JSTL

JSP Standard Tag Library(JSTL) is a standard library of readymade tags. The JSTL contains several tags that can remove scriplet code from a JSP page by providing some ready to use, already implemented common functionalities.

JSTL is divided into 5 groups:

1. **JSTL Core**: JSTL Core provides several core tags such as **if**, **forEach**, **import**, **out** etc. to support some basic scripting task. Url to include JSTL Core Tag inside JSP page is

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

2. **JSTL Formatting**: JSTL Formatting library provides tags to format text, date, number for Internationalized web sites. Url to include JSTL Formatting Tags inside JSP page is

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

3. **JSTL sql**: JSTL SQL library provides support for Relational Database Connection and tags to perform operations like insert, delete, update, select etc. on SQL databases. Url to include JSTL SQL Tag inside JSP page is

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

4. **JSTL XML**: JSTL XML library provides support for XML processing. It provides flow control, transformation features etc. Url to include JSTL XML Tag inside JSP page is

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

5. **JSTL functions**: JSTL functions library provides support for string manipulation. Url to include JSTL Function Tag inside JSP page is

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

JSTL Core Library

The JSTL core library contains several tags that can be used to eliminate the basic scripting overhead such as `for` loop, `if...else` conditions etc. from a JSP Page. Let's study some important tags of JSTL Core library.

- **JSTL if tag**: The if tag is a conditional tag used to evaluate conditional expressions. When a body is supplied with `if` tag, the body is evaluated only when the expression is true. For Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
      <c:if test="${param.name == 'studytonight'}">
      <p>Welcome to ${param.name} </p>
    </c:if>
  </body>
</html>
```

- **JSTL out tag**: The out tag is used to evaluate an expression and write the result to **JspWriter**. For Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
      <c:out value="${param.name}" default="StudyTonight" />
  </body>
  </html>
```

The `value` attribute specifies the expression to be written to the JspWriter. The `default` attribute specifies the value to be written if the expression evaluates null.

- **JSTL forEach tag**: This tag provides a mechanism for iteration within a JSP page. JSTL `forEach` tag works similarly to **enhanced for** loop of Java Technology. You can use this tag to iterate over an existing collection of items. For Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
      <c:forEach var="message" items="${errorMsgs}" >
        <li>${message}</li>
      </c:forEach>
  </body>
</html>
```

Here the attribute **items** have its value as an EL expression which is a collection of error messages. Each item in the iteration will be stored in a variable called **message** which will be available in the body of the **forEach** tag.

- **JSTL choose, when, otherwise tag**: These are conditional tags used to implement conditional operations. If the test condition of the `when` tag evaluates to true, then the content within `when` tag is evaluated, otherwise the content within the `otherwise` tag is evaluated.

  We can also implement `if-else-if` construct by using multiple **when** tag.
  The **when** tags are mutually exclusive, that means the first when tag which evaluates to

true is evaluated and then, the control exits the `choose` block. If none of the when condition evaluates to true, then `otherwise` condition is evaluated. For Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>

    <c:forEach var="tutorial" items="${MyTutorialMap}" begin="0" end="5" varStatus="stat
us">

      <c:choose>

        <c:when test="${status.count %2 == 0 }">
        <p> Divisible by 2 : ${tutorial.key} </p>
        <br/>
        </c:when>

        <c:when test="${status.count %5 == 0 }">
        <p > Divisible by 5 : ${tutorial.key} </p>
        <br/>
        </c:when>

        <c:otherwise>
        <p> Neither divisible by 2 nor 5 : ${tutorial.key} </p><br/>
        </c:otherwise>

      </c:choose>

    </c:forEach>

  </body>
</html>
```

- **JSTL import tag**: `<c:import>` tag is used to dynamically add the contents from the provided URL to the current page, at request time. The URL resource used in the `<c:import>` url attribute can be from outside the web Container. For Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:import url="http://www.example.com/hello.html">>
    <c:param name="showproducts" value="true"/>
    </c:import>
  </body>
</html>
```

- **JSTL url tag**: The JSTL url tag is used to store a url in a variable and also perform url rewriting when necessary. For Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <a href='<c:url value="/home.jsp"/>' > Go Home </a>
  </body>
</html>
```

- **JSTL set tag**: The JSTL set tag is used to store a variable in specified scope or update the property of JavaBean instance. Following is the example of setting the **name** property of a Student bean:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:set target="student" property="name" value="${param.name}" />
  </body>
</html>
```

- **JSTL catch tag**: The JSTL catch tag is used to handle exception and doesn't forward the page to the error page. For Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:catch>
      <% int a = 0;
         int b = 10;
         int c = b/a;
      %>
     </c:catch>
  </body>
</html>
```

Custom Tag

When EL and Standard Action elements aren't enough to remove scriptlet code from your JSP Page, you can use Custom Tags. Custom tags are nothing but user-defined tags.

Custom tags are an excellent way to abstract the complexity of business logic from the presentation of Web pages in a way that is easy for the Web author to use and control. It also allows for reusability as custom tags can be used again and again.

Format of Custom tag

The format of a custom tag can either be empty, called an **Empty tag**, or can contain a body, called a **Body tag**. The number of attributes that a tag will accept depends on the implementation of the Tag Handler class.

Syntax for an Empty Tag is :

```
<tagLibraryPrefix:customTagName attribute1="attributeName"
    attribute2="attributeName" ... />
```

The Syntax for a Custom Body Tag is :

```
<tagLibraryPrefix:customTagName attribute1="attributeName"
    attribute2="attributeName" ... />
  < --Body of custom tag-- >
</tagLibraryPrefix:customTagName>
```

Creating custom tags is considered as a very good practice in JSP world. Always try to create and use your own custom tags from frequently used operations in your JSP application. Let's move to the next lesson and study how to create a Custom tag.

Creating a Custom Tag

To create a Custom Tag the following components are required :

1. The **Tag Handler** class which should extend `SimpleTagSupport`.
2. The **Tag Library Descriptor(TLD)** file
3. Use the Custom Tag in your JSP file

Tag Handler Class

You can create a Tag Handler class in two different ways:

1. By implementing one of three interfaces: `SimpleTag`, `Tag` or `BodyTag`, which define methods that are invoked during the life cycle of the tag.
2. By extending an abstract base class that implements the `SimpleTag`, `Tag`, or `BodyTag`interfaces. The **SimpleTagSupport**, **TagSupport**, and **BodyTagSupport** classes implement the SimpleTag, Tag and BodyTag interfaces. Extending these classes relieves the tag handler class from having to implement all methods in the interfaces and also provides other convenient functionality.

Tag Library Descriptor

A Tag Library Descriptor is an XML document that contains information about a library as a whole and about each tag contained in the library. TLDs are used by the web container to validate the tags and also by JSP page development tools.

Tag library descriptor file must have the extension `.tld` and must be packaged in the **/WEB-INF/**directory or subdirectory of the WAR file or in the **/META-INF/** directory or subdirectory of a tag library packaged in a JAR.

Example of Custom Tag

In our example, we will be creating a Tag Handler class that extends the **TagSupport** class. When we extend this class, we have to override the method `doStartTag()`. There are two other methods of this class namely `doEndTag()` and `release()`, that we can decide to override or not depending on our requirement.

CountMatches.java

```
package com.studytonight.taghandler;


import java.io.IOException;
import javax.servlet.jsp.*;
import org.apache.commons.lang.StringUtils;


public class CountMatches extends TagSupport {
  private String inputstring;
  private String lookupstring;
```

```
  public String getInputstring() {
    return inputstring;
  }

  public void setInputstring(String inputstring) {
    this.inputstring = inputstring;
  }

  public String getLookupstring() {
    return lookupstring;
  }

  public void setLookupstring(String lookupstring) {
    this.lookupstring = lookupstring;
  }

  @Override
  public int doStartTag() throws JspException {
    try {
      JspWriter out = pageContext.getOut();
      out.println(StringUtils.countMatches(inputstring, lookupstring));
    }
    catch (IOException e) {
      e.printStackTrace();
    }
    return SKIP_BODY;
  }
}
```

In the above code, we have an implementation of the `doStartTag()` method which is must if we are extending **TagSupport** class.

We have declared two variables `inputstring` and `lookupstring`. These variables represent the **attributes** of the custom tag. We must provide getter and setter for these variables in order to set the values into these variables that will be provided at the time of using this custom tag. We can also specify whether these attributes are required or not.

CountMatchesDescriptor.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>cntmtchs</shortname>
  <info>Sample taglib for Substr operation</info>
  <uri>http://studytonight.com/jsp/taglib/countmatches</uri>

  <tag>
    <name>countmatches</name>
    <tagclass>com.studytonight.taghandler.CountMatches</tagclass>
    <info>String Utility</info>
    <attribute>
       <name>inputstring</name>
       <required>true</required>
    </attribute>
    <attribute>
       <name>lookupstring</name>
       <required>true</required>
    </attribute>
  </tag>

</taglib>
```

The taglib element specifies the schema, required JSP version and the tags within this tag library. Each **tag** element within the TLD represents an individual custom tag that exist in the library. Each of these tags should have a tag handler class associated with them.

The **uri** element represents a Uniform Resource Identifier that uniquely identifies the tag library.

The two **attribute** elements within the **tag** element represents that the tag has two attributes and the **true** value provided to the **required** element represents that both of these attributes are required for the tag to function properly.

test.jsp

```
<%@taglib prefix="mytag" uri="/WEB-INF/CountMatchesDescriptor.tld"%>
<html>
```

```
  <mytag:countmatches inputstring="Studytonight" lookupstring="t">

  </mytag:countmatches>

</html>
```

If this tag works fine it should print a value `3` in the browser as there 't' occurs 3 times in the word 'Studytonight'.