DESARROLLA
SOFTWARE

www.desarrollasoftware.com

JAVA WEB DEVELOPER

Eric Gustavo Coronel Castillo
gcoronelc.blogspot.com
gcoronelc@Gmail.com

DESARROLLA

SOFTWARE

www.desarrollasoftware.com

TEMA: JSTL

Índice

- Objetivo
- Introducción
- Expression Language (EL)
- Funciones
- Etiquetas Core

Objetivo

Utilizar JSTL en la construcción de aplicaciones web utilizando tecnología Java.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<html>
   <head>
      <title>Demo 01 – demo01.jsp</title>
   </head>
   <body>
      <c:forEach items="${param}" var="item">
         <p>${item}</p>
      </c:forEach>
   </body>
</html>
```

Introducción

¿Qué es JSTL?

JSTL es un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal que es usada comúnmente para escribir páginas JSP.

```
<body>
   <c:set var="mensaje" value="${requestScope.mensaje}" />
   <c:set var="empleadoTO" value="${requestScope.empleadoTO}" />
   <c:out value="${mensaje}"/>
   <c:if test="${empleadoTO != null}">
      <p>Nombre: ${empleadoTO.nombre}</p>
      <p>Paterno: ${empleadoTO.paterno}</p>
      <p>Materno: ${empleadoTO.materno}</p>
   </c:if>
   <c:if test="${empleadoTO == null}">
      <p>No existe el usuario.</p>
   </c:if>
</body>
```

¿Cuál es el problema con los scriptlets JSP?

- El código Java embebido en scriptlets es desordenado.
- -El código Java dentro de scriptlets JSP no pueden ser reutilizados por otros JSP, por lo tanto la lógica común termina siendo re-implementada en múltiples páginas.
- La recuperación de objetos fuera del HTTP request y session es complicada. Es necesario hacer el Casting de objetos y esto ocasiona que tengamos que importar más Clases en los JSP.

Introducción

¿Como mejoran esta situación la librería JSTL?

- Debido a que las etiquetas JSTL son XML, estas etiquetas se integran limpia y uniformemente a las etiquetas HTML.
- JSTL incluyen la mayoría de funcionalidad que será necesaria en una página JSP.
- Las etiquetas JSTL pueden referenciar objetos que se encuentren en el alcance **request** y **session** sin conocer el tipo del objeto y sin necesidad de hacer el **Casting**.
- Los JSP Expression Language (EL) facilitan las llamadas a los métodos **Get** y **Set** en los objetos Java. EL es usado extensamente en la librería JSTL.

Las Librerías JSTL

Descripción	Prefijo	URI por Defecto
Core	c	http://java.sun.com/jsp/jstl/core
XML Processing	x	http://java.sun.com/jsp/jstl/xml
I18N	fmt	http://java.sun.com/jsp/jstl/fmt
Database Access	sql	http://java.sun.com/jsp/jstl/sql
Function	fn	http://java.sun.com/jsp/jstl/functions

- **Instalación de JSTL**

  – Si estamos utilizando NetBeans con GlassFish solo debemos importar las librerías, por ejemplo, si la librería que queremos utilizar es el **core** la instrucción para importarla es:

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

  – Pero si vamos a utilizar un contenedor JEE que no implementa por defecto JSTL tendríamos 3 alternativas, deberíamos optar por una de ellas:

1. Habilitar en el IDE las librerías JSTL.
2. Implementar manualmente JSTL en nuestro proyecto web.
3. Instalar en el contenedor JEE las librerías JSTL.

Expression Language (EL)

Introducción

- Además de las librerías de etiquetas, JSTL define un lenguaje de expresiones, que facilita enormemente el tratamiento de información. JSP usa Java para referenciar atributos dinámicos. Con JSTL ya no es necesario.

- Comparemos por ejemplo la lectura de un parámetro:

  - Con JSP

    ```
    <%= request.getParameter("nombre") %>
    ```

  - Con JSTL

    ```
    ${param.nombre}
    ```

Introducción

- Para acceder al campo de un bean, o a un elemento de una colección (array, o Map), se usa el operador punto, o el operador corchete:

```
${bean.propiedad}
${map.elemento}
${header['User-Agent']}
```

Si un campo de un bean es otro bean, podemos encadenar puntos para acceder a una propiedad del segundo bean:

```
${bean1.bean2.propiedad}
```

- Las expresiones pueden aparecer como parte del valor de un atributo de una etiqueta:

```
<c:out value="${15*4}"/>
<c:out value="${nombre}"/>
<c:if test="${tabla.indice % 2 == 0}">es par</c:if>
```

O independientemente junto a texto estático como el HTML:

```
<input type="text" name="usuario" value="${requestScope.empleado.nombre}"/>
```

Expression Language (EL)

Operadores

- Además de los operadores punto (.) y [ ], EL provee los siguientes operadores:
  - **Aritméticos:** +, - (binario), *, / y div, % y mod, - (unario)
  - **Lógicos:** and, &&, or, ||, not, !
  - **Relacionales:** ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le.
    Las comparaciones se pueden hacer contra otros valores, o contra literales de tipo boolean, string, integer, o de punto flotante
  - **Empty:** El operador empty puede ser usado para determinar si un valor es nulo o vacío.
  - **Condicional:** A ? B : C.
    Evalúa B o C, dependiendo del resultado de la evaluación de A.

Palabras Reservadas

and	eq	gt	true
Instanceof	or	ne	le
false	empty	not	lt
ge	null	div	mod

Expression Language (EL)

Objetos implícitos

Objetos implícitos	contiene
pageScope	Variables de ámbito página.
requestScope	Variables de ámbito **request**.
sessionScope	Variables de ámbito **session**.
applicationScope	Variables de ámbito **application**.
param	Parámetros del request como cadenas.
paramValues	Parámetros del request como arreglo de cadenas.
header	Cabeceras del request HTTP como cadenas.
headerValues	Cabeceras del request HTTP como arreglo de cadenas.
cookie	Valores de las cookies recibidas en el request.
initParam	Parámetros de inicialización de la aplicación Web.
pageContext	El objeto **PageContext** de la página actual.

Expression Language (EL)

Objetos implícitos

- Acceso a Todos los Elementos

```
<c:forEach items="${header}" var="item">
          <c:out value="${item}"/><br/>
</c:forEach>
```

- Acceso por Nombre Utilizando Punto: objeto.propiedad

```
Host: <c:out value="${header.Host}"/> <br/>
```

- Acceso por Nombre con Corchetes:

```
objeto['propiedad']
```

- Usar este formato evita que se interprete el nombre como una expresión. Por ejemplo: queremos la propiedad "Accept-Language" no una resta entre las variables Accept y Language.

```
Host: <c:out value="${header['Host']}"/> <br/>
Accept-Language: <c:out value="${header['Accept-Language']}"/> <br/>
```

Expression Language (EL)

- Objeto: pageContext

Expresión	Descripción	Ejemplo
${pageContext.exception.message}	Devuelve una descripción del error cuando la página en curso es una página de error JSP.	"Algo ha ido mal"
${pageContext.errorData}	Información de un error JSP.	
${pageContext.request.authType}	Tipo de autentificación usado en la página.	BASIC
${pageContext.request.remoteUser}	Identificador del usuario (cuando esta en uso la autentificación del contenedor).	gustavo
${pageContext.request.contextPath}	Nombre (también llamado contexto) de la aplicación.	/cap18
${pageContext.request.cookies}	Array de cookies.	
${pageContext.request.method}	Método HTTP usado para acceder a la página.	GET
${pageContext.request.queryString}	Query de la página (el texto de la URL que viene después del PATH)	p1=valor&p2=valor
${pageContext.request.requestURL}	URL usada para acceder a la página.	http://localhost/app/pagecontext.jsp
${pageContext.session.new}	Contiene **true** si la sesión es nueva, **false** si no lo es.	true
${pageContext.servletContext.serverInfo}	Información sobre el contenedor JSP.	Sun Java System Application Server 9.1_02

Expression Language (EL)

- Objeto: pageContext.errorData
  - Cuando se produce un error la excepción ocurrida se adjunta al contexto de la página, información adicional en forma de bean con nombre errorData. Este bean tiene las siguientes propiedades:

Propiedad	Tipo Java	Descripción
requestURI	String	URI de la petición fallida.
servletName	String	Nombre de la página o servlet que lanzó la excepción.
statusCode	int	Código HTTP del fallo.
throwable	Throwable	La excepción que produjo el fallo.

Expression Language (EL)

- **Objeto: pageContext.errorData**
  - Para declarar una página de error JSP que responda a un código o excepción, podemos usar el web.xml:

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type>
   <location>/errorPage.jsp</location>
</error-page>
```

  - O una declaración para una página concreta:

```
<%@page errorPage="errorPage.jsp" %>
```

  - En la página de error debemos incluir:

```
<%@page isErrorPage="true" %>
```

Expression Language (EL)

- Objeto: pageContext.errorData
  - Un ejemplo de la codificación de la página de error:

```
<body>
   <p>URI del requerimiento fallido:
      ${pageContext.errorData.requestURI}</p>
   <p>Quien lanzo el error:
      ${pageContext.errorData.servletName}</p>
   <p>Código de error: ${pageContext.errorData.statusCode}</p>
   <p>Excepción: ${pageContext.errorData.throwable.class}</p>
   <p>Mensaje: ${pageContext.errorData.throwable.message}</p>
</body>
```

Funciones

- Las funciones representan un modo de extender la funcionalidad del lenguaje de expresiones. Se usan dentro de cualquier expresión EL, ejemplo:

  Mi nombre de usuario tiene ${fn:length(username)} letras.

- A continuación tenemos la lista de funciones disponibles:

  - contains(string, substring) -> boolean
  - containsIgnoreCase(string, substring) -> boolean
  - endsWith(string, suffix) -> boolean
  - escapeXml(string) -> String
  - indexOf(string, substring) -> int
  - join(collection, separator) -> String
  - length(collection) -> int
  - replace(inputString, beforeSubstring, afterSubstring) -> String
  - split(string, separator) -> String[]
  - startsWith(string, prefix) -> boolean
  - substring(string, beginIndex, endIndex) -> String
  - substringAfter(string, substring) -> String
  - substringBefore(string, substring) -> String
  - toLowerCase(string) -> String
  - toUpperCase(string) -> String
  - trim(string) -> String

Etiquetas Core

- c:out
  - Muestra el resultado de una expresión. Su funcionalidad es equivalente a la de <%= %>.

Atributo	Descripción	Requerido	Por defecto
value	información a mostrar	sí	ninguno
default	información a mostrar por defecto	no	cuerpo
escapeXml	true si debe convertir caracteres especiales a sus correspondientes entidades (por ejemplo, > para <).	no	true

Etiquetas Core

- c:set

Atributo	Descripción	Requerido	Por defecto
value	Información a grabar.	no	cuerpo
target	Nombre de un bean cuya propiedad será modificada	no	ninguno
property	Propiedad a modificar	no	ninguna
var	Nombre de la variable en la que guardar.	no	ninguno
scope	Ámbito de la variable en la que grabar la información (page, request, session, o application)	no	page

- c:remove

Atributo	Descripción	Requerido	Por defecto
var	Nombre de la variable a quitar	sí	--
scope	Ámbito de la variable a quitar.	no	todos los ámbitos

Etiquetas Core

- **c:if**

Atributo	Descripción	Requerido	Por defecto
test	Condición a evaluar. Solo procesa el cuerpo si es true.	sí	--
var	Nombre de la variable para la valor resultante de la evaluación de la condición. El tipo de la variable es booleano.	no	ninguno
scope	Ámbito de la variable.	no	page

  - En el cuerpo es posible colocar otras etiquetas, incluyendo otras **<c:if>**. Es útil guardar el resultado de evaluar la condición para evitar repetir los cálculos.

```
<c:if test="${cadena == null}">
    La cadena es nula o no definida.
</c:if>
```

Etiquetas Core

- **c:choose, c:when, c:otherwise**
  - <c:choose> no tiene atributos. Acepta como hijos uno o más <c:when>.
  - <c:when> tiene un único atributo:

Atributo	Descripción	Requerido	Por defecto
Test	Condición a evaluar.	sí	--

  - <c:otherwise> no tiene atributos.

Etiquetas Core

- c:choose, c:when, c:otherwise

```
<body>
   <c:set var="nota" value="${param.nota}"/>
   Nota:<c:out value="${nota}"/><br>
   Condición:
   <c:choose>
      <c:when test="${nota >= 14}">Aprobado</c:when>
      <c:when test="${nota >= 11}">Asistente</c:when>
      <c:when test="${nota >= 0}">Desaprobado</c:when>
      <c:otherwise>Nota Fuera de Rango</c:otherwise>
   </c:choose>
</body>
```

Etiquetas Core

- **c:forEach**

Sintaxis 1:

```
<c:forEach [var=”nombreVariable”] items=”colección”
   [varStatus=”variableEstado”]
   [begin=”inicio”] [end=”fin”] [step=”paso”]>

   Contenido del Cuerpo

</c:forEach>
```

Sintaxis 2:

```
<c:forEach [var=”nombreVariable”] [varStatus=”variableEstado”]
   begin=”inicio” end=”fin” [step=”paso”]>

   Contenido del Cuerpo

</c:forEach>
```

Etiquetas Core

- **c:forEach**

Atributo	Descripción	Requerido	Por defecto
items	Colección sobre la que se itera.	no	ninguno
begin	Elemento con el que empezar (0=primero).	no	0
end	Elemento con el que terminar (0=primero).	no	último
step	Procesa solo cada step elementos.	no	1 (todos)
var	Nombre del atributo con el que exponer el elemento actual.	no	ninguno
varStatus	Nombre de la variable con la que exponer el estado de la iteración.	no	ninguno

Etiquetas Core

- **c:forEach**
  - La variable **varStatus** tiene propiedades que describen el estado de la iteración:

Atributo	Tipo	Requerido
begin	número	El valor del atributo begin.
current	número	El elemento actual.
end	número	El valor del atributo end.
index	número	Índice del elemento actual dentro de la colección.
count	número	Número de iteración (empezando en 1).
first	boolean	Indica si estamos en la primera iteración.
last	boolean	Indica si estamos en la última iteración.
step	número	El valor del atributo step.

Etiquetas Core

- **c:forEach**
  - Muestra el mensaje "Alianza Campeón" 10 veces.

```
<c:forEach var="n" begin="1" end="10">
   <p>${n} Alianza Campeón</p>
</c:forEach>
```

  - Imprime el nombre y apellido paterno de una lista de empleados.

```
<c:set var="mensaje" value="${requestScope.mensaje}" />
<c:set var="lista" value="${requestScope.lista}" />
<c:out value="${mensaje}"/><br>
<c:if test="${fn:length(lista) == 0}">
   <c:out value="No hay elementos."/><br>
</c:if>
<c:forEach items="${lista}" var="item">
   ${item.nombre} ${item.paterno}<br>
</c:forEach>
```

Etiquetas core

- **Etiqueta: param**

Sintaxis:

– El valor del parámetro se especifica en el atributo “value”.

```
<c:param name=”nombreParámetro” value=”valorParámetro”/>
```

– El valor del parámetro se especifica en el contenido del cuerpo.

```
<c:param name=”nombreParámetro”>
   valorParámetro
</c:param>
```

Etiquetas core

- **Etiqueta: redirect**

Sintaxis:

– **Sin contenido en el cuerpo.**

```
<c:redirect url=”recurso” [context=”contexto”]/>
```

– **Con contenido en el cuerpo para especificar el valor de los parámetros.**

```
<c:redirect url=”recurso” [context=”contexto”]>
    Etiquetas c:param
</c:redirect>
```

Etiquetas core

- **Etiqueta: url**

  Sintaxis:

  – **Sin contenido en el cuerpo.**

```
<c:url value=”valor” [context=”contexto”]
   [var=”nombreVariable”] [scope=”{page|request|session|application}”]/>
```

  – **Con contenido en el cuerpo para especificar los parámetros.**

```
<c:url value=”valor” [context=”contexto”]
   [var=”nombreVariable”] [scope=”{page|request|session|application}”]>

   Etiquetas <c:param>

</c:url>
```

Etiquetas core

- **Etiqueta: url**

A continuación tenemos un ejemplo ilustrativo:

```
<c:url value="demo05.jsp" var="destino">
   <c:param name="nota" value="15"/>
</c:url>
<a href="${destino}">Enlace</a>
```

Etiquetas core

- **Etiqueta: catch**

Sintaxis:

```
<c:catch [var=”nombreVariable”]>
  Acciones Anidadas.
</c:catch>
```

- **Etiqueta: catch**

A continuación tenemos un ejemplo ilustrativo.

```
<c:catch var="error1">
  <fmt:parseNumber var="total" value="ABCD"/>
  Importe: ${dato.total}<br>
</c:catch>
<c:if test="${not empty error1}">
  Lo sentimos, no existe el dato solicitado.<br>
  Error: ${error1}
</c:if>
```

Proyecto 1

Desarrollar un proyecto que permita generar la tabla de multiplicar de un número.

El usuario es quien ingresa el número del que quiere la tabla de multiplicar.

Proyecto 2

Desarrollar un proyecto que permita al usuario ingresar dos puntos del plano cartesiano.

La aplicación luego debe reportar lo siguiente:

- De cada punto debe reportar en que cuadrante se encuentra.
- La distancia entre los dos puntos.