo"/>
<c:set var="aValue" value="bar"/>
<jsp:invoke fragment="item"/>
```

The `<jsp:doBody/>` tag is exactly the same as `<jsp:invoke/>`, except that it operates on the fragment provided by the `<jsp:body>` element.
The **dynamic-attributes** Attribute

- Declares that this tag will accept dynamic attributes.
- Similar to having
  
  `<dynamic-attributes>true</dynamic-attributes>`

  specified for a tag a TLD file.
- The attribute’s value defines the name of a scripting variable that will contain a **Map** of the dynamic attributes.
<%@ variable %>

- Defines scripting variables exposed by the tag.
- Analogous to the <variable> element in a TLD.
- Syntax:

```jsp
<%@ variable
  ( name-given="variable-name" | name-from-attribute="attribute name"
    alias="local-name" )
[variable-class="type" (default: String)]
[declare="true|false"]
[scope="AT_BEGIN|AT_END|NESTED"]
[description="description"] %>```
Putting It All Together

example.tag:

```jsp
<%@ tag dynamic-attributes="attrMap" body-content="scriptless" %>
<%@ attribute name="bodyColor" required="true" %>
<%@ attribute name="listopen" fragment="true" required="true" %>
<%@ attribute name="listitem" fragment="true" required="true" %>
<%@ attribute name="listclose" fragment="true" required="true" %>
<%@ variable name-given="aKey" scope="NESTED" %>
<%@ variable name-given="aValue" scope="NESTED" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:invoke fragment="listopen"/>
<c:forEach var="attr" items="${attrMap}">
    <c:set var="aKey" value="${attr.key}"/>
    <c:set var="aValue" value="${attr.value}"/>
    <jsp:invoke fragment="listitem"/>
</c:forEach>
<jsp:invoke fragment="listclose"/>
<font color="${bodyColor}"><jsp:doBody/></font>
```
Putting It All Together

Example.jsp:

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="my" tagdir="/WEB-INF/tags/css" %>
<my:example bodyColor="blue"
    first="1" second="2" third="3" fourth="4" fifth="5" sixth="6">
    <jsp:attribute name="listopen">
    <table border="1">
    <caption>Dynamic Attributes</caption>
    <tr><TH>Name</TH><TH>Value</TH></tr>
    <jsp:attribute name="listclose"></table></jsp:attribute>
    <jsp:attribute name="listitem">
    <tr><td>${aKey}</td><td>${aValue}</td></tr>
    </jsp:attribute>
    </jsp:attribute>
    </jsp:attribute>
    </jsp:attribute>
    <jsp:body>This is body text.</jsp:body>
</my:example>
```
Sample Output

- Accessing example.jsp results in:

<table>
<thead>
<tr>
<th>Dynamic Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>fifth</td>
</tr>
<tr>
<td>first</td>
</tr>
<tr>
<td>sixth</td>
</tr>
<tr>
<td>third</td>
</tr>
<tr>
<td>fourth</td>
</tr>
<tr>
<td>second</td>
</tr>
</tbody>
</table>

This is body text.
EL Not Evaluating?

- If EL is not evaluating, it likely means that you are modifying a webapp from Servlet 2.3 or earlier.
- Solution is to update `web.xml`, or add an `isELIgnored="false"` attribute to tag and page directives as necessary.
Q&A and Other Examples

- In our remaining time today, let’s:
  - Look more closely at how the tags and pages thus far used were deployed on Tomcat 5.
  - Walk through other custom tags, and their uses.
  - Discuss how to implement tags of interest to attendees.
  - Address other questions that you wish to ask.
- Please turn in the session evaluation.
- Thanks for coming. 😊
Resources

- **Easy Custom Tags with Tag Files**
  [http://today.java.net/pub/a/today/2003/11/14/tagfiles.html](http://today.java.net/pub/a/today/2003/11/14/tagfiles.html)

- **Creating JSP 2.0 Tag Files**

- **Playing Tag with JSP 2.0**
  [http://www.vsj.co.uk/articles/display.asp?id=408](http://www.vsj.co.uk/articles/display.asp?id=408)
Related Sessions

- “Introduction to Portlet Programming with JSR-168” – Noel J. Bergman
- “Comparing Java Web Frameworks: JSF, Struts, Spring, Tapestry and WebWork” – Matt Raible