JSP – Standard Tag Library (JSTL)

Main point 1 Preview

The JSP Standard Tag Library provides convenient action tags for many common operations on a JSP page. JSTL combined with the EL provides us way to wrote clean JSP code.

Science of Consciousness: Purification leads to progress.

JSTL

- The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP use cases.
- JSTL has support for common tasks such as conditionals and iterations, tags for formatting outputs, internationalization tags, and even SQL tags.
- There are five tag libraries in the Java Standard Tag Library specification:
  - Core (`c`)
  - Formatting (`fmt`)
  - Functions (`fn`)
  - SQL (`sql`)
  - XML (`x`)

Using the core tag library

- The Core tag library, contains nearly all the core functionality you need to replace the Java code in your JSPs.
- This includes tools for conditional programming, looping and iterating, and outputting content.
- Before you can use core tag library, you first include it to your page and give the appropriate prefix (c) which you will use later on your JSP code to refer this library.

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

Tags in the Core library

- There are many tags in Core library, some more commonly used than others.

- `<c:out>`

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

  - Outputting content to your JSPs.
    - But you can do it simply using EL expressions?
  - By default, escapes reserved XML characters (<, >, ', ", and &)
    - This behavior can be disabled by setting escapeXml attribute to false.

```
<c:out value="${someVariable}" escapeXml="false" />
```

    - However, in most cases you never want to do it.
    - Default escaping protects your site from cross-site scripting and various injection attacks, and also helps prevent unexpected special characters from breaking the functionality of your site.

<c:url>

- <c:url>
  - The <c:url> tag properly encodes URLs, and rewrites them if necessary to add the session ID, and can also output URLs in your JSP.
  - If the URL is a relative URL, the tag prepends the URL with the context path for your application so that the browser receives the correct absolute URL.

```
<a href="<c:url value="/view.jsp">
   <c:param name="forumId" value="12" />
</c:url>">Product Forum</a>
```

  - It might be useful even when URL is an absolute URL, if you had query parameters that you need to include in the url.

```
<c:url value="http://www.goog.com">
    <c:param name="q" value="${searchString}" />
</c:url>
```

    - In this case the <c:url> tag will properly form and encode the query string.

<c:url>

- It also has `context` attribute, which you can use to define how your url gets resolved.

```
<c:url value="/index.html" context="/" />
<c:url value="/item.jsp?itemId=15" context="/store" />
```

- If you have URL you are going to re-use, you can save the resulting URL to a scoped variable instead:

```
<c:url value="/index.jsp" var="homepageUrl" />
<c:url value="/index.jsp" var="homepageUrl" scope="request" />
```

  - By default it is saved to the page scope which is normally sufficient.

- For maximum safety, flexibility, and portability, it is recommended that all URLs in JSPs get encoded with `<c:url>` unless the URL is an external URL with no query parameters.

<c:if>

```
<c:if test="${something == somethingElse}">
    execute only if test is true
</c:if>
```

If you have some complex condition that you want to test only once but use multiple times on the page, you can save it to a variable using the `var` attribute (and optionally specify a different scope):

```
<c:if test="${someComplexExpressionIsTrue}" var="itWasTrue" />
...
<c:if test="${itWasTrue}">
    do something
</c:if>
...
<c:if test="${itWasTrue}">
    do something else
</c:if>
```

There is no `<c:else>` for more complex if/else-if/else logic, you use `<c:choose>`, `<c:when>`, and `<c:otherwise>`

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

```
<c:choose>
    <c:when test="${something}">
        "if"
    </c:when>
    <c:when test="${somethingElse}">
        "else if"
    </c:when>
    ...
    <c:otherwise>
        "else"
    </c:otherwise>
</c:choose>
```

Only one <c:when> tag will have its contents evaluated: The first one whose test is true. The <c:choose> short-circuits to the end after a <c:when> tag has evaluated to true.

<c:forEach>

```
for(int i = 0; i < 100; i++)
{
    out.println("Line " + i + "<br />");
}
```

The equivalent <c:forEach> tag is a follows:

```
<c:forEach var="i" begin="0" end="100">
    Line ${i}<br />
</c:forEach>
```

You can also increment i by more than 1, if you want to, using the step attribute (which must be greater than or equal to 1):

```
<c:forEach var="i" begin="0" end="100" step="3">
    Line ${i}<br />
</c:forEach>
```

<c:forEach>

- While using <c:forEach> to iterate over some collection, items attribute represent the collection and var represent current item in loop.

```
<c:forEach items="${users}" var="user">
    ${user.lastName}, ${user.firstName}<br />
</c:forEach>
```

- The expression within items must evaluate to some Collection, Map, Iterator, Enumeration, object array, or primitive array. If items is a Map, the Map.Entrys are iterated by calling the entrySet method.
- You can skip collections elements in <c:forEach> using the step attribute, just like you would for iterating over numbers. You can also use the begin attribute to begin iteration at the specified index (inclusive) and the end attribute to end iteration at the specified index (inclusive).

<c:forEach>

- Finally, whether you use <c:forEach> as a for loop of numbers or a for-each loop over a collection of objects, you can use the varStatus attribute to make a variable available within the loop containing the current status of the iteration.

```
<c:forEach items="${users}" var="user" varStatus="status">
    ${status.begin} // begin attribute from the loop tag
    ${status.end} // end attribute form the loop tag
    ${status.step} // step attribute from the loop tag
    ${status.count} // count of the n umber of iterations performed so far
    ${status.current} // current item from the iteration
    ${status.index} // current index from the iteraton
    ${status.first} // true if the current iteration is the first one
    ${status.last} // true if the current iteration is the last one
</c:forEach>
```

<c:redirect>

- This tag redirects the user to another URL.
  - After adding the HTTP Location header to the response and changing the HTTP response status code, it aborts execution of the JSP.
- Because it changes response headers, <c:redirect> must be called before the response has started streaming back to the client. Otherwise, it is not successful in redirecting the client, and the client instead receives a truncated response.

```
<c:redirect url="http://www.example.com/" />

<c:redirect url="/tickets">
    <c:param name="action" value="view" />
    <c:param name="ticketId" value="${ticketId}" />
</c:redirect>

<c:redirect url="/browse" context="/store" />
```

Main point 1

The JSP Standard Tag Library provides convenient action tags for many common operations on a JSP page. JSTL combined with the EL provides us way to wrote clean JSP code.

Science of Consciousness: Purification leads to progress.

Main point 2 Preview

Developers can create custom tags for JSP that use a Java tag handler to provide any desired functionality in a simple tag definition for JSP authors.

Science of Consciousness: Knowledge of custom tags gives developers a means to create any JSP tag functionality. Knowledge and experience of the unified field gives us access to the source of all possibilities.

Custom Tags

Actions Supported by All the Laws of Nature

UNDERSTANDING TLDS, TAG FILES, AND TAG HANDLERS

- All JSP tags result in execution of some tag handler.
  - The tag handler is an implementation of `javax.servlet.jsp.tagext.Tag` or `javax.servlet.jsp.tagext.SimpleTag` and contains the Java code necessary to achieve the tag's wanted behavior.
- A tag handler is specified within a tag definition in a TLD, and the container uses this information to map a tag in a JSP to the Java code that should execute in place of that tag.

Custom Tags

- A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler.
- To write a custom tag you can simply extend **`SimpleTagSupport`** class and override the **`doTag()`** method, where you can place your code to generate content for the tag.

Build your TLD file

WEB-INF/custom.tld

```
<taglib ... >
  <tlib-version>1.0</tlib-version>
  <short-name>My-Custom-Tags</short-name>
  <uri>http://cs.mum.edu</uri>
  <tag>
    <name>tag1</name>
    <tag-class>com.test.MyTag</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

JSP

```
<%@ taglib prefix="ct" uri="http://cs.mum.edu"%>
<ct:tag1/>
```

Build your Java Class

```
public class MyTag extends SimpleTagSupport {

  public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();
    out.println("My very first custom tag.");
  }
}
```

Custom Tags with Body

custom.tld

```
<tag> ... <name>tag2</name>
        <body-content>scriptless</body-content>
...</tag>
```

JAVA

```
public class MyTag2 extends SimpleTagSupport {
    StringWriter sw = new StringWriter();
    public void doTag()
        throws JspException, IOException
     {
        getJspBody().invoke(sw); // capture body content into StringWriter
        getJspContext().getOut().println(sw.toString());
     }
}
```

JSP

```
<ct:tag2>
    Hello from the tag body.
</ct:tag2>
```

Custom Tag with Attributes

custom.tld

```
<tag>... <name>tag3</name> ...
    <attribute>
        <name>message</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

false: String literal
true: EL, Script expressions

JSP

```
<ct:tag3 message="Hello From Attributes :)" />
<ct:tag3 message='<%= request.getHeader("User-Agent") %>' />
<ct:tag3 message="${pageContext.request.localAddr}" />
```

Custom Tag with Attributes

JAVA

```
public class MyTag3 extends SimpleTagSupport {

   private String message;

   public void setMessage(String msg) {
      this.message = msg;
   }

   public void doTag()
      throws JspException, IOException
    {
       if (message != null) {
          JspWriter out = getJspContext().getOut();
          out.println( message );
       }
   }
}
```

Label Example

Create a custom tag that generates a SPAN element with color red.
We need to read two properties:

- Foreground color
- Text

JSP

```
<ct:label foreColor='red' text='Hello'/>

<!-- will generate the following HTML:
<span style='color:red'>Hello</span>  -->
```

Label Example

custom.tld

```
<taglib>
...
  <tag>
    <description>Generates a label</description>
    <name>label</name>
    <tag-class>com.test.MyTag4</tag-class>
    <body-content>empty</body-content>
    <attribute>
        <name>foreColor</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>text</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    </tag>
</taglib>
```

Label Example

```
public class MyTag4 extends SimpleTagSupport {
    String foreColor; String text;

    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        if (foreColor != null)
        out.write(String.format("<span style='color:%s'>%s</span>",
        foreColor, text));
        else
        out.write(String.format("<span>%s</span>", text));
    }
    // Setters
    public void setForeColor(String foreColor) {
        this.foreColor = foreColor;
    }
    public void setText(String text) {
        this.text = text;
    }
}
```

Why JSP custom tags are important

- The main purpose of most JSP custom tags is to provide an easy mechanism to dynamically "generate markup" for common processing tasks
- The basic concept of components for markup generation will be found in most web app frameworks:
  - e.g., JSF is a component based framework
  - All JSF tags will be implemented as "custom tags"
  - Slightly more complex because have to satisfy requirements of the JSF framework
  - Angular, React and most of other modern SPA frameworks are also component based.

Main point 2 Preview

Developers can create custom tags for JSP that use a Java tag handler to provide any desired functionality in a simple tag definition for JSP authors.

Science of Consciousness: Knowledge of custom tags gives developers a means to create any JSP tag functionality. Knowledge and experience of the unified field gives us access to the source of all possibilities.

Exercise

Create a custom tag `<ct:currentDateTime>` that accepts two attributes (color and size) that prints the current date and time.

Example:

JSP: `<ct:currentDateTime color="red" size="12px" />`

HTML: `<span style="color: red; font-size: 12px;">Mon 2016.04.04 at 04:14:09 PM PDT</span>`

You may use the following code snippet:

```
Date dNow = new Date();
SimpleDateFormat ft = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
System.out.println("Current Date: " + ft.format(dNow));
```

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Actions Supported by All the Laws of nature

1. Custom tags are easy for JSP page authors to use.

2. Java developers can create custom tags for any functionality.

---

3. **Transcendental consciousness** is the experience of the home of all the laws of nature.

4. **Impulses within the transcendental field:** Thoughts that arise from this transcendental field will be naturally in accord with all the laws of nature because this transcendental field is the unified field from which the laws of nature arise.

5. **Wholeness moving within itself:** A developer who understands the basic coding mechanisms underlying custom tags will feel a deep level of comfort and connection with JSP pages and web application technologies and frameworks that rely on them. In a similar manner, in unity consciousness we appreciate that the pure consciousness we experience as our deepest Self, is also the same unified field of consciousness that is expressed as the rest of the manifest world and feel a deep level of comfort and connection with that.