JSTL Taglib

¡Comencemos!

¿Qué aprenderás?

- Conocer los JSTL
- Instalar librería JSTL
- Conocer las funcionalidades de JSTL

Introducción

Anteriormente, se estudiaron en contexto de introducción las funcionalidades básicas de la tecnología JSTL, su instalación y el uso básico de esta tecnología. En esta oportunidad vamos a trabajar con más detalle utilizando esta librería para aprender sus beneficios.

¿Qué son los JSTL?

Básicamente JSTL es la librería estándar de la tecnología *Java Server Page*. Cuenta con una variedad de tags que extienden la funcionalidad de un JSP permitiendo un mayor orden y más claridad en la creación de sitios web con *jsp*.

Utilizando JSTL los *jsp* subirán de nivel, aumentando la legibilidad y la mantenibilidad del código al mismo tiempo que permite al desarrollador enfocarse en las mejores prácticas de programación.

En el capítulo anterior se estudiaron los scriptlets que permiten insertar lógica java en un mismo jsp y como se pudo ver aún siendo una muy buena manera de generar código ordenado y mantenible, aun así mantiene cierta dependencia y mezcla de responsabilidades en el archivo. Con JSTL el código se reduce drásticamente y es mucho más claro determinar dónde termina una capa y comienza otra.

Instalación de la librería JSTL

La librería JSTL está compuesta por una serie de clases empaquetadas en forma de jar. Para poder utilizarla es necesario instalar el JAR en el classpath y además importar las librerías a la carpeta *lib* del proyecto. Al final del procedimiento, la estructura de carpetas debe lucir como la imagen siguiente:

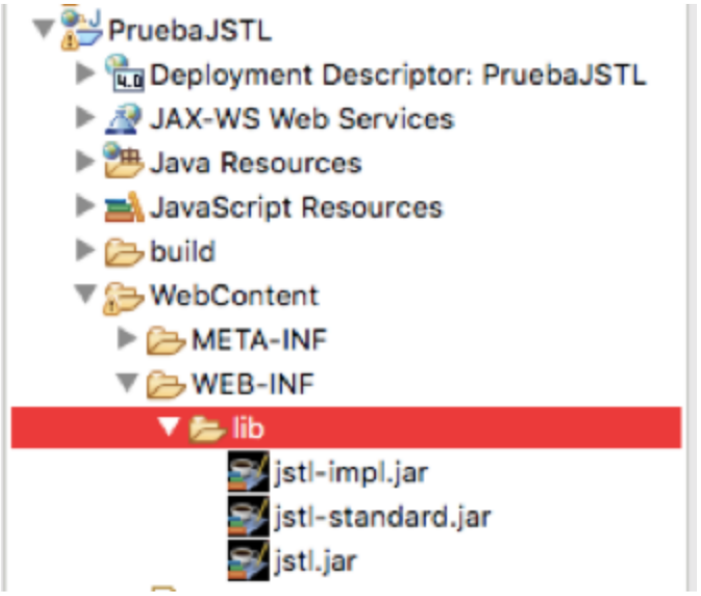

Imagen 1. Librerías JSLT.
Fuente: Desafío Latam

Para agregar la librería al ClassPath del proyecto, primero debes ir a la opción *Build Path* y luego a *Configure Build Path*.

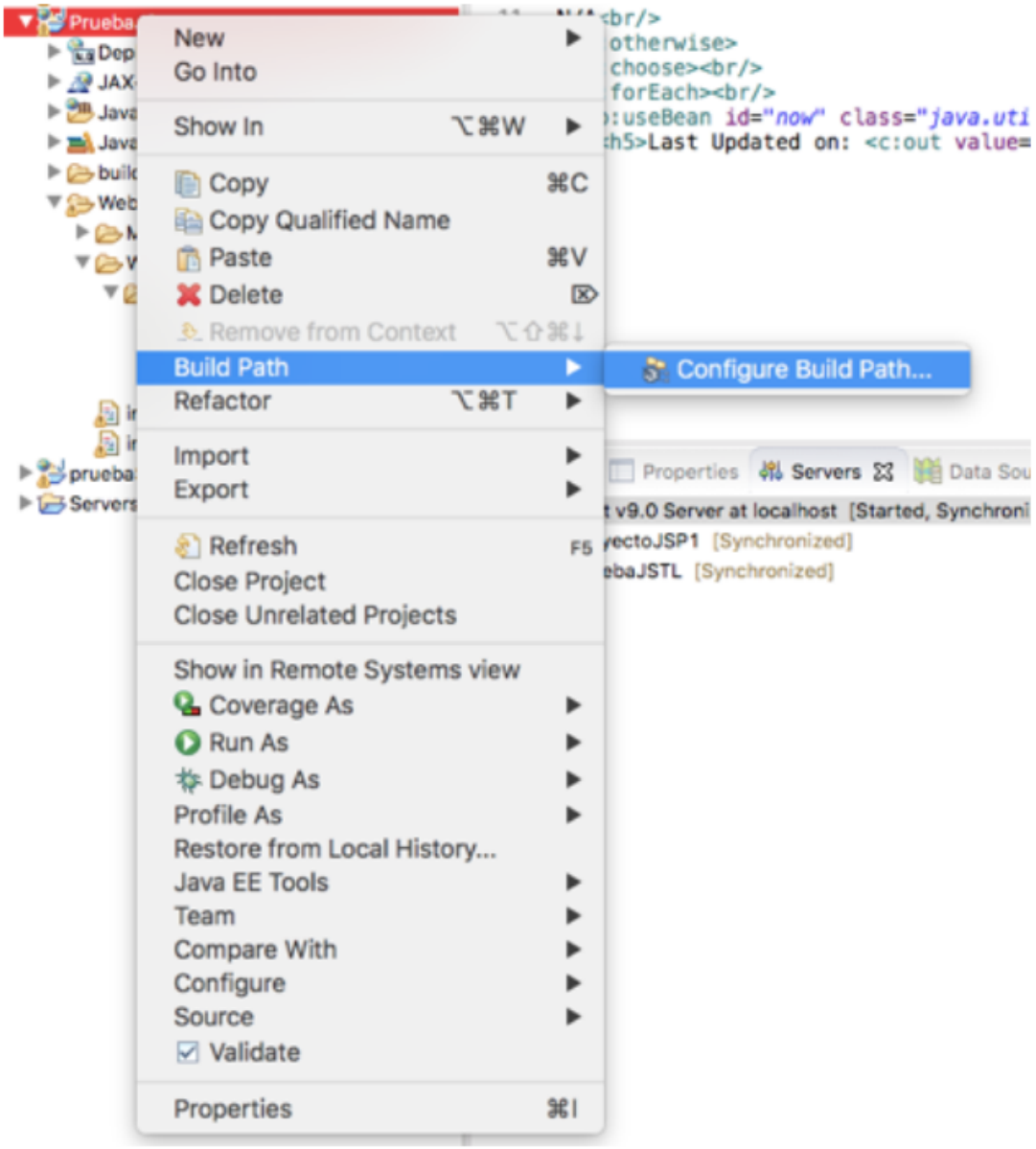

Imagen 2. Configure build path.
Fuente: Desafío Latam

En la pantalla siguiente, debes seleccionar la opción *Add External Jars*. Luego, debes ubicar la carpeta de instalación de apache tomcat (está instalada en tu equipo, dentro de la carpeta tomcat) y dentro de la carpeta lib, agregar el jar de nombre servlet-api.jar. Este *jar* es necesario para poder utilizar JSTL.

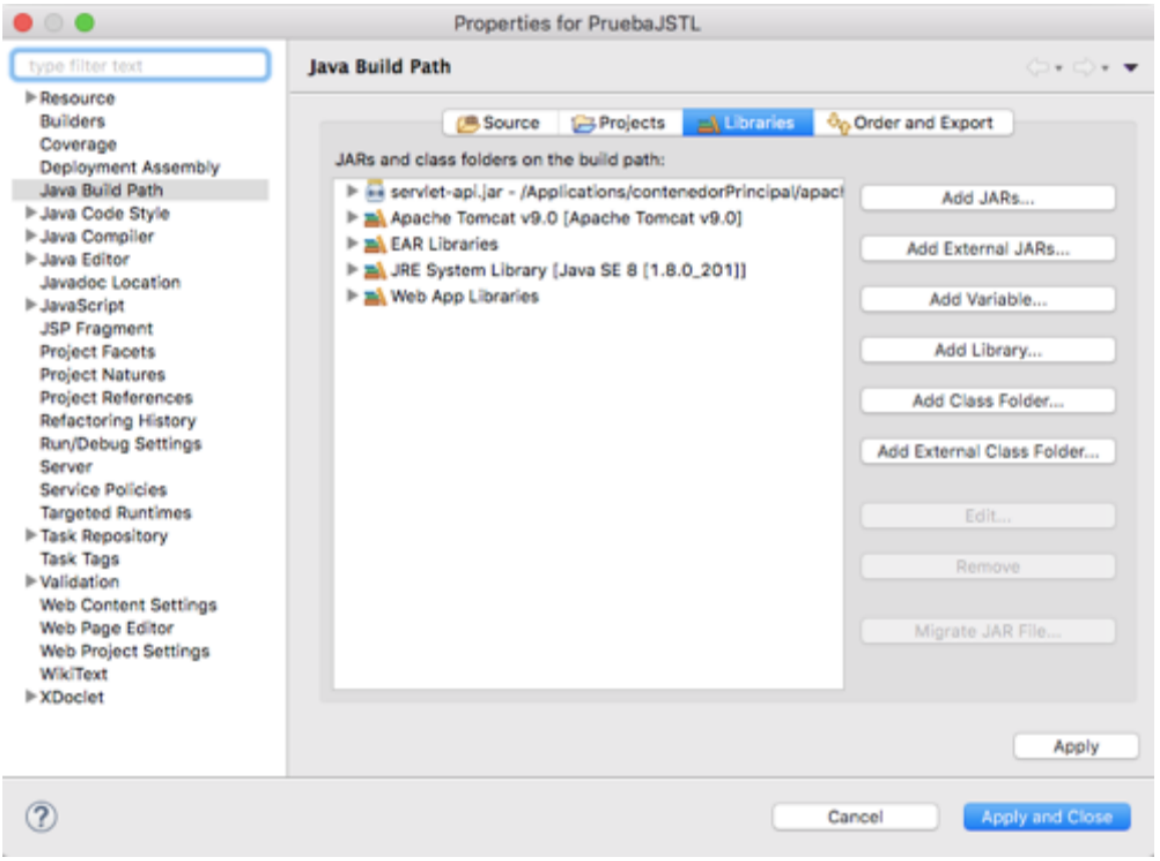

Imagen 3. Java Build Path.
Fuente: Desafío Latam

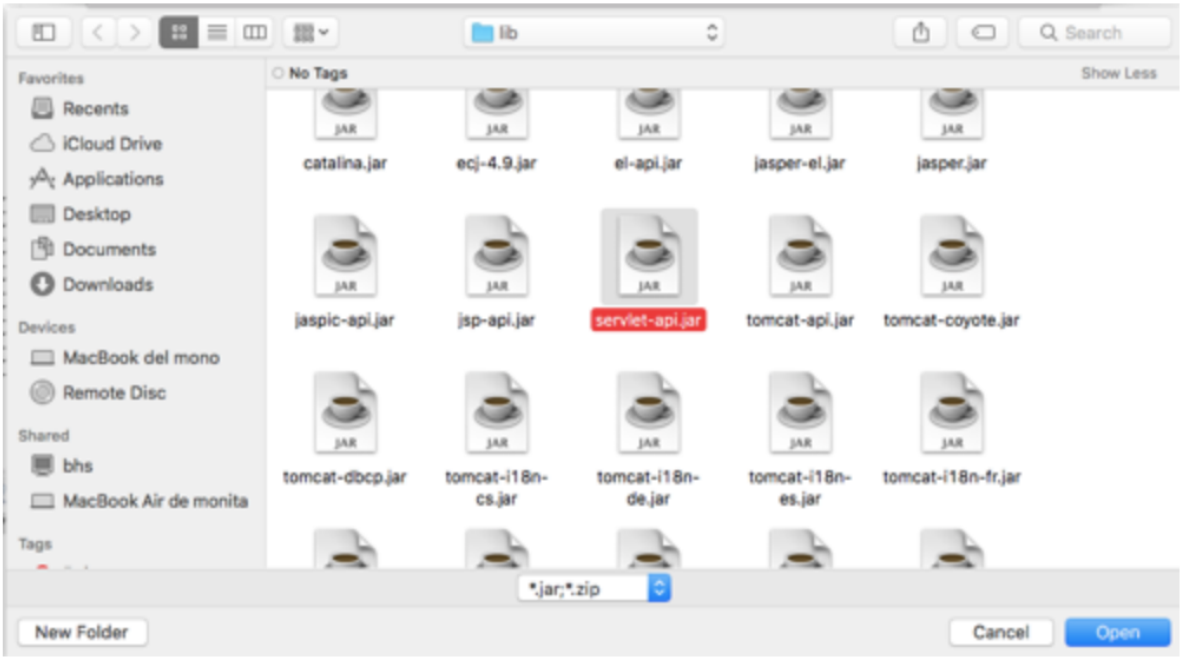

Imagen 4. Librería servlet-api.jar.
Fuente: Desafío Latam

Al seleccionar el jar, debe quedar agregado en la pantalla junto a las otras librerías.

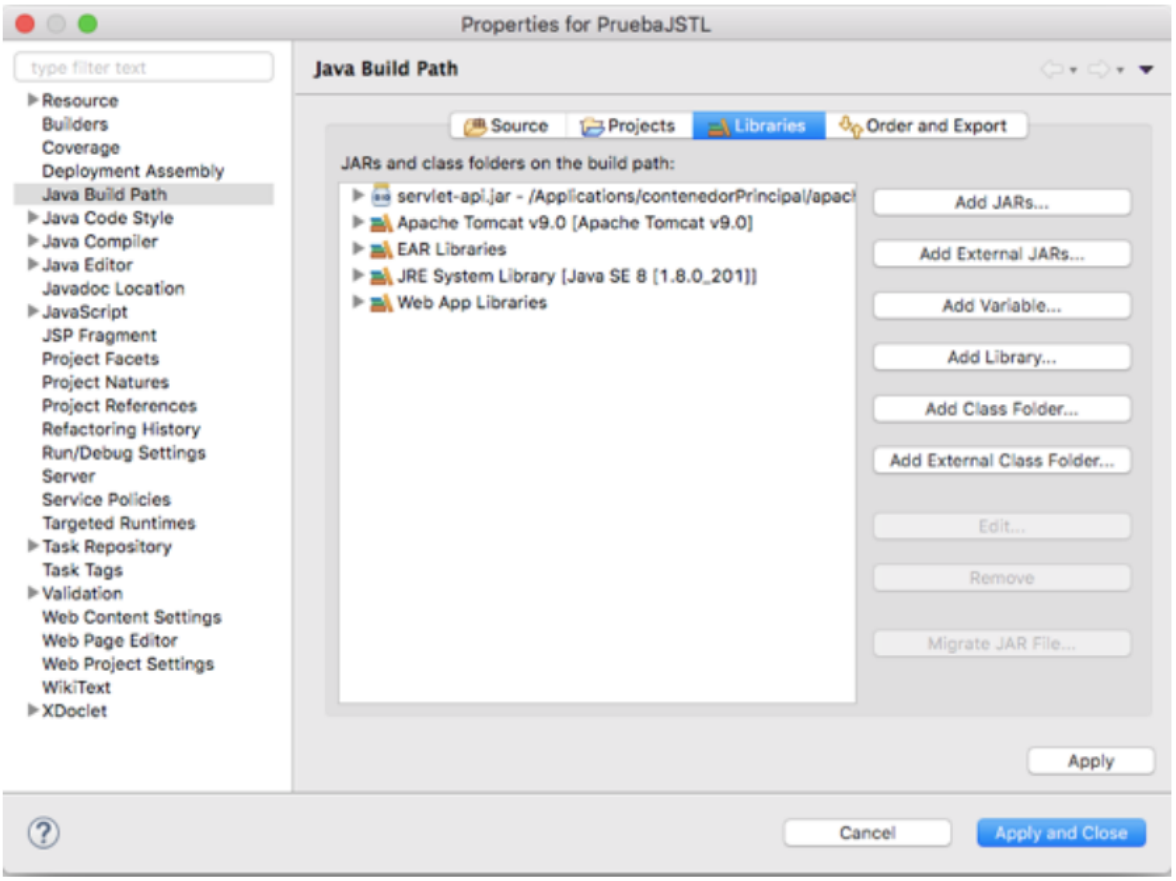

Imagen 5. Resultado después de importar la librería.
Fuente: Desafío Latam

Ahora, es tiempo de descargar las librerías necesarias. Para poder instalar los jar primero debemos crear una carpeta con un nombre, y dentro de ella debes alojar los siguientes jar:

Imagen 6. Librerías JSTL.
Fuente: Desafío Latam

Para descargarlos, dirígete a la dirección:

http://www.java2s.com/Code/Jar/j/Downloadjstlstandardjar.htm

Dentro de la página, busca cada uno de los jar mostrados anteriormente.

Guardas los 3 jar en una carpeta, para luego volver a eclipse y agregarlos al ClassPath según se vio un poco más arriba.

Después de instalar las librerías en el torneo, debes agregar cada jar dentro de la carpeta lib.

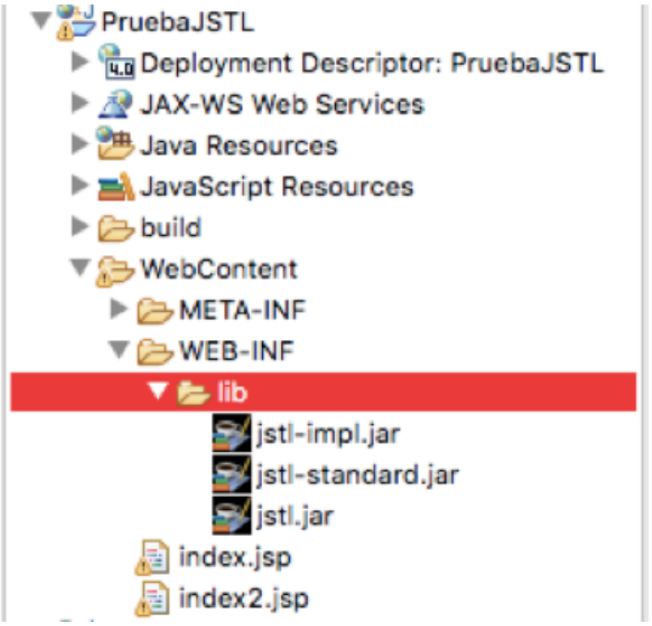

Imagen 7. librerías JSTL.
Fuente: Desafío Latam

Al tener los archivos tanto en el classPath y en la carpeta lib, ya es posible utilizar JSTL con total libertad.

Funcionalidades JSTL

La librería JSTL encapsula mucha de la funcionalidad de la tecnología JSP estándar a base de etiquetas (tags), que permiten aplicar funcionalidades con mayor facilidad y menos código que con java server page puro. Siguiendo por esta línea es posible definir a los JSTL como un set de funcionalidades comunes que están basadas en jsp 1.2 y servlets 2.3.

Comúnmente se define a JSTL como una librería individual, pero en realidad está compuesta por múltiples 'Tags Libraries', cada uno con sus características particulares y finalidades totalmente distintas pero a la vez conectadas en una sola misión, proveer una forma fácil de generar la vista de un sistema de información. Las librerías de tags jstl se dividen en:

- Core: http://java.sun.com/jsp/jstl/core
- XML: http://java.sun.com/jsp/jstl/xml
- SQL: http://java.sun.com/jsp/jstl/sql
- Internacionalización: http://java.sun.com/jsp/jstl/fmt
- Functions: http://java.sun.com/jsp/jstl/functions

Las distintas funcionalidades de estos TagLibs son agrupados en área, subfunción y un prefijo específico tal como se muestran a continuación:

Core:
Prefijo: **c**
Manejo de variables
Control de flujo
Administración de URL
Misceláneos

XMl:
Prefijo: **x**
Core
Control de flujo
Transformación

I18N (Internacionalización):
Prefijo: **fmt**
Local
Formateo de mensajes
Formateo de números y fechas

Database:
Prefijo: **sql**
SQL

Function:
Prefijo: **fn**
Collection
Manipulación de String

Para poder utilizar estas funcionalidades es necesario añadir a la página jsp la directiva correspondiente al tag que se desea utilizar. Por ejemplo si se desea trabajar con el tag del Core se debe instanciar con el siguiente código. En ejemplos posteriores se verá en acción.

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

Core JSTL

La librería CORE de jstl, provee acciones de propósito general que obtienen y setean variables de ámbito global en la aplicación, escriben valores mediante el *JspWriter* (pintan en el *html* valores procesados por *java*) y manejan excepciones de captura. También es posible trabajar con sentencias condicionales, iterativas y además trabajar con el manejo de las URL de respuesta y solicitudes.

Las funcionalidades que componen el TagLib Core son las siguientes:

Soporte a variables:
tags: remove,set

Control de flujo:
tags: choose,when, otherwise,forEach, forTokens, if

Administración de url:
tags: import,param,redirect,url.

Misceláneos:
tags: catch, out

Escribir un mensaje en un jsp es tan simple como escribir el código siguiente utilizando el prefijo c:

```
    < c:out value="Hola, estoy usando jstl" />
```

Para demostrar el funcionamiento de este tag, se detalla un ejemplo utilizando el prefijo c correspondiente al core, para implementar un ciclo iterativo *forEach* de un arraylist en un jsp. Poner atención a la simplicidad del código, comparado con el uso de *scriptlet*.

Crear un proyecto *Dynamic Web Project* de nombre *Patrones_Diseño_Sesion1_ejemplo007* y un jsp de nombre *vista.jsp*.

Código de archivo vista.jsp

```
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
    <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="ISO-8859-1">
    <title>Uso de forEach con JSTL</title>
    </head>
    <body>
      <c:forEach var="elementos" items="${requestScope.listaDeCompras}">
            <c:out value="${elementos}" />
      </c:forEach>
    </body>
    </html>
```

Se puede apreciar la directiva que importa el prefijo c, y dentro del body se puede ver como el prefijo es capaz de usar el forEach haciendo referencia al requestScope.listaDeCompras que es un arrayList creado en un servlet, el cual se expone a continuación.

Código de archivo ListadoGenericoServlet.java

```
    /**
     * Servlet implementation class ListadoGenericoServlet
     */
    @WebServlet("/ListadoGenericoServlet")
    public class ListadoGenericoServlet extends HttpServlet {
      private static final long serialVersionUID = 1L;
      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
            List<String> listaDeCompras = new ArrayList<String>();
            listaDeCompras.add("Manzanas");
            listaDeCompras.add("Peras");
            listaDeCompras.add("Uvas");
            listaDeCompras.add("Mandarinas");
            listaDeCompras.add("Platanos");
            listaDeCompras.add("Lechugas");
            listaDeCompras.add("Mangos");

            request.setAttribute("listaDeCompras", listaDeCompras);
            RequestDispatcher reenviador =
request.getRequestDispatcher("vista.jsp");
            reenviador.forward(request, response);
      }
    }
```

Al ejecutar el ejemplo, solamente se despliega por pantalla los elementos del arrayList, utilizando el poder de jstl. Como ejecutamos un servlet, no olvidar de ejecutar directamente el archivo ListadoGenericoServlet.java para que el ejercicio funcione.

XML

La librería de XML otorga las herramientas básicas necesarias para manipular este tipo de documentos utilizando características que permiten trabajar con el análisis de sintaxis y escritura de documentos xml, manejo del control de flujo y transformaciones. Para ilustrar esta librería se implementa una funcionalidad básica de lectura y despliegue de datos desde un archivo .xml a la página jsp. Para poder utilizar esta librería primero que nada hay que importarlas mediante los tags:

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

TAG XML	Descripción
x:out	Tag similar a <%= %>, pero para xml.
x:parse	Se utiliza para analizar los datos xml especificados en el cuerpo de la etiqueta o en un atributo.
x:choose	Tag condicional.
x:when	Subtag en caso de que la primera condición no se cumpla.
x:otherwise	Sub subtag en caso de que when no se cumpla.
x:if	Tag condicional.
x:transform	Transforma xml a xls.
x:params	Se utiliza junto con la etiqueta de transformación para configurar el parámetro en la hoja de estilo XSLT.
x:set	Usado para setear valores.

Tabla 1. Lista de tags disponibles para el módulo xml.
Fuente: Desafío Latam

Ejercicio guiado

Se implementará un proyecto de ejemplo que usará los tags de xml para poder leer el documento y mostrarlos en un JSP, llamado *HolaMundoJstlXml*

Crear un proyecto *Dynamic Web Project* de nombre *Patrones_Diseño_Sesion1_ejemplo008* y un jsp de nombre *index.jsp* y un archivo *autores.xml*.

Código de archivo autores.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<autores>
  <libros>
        <autor>William Faulkner</autor>
        <titulo>El sonido y la furia</titulo>
        <paginas>300</paginas>
  </libros>
  <libros>
        <autor>Oscar Wilde</autor>
        <titulo>De profundis</titulo>
        <paginas>200</paginas>
  </libros>
  <libros>
        <autor>Franz Kafka</autor>
        <titulo>La Metamorfosi</titulo>
        <paginas>394</paginas>
  </libros>
  <libros>
        <autor>James Joyce</autor>
        <titulo>Eveline</titulo>
        <paginas>459</paginas>
  </libros>
</autores>
```

El código anterior contiene los datos de autores de libros, los cuales cuentan con el autor, el título y las páginas del libro.

Un archivo *xml* en palabras simples es un tipo de documento *universal* que permite la transferencia de datos entre sistemas, independiente del lenguaje de programación que implementen. Se dice que es universal ya que todos los sistemas son capaces de leer archivos en este formato, ya sean sistemas hechos en java, asp.net, php, etc.

Pensemos que un sistema hecho en php nos envía un archivo xml con datos de libros a nuestro proyecto jsp, y nosotros lo leemos mediante la librería JSTL.

Ahora se implementa un archivo jsp que tendrá por misión capturar los datos del xml de autores y mostrarlos en una tabla al usuario.

Código de archivo index.jsp

```
    <body>
      <c:import var="lib" url="autores.xml" />
      <x:parse var="doc" xml='${lib}' />
      <div class="container">
            <h2>Estos elementos son leidos desde un archivo XML con
jstl</h2>
            <table class="table table-dark">
                  <thead>
                        <tr>
                              <th scope="col">Titulo</th>
                              <th scope="col">Autor</th>
                              <th scope="col">Paginas</th>
                        </tr>
                  </thead>
                  <tbody>
                  <x:forEach var="ob" select="$doc/autores/libros">
                        <tr>
                              <td><x:out select="$ob/titulo" /></td>
                              <td><x:out select="$ob/autor" /></td>
                              <td><x:out select="$ob/paginas" /></td>
                        </tr>
                  </x:forEach>
                  </tbody>
            </table>
      </div>
    </body>
```

El código anterior crea la página JSP que lee los datos del xml mediante los tags del módulo xml de jstl.

- **<c:import var="lib" url="autores.xml" />**: Hace referencia al archivo xml del cual se quiere obtener datos, el cual es el xml expuesto anteriormente nombre autores.xml.

- **<x:parse var="doc" xml='${lib}' />**: Asigna a la variable doc la referencia del documento autores.xml que anteriormente fue guardado en la variable 'lib'.

- **<x:forEach var="ob" select="$doc/autores/libros">**: Se ve la implementación jstl de un ciclo de iteración foreach el cual obtiene el documento xml y lo asigna a la variable ob. Esta variable ob es la que mantiene en memoria todos los elementos del xml y el foreach los recorre, asignando en cada vuelta el valor correspondiente y desplegando gracias al tag x:out.

- Como el foreach está encerrando a las etiquetas de tablas **<tr>**, por cada vuelta del ciclo se creará una fila html y en ella imprimirá el valor rescatado del xml, formando la tabla de forma automática.

Como resultado de esta implementación, el jsp genera la tabla y despliega los datos que se recibieron desde el archivo xml.

Imagen 8. index.jsp.
Fuente: Desafío Latam

SQL

La librería sql provee la capacidad para interactuar con bases de datos. Incluye el manejo de *dataSources*, querys, updates y transacciones. Usando las etiquetas SQL junto a las capacidades del core que permiten hacer ciclos permiten fácilmente manipular datos rescatados de una base de datos. Se muestra un ejemplo de código de manipulación de datos.

Crear un proyecto Dynamic Web Project de nombre *Patrones_Diseño_Sesion1_ejemplo009* y un jsp de nombre index.jsp.

```
<sql:query var="listaLibros" dataSource="${datasource}">
SELECT * FROM libros WHERE title = 'JSTL' ORDER BY autor
</sql:query>
<table>
<c:forEach var="libro" items="${listaLibros.row}">
<tr>
<td><c:out value="${libro.titulo}" />
</td><td><c:out value="${libro.autor}" />
</td></tr></c:forEach></table>
```

El código anterior se compone de:

- En la etiqueta sql se dispone la sentencia que manipulará la base de datos, ya sea obteniendo los valores o en su defecto modificando.
- La query SQL que se muestra es un simple select a una tabla de nombre libros.

A continuación, se genera una tabla común y corriente, pero utilizando etiquetas jstl se genera un ciclo con la sentencia foreach, la cual recorre una lista de nombre *listaLibros* y asigna cada valor a la variable libro. Al finalizar solo es necesario imprimir por pantalla cada libro mediante la definición libro.título y libro.autor.

Para una demostración más palpable de su uso implementemos un ejemplo. Primero utilizaremos las tablas de departamento generadas del siguiente script en su *sql developer*:

```
    CREATE TABLE DEPARTAMENTO (NUMDEPTO NUMBER PRIMARY KEY NOT NULL,
NOMDEPTO VARCHAR(100), UBICACIONDPTO VARCHAR(100));

    CREATE TABLE EMPLEADO (NUMEMPLEADO NUMBER PRIMARY KEY, NOMBRE
VARCHAR(100), NUMDEPTO INTEGER,
    FOREIGN KEY (NUMDEPTO) REFERENCES DEPARTAMENTO (NUMDEPTO) );

    INSERT INTO DEPARTAMENTO VALUES(11,'informatica', 'Chile');
    INSERT INTO DEPARTAMENTO VALUES(12,'contabilidad', 'Chile');
    INSERT INTO DEPARTAMENTO VALUES(13,'mecanica', 'Chile');
    INSERT INTO DEPARTAMENTO VALUES(14,'plomeria', 'Chile');

    INSERT INTO EMPLEADO VALUES (0101, 'Matias Zarate', 11);
```

A continuación se genera el index.jsp con el siguiente código.

```
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
    <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
    <html>
      <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
        <title>Prueba SQL</title>
      </head>
      <body>
        <h1>Demostración del uso del tag sql:</h1>
        <h2>Registros correspondientes a la tabla departamentos</h2>
        <sql:setDataSource var="myDS"
driver="oracle.jdbc.driver.OracleDriver"
          url="jdbc:oracle:thin:@//localhost:1521/xe"
          user="Empresa_DB"  password="empresa"/>

        <sql:query dataSource="${myDS}" var="result">
          select * from DEPARTAMENTO
```

```
        </sql:query>
        <table border="1">
          <c:forEach var="row" items="${result.rows}">
            <tr>
              <td><c:out value="${row.NUMDEPTO}"/></td>
              <td><c:out value="${row.NOMDEPTO}"/></td>
              <td><c:out value="${row.UBICACIONDPTO}"/></td>
            </tr>
          </c:forEach>
       </table>
      </body>
    </html>
```

Notar que en la línea *sql:setDataSource* se define la variable datasource, el driver correspondiente a oracle, la url de conexión, el nombre del usuario de base de datos y su password. Si los datos están correctos se debería desplegar en el navegador los registros de la tabla departamento.

Internacionalización

El prefijo *ftm* tiene como objetivo dar funcionalidad de formateo de mensajes, formateo de números y fechas e internacionalización.

Para utilizar internacionalización basta con importar las librerías correspondientes:

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

Este tag tiene múltiples funciones, las cuales examinaremos a continuación.

Uso de *tag fmt: parseNumber:* Este tag es usado para generar la transformación de números en formato String a monedas, porcentajes o números. Coloquialmente esto se conoce como casteo de tipos de datos.

Crear un proyecto *Dynamic Web Project* de nombre *Patrones_Diseño_Sesion1_ejemplo010* y un jsp de nombre t*ransformadorMonedas.jsp*.

```
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

    <html>
      <head>
     <title>Demostración parse number tag</title>
      </head>
      <body>
    <h3>Uso del tag ftm para conversión de monedas:</h3>

    <c:set var="Cantidad" value="789.154" />

        <fmt:parseNumber var="j" type="number" value="${Cantidad}" />
        <p><i>La cantidad es:</i>  <c:out value="${j}" /></p>

        <fmt:parseNumber var="j" integerOnly="true" type="number"
value="${Cantidad}" />
        <p><i>La cantidad es:</i>  <c:out value="${j}" /></p>
      </body>
    </html>
```

El código anterior se encarga de manejar el cómo se despliegan los valores numéricos en cada etiqueta, generando tanto valores enteros como decimales.

Ahora, pensemos en un sistema web que pueda ser visualizado en inglés, chino y alemán. Cada vez que escribas una palabra, por ejemplo 'hola' tiene que tener su traducción *hello*. A primera vista esta técnica es muy poco eficiente, además de complicada de implementar, por lo cual se recurre a los archivos de propiedades (.properties) que contendrán las palabras de algún idioma que se configure.

Crear un proyecto Dynamic Web Project de nombre *Patrones_Diseño_Sesion1_ejemplo011* y un jsp de nombre bienvenido.jsp

Luego en el JSP tenemos que hacer referencia a la ubicación de estos archivos de propiedades mediante:

```
    fmt:setBundle basename="bundle.fichero" />
```

Para luego desde la página utilizando fmt:message trabajar con los distintos textos traducidos desde el archivo.properties.

```
    <p>Idioma del navegador: <fmt:message key="idioma"/></p>
```

En donde el idioma es la clave que se generó en los archivos .properties que contienen las traducciones.

Para realizar este ejercicio, generamos un proyecto *Dynamic Web Project*, y se crea una *pagina.jsp* de nombre *bienvenido.jsp* con el siguiente código:

```
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
    <%@ page isELIgnored="false" %>

    <fmt:setBundle basename="messages" />

    <html>
    <head>
        <title>Distintos Idiomas</title>
    </head>
    <body>
        <h2>
            <fmt:message key="label.welcome" />
        </h2>
    </body>
    </html>
```

Con la página *jsp* implementada, vamos a crear tres archivos .properties en la carpeta resources (si no existe, crearla en Java Resources/resources):

- messages.properties: Idioma Ingles
- messages_de.properties: Idioma Alemán
- messages_zh.properties: Idioma Chino

Cada uno de estos archivos son propiedades que el servidor podrá detectar para así aplicar su configuración. El contenido de cada uno de estos archivos es:

messages.properties

```
label.welcome = Welcome
```

messages_de.properties

```
label.welcome = Hallo
```

messages_zh.properties

```
label.welcome = nijau
```

Por ejemplo, el archivo *messages_de.properties* tiene la propiedad *label.welcome = hallo*, la cual dependiendo de la configuración regional o de la llamada explícita que haremos en el jsp desplegará tal saludo en ese idioma. Volvamos al código anterior y agregamos la instrucción para llamar a alguno de estos archivos .properties.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page isELIgnored="false" %>

<fmt:setLocale value="zh"/>
<fmt:setBundle basename="messages" />

<html>
<head>
    <title>Distintos Idiomas</title>
</head>
<body>
    <h2>
        <fmt:message key="label.welcome" />
    </h2>
</body>
</html>
```

La nueva instrucción es: **<fmt:setLocale value="de"/>** que hace referencia a 'de' que corresponde al prefijo del archivo *messages_de.properties*. Si se ejecuta el programa se desplegará en el navegador el mensaje correspondiente al idioma Alemán 'Hallo'.