Web Component Development Using Java

Session: 12

JavaServer Pages Standard Tag Library

Objectives

- Explain the concept and need for JSTL
- List the advantages of using JSTL
- Describe the different tag libraries available in JSTL
- Explain how to configure JSTL library in NetBeans
- Explain general purpose tags
- Explain decision-making in the tags
- Explain iteration tags in the core tag library
- Explain the different tags available in the SQL tag library

Introduction

How a Web designer can design a Web page to display a dynamic list in a Shopping cart site with selected or removed products?

- To help Web designers, Sun developed a pre-defined tag library.
- The library contains tags related to the core common functionalities performed in Java applications.
- The pre-defined tag library is known as **JavaServer Pages Standard Tag Library (JSTL)**.
- JSTL:
  - Helps to reuse standard tags that work in the similar manner in every Java Web application.
  - Allows programming using tags rather than scriptlet code.

Designing JSP Pages with JSTL 1-2

- The roles found in designing and developing a Web applications are as follows:

Web Designers

- They are involved in creating a view part of the application.
- The views are basically HTML pages.

Web Component Developers

- They are responsible for developing the controller of the application.
- The controller is basically a Servlet written in Java language.

Business Component Developers

- They are responsible for creating the model for the application.
- The model is a component, such as a Java class or an Enterprise JavaBean (EJB) that are used to process the business logic of an application.

Designing JSP Pages with JSTL 2-2

- The benefits of using JSTL are as follows:

JSTL help Web designers to integrate the Java technology code into JSP, without the need to write complex scriptlet code.

JSTL provides most of the functionality necessary for the development of JSP application.

JSTL tags helps to create a program with business logic that saves lots of development time.

JSTL is similar to HTML and therefore HTML programmers easily start programming using JSTL.

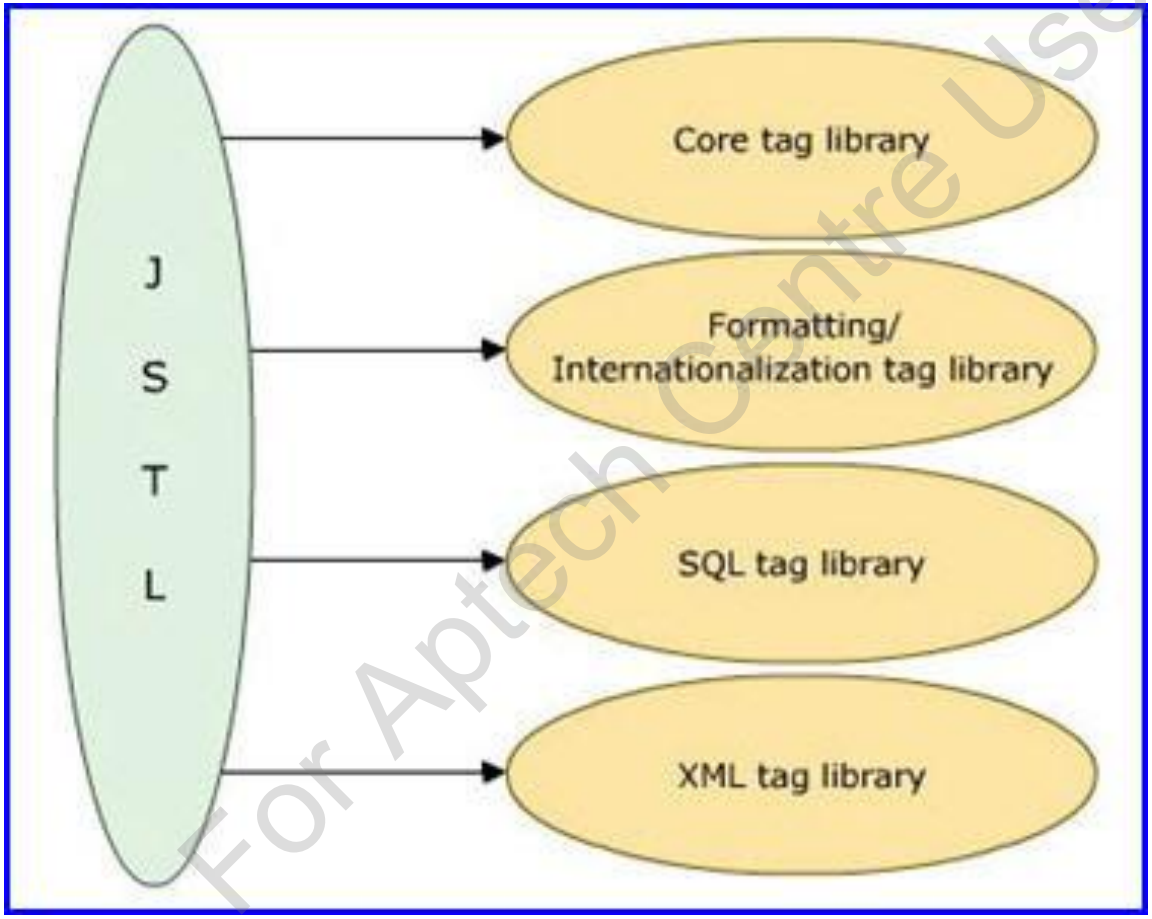
JSTL is expressed in XML-compliant tags and therefore it is easier for HTML generation tools to parse the JSTL code contained within the document.

JSTL tags provides a consistent approach to formatting of numbers and strings, and internationalization (I18N) support features in JSP scriptlet code.

JSTL provides mechanism that enables the programmer to develop own custom tags.

JSTL Tag Libraries 1-3

- JSTL provide various tag libraries that can be used for many functionalities.
- Following figure shows the JSTL tag libraries.



JSTL Tag Libraries 2-3

- **Core Tag Library**
  - It contain tags for looping, expression evaluation, and basic input/output.
  - It can be declared by `<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>`.

- **Formatting/Internationalization (I18N) Tag Library**
  - It contain tags used to parse the data such as dates and time, based on the current location.
  - It can be declared by `<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>`.

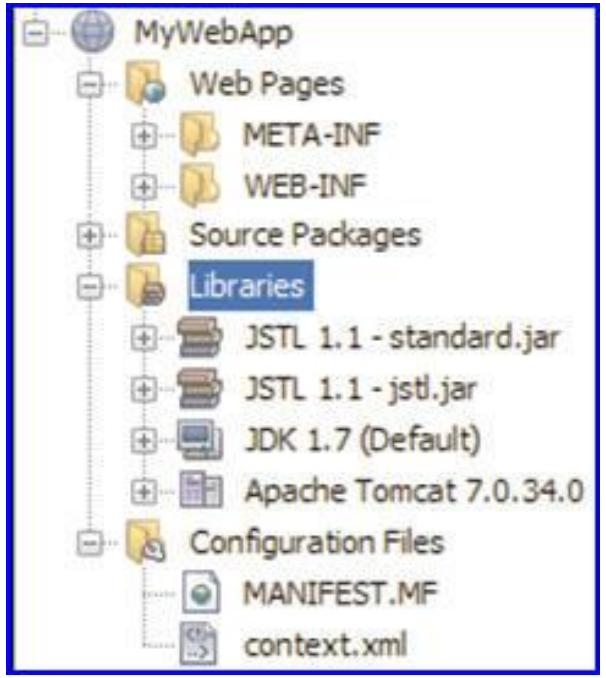
JSTL Tag Libraries 3-3

- **SQLTag Library**
  - It contains tags used to access SQL databases.
  - It provides an interface for executing SQL queries on database through JSP.
  - It can be declared by `<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>`.

- **XML Tag Library**
  - It contains tags for accessing XML elements.
  - It can be declared by `<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>`.

JSTL Library in NetBeans

- To use the JSTL tags on the JSP page, it is required to include the JSTL library in the Web application.
- To do so, perform the following steps:
  - Right-click **Libraries** and select **Add Library**.
  - Select **JSTL 1.1** under **Available Libraries** and click **Add Library**. The `JSTL 1.1- standard.jar` and `JSTL 1.1- jstl.jar` are added in the project.
  - Following figure shows the JSTL library added in **MyWebApp** application project created in NetBeans IDE.



Core Tag Library

- It provides support for conditional logic, iteration, forward or redirect response, URL, catch exception, and so on.
- It is the most frequently used JSTL tag library containing core group of tags.
- The core tag library tags are prefixed with `c`.
- Some of the Core JSTL tags include:
  - General Purpose Tags
  - Decision-making Tags
  - Iteration Tags

General Purpose Tags 1-7

- They are used to set, remove, and display variable values created within a JSP page.
- The core tag library contains tags for getting, setting, and displaying attribute values.
- The general purpose tags are as follows:

<c:set>

- It assigns a value to a variable in scope.
- **Syntax:**

```
<c:set var = "varName" value = "expression"
scope = "page/request/ session/ application"/>
```

where,

- `value` specifies the expression.
- `var` specifies the name of the exported scope to hold the value specified in the tag.
- `scope` specifies the scope of variable such as page, request, session, and application. The default scope is `page`.

- **Example:** `<c:set var = "sessionvariable" value = "${80+8}" scope = "session" />`

General Purpose Tags 2-7

<c:remove>

- This is an empty tag used to remove a scoped variable.
- **Syntax:**

```
<c:remove var = "varName" scope =
"page/request/session/application"/>
```

where,

- `var` specifies the name of the variable to be removed.
- `scope` specifies the scope of the variable.

- **Example:** `<c:remove var="simple" scope="page"/>`

General Purpose Tags 3-7

<c:out>

- It is used to evaluate an expression and store the result in the current `JspWriter` object.
- **Syntax:**

```
<c:out value = "value" escapeXml =
"boolean" default = "defaultValue"/>
```

where,

- `value` specifies the expression.
- `default` specifies the default value if the value of the result is null.
- `escapeXml` determines that special characters in the result, such as <, >, &, ', and " should be converted to their character entity codes. The default value of `escapeXml` is true.

- **Example:**

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

General Purpose Tags 4-7

<c:catch>

- It provides an exception handling functionality such as try-catch, inside JSP pages without using scriptlets.
- **Syntax:**

```
<c:catch [var="varName"]>
    nested actions
    . . .
</c:catch>
```

where,

- `var` specifies the name of the variable to be caught.

General Purpose Tags 5-7

- The code snippet demonstrates how to catch the exception using `<c:catch>` tag.

```
<body>

<c:catch var="e">
   100 divided by 0 is
   <c:out value="${100/0}" />
<br />

</c:catch>
```

- The `catch` tag provides an error handling mechanism for the division operation.
- The exception raised by dividing the number from 0 is stored in the variable `var`.

General Purpose Tags 6-7

- The code snippet demonstrates the core tags used in login application.

```
<!-- welcome.html -->
<body>

<form action="login.jsp">
 User Name:
 <input type="text" name="username"/>
 <br/> <br/>
 Password:
 <input type="password" name="password"/>
 <br/> <br/>
 <input type="submit" value="Submit" name="Submit">
</form>
</body>
```

General Purpose Tags 7-7

```
<!-- login.jsp -->
. . .
<body>
        <c:set var="name" value="${param.username}"/>
        <c:set var="password" value="${param.password}"/>

        Entered Name: <c:out value="${name}"/> <br/> <br/>
        Entered Password: <c:out value="${password}"/>

</body>
```

- The `login.jsp` page assigns the variables `name` and `password` with the values from the request parameters `username` and `password` respectively.
- The `out` tag outputs the value of `name` and `password` respectively.

Decision-making Tags 1-5

- JSTL provides decision-making tags to support conditions in a JSP page.
- These tags are necessary as the contents or the output of the JSP page is often conditional based on the value of the dynamic application data.
- The two types of decision-making tags are as follows:
  - `<c:if>`
  - `<c:choose>`

Decision-making Tags 2-5

<c:if>

- The tag is used for conditional execution of the code.
- This tag is a container tag.
- It allows the execution of the body if the test attribute evaluates to true.
- **Syntax:**

```
<c:if test = "testCondition" var = "varName"
scope = "page/request/session/application">

        Body Content

</c:if>
```

where,

- `test` specifies the test condition.
- `var` specifies the name of the variable of the test condition.
- `scope` specifies the scope of the variable, `var`.

- In `<c:if>` tag, attribute `var` and `scope` are optional.

Decision-making Tags 3-5

- The code snippet demonstrates the use of `if` tag to evaluate.

```
<!-- Test the username -->
        <c:if test="${name=='admin'}">
            <h4> Valid User </h4>
        </c:if>
```

- The code uses `if` tag to check the value of the name variable.
- If the condition evaluates to true, that is, the value assigned to the `name` variable is `admin`, then the body gets executed.

Decision-making Tags 4-5

<c:choose>

- The tag is similar to the `switch` statement in Java.
- The <c:choose> tag performs conditional block execution.
- The <c:choose> tag processes body of the <c:when> tag.
- Multiple <c:when> tags can be embedded in a <c:choose> tag.
- If none of the conditions evaluates to true, then the body of <c:otherwise> tag is processed.
- **Syntax:**

```
<c:choose>
  <c:when test = "testcondition">
        Body Content
  </c:when>
   . . .
     . . .
   <c:otherwise>
       Body Content
  </c:otherwise>
</c:choose>
```

where,

- <c:when> is the body of <c:choose>. It will execute the body content if the test condition evaluates to true.
- <c:otherwise> is executed when none of the test conditions of <c:when> evaluates to true.

Decision-making Tags 5-5

- The code snippet demonstrates the `choose` tag.

```
// Performs the checking of the condition
    <c:choose>
                <c:when test="${name == 'admin'}">
                    <h4> Valid User </h4>
                </c:when>
                <c:otherwise>
                    <h4> Invalid User </h4>
                </c:otherwise>
      </c:choose>
```

- The `choose` tag performs evaluation of the condition to test if name entered is `'admin'`.
- If the condition evaluates to true, then the nested when tag gets executed, else the nested `otherwise` tag gets executed.

Iteration Tags 1-5

- The iteration tag is required for performing looping function.
- The object can be retrieved from a collection in the JavaBeans components and assigned to a scripting variable by using iteration tags.
- The two types of Iteration tags are as follows:
  - `<c:forEach>`
  - `<c:forTokens>`

Iteration Tags 2-5

<c:forEach>

- This tag is used to repeat the body content over a collection of objects.
- The iteration will continue for a number of times specified by the user in the code.
- **Syntax:**

```
<c:forEach var = "varName" item = "collection"
varStatus = "varStatusName" begin = "begin" end =
"end" step = "step">

        Body Content

</c:forEach>
```

where,

- `var` specifies the name of the exported scoped variable.
- `item` specifies the collection of items to iterate over.
- `varStatus` specifies the name of the variable for the status of iteration.
- `begin` specifies the index from which the iteration is to begin.
- `end` specifies the index at which the iteration is to end.
- `step` specifies that iteration will process every item of the collection.

Iteration Tags 3-5

- The code snippet demonstrates the `forEach` tag.

```
// Displays objects from collection
"companies"
<c:forEach var= "company" items =
"${companies"> ${company} <br>
</c:forEach>

// Displays values from 10 to 100
<c:forEach var="i" begin="10" end="100">
      ${i}
</c:forEach>
```

Iteration Tags 4-5

<c:forTokens>

- It is used to iterate over a collection of tokens separated by user-specified delimiters.
- It is a container tag.
- **Syntax:**

```
<c:forTokens items = "stringofToken" delims =
"delimiters" var = "varName" varStatus =
"varStatusName" begin = "begin" end = "end" step =
"step">
            Body Content
</c:forTokens>
```

where,

- `Items` specifies the string of value to iterate.
- `delims` specifies the character that separates the tokens in the string.
- `var` specifies the name of the scope variable for the item of iteration.
- `varStatus` specifies the name of the scope variable for the status of iteration.

Iteration Tags 5-5

- The code snippet demonstrates the <c:forTokens> tag.

```
// Displays each token at a time separated by
// ',' delimiter
<c:forTokens var="token" items="Tom,Dick,Harry"
delims=",">

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

SQL Tag Library

- JSTL SQL tag library is used to interact with other databases such as Oracle, MySQL, or Microsoft SQL Server.
- These tags are prefixed by `sql`.
- `SQL` tags include:

`<sql:setDataSource>`
`<sql:query>`
`<sql:update>`
`<sql:transaction>`
`<sql:param>`

<sql:setDataSource> 1-3

- JSTL SQL tags are used to access databases.
- They are designed for low-volume Web-based applications.
- SQL tag library provides tags that allow direct database access within a JSP page.
- The JSTL SQL tag provides the following functionalities:

Passing Database Queries	• The functionality allows executing queries using the `<sql:query>` tag.
Accessing Query Results	• The functionality allows the users to access results for queries.
Database Modifications	• The functionality helps in modifying database using the `<sql:update>` tag.

<sql:setDataSource> 2-3

❖ Syntax:

```
<sql:setDataSource dataSource = "datasource" | url
= "jdbcurl"
driver = "driverclassdriver"
user = "username" password = "userpwd"
var = "varname"
scope = "page/request/session/application"/>
```

where,

- `dataSource` can either be the path to Java Naming and Directory Interface (JNDI) resource or a JDBC parameter string.
- `url` is the URL associated with the database.
- `driver` is a JDBC parameter and takes the driver class name.
- `user` takes the user of the database.
- `password` takes the user password.
- `var` is the name of exported scoped variable for the data source specified.
- `scope` specifies the scope of the variable.

<sql:setDataSource> 3-3

- The code snippet demonstrates the use of `<sql:setDataSource>` sql tag to make the connection to SQL Server database.

```
// Sets the data source for the database
<sql:setDataSource

driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"

url="jdbc:microsoft:sqlserver://10.1.3.27:1433;DataBa
seName=pubs;" user="sa" password="playware"
var="conn"/>
```

- In the code:
  - `<sql:setDataSource>` is used to set a data source for the database.
  - This is an empty tag and allows the user to set data source information for the database.
  - The `url` attribute provides the JDBC url string for the SQL Server database.

<sql:query> 1-3

- The `<sql:query>` tag searches the database and returns a result set containing rows of data.
- The tag can either be an empty tag or a container tag.
- The `SELECT` statement is used to select data from the table.
- **Syntax:**

```
<sql:query sql = "sqlQuery" var = "varName" scope
= "{page|request|session|application}" dataSource
= "dataSource" maxRows = "maxRows" startRow =
"startRow"/>
```

where:

- `sql` specifies the SQL query statement.
- `var` specifies the name of the exported scope variable for the query result.
- `scope` specifies the scope of the variable.
- `dataSource` specifies the data source associated with the database to query.
- `maxRows` specifies the maximum number of rows to be included in the result.
- `startRow` specifies the row starting at the specified index.

<sql:query> 2-3

❖ **Syntax:**

```
<sql:query var = "varName" dataSource =
"dataSource" scope =
"{page|request|session|application}" maxRows =
"maxRows" startRow = "startRow">

      SQL Statement
      <sql:param/>

</sql:query>
```

where,

- `param` takes the parameter for the query.

<sql:query> 3-3

- The code snippet demonstrates the use of `<sql:query>` tag.

```
// Gets the records about the 'product' from data
source 'conn'
<sql:query var="products" dataSource="${conn}">
            select * from Products
</sql:query>
```

<sql:update> 1-3

- The `<sql:update>` tag executes the `INSERT`, `UPDATE`, and `DELETE` statements.
- It returns 0, if no rows are affected by the DML statements.
- **Syntax:**

```
<sql:update sql = "sqlUpdate" dataSource =
"dataSource" var = "varName" scope =
"{page|request|session|application}"/>
```

where,

- `sql` specifies the update, insert, or delete statement.
- `dataSource` specifies the data source associated with the database to update.
- `var` specifies the name of the exported scope variable for the result of the database update.
- `scope` specifies the scope of variable, such as page, request, session, or application.

<sql:update> 2-3

❖ **Syntax:**

```
<sql:update dataSource = "dataSource" var =
"varName" scope =
"{page|request|session|application}">

      SQL Statement
      <sql:param value = "value"/>


</sql:update>
```

where,

- `update` is the `UPDATE` statement in SQL.
- `param` takes the parameter for the query.

<sql:update> 3-3

- The code snippet demonstrates the use of `<sql:update>` tag.

```
// Adds the product details in the Product table

<sql:update var="newrow" dataSource="${conn}"

INSERT INTO Products(ProductName, ProductType, Price,
Brand, Description)

VALUES('Jeans', 'Clothes', '1000', 'Lee', 'Good Quality
Jeans')

</sql:update>
```

<sql:transaction> 1-2

- The `<sql:transaction>` is used to establish a transaction context for `<sql:query>` and `<sql:update>` tags.
- The connection object is obtained from `<sql:transaction>`.
- This tag is responsible for managing access to the database.
- **Syntax:**

```
<sql:transaction dataSource = "dataSource" isolation =
isolationLevel>
        <sql:update> or <sql:query > statements
</sql:transaction>
```

where,

- `dataSource` sets the SQL data source which can be a string or a data source object.
- `isolation` sets the transaction isolation level. Isolation level can be `read_committed`, `read_uncommitted`, `repeatable_read`, or `serializable`.

<sql:transaction> 2-2

- The code snippet demonstrates the use of `<sql:transaction>` tag.

```
/** The code snippet performs transaction by first accessing a
data source to create a table and then inserting a row. It then
performs a SQL queries on the table
**/
<sql:transaction dataSource="${mydatasource}">
        <sql:update var="newTable">
                create table emp (
                id int primary key,
                name varchar(80)
                )
        </sql:update>
        <sql:update var="updateCount">
                INSERT INTO emp VALUES (1,'Jenny')
        </sql:update>
        <sql:update var="updateCount">
                INSERT INTO emp VALUES (2,'Christina')
        </sql:update>
        ...
        <sql:query var="empQuery">
                SELECT * FROM emp
        </sql:query>
</sql:transaction>
```

<sql:param> 1-3

- `<sql:param>` is used to set values for parameters markers ('?') in SQL statements.
- It acts as a sub tag for `<sql:query>` and `<sql:update>`.
- **Syntax:**

```
<sql:param value = "value"/>
```

where,

- `value` sets the value for the parameter.

<sql:param> 2-3

- The code snippet demonstrates how to use the <sql:param> in database updation.

```
//The code snippet sets the values for the
parameters//
<sql:update>
      INSERT INTO Employee (DemoFirstName,
DemoLastName, EmpDate)
      VALUES(?, ?, ?)
      <sql:param value="${param. DemofirstName}"
/>
      <sql:param value="${param. DemolastName}" />
      <sql:dateParam value="${empDemoDate}"
type="date" />
</sql:update>
```

<sql:param> 3-3

- Following figure depicts the use of `<sql:param>` tag.

```
<sql:query var="product"
  sql="select * from PUBLIC.product where id = ?" >
  <sql:param value="${productId}" />
</sql:query>

<c:forEach var="productRow" begin="0"
  items="${product.rows}">
  <sql:update var="product" sql="update PUBLIC. product set
    inventory = inventory - ? where id = ?" >
    <sql:param value="${item.quantity}" />
    <sql:param value="${productId}" />
  </sql:update>
</c:forEach>
```

Summary

- JSTL provides a set of reusable standard tags.
- JSTL standard tag library works in a similar manner everywhere and this makes the iteration over the collection using scriptlets unnecessary.
- JSTL allows programming using tags rather than scriptlet code.
- The core tag library has general purpose tags that are used to manipulate scoped variables created within a JSP page.
- Decision-making tags are used to do conditional processing of code in a JSP page.
- Iteration tags are used to iterate over a collection of objects multiple times.
- SQL Tag Library is useful in performing database queries. It allows easy access to query results.
- The database statements, such as insert, update, and delete can be performed by SQL tags.