More on JSP

JSP Predefined Tags

<jsp:forward>

<jsp:include>

<jsp:param>

<jsp:plugin>

<jsp:useBean>

<jsp:getProperty>

<jsp:setProperty>

} Used for Java Beans

JSP Predefined Tag Example <jsp:include>

- Standard Action Example: **<JSP: include>** tag
- Example:

```
<HTML>
        <BODY>
                Going to include hello.jsp...<BR>
                 <jsp:include page="hello.jsp"/>
     </BODY>
</HTML>
```

Executes the included JSP page and adds its output into the this page

JSP Predefined Tag Example <jsp:include>

What's Difference from Using the 'include' directive?

e.g. `<%@ include file = 'hello.jsp' %>`

- The include directive includes the contents of another file at compilation time. Good for including common *static* code e.g. header file, footer file. Good on performance → included only once.

- But, what if including *dynamic* common code (e.g. a navigation bar where links are read from the dB?).. need to re-run the file *each time a request is made* -→ `JSP: include`

- `JSP: include` incorporates the *output* of the included JSP file at run time

JSP Predefined Tag Example <jsp:forward>

- Standard Action Example: **<JSP: forward>** tag
- Stops processing of one page and starts processing the page specified by the page attribute

Example:

```
<HTML>
      <BODY>
            <jsp:forward page="errorpage.jsp"/>
    </BODY>
</HTML>
```

JSP Predefined Tag Example <jsp:param>

- Standard Action Example: **<JSP: param>** tag
- Can be used to pass parameters when using `<jsp:include>` or `<JSP:forward>`
- Example

```
<jsp:forward page="login.jsp">
   <jsp:param name="username" value="jsmith" />
 </jsp:include>
```

Executes a login page
jsp:param passes in username to the login page

Standard Actions

- <jsp:useBean> : associates an instance of a java object with a newly declard scripting variable of the same id
  - <jsp:useBean id="name" scope="page|request|session|application" class="className" />
- <jsp:setProperty>
  - <jsp:setProperty name="beanid" property="*" />
- <jsp:getProperty> :action places the value of a Bean instance property, converted to a string, into the implicit out object
  - <jsp:getProperty name="beanid" property="propertyName" />
- <jsp:param>
  - <jsp:param name="paramname" value="paramvalue" />

- <jsp:include> :Include static or dynamic (jsp) pages with optional parameters to pass to the included page.
  - <jsp:include page="filename" />
  - <jsp:include page="urlSpec">
    <jsp:param name paramname" value="value">
    </jsp:include>

- <jsp:forward> : allows the runtime dispatch of the current request to a static resource, jsp pages or java servlet in the same context as the current page.
  - <jsp:forward page="url" />
  - <jsp:include page="urlSpec">
    <jsp:param name=paramname" value="value">
    </jsp:forward>

JSP and Scope

- *Page* - objects with page scope are accessible only within the page where they are created
- *Request* - objects with request scope are accessible from pages processing the same request where they were created
- *Session* - objects with session scope are accessible from pages processing requests that are in the same session as the one in which they were created
- *Application* - objects with application scope are accessible from pages processing requests that are in the same application as the one in which they were created
- All the different scopes behave as a single name space

Implicit Objects

- These objects do not need to be declared or instantiated by the JSP author, but are provided by the container (jsp engine) in the implementation class
- request Object (javax.servlet.ServletRequest)
- response Object (javax.servlet.ServletResponse)
- session Object (javax.servlet.http.HttpSession)
- application Object
- out Object
- config Object
- page Object
- pageContext Object (javax.servlet.jsp.PageContext)
- exception

Auto-Generated Servlet Code

```
JspFactory _jspxFactory = null;
javax.servlet.jsp.PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
JspWriter out = null;
Object page = this;
JspWriter _jspx_out = null;
try {
 _jspxFactory = JspFactory.getDefaultFactory(); response.setContentType("text/xml;charset=UTF-8");
pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
application = pageContext.getServletContext(); config = pageContext.getServletConfig();
session = pageContext.getSession();
out = pageContext.getOut();
_jspx_out = out;

b = (CacheBean) pageContext.getAttribute("b", PageContext.PAGE_SCOPE);
b = (CacheBean) java.beans.Beans.instantiate( this.getClass().getClassLoader(), "CacheBean");
pageContext.setAttribute("b", b, PageContext.PAGE_SCOPE);
```

JSP Standard Tag Library (JSTL)

JSTL

- JSP 1.2 introduced supported for a special tag library called the JSP Standard Tag Library (JSTL)
- Version 1.0 released in June 2002
  Version 1.1 released in June 2004
- The JSTL saves programmers from having to develop custom tag libraries for a range of common tasks, such as if statements, conditional loops etc.
- Enables developers to produce more maintainable and simpler JSP code
- **Important development for JSP technology**

JSTL

- The JSP Standard Tag Library groups actions into four libraries as follows:

Library	**Contents**
Core	Core functions such as conditional processing and looping, important data from external environments etc
Formatting	Format and parse information
SQL	read and write relational database data
Xml	Processing of XML data

JSTL

- To use any of these libraries in a JSP, need to declare using the taglib directive in the JSP page, specifying the URI and the Prefix

Library	Prefix	URI
Core	c	http://java.sun.com/jsp/jstl/core
Formatting	fmt	http://java.sun.com/jsp/jstl/fmt
SQL	sql	http://java.sun.com/jsp/jstl/sql
XMl	xml	http://java.sun.com/jsp/jstl/xml

Example of declaring use of core library:

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

JSTL: Example

Example: JSP page using JSTL that outputs 1 to 10 on a webpage using the <c:forEach> and <c:out> tags of the **core library**

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

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

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

A taglib directive declare use of core library

JSTL tag examples

JSTL: Example <c:foreach>

Looking more closely at <c:forEach tag> …..

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

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

The <forEach> tag enables loop logic. In this case, will look through 10 times. Equivalent to java "for" loop
Closing tag <c:forEach> placed after the end of body of loop

JSTL: Example <c:foreach>

All JSTL tags have a set of attributes (similar to HTML tags..)

e.g. <c:foreach> tag has 6 attributes:

```
var, items, varStatus,  begin, end, step
```

The full details for each attribute is in the JSTL specification document.

Willl need to use this document to verify WHICH tag should be used and HOW is should be used

JSTL: Example <c:out>

<c:out> .. outputs a value to webpage.

Usually uses just one attribute `value`

Examples:

```
<c:out value="${i}" />

<c:out value="The result of 1 + 2 is ${1+2}" />

<c:out value="param.userName" />
```

JSTL: Example <c:if>

<c:if> .. evaluates a condition. Uses an attribute `test` to hold the condition

Example :

```
<%-- Simple if conditions --%>
<c:if test='${param.p == "someValue"}'>
    Generate this template text if p equals
    someValue
</c:if>
```

Example 2

```
<c:if test='${param.p}'>
    Generate this template text if p equals "true"
</c:if>
```

JSTL: Multiple 'if' conditions

An if/else action requires the use of the <c:choose> tag

Syntax :

```
<c:choose>
   body content (<when> and <otherwise> subtags)
</c:choose>
```

JSTL: Multiple 'if' conditions

Uses <c:choose>, <c:when> and <c:otherwise>

Example:

```
<c:choose>
  <c:when test='${param.p == "0"}'>
      <c:out value = "zero recorded"/>
  </c:when>
  <c:when test='${param.p == "1"}'> Generate this
      <c:out value = "single value"/>
  </c:when>
  <c:otherwise>
    <c:out value = "Set to ${param.p}"/>
  </c:otherwise>
</c:choose>
```

JSTL: Other core <c:..> actions

Other examples: (NOT a complete list!)

`<c:set>` ….sets the value of a variable
`<c:remove>` ….removes a scoped variable
`<c:catch>` ….catches an exception
`<c:url>` ….. encodes a URL
`<c:import>`… imports the content of a resource
`<c:redirect>`.. redirects to another URL
`<c:param>`.. adds a request parameter to other actions

JSTL: <fmt:……> example

Library	Prefix	URI
Core	c	http://java.sun.com/jsp/jstl/core
Formatting	**fmt**	**http://java.sun.com/jsp/jstl/fmt**
SQL	sql	http://java.sun.com/jsp/jstl/sql
XMl	xml	http://java.sun.com/jsp/jstl/xml

JSTL contains a set of actions in the Formatting library – these tags are useful for formatting numbers, times and dates

e.g. `<fmt:parseDate>`

JSTL: <fmt:parseDate> example

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


<html>  etc etc

<fmt:parseDate value= ${param.empDate}"  var =
"parsedEmpDate" pattern = "yyyy-MM-dd" />

etc etc

</html>
```

The fmt:parseDate action takes the date or time string specified by the value attribute (e.g. 2001-09-28) , interprets it according to the pattern defined by the pattern attribute and saves it in a variable called "parsedEmpDate"

JSTL: other <fmt> actions

Other examples:

`<fmt:formatNumber>` - formats a numeric value
e.g. number of digits, currency, decimal place
e.g. `<fmt:formatNumber value="12.3" pattern=".000"/>`
will output "12.300"

`<fmt:formatDate>` --formats a date and time

JSTL: Expression language

- Up to now, could only use Java expressions to assign dynamic values → syntax errors common
- JSTL now provides an expression language (EL) to support the tags → simpler syntax, less errors
- The EL is now part of the JSP specification (as of versions JSP 2.0) – can be used in JSTL tags or directly in JSP pages.

JSTL: Expression language

- All EL expressions are evaluated at runtime
- The EL usually handles data type conversion and null values --→ easy to use
- An EL expression always starts with a ${ and ends with a }

JSTL: Expression language

- The expression can include
  - literals (“1”, “100” etc)
  - variables
  - implicit variables

Examples:

```
<c:out value  = “${1+2+3}” />
```

expression

```
<c:if test = “${param.Address == “D6”}” />
```

JSTL: Expression language - operators

==	>	+
!=	<=	-
<	>=	*
		/ or div

- Logical operators consist of &&, ||, and !
- The **empty** operator is a prefix operator that can used to determine if a value is null or empty. For example:

```
<c:if test=”${empty param.name}”>
     Please specify your name.
</c:if>
```

JSP Implicit objects

- In JSP, need to be able to access information about the environment in which the page is running e.g. the parameters passed in a request for a form, the browser type of the user, etc.
- Implicit objects are a set of Java objects that the JSP Container makes available to developers in each page. These objects may be accessed as built-in variables via scripting elements

JSTL implicit variables

- The JSTL EL allows these objects to be accessed as 'Implicit Variables'
- Implicit variable are just pre-agreed fixed variable names that can be used in JSTL Expressions
-
- --→ Think of as "variables that are automatically available to your JSP page"..

JSTL: Expression language – Implicit Objects

- Very common implicit object is `param`
- `param` refers to parameter passed in a request message (e.g. information entered into a form by a user).
- e.g. `<c:out value = "${param.userName}"/>`
- Further Examples of using `param` in next topic

Comparing JSTL and Scriptlets

JSTL removes complexity by using tags instead of java code (abstraction)

JSP pages using JSTL usually easier to maintain

JSTL allows HTML 'tag' developers to 'program'

JSTL often more difficult to debug

Note: Using JSTL *does not* eliminate scriplets entirely.. may still need them sometimes for more complex logic

References

1. “JSP: Action Elements and JSTL,” Susan McKeeve, Dublin Institute of Technology.
2. “Servlets and JSP,” Ilmi Yoon, San Francisco State University.