	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

DESARROLLAR APLICACIONES WEB UTILIZANDO LA ARQUITECTURA MODELO 1

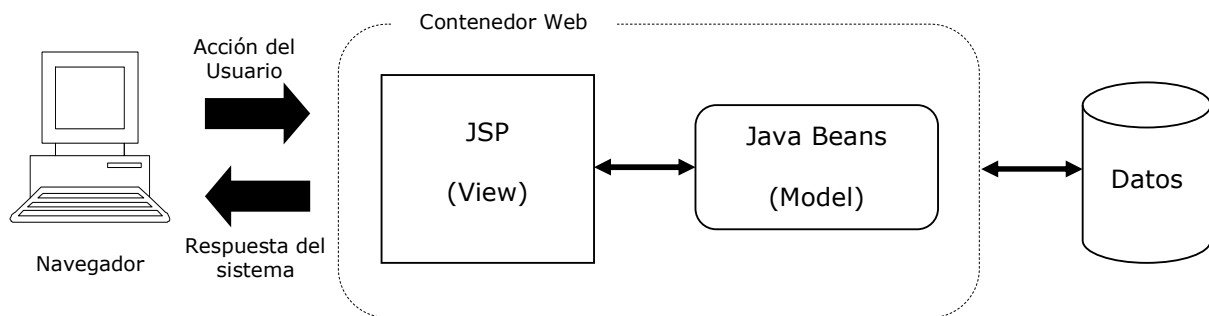
Observando cualquier aplicación Web debería reconocerse la presencia de algún esquema de navegación Web que dicta cómo fluyen las páginas junto con los datos que deben procesarse con ellas. Algunas aplicaciones emplean prácticas de Arquitectura Modelo 1 que codifican por hardware esos flujos de las páginas que componen las aplicaciones Web. Otros utilizan prácticas de Modelo 2 que incrustan los atributos de flujo en un archivo externo para que el mantenimiento y las rutas de navegación puedan manejarse fuera del propio código. Inevitablemente, durante la fase de diseño de un programa, se tendrá que hacer suposiciones sobre cuál de los métodos será puesto en práctica antes de desplegar el software. En la toma de decisión se podrían utilizar experiencias externas, como la experimentación y las lecciones aprendidas en trabajos anteriores, pero lo más probable es que los plazos de entrega, la madurez del personal y los plazos afecten a la decisión.

¿Qué es Modelo 1? ¿Por qué usarlo?

La Arquitectura Modelo 1 es un enfoque “centrado en páginas” en el que distintos componentes Web manejan, individualmente, los flujos de páginas. Esto significa que el procesamiento de solicitudes y respuestas se codifica por hardware en las páginas para acomodar la navegación del usuario en una aplicación Web.

Por supuesto, esto supone problemas de mantenimiento cuando se necesitan modificaciones lógicas para acomodar cambios de requisitos y necesidades del usuario final. Esos cambios obligan a los desarrolladores a hacer malabarisos por el código para garantizar que todos los flujos lógicos se manejan adecuadamente cuando los usuarios navegan por una aplicación Web. Junto con las responsabilidades propias del mantenimiento del flujo de navegación en los despliegues de Modelo 1, es necesario gestionar puntos relativos a la seguridad y al estado de la aplicación.

Los problemas a los que se enfrenta la Arquitectura Modelo 1 son difíciles decisiones de diseño que hacen frente al comienzo de un proyecto, pero las limitaciones de la experiencia en desarrollo de un equipo, el alcance de una aplicación y el tiempo de entrega pueden persuadir de adoptar esta filosofía de desarrollo. Adoptar la filosofía de Modelo 1 no es, necesariamente, una mala decisión, dependiendo de la estimación de lo que entregará un equipo, y como lo hará en un plazo de entrega apretado.

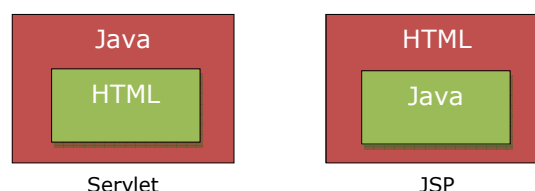



En resumen, los problemas principales de esta arquitectura son:

- (1) *Problemas de mantenimiento:* En general contiene varias páginas JSP que contienen numerosos scriptlets.
- (2) *Problemas de reusabilidad:* En general la lógica está embebida en las páginas JSP y torna complicado reusar la lógica de negocio (en general se corta y se pega).
- (3) *Problemas de seguridad:* Debido a que las páginas JSP son responsables por el manejo de todo el procesamiento, muy probablemente la página maneja funciones de logueo y lógica en forma embebida.

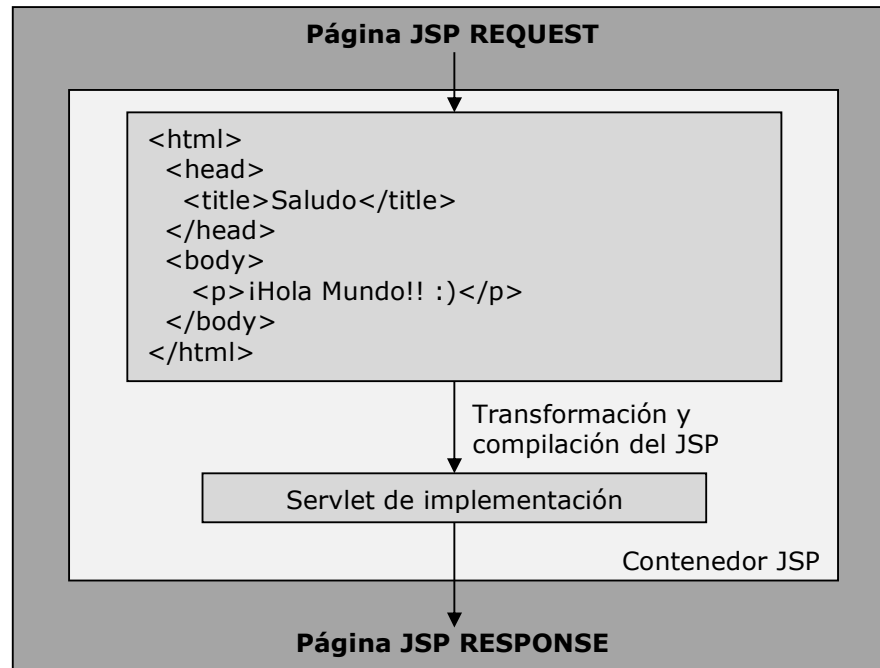
JSP (Java Server Page)

La tecnología JSP combina HTML, XML y Java Servlet, facilitando la creación de contenido dinámico. Si programáramos un Servlet y una página JSP que generaran el mismo código HTML, veríamos que en un Servlet el HTML es generado desde el código Java, mientras que en un JSP el código Java es embebido en el código HTML.



	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

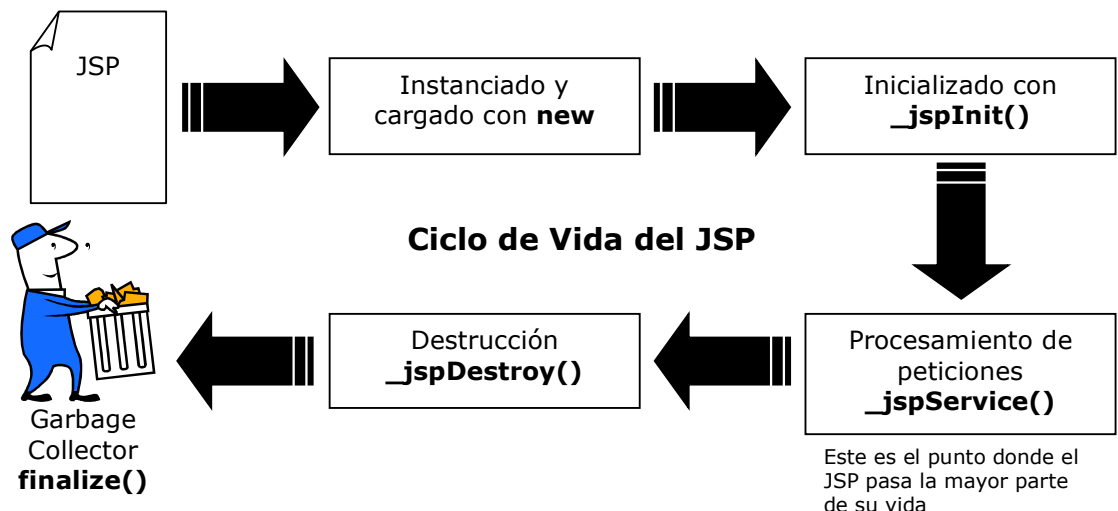
JSP es una extensión de Servlet. La página JSP es implementada a través de la clase `javax.servlet.Servlet` de acuerdo a la especificación Servlet 2.5




Ciclo de Vida del JSP

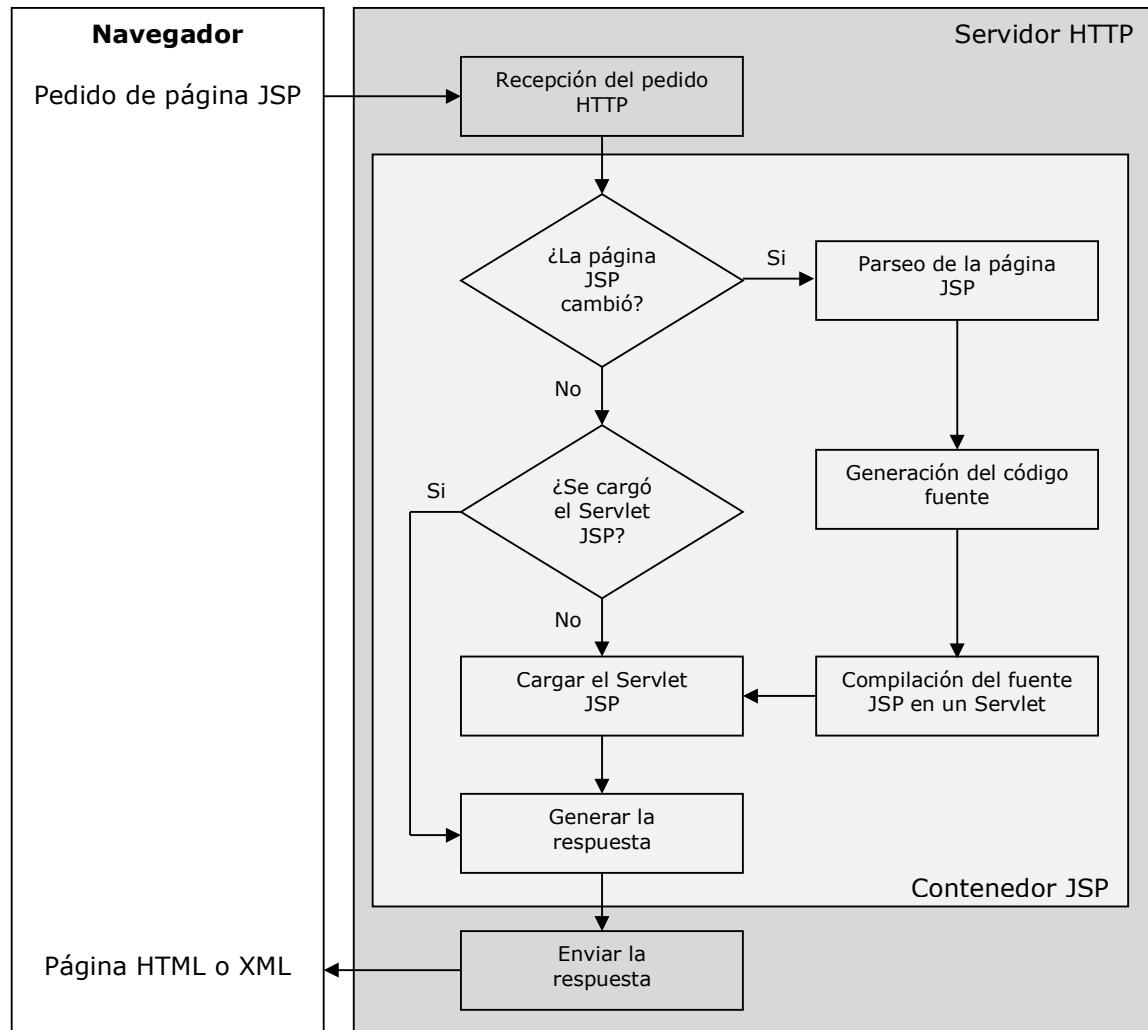
El contenedor JSP es el encargado de regular y controlar el ciclo de vida de un JSP. El ciclo de vida consiste de cuatro fases:

- *Traslación*: Si la sintaxis es correcta se traslada al Servlet de implementación.
- *Inicialización*: El contenedor JSP carga el Servlet de implementación y crea una instancia del Servlet para procesar el pedido. Se invoca al método `_jspInit()`
- *Ejecución*: Después de que el contenedor cargó e inicializó el Servlet de implementación, el pedido es procesado. Para resolver el pedido se llama al método `_jspService()`
- *Finalización*: El contenedor invoca el método `_jspDestroy()`. Después de que este método es invocado el Servlet de implementación no puede responder a ningún otro pedido.



	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

¿Cómo trabaja una página JSP?



JSP (Java Server Page) y JSTL (JSP Standard Tag Library), son importantes componentes de aplicación Web para construir contenido dinámico en plataformas J2EE. Los scripts JSP pueden construir fácilmente contenido HTML y acceder a propiedades JavaBean a través de bibliotecas EL (Expression Language). Los componentes JSTL encapsulan funciones que permiten a los desarrolladores repetir a través de datos, realizar operaciones de transformación XSLT y acceder a datos de objeto y base de datos. Las dos tecnologías pueden combinarse para dar formas a componentes a nivel de presentación que mostrar y con los que interactuar con modelos de datos de back-end.


EL (Expression Language)

La especificación de JSP 2.0 introdujo Expression Language (EL), fue diseñado para ser más amigable que el manejo de scriptlets. Es similar a JavaScript y a XPath, de ahí que fue inspirado en ECMAScript que es la versión estándar de JavaScript.

EL no es un lenguaje de programación en sí mismo, su único propósito es:

- Referenciar objetos y sus propiedades.
- Escribir expresiones simples.

Las expresiones EL se pueden utilizar con tres valores de atributos distintos. En primer lugar, pueden aplicarse cuando un valor de atributo tiene una sola expresión. En segundo lugar, pueden utilizarse cuando el valor del atributo contiene una o más expresiones rodeadas o separadas por texto. Por último, pueden utilizarse si el valor del atributo solo contiene texto.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.2 VIGENCIA: 06-08-2009
APUNTE DE ARQUITECTURA MODELO 1		

Expresiones EL	Implementación
Una sola expresión	<xyz.tag value="{expresión}"/>
Una o más expresiones	<xyz.tag value="texto{expresión}texto{expresión}"/>
Solo texto	<xyz.tag value="texto"/>

Como se dijo anteriormente, EL ha sido creado para reemplazar a los scriptlets.

Para poder deshabilitar el uso de scriptlets, se debe editar el archivo web.xml, y agregar la siguiente directiva:

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-enabled>false</scripting-enabled>
  </jsp-property-group>
</jsp-config>
```

A su vez, se puede pretender deshabilitar el uso de EL, para ello en el archivo web.xml se debe especificar:

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <el-enabled>false</el-enabled>
  </jsp-property-group>
</jsp-config>
```

Esta directiva puede directamente aplicarse a la página JSP incluyendo:

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

Las expresiones se componen de:

- *Identificadores.* Hay once identificadores reservados que corresponden a once objetos implícitos. El resto de identificadores sirven para crear variables.
- *Literales.* Son números, cadenas delimitadas por comillas simples o dobles, y los valores true, false, y null.
- *Operadores.* Permiten comparar y operar con identificadores y literales.
- *Operadores de acceso.* Se usan para referenciar propiedades de los objetos.

Operadores

Operadores matemáticos


Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Multiplicación	*	{2*4}	8
División	/ o div	{5/2}	2.5
Resto de una división entera	% o mod	{5%2}	1
Suma	+	{2+2}	4
Resta	-	{7-2}	5

Operadores de comparación

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Igualdad	== o eq	{2 == '2'}	Verdadero
Desigualdad	!= o ne	{2 != 2}	Falso
Menor que	< o lt	{2 < 2}	Falso
Mayor que	> o gt	{3 > 2}	Verdadero
Menor o igual que	<= o le	{2 <= 2}	Verdadero
Mayor o igual que	>= o ge	{1 >= 2}	Falso

Operadores lógicos

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Negación	! o not	{!(2 == 2)}	Falso
Y	&& o and	{(2 == 2) && (2 >= 0)}	Verdadero

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Ó	o or	$\$(2 == 2) (2 != 2)$	Verdadero
---	------	--------------------------	-----------

Operadores de acceso

Para acceder a object.property se emplea: object['property']. Por ejemplo, $\$(param['login'])$

Operador empty

El operador empty comprueba si una colección o cadena es vacía o nula. Por ejemplo, $\$(empty param.login)$
Otra manera de hacerlo es usando la palabra clave null. Por ejemplo, $\$(param.login == null)$

Objetos implícitos

Estos objetos no requieren ser declarados ya que se declaran automáticamente.

Objeto	Descripción
pageContext	Accede al objeto PageContext que da acceso a todos los espacios de nombre asociados con una página JSP.
pageScope	Map que contiene valores y nombres de atributos centrados en la página.
requestScope	Map que contiene valores y nombres de atributos centrados en la solicitud.
sessionScope	Map que contiene valores y nombres de atributos centrados en la sesión.
applicationScope	Map que contiene valores y nombres de atributos centrados en la aplicación.
param	Map que relaciona nombres de parámetros con valores de parámetro String.
paramValues	Map que relaciona nombres de parámetros con un componente de vector String de todos los valores de ese parámetro.
header	Map que contiene nombres de encabezado con valores de parámetro String.
headerValues	Map que contiene nombres de encabezado de un componente de vector String.
cookie	Map que contiene objetos cookie Web.
initParam	Map que alberga nombres de parámetros de inicialización de la página Web.

Funciones EL

Las funciones representan un modo de extender la funcionalidad del lenguaje de expresiones.

Una función EL es mapeada a un método **estático** de una clase Java. El mapeo se especifica dentro de un TLD (Tag Library Descriptor).

Una función en EL puede tomar cualquier número de parámetros. Estos deben ser declarados en el TLD.

Debe respetarse que los nombres de las funciones sean únicos, si no se generará un error al momento de traslación.

Dentro del TLD los parámetros y el tipo de retorno deben ser clases Java totalmente calificadas, por ejemplo, java.lang.String. La sintaxis de la firma de una función EL (<function-signature>) es,

return_type static_function_name(parameter_1_type, ..., parameter_n_type)

Si la firma de la función definida en el TLD no aparece exactamente con la función en la clase Java, un error a tiempo de traslación ocurrirá.

Un TLD es un archivo XML que declara una librería de tags, describiendo la clase que implementa cada tag. Cada TLD puede describir cero o más funciones estáticas. A cada función se le asigna un nombre y método que implementa de la clase Java.

Veamos un ejemplo sencillo,

Clase Java

```
package src;


public class MyFunctions {

    public static String lower(String param) {
        return param.toLowerCase();
    }

}
```

Archivo TLD

```
<?xml version="1.0" encoding="UTF-8"?>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>myfunctionel</short-name>
  <uri>/WEB-INF/tlds/myFunctionEL</uri>
  <function>
    <name>toLower</name>
    <function-class>src.MyFunctions</function-class>
    <function-signature>java.lang.String lower(java.lang.String)</function-signature>
  </function>
</taglib>
```

Página JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="fn" uri="/WEB-INF/tlds/myFunctionsEL.tld"%>

<!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=UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <b>${fn:toLower("¡HOLA MUNDO!! :")}</b>
  </body>
</html>
```

Resultado

¡hola mundo!! :)

Funciones EL vs. Custom Tags

De acuerdo a las siguientes preguntas podemos determinar si es conveniente utilizar funciones EL o custom tags.

- (1) ¿Se necesita conocimiento del ambiente? Sí, emplee tags. Un tag provee un fácil acceso al pageContext y otras variables, las funciones no, para accederlo deben pasarse como parámetros.
- (2) ¿Se requiere comportamiento interactivo sobre el cuerpo del HTML? Sí, emplee tags.
- (3) ¿Se quiere proveer un pequeño pedazo de código reusable que actúa con uno o más argumentos? Sí, emplee una función EL.
- (4) ¿Se quiere reusar código Java existente? Sí, emplee una función EL.

JavaBeans

Un JavaBean es un modelo de componente de software reusable en lenguaje Java, que puede ser manipulado visualmente por una herramienta de programación. Cualquier clase Java que adhiere a ciertas convenciones en lo relativo a las propiedades, métodos y eventos puede ser un JavaBean. Los componentes JavaBean son conocidos como beans.

Un bean es un componente, portable, visible o no, autocontenido y escrito en Java que cumple con las recomendaciones de la especificación JavaBean, en lo referente a los protocolos de comunicación y configuración del objeto generado.

Un bean es una clase Java que sigue determinadas convenciones, de manera que pueda ser empleada como un componente por herramientas de desarrollo, para la construcción de aplicaciones.

Características

- (1) **Introspección:** Las herramientas de construcción de beans pueden descubrir las características de un bean (propiedades, métodos y eventos) y analizar cómo trabaja. La información de una clase, tales como nombre de los métodos, parámetros y tipos de retorno, pueden ser descubiertas por otras clases.
- (2) **Propiedades:** Una propiedad es un atributo de un bean que puede ser leído o escrito por el cliente del bean, a través de métodos regulares cuyos nombres responden a los lineamientos de construcción de JavaBeans.
- (3) **Customización:** Permite al desarrollador establecer y personalizar nuevas propiedades que gobiernan el comportamiento del bean.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

- (4) Eventos: Permite a un bean comunicarse con otros beans.
- (5) Persistencia: Habilita al bean a salvar y recuperar su estado. Debe implementar la interface `java.io.Serializable`

Convenciones

- (1) Un JavaBean es una clase cuyo constructor no tiene argumentos.
- (2) No puede tener variables de instancias públicas, también conocidos como atributos de la clase.
- (3) Los valores persistentes deberán ser accedidos a través de los métodos getter y setter, conocidos como métodos de acceso.
- (4) Los nombres de los métodos getter o setter, se componen de la palabra get o set, más el nombre de la propiedad con su primera letra en mayúscula.
- (5) Un método regular getter no tiene parámetros y retorna un valor del tipo de la propiedad.
- (6) Un método setter tiene un único parámetro que es del tipo de la propiedad y tiene un retorno de tipo void.
- (7) Una propiedad legible debe tener un método getter.
- (8) Una propiedad escribible debe tener un método setter.
- (9) Dependiendo de la combinación de métodos getter y setter, una propiedad es solo lectura, solo escritura o lectura y escritura.
- (10) Una propiedad solo lectura no necesariamente debe coincidir con una variable de instancia. Puede ser un valor computado resultado de combinar varias variables de instancia.
- (11) El tipo de una propiedad puede ser una clase, una interface o un tipo primitivo.
- (12) Los beans pueden tener propiedades de múltiples valores representados por un vector de cualquier tipo. Esto es denominado una "propiedad indexada".
- (13) Dos tipos de métodos de acceso pueden tener una propiedad indexada:
 - a. Métodos de lectura y escritura de todo el vector.
 - b. Métodos de lectura y escritura de un elemento del vector.
- (14) En una propiedad booleana se puede emplear el método getter regular. Pero la recomendación es usar la palabra `is` combinada con el nombre de la propiedad con su primera letra en mayúscula.
- (15) El método setter de una propiedad booleana sigue el patrón general.

Veamos algunos ejemplos,

```
package src;
```

```
public class ProductoBean implements java.io.Serializable {
```

```
    private String nomProducto;
    private int cantidad;
    private float precioUnitario;
```

```
    public ProductoBean() {
    }
```

```
    public int getCantidad() {
        return cantidad;
    }
```

```
    public String getNomProducto() {
        return nomProducto;
    }
```


```
    public float getPrecioUnitario() {
        return precioUnitario;
    }
```

```
    public float getPrecioTotal() {
        return precioUnitario * cantidad;
    }
```

```
    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }
```

```
    public void setNomProducto(String nomProducto) {
        this.nomProducto = nomProducto;
    }
```

```
    public void setPrecioUnitario(float precioUnitario) {
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

        this.precioUnitario = precioUnitario;
    }

}

package src;

public class CategoriasBean {

    private String[] categorias;

    public CategoriasBean() {
        this.categorias = new String[10];
    }

    public String getCategoria(int posicion) {
        return categorias[posicion];
    }

    public void setCategoria(String categoria, int posicion) {
        this.categorias[posicion] = categoria;
    }

}

```

Tipos de beans

Existen dos tipos:

- (1) Value beans: Encapsula toda la información acerca de una entidad, tal como un usuario, producto, etc.
- (2) Utility beans: Efectúa alguna acción, tal como guardar información en la base de datos o enviar un e-mail. "Utility beans" pueden emplear "Value beans" como entrada o producirlos como resultado de una acción.

Empleo de beans

Para declarar el uso de una instancia bean en una página JSP se usa la directiva:

Sintaxis 1

<jsp:useBean id="name" scope="scope" Bean-Specification/>

Sintaxis 2

<jsp:useBean id="name" scope="scope" Bean-Specification>
Creation-Body
</jsp:useBean>

id: Nombre con el cual se referenciará el bean dentro del JSP. Atributo obligatorio.

scope: Ámbito de visibilidad del bean, entre ellos:

Ámbito	Duración
page	El bean solo será válido dentro de la página JSP y será regenerado en cada nueva petición.
request	El bean será válido a lo largo de la petición y estará disponible para incluirse o reenviarse a otras páginas JSP.
session	El bean se asocia a una sesión en particular, la cual es responsable de su creación. El bean será válido mientras la sesión exista.
application	El bean es común a todas las sesiones y será válido hasta que la aplicación Web finalice.

Bean-Specification: Atributos adicionales que sirven para declarar el bean. Debe considerarse que el uso de los mismos es mutuamente excluyente, es decir, que existen determinadas combinaciones a respetar. Entre ellos encontramos:

Atributo	Descripción
class="className"	Clase Java que implementa al objeto.
type="typeName"	Especifica el tipo de la variable que identifica al bean. Este debe corresponderse con el nombre de la clase Java o superclase o interface que implementa la clase Java.
beanName="beanName"	El nombre del bean en caso de que esté serializado (persistencia), como lo suministraríamos en el método <code>java.beans.Beans.instantiate()</code> .

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Los atributos antes especificados deben aparecer en las siguientes combinaciones:

- class
- type
- class y type
- beanName y type

Creation-Body: Se ejecuta solo si el bean no existe y sirve para inicializar el mismo. El bean solo es creado si "no es encontrado dentro del scope establecido".

Para establecer un valor a una propiedad del bean, se usa:

<jsp:setProperty name="beanId" property="propertyName" Property-Specification />

name: Nombre con el cual se identifica el bean en la página JSP.

property: Nombre de la propiedad del bean a establecer su valor.

Property-Specification: Atributos adicionales cuyo uso es mutuamente excluyente. Entre ellos encontramos:

Atributo	Descripción
value="value"	Valor a establecer.
param="param"	Nombre del parámetro disponible en el HttpServletRequest cuyo valor será asignado a la propiedad.

Si no se especifica ninguno de estos atributos, se usa por defecto "param"

Para obtener el valor actual de una propiedad de un bean, se usa:

<jsp:getProperty name="beanId" property="propertyName" />

name: Nombre con el cual se identifica el bean en la página JSP.

property: Nombre de la propiedad del bean a establecer su valor.

Con todo lo antes expuesto veamos un ejemplo sencillo,

Clase Java

package src;

```
public class ContadorBean {
    private int contador;

    public ContadorBean() {
        this.contador = 0;
    }


    public void setContador(int c) {
        this.contador += c;
    }

    public int getContador() {
        return this.contador;
    }
}
```

Página JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

"http://www.w3.org/TR/html4/loose.dtd">

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Beans</title>
</head>
<body>
  <jsp:useBean id="cont" class="src.ContadorBean" scope="session" />
  <jsp:setProperty name="cont" property="contador" value="1"/>

  <h2>Contador: <jsp:getProperty name="cont" property="contador"/></h2>
  <h2>Contador: ${cont.contador}</h2>

  <a href="index.jsp" target="_self">Volver a cargar</a>
</body>
</html>
```

Resultado

Al estar declarado el bean con scope de “sesión”, cada vez que se recargue la página se irá incrementando el contador.

Inicialización de un JavaBean

Existen dos posibilidades para inicializar un bean, veamos ejemplos para explicar cada una de ellas.

Clase Java

```
package src;

public class PersonaBean {

    private String apellido, nombre;

    public PersonaBean() { }

    public void setApellido(String a) {
        this.apellido = a;
    }

    public void setNombre(String n) {
        this.nombre = n;
    }

    public String getApellido() {
        return this.apellido;
    }

    public String getNombre() {
        return this.nombre;
    }

    public String getNombreCompleto() {
        return this.apellido + ", " + this.nombre;
    }


}
```

Posibilidad 1: Inicializar cada propiedad del bean de acuerdo al nombre de los parámetros del formulario procesado.

Página JSP 1: index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Beans</title>
</head>
<body>
  <form action="resultado.jsp" method="post">
    <p>Apellido: <input type="text" name="ape" maxlength="100" size="109"/></p>
    <p>Nombre: <input type="text" name="nom" maxlength="100" size="109"/></p>
    <input type="submit" value="Mostrar datos ingresados"/>
  </form>
</body>
</html>
```

Página JSP 2: resultado.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!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=UTF-8">
  <title>Resultado</title>
</head>
<body>
  <jsp:useBean id="per" class="classes.PersonaBean" scope="request">
    <jsp:setProperty name="per" property="apellido" param="ape"/>
    <jsp:setProperty name="per" property="nombre" param="nom"/>
  </jsp:useBean>
  <h3>Par&aacute;metros Recibidos</h3>
  <p><b>Apellido</b>: ${param["ape"]}</p>
  <p><b>Nombre</b>: ${param["nom"]}</p>
  <h3>Directivas JSP</h3>
  <p><b>Apellido</b>: <jsp:getProperty name="per" property="apellido"/></p>
  <p><b>Nombre</b>: <jsp:getProperty name="per" property="nombre"/></p>
  <p><b>Nombre Completo</b>: <jsp:getProperty name="per" property="nombreCompleto" /></p>
  <h3>Scriptlets</h3>
  <p><b>Apellido</b>: <%= per.getApellido() %></p>
  <p><b>Nombre</b>: <%= per.getNombre() %></p>
  <p><b>Nombre Completo</b>: <%= per.getNombreCompleto() %></p>
  <h3>EL</h3>
  <p><b>Apellido</b>: ${per.apellido}</p>
  <p><b>Nombre</b>: ${per.nombre}</p>
  <p><b>Nombre Completo</b>: ${per.nombreCompleto}</p>
  <br>
  <a href="index.jsp">Volver</a>
</body>
</html>
```


Posibilidad 2: Inicializar todas las propiedades del bean usando en el atributo "property" un asterisco (*). Esta posibilidad solo es válida si "todos" los parámetros del formulario procesado se denominan igual que las propiedades del bean.

Página JSP 1: index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!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=UTF-8">
  <title>Beans</title>
</head>
<body>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.2 VIGENCIA: 06-08-2009
APUNTE DE ARQUITECTURA MODELO 1		

```

<form action="resultado.jsp" method="post">
  <p>Apellido: <input type="text" name="apellido" maxlength="100" size="109"/></p>
  <p>Nombre: <input type="text" name="nombre" maxlength="100" size="109"/></p>
  <input type="submit" value="Mostrar datos ingresados"/>
</form>
</body>
</html>

```

Página JSP 2: resultado.jsp

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!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=UTF-8">
    <title>Resultado</title>
  </head>
  <body>
    <jsp:useBean id="per" class="classes.PersonaBean" scope="request">
      <jsp:setProperty name="per" property="*" />
    </jsp:useBean>
    <h3>Parámetros Recibidos</h3>
    <p><b>Apellido</b>: ${param["apellido"]}</p>
    <p><b>Nombre</b>: ${param["nombre"]}</p>
    <h3>Directivas JSP</h3>
    <p><b>Apellido</b>: <jsp:getProperty name="per" property="apellido"/></p>
    <p><b>Nombre</b>: <jsp:getProperty name="per" property="nombre"/></p>
    <p><b>Nombre Completo</b>: <jsp:getProperty name="per" property="nombreCompleto" /></p>
    <h3>Scriptlets</h3>
    <p><b>Apellido</b>: <%= per.getApellido() %></p>
    <p><b>Nombre</b>: <%= per.getNombre() %></p>
    <p><b>Nombre Completo</b>: <%= per.getNombreCompleto() %></p>
    <h3>EL</h3>
    <p><b>Apellido</b>: ${per.apellido}</p>
    <p><b>Nombre</b>: ${per.nombre}</p>
    <p><b>Nombre Completo</b>: ${per.nombreCompleto}</p>
    <br>
    <a href="index.jsp">Volver</a>
  </body>
</html>

```

Veamos como trabajamos para inicializar un bean con propiedades indexadas.

Clase Java

```

package src;

public class CategoriasBean {

  private String[] categorias;


  public CategoriasBean() {
    this.categorias = new String[100];
  }

  public String getCategoria(int pos) {
    return categorias[pos];
  }

  public String[] getCategorias() {
    return this.categorias;
  }

  public void setCategoria(String categoria, int pos) {
    this.categorias[pos] = categoria;
  }
}

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

    }

    public void setCategorias(String[] c) {
        this.categorias =c;
    }
}

```

Página JSP 1: index.jsp

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!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=UTF-8">
<title>Beans</title>
</head>
<body>
<h1>Categorías</h1>
<form action="resultado.jsp" method="post">
<p><input type="checkbox" id="A" name="categoria" value="A"/><label for="A">Categoría A</label></p>
<p><input type="checkbox" id="B" name="categoria" value="B"/><label for="B">Categoría B</label></p>
<p><input type="checkbox" id="C" name="categoria" value="C"/><label for="C">Categoría C</label></p>
<p><input type="checkbox" id="D" name="categoria" value="D"/><label for="D">Categoría D</label></p>
<p><input type="checkbox" id="E" name="categoria" value="E"/><label for="E">Categoría E</label></p>
<input type="submit" value="Continuar..." />
</form>
</body>
</html>

```

Página JSP 2: resultado.jsp

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>


<!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=UTF-8">
<title>Resultado</title>
</head>
<body>
<h1>Categorías Seleccionadas</h1>
<jsp:useBean id="cat1" class="src.CategoriasBean" scope="request">
<%
String[] c = request.getParameterValues("categoria");

for(int i = 0; i < c.length; i++) {
    cat1.setCategoria(c[i], i);
}
%>
</jsp:useBean>
<p>Primera Categoría Elegida (Scriptlet): <%= cat1.getCategoria(0)%> </p>

<jsp:useBean id="cat2" class="src.CategoriasBean" scope="request">
<jsp:setProperty name="cat2" property="categorias" value="${paramValues['categoria']}" />
</jsp:useBean>
<p>Primera Categoría Elegida (EL): ${cat2.categorias[0]}</p>
<a href="index.jsp">Volver</a>

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

</body>
</html>

JSTL

JSP Standard Tag Library (JSTL) encapsula en etiquetas simples funcionalidades básicas de muchas aplicaciones Web. Por ejemplo, en lugar de iterar sobre listas usando un scriptlet, JSTL cuenta con etiquetas que permiten iterar de la misma forma en todas partes. Esta estandarización posibilita aprender una sola etiqueta y utilizarla en múltiples contenedores JSP. Además, cuando las etiquetas son estándar, los contenedores pueden reconocer y optimizar sus implementaciones.

JSTL es una colección de cuatro librerías de tags:

- *Core*: Acciones de propósito generales, tales como iteraciones y condicionales.
- *XML*: Manipulación de documentos XML.
- *FMT*: Internacionalización y formato de configuración regional como moneda y fechas.
- *SQL*: Acceso a base de datos relacionales.

Debido a que JSTL 1.0 fue liberado antes que JSP 2.0, este soportaba completamente los contenedores JSP 1.2 y no EL. Por ello, para soportar ambos mundos, scripting y EL, se crearon un conjunto de librerías gemelas. Los URI de las dos librerías son similares, pero a las librerías de runtime se le agregaba "_rt" tanto al URI como al PREFIX. Por ello, fue posible mezclar el uso de acciones runtime y EL.

A partir, JSTL 1.1 se incorporó extensamente el uso de EL.

En la actualidad, la última versión estable de JSP es la 2.1 y de JSTL es la 1.2.

(1) Tag Libraries:

Librería	URI	PREFIX
Core	http://java.sun.com/jsp/jstl/core	c
XML	http://java.sun.com/jsp/jstl/xml	x
FMT	http://java.sun.com/jsp/jstl/fmt	fmt
SQL	http://java.sun.com/jsp/jstl/sql	sql

(2) Runtime Tag Libraries (Solo si se trabaja con JSTL 1.0):

Librería	URI	PREFIX
Core	http://java.sun.com/jstl/core_rt	c_rt
XML	http://java.sun.com/jstl/xml_rt	x_rt
FMT	http://java.sun.com/jstl/fmt_rt	fmt_rt
SQL	http://java.sun.com/jstl/sql_rt	sql_rt

Core Tag Library

La librería JSTL Core cubre las siguientes áreas funcionales:

- Manipulación de variables
- Condicionales
- Looping e iteraciones
- Manipulación de URL

Manipulación de variables

Sintaxis 1

<c:out value="expression" Out-Specification />


Sintaxis 2

**<c:out value="expression" Out-Specification>
 Out-Body
</c:out>**

value: Expresión que se mostrará. Es equivalente a <%= expresión %>

Out-Specification: Atributos adicionales, entre los que encontramos:

Atributo	Descripción
default="expression" o Out-Body	Expresión que se mostrará por defecto en el caso de que el atributo "value" sea una cadena vacía o nula.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

escapeXml="boolean"	Indica si la expresión que se visualiza debe ser tratada o no como HTML o XML. En el caso de que el valor sea "false", se considera que la expresión es HTML o XML por lo que será interpretada por parte del navegador, de lo contrario, el valor expresado se muestra en pantalla.
---------------------	--

Cuando el nombre de una variable aparece en una expresión, la variable se busca en el siguiente orden:

- (1) page
- (2) request
- (3) session
- (4) application

Si la variable no es encontrada, no se retorna nada.

Para acceder a una variable dentro de un alcance en particular, procedemos de la siguiente manera:

```

${pageScope.variable}
${requestScope.variable}
${sessionScope.variable}
${applicationScope.variable}

```

Veamos algunos ejemplos sencillos,

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<c:out value="¡Hola Mundo!! :)" />
<br>
<c:out value="${producto.id}" default="Sin definir" />
<br>
<c:out value="${param.nombre}">
No informado
</c:out>
<br>
<c:out value="<b>Texto en negrita</b>" default="Sin valor" escapeXml="false" />
<br>
<c:out value="<b>Texto en negrita</b>" default="Sin valor" escapeXml="true" />
</body>
</html>

```

Sintaxis 1

```
<c:set var="name" value="value" scope="Var-Scope" />
```

Sintaxis 2

```

<c:set var="name" scope="Var-Scope">
value
</c:set>

```


Establece el valor de una variable en un ámbito en particular.

var: Nombre de la variable a definir.

value: Valor asignado a la variable.

scope: Ámbito de visibilidad de la variable, entre ellos:

Ámbito	Duración
--------	----------

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

page	La variable solo será válida dentro de la página JSP y será regenerada en cada nueva petición.
request	La variable será válida a lo largo de la petición y estará disponible para incluirse o reenviarse a otras páginas JSP.
session	La variable se asocia a una sesión en particular, la cual es responsable de su creación. La variable será válida mientras la sesión exista.
application	La variable es común a todas las sesiones y será válida hasta que la aplicación Web finalice.

Veamos un ejemplo sencillo,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<c:set var="browser" value="${header['User-Agent']}" scope="session" />
<p>Navegador usado: <c:out value="${browser}"/></p>
</body>
</html>
```

Veamos como declarar una variable con un valor por defecto,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<c:set var="sessionid" scope="session">
<c:out value="${cookie['JSESSIONID'].value}" default="---"/>
</c:set>
<p>Identificador de Sesión: ${sessionid}</p>
</body>
</html>
```

<c:remove var="name" scope="Var-Scope">

Remueve una variable de un determinado alcance.

var: Nombre de la variable a eliminar.


scope: Ámbito en que se elimina la variable. Este puede ser page, request, session o application.

Tomando como base un de los ejemplos antes planteados,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
```


	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Ejemplo</title>
</head>
<body>
  <c:set var="browser" value="${header['User-Agent']}" scope="session" />
  <p>Navegador usado: <c:out value="${browser}"/></p>
  <c:remove var="browser" scope="session"/>
  <p>Variable removida</p>
  <p>Navegador usado: <c:out value="${browser}"/></p>
</body>
</html>

```

```

<c:catch var="name">
  catch-body
</c:catch>

```

Provee un mecanismo para capturar cualquier excepción java.lang.Throwable que puede ocurrir durante la ejecución de cualquier acción anidada.

var: Nombre de la variable que se define al ejecutar la etiqueta.

Condicionales

```

<c:if test="expression" var="name" scope="scope">
  If-Body
</c:if>

```

var: Nombre de la variable que contiene el retorno de la evaluación de la condición.

scope: Ámbito en el que es válida la variable. Para más detalles ver la etiqueta <c:set>

test: Expresión a evaluar. Si el resultado de evaluar la expresión es "true", el contenido del cuerpo será procesado en forma estándar por el contenedor JSP y cualquier salida será retornada al corriente JspWriter.

Veamos un ejemplo,

```


<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!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=UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1>Validador</h1>
    <form action="index.jsp" method="post">
      <p>Ingresa un valor: <input type="text" name="nro" maxlength="10" size="7"/></p>
      <input type="submit" value="Procesar"/>
    </form>

    <c:set var="i" value="${param.nro}" scope="request" />
    <c:catch var="exception">
      <c:if test="${!empty i}">
        <c:if test="${i mod 2 eq 0}">
          <p>El valor ingresado es par</p>
        </c:if>
        <c:if test="${i mod 2 ne 0}">
          <p>El valor ingresado es impar</p>
        </c:if>
      </c:if>
    </c:catch>
    <c:if test="${!empty exception}">

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

    <p>El valor ingresado debe ser un número</p>
  </c:if>
</body>
</html>

```

```

<c:choose>
  <c:when test="expression">
    When-Body 1
  </c:when>
  <c:when test=" expression">
    When-Body 2
  </c:when>
  ...
  <c:when test=" expression">
    When-Body n
  </c:when>
  <c:otherwise>
    Otherwise-Body
  </c:otherwise>
</c:choose>

```

Un número ilimitado de <c:when> puede existir en una acción <c:choose> pero solo puede haber una única etiqueta <c:otherwise>

Cuando se rempazan varios <c:if> por <c:choose> podemos notar:

- (1) El código es más legible y autodescriptivo para una persona que no se dedica a programar.
- (2) El código es más eficiente ya que una vez que una expresión es evaluada y resulta verdadera, no continúa evaluando todas las otras expresiones que siguen a continuación.
- (3) Con el empleo de <c:choose> nos aseguramos que las acciones son mutuamente excluyentes, mientras que un error en la lógica de los if, puede permitir ejecutar dos o más condiciones en forma simultánea.

Veamos el ejemplo anterior,

```


<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!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=UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1>Validador</h1>
    <form action="index.jsp" method="post">
      <p>Ingresa un valor: <input type="text" name="nro" maxlength="10" size="7"/></p>
      <input type="submit" value="Procesar"/>
    </form>

    <c:set var="i" value="${param.nro}" scope="request" />
    <c:catch var="exception">
      <c:if test="${!empty i}">
        <c:choose>
          <c:when test="${i mod 2 eq 0}">
            <p>El valor ingresado es par</p>
          </c:when>
          <c:otherwise>
            <p>El valor ingresado es impar</p>
          </c:otherwise>
        </c:choose>
      </c:if>
    </c:catch>
    <c:if test="${!empty exception}">
      <p>El valor ingresado debe ser un número</p>
    </c:if>
  </body>
</html>

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

</c:if>
</body>
</html>

```

Looping e iteraciones

Sintaxis 1: Iteración sobre una colección de objetos

```

<c:forEach [var="varName"] [varStatus="varStatusName"] items="collection" [begin="begin"]
[end="end"] [step="step"]>
  forEach-body
</c:forEach>

```

Sintaxis 2: Iteración de un número fijo de veces

```

<c:forEach [var="varName"] [varStatus="varStatusName"] begin="begin" end="end" [step="step"]>
  forEach-body
</c:forEach>

```

var: Nombre de la variable que contiene el valor actual de la iteración.

varStatus: Nombre de la variable que representa el estado de la iteración. Contiene información acerca de la iteración en curso. Sus propiedades son:

- index: Índice del actual ítem en la iteración
- count: Posición del ciclo de iteración
- first: Determina si el ciclo actual es el primero
- last: Determina si el ciclo actual es el último

items: Puede ser un objeto de uno de los siguiente tipos:

- Un vector
- Una implementación de java.util.Collection
- Una implementación de java.util.Iterator
- Una implementación de java.util.Enumeration
- Una implementación de java.util.Map
- Una cadena de valores separados por coma

Si es nulo ninguna iteración es llevada a cabo

begin: Si es especificado, debe ser un valor mayor o igual a cero. Si este se informa, la iteración comienza en el ítem que coincide con el valor especificado, de lo contrario, comienza en cero.

end: Si es especificado, debe ser mayor o igual al valor del atributo "begin". Si este se informa, la iteración termina cuando la posición del ítem tratado coincide.

step: Si es especificado, debe ser mayor o igual a uno. El valor por defecto es uno. Representa el incremento aplicado en cada iteración.

Veamos algunos ejemplos,

Ejemplo 1


```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Libros</h1>
<form action="index.jsp" method="post">
<input type="checkbox" id="bookA" name="books" value="A"/><label for="bookA">Libro A</label>
<input type="checkbox" id="bookB" name="books" value="B"/><label for="bookB">Libro B</label>
<input type="checkbox" id="bookC" name="books" value="C"/><label for="bookC">Libro C</label>

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009	

```

<input type="checkbox" id="bookD" name="books" value="D"/><label for="bookD">Libro D</label>
<br>
<input type="submit" value="Mostrar"/>
</form>

<c:set var="bo" value="${paramValues.books}"/>
<c:if test="${!empty bo}">
  <ol>
    <c:forEach items="${bo}" var="b">
      <c:if test="${!empty b}">
        <li>${b}</li>
      </c:if>
    </c:forEach>
  </ol>
</c:if>
</body>
</html>

```

Ejemplo 2

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!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=UTF-8">
  <title>Ejemplo</title>
</head>
<body>
  <h1>Múltiplos</h1>
  <table border="1" cellspacing="0" cellpadding="1">
    <c:forEach var="row" begin="1" end="10">
      <tr>
        <c:forEach var="col" begin="1" end="10">
          <td>${row * col}</td>
        </c:forEach>
      </tr>
    </c:forEach>
  </table>
</body>
</html>

```

```

<c:forTokens [var="varName"] [varStatus="varStatusName"] items="stringOfTokens"
delims="delimiters" [begin="begin"] [end="end"] [step="step"]>
  forTokens-body
</c:forTokens>

```

Interactúa sobre una cadena separada por un conjunto de delimitadores.


var: Nombre de la variable que contiene el valor actual de la iteración.

varStatus: Nombre de la variable que representa el estado de la iteración. Contiene información acerca de la iteración en curso. Sus propiedades son:

- index: Índice del actual ítem en la iteración
- count: Posición del ciclo de iteración
- first: Determina si el ciclo actual es el primero
- last: Determina si el ciclo actual es el último

items: Cadena a tratar. Atributo obligatorio.

delims: Separador utilizado en la cadena. Atributo obligatorio.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

begin: Si es especificado, debe ser un valor mayor o igual a cero. Si este se informa, la iteración comienza en el ítem que coincide con el valor especificado, de lo contrario, comienza en cero.

end: Si es especificado, debe ser mayor o igual al valor del atributo "begin". Si este se informa, la iteración termina cuando la posición del ítem tratado coincide.

step: Si es especificado, debe ser mayor o igual a uno. El valor por defecto es uno. Representa el incremento aplicado en cada iteración.

Veamos un ejemplo sencillo,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!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=UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <ul>
      <c:forEach items="a;b;c;d;e;f" delims=";" var="current" begin="2" end="4">
        <li>${current}</li>
      </c:forEach>
    </ul>
  </body>
</html>
```

Manipulación de URL

<c:import var="name" url="url" />

Importa el contenido de un recurso basado en un URL. Reemplaza a <jsp:include>. La etiqueta <jsp:include> permite incluir archivos solo desde la aplicación Web actual, mientras que esta etiqueta permite incluir archivos desde otros sitios Web.

<c:url value="url" />

Provee una forma sencilla de construir correctamente una URL aplicando las reglas de reescritura.

<c:redirect url="url" />

Envía una redirección al cliente.

<c:param name="name" value="value" />


Permite definir parámetros en <c:import>, <c:url> o <c:redirect>, usando la siguiente sintaxis:

```
<c:url value="url" >
  <c:param name="param1">value1</c:param>
  <c:param name="param2">value2</c:param>
</c:url>
```

XML Tag Library

XML proporciona un medio flexible para representar datos estructurados. Como resultado de ello, es especialmente adecuado para el intercambio de datos entre sistemas débilmente acoplados. Esto a su vez hace que sea una tecnología de integración atractiva para aplicaciones Web.

El primer paso para interactuar con los datos XML es recuperarlo como un documento XML y armar una estructura de datos para acceder al contenido del mismo. Después de que el documento ha sido analizado se puede, opcionalmente,

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.2 VIGENCIA: 06-08-2009
APUNTE DE ARQUITECTURA MODELO 1		

transformar para producir un nuevo documento XML, al que se le pueden aplicar las mismas operaciones. Finalmente, los datos del documento se pueden extraer y mostrar o utilizar como entrada para realizar operaciones adicionales.

Estas tareas se reflejan en las etiquetas JSTL utilizadas para la manipulación de XML. Los documentos XML se recuperan mediante la etiqueta `<c:import>` de la librería "Core". La etiqueta `<x:parse>` se utiliza para analizar el documento, con soporte para las tecnologías estándar de análisis XML como son el Modelo de Objetos de Documento (DOM) y la API Simple para XML (SAX). La etiqueta `<x:transform>` está disponible para la transformación de documentos XML y se basa en la tecnología estándar para la transformación de datos XML, Extensible Stylesheet Language (XSL). Por último, varias etiquetas se proporcionan para acceder y manipular datos XML, los cuales utilizan un nuevo estándar, el lenguaje XML Path (XPath).

A su vez, la librería posee otras etiquetas muy similares a las "etiquetas fundamentales". Por ejemplo, al igual que `<c:out>` tenemos `<x:out>`. Del mismo modo, `<x:forEach>`, `<x:if>`, `<x:when>`, etc. Por lo tanto, si hemos entendido la sintaxis de la librería "Core", las etiquetas XML no serán difíciles de utilizar.

Parseo de XML

Sintaxis 1

```
<x:parse xml="xml" var="name" scope="scope" filter="expression" systemId="expression"/>
```

Sintaxis 2

```
<x:parse var="name" scope="scope" filter="expression" systemId="expression">  
xml  
</x:parse>
```

xml: Su valor debe ser una cadena que contiene el documento XML para analizar o una instancia de java.io.Reader. Este puede ser especificado directamente en el cuerpo de la etiqueta `<c:parse>` sin usar el atributo correspondiente.

var: Nombre de la variable que referencia al documento XML.

scope: Ámbito de visibilidad de la variable. Los valores posibles son los ya estudiados anteriormente.

filter: Se especifica una instancia de la clase org.xml.sax.XMLFilter para filtrar el documento antes de analizarlo. Este atributo es especialmente útil si el documento que se va a analizar es muy grande y solo se requiere una pequeña porción de este.

systemId: Indica la URI para el documento que se analiza y resuelve cualquier ruta relativa que esté presente en el mismo. Este atributo es necesario si el código XML que se está analizando utiliza direcciones URL relativas para hacer referencia a otros documentos o recursos a los cuales se necesita tener acceso durante el proceso de análisis.

Si se requiere realizar operaciones sobre el documento analizado, el cual debe adherirse a un interfaz estándar, la sintaxis de la etiqueta es la siguiente:

Sintaxis 1

```
<x:parse xml="xml" varDom="name" scopeDom="scope" filter="expression" systemId="expression"/>
```

Sintaxis 2

```
<x:parse varDom="name" scopeDom="scope" filter="expression" systemId="expression">  
xml  
</x:parse>
```

Cuando se utiliza esta versión de `<x:parse>`, el objeto que representa el documento XML analizado debe implementar la interfaz org.w3c.dom.Document.

varDom: Nombre de la variable que referencia al documento XML.


scopeDom: Ámbito de visibilidad de la variable. Los valores posibles son los ya estudiados anteriormente.

Trabajando con el contenido XML

A menudo sólo ciertos elementos de los datos contenidos en el documento XML serán de interés para una aplicación particular. Por este motivo, la librería incluye varias etiquetas XML para acceder y manipular los elementos individuales de los documentos XML.

Estas etiquetas se basan en las etiquetas de la librería "Core", si bien estas últimas utilizan expresiones EL, sus homólogos para manipular los documentos XML usan expresiones XPath.

XPath es una notación estándar para hacer referencia a los elementos de un documento XML, sus atributos y contenido. Como su nombre lo indica, esta notación se asemeja a las rutas de archivos del sistema en el sentido de

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

que los componentes de una declaración de XPath se delimitan mediante barras. Estos componentes se mapean con nodos del documento XML. Además, los asteriscos se pueden utilizar como comodines para considerar varios nodos, y las expresiones entre corchetes se pueden usar para que coincidan con los valores de los atributos, en este último caso es requerido especificar índices.

Para mostrar los datos de un elemento del documento XML, se utiliza la etiqueta `<x:out>`, que es análoga a `<c:out>`. Los atributos de esta etiqueta son "select" y "escapeXml".

`<x:out select="XPathExpression" escapeXml="boolean"/>`

La diferencia, por supuesto, es que el valor del atributo "select" debe ser una expresión XPath, mientras que el atributo "value" de `<c:out>` debe ser una expresión EL. El significado del atributo `escapeXml` es el mismo en ambas etiquetas.

Además de `<x:out>`, la librería incluye las siguientes etiquetas para la manipulación de datos XML:

- `<x:set>` para asignar el valor de una expresión XPath a una variable JSTL bajo un determinado alcance.
- `<x:if>` para evaluar una expresión XPath.
- `<x:choose>`, `<x:when>` y `<x:otherwise>` para evaluar distintas expresiones XPath mutuamente excluyentes.
- `<x:forEach>` para iterar sobre los múltiples elementos de una expresión XPath.

En todos los casos, se utiliza el atributo "select" en el cual se debe especificar una expresión XPath.

Veamos algunos ejemplos,

Ejemplo 1

Página JSP

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


<!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>Ejemplo</title>
</head>
<body>
<c:import var="rssFeed" url="http://slashdot.org/slashdot.rdf"/>
<x:parse var="rss" xml="{rssFeed}"/>
<h1>
<a href="<x:out select="$rss//*[name()='channel']/*[name()='link']" escapeXml="false"/>"
target="_blank">
<x:out select="$rss//*[name()='channel']/*[name()='title']" escapeXml="false"/>
</a>
</h1>
<ul>
<x:forEach select="$rss//*[name()='item']">
<li>
<a href="<x:out select="."/*[name()='link']"/>" target="_blank">
<x:out select="."/*[name()='title']" escapeXml="false"/>
</a>
</li>
</x:forEach>
</ul>
</body>
</html>
```

Ejemplo 2

Archivo XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

<cereales>
  <cereal>Cebada</cereal>
  <cereal>Avena</cereal>
  <cereal>Mijo</cereal>
  <cereal>Maíz</cereal>
  <cereal>Arroz</cereal>
  <cereal>Centeno</cereal>
  <cereal>Trigo</cereal>
</cereales>
</root>

```

Página JSP

```

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

<!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>Ejemplo</title>
  </head>
  <body>
    <h1>Cereales</h1>
    <c:import var="cereales" url="/WEB-INF/cereales.xml"/>
    <x:parse var="cer" xml="{cereales}"/>
    <ol>
      <x:forEach var="c" select="$cer/root/cereales/cereal">
        <li><x:out select="$c" escapeXml="false" /></li>
      </x:forEach>
    </ol>
  </body>
</html>

```

Ejemplo 3

Archivo XML

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
  <producto>
    <id>1</id>
    <nom>Lapicera</nom>
    <cant>10</cant>
  </producto>
  <producto>
    <id>2</id>
    <nom>Plasticola</nom>
    <cant>4</cant>
  </producto>
  <producto>
    <id>3</id>
    <nom>Lápiz</nom>
    <cant>0</cant>
  </producto>
  <producto>
    <id>4</id>
    <nom>Resma de Papel A4</nom>
    <cant>1</cant>
  </producto>
</root>


```

Página JSP

```

<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>

```


	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

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

<!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>Ejemplo</title>
  </head>
  <body>
    <h1>Productos faltantes</h1>
    <c:import var="stock" url="/WEB-INF/stock.xml"/>
    <x:parse var="st" xml="{stock}"/>
    <table border="1" cellpadding="0" cellspacing="1">
      <tr>
        <th>Id.</th>
        <th>Nombre</th>
      </tr>
      <x:forEach var="s" select="$st/root/producto">
        <x:if select="$s/cant = 0">
          <tr>
            <td><x:out select="$s/id" escapeXml="false" /></td>
            <td><x:out select="$s/nom" escapeXml="false" /></td>
          </tr>
        </x:if>
      </x:forEach>
    </table>
  </body>
</html>

```

Transformando XML

Para poder transformar XML se utilizan hojas de estilo XSL. JSTL apoya esta operación mediante el uso de la etiqueta `<x:transform>`. Como fue el caso de `<x:parse>`, la etiqueta `<x:transform>` admite varias formas diferentes. Entre ellas:

Sintaxis 1

```

<x:transform xml="xml" xslt="expression" var="name" scope="scope" xmlSystemId="expression"
xsltSystemId="expression">
  <x:param name="expression" value="expression" />
  ...
</x:transform>

```

Sintaxis 2

```

<x:transform xslt="expression" var="name" scope="scope" xmlSystemId="expression"
xsltSystemId="expression">
  xml
  <x:param name="expression" value="expression" />
  ...
</x:transform>

```


xml: Su valor debe ser una cadena que contiene el documento XML a transformar o una instancia de `java.io.Reader`. También puede ser una instancia de la clase `org.w3c.dom.Document` o `javax.xml.transform.Source`, o finalmente, puede ser el valor de una variable asignada a través del atributo `var` o `varDom` de la etiqueta `<x:parse>`. A su vez, el valor puede ser especificado directamente en el cuerpo de la etiqueta `<x:transform>` sin usar el atributo correspondiente.

xslt: Define la hoja de estilo con la que se transforma el documento XML. Este atributo es obligatorio.

var: Nombre de la variable que referencia al documento XML ya transformado.

scope: Ámbito de visibilidad de la variable. Los valores posibles son los ya estudiados anteriormente.

xmlSystemId: Indica la URI para el documento que se transforma y resuelve cualquier ruta relativa que esté presente en el mismo. Este atributo es necesario si el código XML que se está transformando utiliza direcciones URL relativas

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

para hacer referencia a otros documentos o recursos a los cuales se necesita tener acceso durante el proceso de transformación.

xsltSystemId: Indica la URI para la hoja de estilos y resuelve cualquier ruta relativa que esté presente en la misma. Este atributo es necesario si el código XSL presenta direcciones URL relativas que hacen referencia a otras hojas de estilos a las cuales se necesita tener acceso.

Si el manejador de hojas de estilos para la transformación del documento recibe parámetros, estos deben ser especificados a través de la etiqueta <x:param>.

Un detalle a tener en cuenta, es que si se especifica el documento XML en el cuerpo de la etiqueta <x:transform>, este debe aparecer antes que la definición de parámetros.

La etiqueta <x:param> posee dos atributos "name" y "value", ambos obligatorios.

De los ejemplos antes desarrollados,

Ejemplo 1

Archivo XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo</title>
      </head>
      <body>
        <h1>
          <a target="_blank">
            <xsl:attribute name="href">
              <xsl:value-of select="//*[name()='channel']/*[name()='link']"/>
            </xsl:attribute>
            <xsl:value-of select="//*[name()='channel']/*[name()='title']" />
          </a>
        </h1>
        <ul>
          <xsl:for-each select="//*[name()='item']">
            <li>
              <a target="_blank">
                <xsl:attribute name="href">
                  <xsl:value-of select=".*[name()='link']"/>
                </xsl:attribute>
                <xsl:value-of select=".*[name()='title']" />
              </a>
            </li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```


Página JSP

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<c:import var="rssFeed" url="http://slashdot.org/slashdot.rdf"/>
<c:import var="rssFeedHtml" url="/WEB-INF/slashdot.xml"/>
<x:transform xml="{rssFeed}" xslt="{rssFeedHtml}"/>
```

Ejemplo 2

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Archivo XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo</title>
      </head>
      <body>
        <h1>Cereales</h1>
        <ul>
          <xsl:for-each select="root/cereales/cereal">
            <li><xsl:value-of select="text()"/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Página JSP

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


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<c:import var="cereales" url="/WEB-INF/cereales.xml"/>
<c:import var="cerealesHtml" url="/WEB-INF/cereales.xml"/>
<x:transform xml="{cereales}" xslt="{cerealesHtml}"/>
```

Ejemplo 3

Archivo XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo</title>
      </head>
      <body>
        <h1>Productos faltantes</h1>
        <table border="1" cellpadding="0" cellspacing="1">
          <tr>
            <th>Id.</th>
            <th>Nombre</th>
          </tr>
          <xsl:for-each select="root/producto">
            <xsl:if test="./cant = 0">
              <tr>
                <td><xsl:value-of select="./id"/></td>
                <td><xsl:value-of select="./nom"/></td>
              </tr>
            </xsl:if>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Página JSP

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<c:import var="stock" url="/WEB-INF/stock.xml"/>
<c:import var="stockHtml" url="/WEB-INF/stock.xml"/>
<x:transform xml="${stock}" xslt="${stockHtml}"/>
```

FMT Tag Library

Las etiquetas de la librería FMT están divididas en cuatro grupos. El primer grupo permite establecer la internacionalización del resto de las etiquetas que se utilicen. En otras palabras, permite establecer explícitamente la configuración regional y la zona horaria que las etiquetas FMT utilizarán para dar formato. El segundo y tercer grupo permiten dar formato a las fechas y números, respectivamente. Y el último grupo, permite internacionalizar los mensajes de texto.

Internacionalización

La configuración regional utilizada por las etiquetas JSTL, se determina normalmente mediante la evaluación del atributo "Accept-Language" de la cabecera HTTP enviada como respuesta a la petición del usuario. Si esta información no está presente, entonces JSTL proporciona un conjunto de variables de configuración JSP para configurar un idioma predeterminado. Si a su vez, estas variables de configuración no han sido establecidas, la máquina virtual toma la configuración regional por defecto del Sistema Operativo donde se está ejecutando el contenedor JSP.

La etiqueta JSTL utilizada para tal fin es <fmt:setLocale> cuya sintaxis es:

```
<fmt:setLocale value="expression" scope="scope" variant="expression"/>
```

value: Cadena con el nombre de la localización seleccionada o una instancia de la clase java.util.Locale. El nombre está construido por el código ISO del país en minúscula seguido opcionalmente por un guión más el código ISO del lenguaje en mayúscula. Este atributo es el único obligatorio.

scope: Ámbito de visibilidad de la localización definida. Los valores posibles son los ya estudiados anteriormente.

variant: Permite personalizar la configuración regional a un navegador específico. Por ejemplo, Mac y Win son variantes de los nombres Apple Macintosh y Microsoft Windows, respectivamente.

Después que el contenedor JSP procesa este fragmento, las preferencias del idioma definidas por el navegador serán ignoradas.

La etiqueta <fmt:setTimeZone> permite fijar la zona horaria predeterminada para el uso de las otras etiquetas FMT. Su sintaxis es:

```
<fmt:setTimeZone value="expression" var="name" scope="scope"/>
```


value: Cadena con el nombre de la zona horaria seleccionada o una instancia de la clase java.util.TimeZone. Desafortunadamente, no hay norma para la definición del huso horario, por lo que se utilizan los definidos por la plataforma Java, los valores válidos se pueden tomar de la ejecución del método getAvailableIDs(). Este atributo es el único obligatorio.

var: Nombre de la variable que referencia el huso horario que se declara.

scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

También se puede utilizar la etiqueta <fmt:timeZone> donde la aplicación del huso horario es válida dentro del cuerpo de la etiqueta. Su sintaxis es:

```
<fmt:timeZone value="expression">
  TimeZone-Body
</fmt:timeZone>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Fechas

La librería FMT incluye dos etiquetas para interactuar con la fecha y hora: `<fmt:formatDate>` y `<fmt:parseDate>`. Como sus nombres lo indican, se utiliza `<fmt:formatDate>` para dar formato y mostrar la fecha y hora, mientras que `<fmt:parseDate>` se utiliza para analizar la fecha y los valores de la hora.

La sintaxis de la etiqueta `<fmt:formatDate>` es:

`<fmt:formatDate value="expression" timeZone="expression" type="field" dateStyle="style" timeStyle="style" pattern="expression" var="name" scope="scope"/>`

value: Es una instancia de la clase `java.util.Date`. Este atributo es el único obligatorio.

timeZone: Permite definir bajo que huso horario se mostrará la fecha y hora especificada. Si no es definida se toma la utilizada por la máquina virtual, es decir, la configuración horaria del Sistema Operativo.

type: Indica que campos de la fecha se mostrarán. Los valores posibles son: "time", "date" o "both". El valor por defecto es "date".

dateStyle: Permite definir el formato utilizado en la fecha. Los valores posibles son: "default", "short", "medium", "long", y "full". El valor por defecto es "default".

timeStyle: Permite definir el formato utilizado en la hora. Los valores posibles son: "default", "short", "medium", "long", y "full". El valor por defecto es "default".

pattern: En lugar de definir un estilo, se puede utilizar un patrón. El patrón se basa en las convenciones de la clase `java.text.SimpleDateFormat`.

var: Nombre de la variable que referencia al formato que se declara.

scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

Veamos un ejemplo,


```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Formatos de Fechas</h1>
<jsp:useBean id="hoy" class="java.util.Date" />
<fmt:setLocale value="en_US" />
<fmt:setTimeZone value="GMT" var="gmt" />
<p>Fecha completa (en_US): <fmt:formatDate value="${hoy}" dateStyle="full" /></p>
<p>Fecha con formato MM/d/yyyy hh:mm (en_US): <fmt:formatDate value="${hoy}"
timeZone="${gmt}" dateStyle="full" timeStyle="medium" pattern="MM/d/yyyy hh:mm"/></p>
<br>
<fmt:setLocale value="es_AR" />
<p>Fecha completa (es_AR): <fmt:formatDate value="${hoy}" dateStyle="full" /></p>
<p>Fecha con formato d/MM/yyyy hh:mm (es_AR): <fmt:formatDate value="${hoy}"
dateStyle="full" timeStyle="medium" pattern="d/MM/yyyy hh:mm"/></p>
</body>
</html>
```

La etiqueta `<fmt:parseDate>` presenta dos posibles sintaxis:

Sintaxis 1

`<fmt:parseDate value="expression" type="field" dateStyle="style" timeStyle="style" pattern="expression" timeZone="expression" parseLocale="expression" var="name" scope="scope"/>`

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Sintaxis 2

```
<fmt:parseDate type="field" dateStyle="style" timeStyle="style" pattern="expression"
timeZone="expression" parseLocale="expression" var="name" scope="scope">
  ParseDate-Body
</fmt:parseDate>
```

value: Cadena de caracteres que puede ser una fecha, hora o ambas. Este atributo es el único obligatorio.

timeZone: Permite definir bajo que huso horario se mostrará la fecha y hora especificada. Si no es definida se toma la utilizada por la máquina virtual, es decir, la configuración horaria del Sistema Operativo.

type: Indica que campos de la fecha se mostrarán. Los valores posibles son: "time", "date" o "both". El valor por defecto es "date".

dateStyle: Permite definir el formato utilizado en la fecha. Los valores posibles son: "default", "short", "medium", "long", y "full". El valor por defecto es "default".

timeStyle: Permite definir el formato utilizado en la hora. Los valores posibles son: "default", "short", "medium", "long", y "full". El valor por defecto es "default".

pattern: En lugar de definir un estilo, se puede utilizar un patrón. El patrón se basa en las convenciones de la clase java.text.SimpleDateFormat.

parseLocale: Permite definir la configuración regional que se utilizará. Debe ser una cadena con el nombre de la localización seleccionada o una instancia de la clase java.util.Locale.

var: Nombre de la variable que referencia al objeto Date creado.


scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

ParseDate-Body: Cuando se define no se debe utilizar el atributo "value", el valor especificado debe ser según la definición de este último.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Parseo de Fecha</h1>
<c:set var="fechaNac" value="14/10/1980 17:12:00" />
<c:catch var="ex">
  <fmt:parseDate parseLocale="es_AR" type="both" dateStyle="short" timeStyle="short"
var="fechaNacP">
    <fmt:formatDate value="${fechaNac}" dateStyle="short" timeStyle="short" type="both"
/></p>
  </c:catch>
<c:if test="${!empty ex}">
  <p>La fecha informada no es válida. Error: ${ex}</p>
</c:if>
</body>
</html>
```

Números

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Así como las etiquetas `<fmt:formatDate>` y `<fmt:parseDate>` son utilizadas para dar formato y analizar fechas, las etiquetas `<fmt:formatNumber>` y `<fmt:parseNumber>` realizan las mismas funciones sobre los datos numéricos.

La etiqueta `<fmt:formatNumber>` se utiliza para mostrar datos numéricos, incluyendo las monedas y los porcentajes, de acuerdo a la configuración regional especificada. Su sintaxis es:

```
<fmt:formatNumber value="expression" type="type" pattern="expression" currencyCode="expression"
currencySymbol="expression" maxIntegerDigits="expression" minIntegerDigits="expression"
maxFractionDigits="expression" minFractionDigits="expression" groupingUsed="expression"
var="name" scope="scope"/>
```

value: Número al que se dará formato. Este atributo es el único obligatorio.

type: Indica el tipo de formato con el que se tratará el número. Los valores posibles son: "number", "currency" o "percent". El valor por defecto es "number".

pattern: Este atributo tiene prioridad sobre "type". Es un patrón basado en las convenciones de la clase `java.text.DecimalFormat`.

currencyCode: Cuando el atributo "type" tiene como valor "currency", se pueden utilizar este atributo para especificar explícitamente la moneda a utilizar. Los valores posibles se basan en la norma ISO de definición de monedas. Puede ser una instancia de la clase `java.util.Currency`.

currencySymbol: Sirve para explicitar el símbolo de la moneda. El atributo `currencyCode` tiene prioridad sobre este atributo.

maxIntegerDigits, minIntegerDigits, maxFractionDigits y minFractionDigits: Estos atributos se utilizan para controlar el número de dígitos desplegados antes y después del punto decimal. El valor a especificar debe ser número entero.

groupingUsed: Es un valor booleano y controla si los dígitos antes del punto decimal se agrupan. Su valor por defecto es "true".


var: Nombre de la variable que referencia al formato que se declara.

scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

Veamos un ejemplo,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Formato de Números</h1>
<h3>Número 712.03</h3>
<c:set var="val1" value="712.03" />
<fmt:setLocale value="en_US"/>
<p>Formato (en_US): <fmt:formatNumber value="${val1}" /></p>
<fmt:setLocale value="es_AR"/>
<p>Formato (es_AR): <fmt:formatNumber value="${val1}" /></p>
<fmt:setLocale value="de_DE"/>
<p>Formato (de_DE): <fmt:formatNumber value="${val1}" /></p>
<br>
<h3>Porcentaje 63.5</h3>
<c:set var="val2" value="63.5" />
<fmt:setLocale value="en_US"/>
<p>Porcentaje (en_US): <fmt:formatNumber value="${val2}" type="percent" /></p>
<fmt:setLocale value="es_AR"/>
<p>Porcentaje (es_AR): <fmt:formatNumber value="${val2}" type="percent" /></p>
<fmt:setLocale value="de_DE"/>
<p>Porcentaje (de_DE): <fmt:formatNumber value="${val2}" type="percent" /></p>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

<br>
<h3>Moneda 631.09</h3>
<c:set var="val3" value="631.09" />
<fmt:setLocale value="en_US"/>
<p>Moneda (en_US): <fmt:formatNumber value="{val3}" type="currency" /></p>
<fmt:setLocale value="es_AR"/>
<p>Moneda (es_AR): <fmt:formatNumber value="{val3}" type="currency" /></p>
<fmt:setLocale value="de_DE"/>
<p>Moneda (de_DE): <fmt:formatNumber value="{val3}" type="currency" /></p>
<h3>Uso de un patrón 123456789</h3>
<c:set var="val4" value="123456789" />
<p>###.###E0: <fmt:formatNumber value="{val4}" pattern="###.###E0" /></p>
</body>
</html>

```

La etiqueta `<fmt:parseNumber>` permite analizar un número proporcionado a través del atributo "value" o en el cuerpo de la etiqueta según la configuración regional especificada. Su resultado es una instancia de la clase `java.lang.Number`. Presenta dos posibles sintaxis:

Sintaxis 1

```
<fmt:parseNumber value="expression" type="type" pattern="expression" parseLocale="expression"
integerOnly="expression" var="name" scope="scope"/>
```

Sintaxis 2

```
<fmt:parseNumber type="type" pattern="expression" parseLocale="expression"
integerOnly="expression" var="name" scope="scope">
  ParseNumber-Body
</fmt:parseNumber>
```

value: Número que se analizará. Este atributo es el único obligatorio.

type: Indica cómo se tratará al número. Los valores posibles son: "number", "currency" o "percent". El valor por defecto es "number".

pattern: Este atributo tiene prioridad sobre "type". Es un patrón basado en las convenciones de la clase `java.text.DecimalFormat`.

parseLocale: Permite definir la configuración regional que se utilizará. Debe ser una cadena con el nombre de la localización seleccionada o una instancia de la clase `java.util.Locale`.

integerOnly: Es un valor booleano e indica si solo debe analizarse la parte entera del número, ignorando los dígitos decimales. Su valor por defecto es "false".

var: Nombre de la variable que referencia al objeto `Number` creado.

scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

ParseNumber-Body: Cuando se define no se debe utilizar el atributo "value", el valor especificado debe ser según la definición de este último.


Veamos un ejemplo sencillo,

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Parseo de Números</h1>
<h3>Parseo de moneda</h3>
<fmt:parseNumber var="val1" type="currency" value="$123456,789" parseLocale="es_AR" />
<p>${val1}</p>

```


	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

<br>
<h3>Parseo de enteros</h3>
<fmt:parseNumber var="val2" value="123456,789" integerOnly="true" parseLocale="es_AR" />
<p>${val2}</p>
<br>
<h3>Parseo de números</h3>
<fmt:parseNumber var="val3" value="123456,789" parseLocale="es_AR" />
<p>${val3}</p>
<br>
<h3>Parseo de porcentajes</h3>
<fmt:parseNumber var="val4" type="percent" value="123456,789%" parseLocale="es_AR" />
<p>${val4}</p>
</body>
</html>

```

Texto

La posibilidad de internacionalizar texto se logra a través de la etiqueta `<fmt:message>`. Esta etiqueta permite recuperar mensajes de texto desde un paquete de recursos teniendo en cuenta la configuración regional especificada para mostrarlo en una página JSP.

Los paquetes de recursos para almacenar los mensajes de acuerdo a una configuración regional pueden ser una clase o archivo de propiedades cuyo nombre se adhiere a una determinada nomenclatura, en la que se combina un nombre de archivo más la configuración regional. Por ejemplo, podríamos tener el archivo `MisMensajes_es.properties` para mensajes en español y `MisMensajes_en.properties` para mensajes en inglés. Hasta llegado el caso se puede especificar para un determinado país, `MisMensajes_es_AR.properties`.

En cada uno de estos archivos se definen las mismas propiedades pero los valores de estas propiedades se pueden personalizar por idioma o dialecto según corresponda. A cada propiedad se le pueden especificar parámetros que se definen secuencialmente comenzando con cero y encerrando el número entre llaves.

Por ejemplo,

`MisMensajes_es.properties`

```

saludo=iHola {0}!!!
mensaje=iQué tenga un buen día!!! :)

```

`MisMensajes_en.properties`

```

saludo=Hi {0}!!!
mensaje=Have a nice day!!! :)

```

Nota: Estos archivos serán almacenados en un paquete asociado al proyecto con el nombre "properties".

El primer paso para mostrar dicho contenido es especificar el paquete de recursos. La librería FMT posee dos etiquetas para lograr esto `<fmt:setBundle>` y `<fmt:bundle>`.

La etiqueta `<fmt:setBundle>` establece un paquete de recursos por defecto para ser utilizados por la etiqueta `<fmt:message>` dentro de un alcance determinado. Mientras que la etiqueta `<fmt:bundle>` establece el paquete de recursos para ser usado dentro del cuerpo de la etiqueta, por lo que las etiquetas `<fmt:message>` se encuentran anidadas dentro de esta.

```
<fmt:setBundle basename="expression" var="name" scope="scope"/>
```

basename: Atributo obligatorio que permite identificar el paquete de recursos a utilizar. Debe tenerse en cuenta que el nombre a especificar no debe contener la configuración regional, es decir, en nuestro ejemplo se debe especificar solamente "properties.MisMensajes"

var: Nombre de la variable que referencia al paquete de recursos.


scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente.

Por su parte la etiqueta `<fmt:bundle>` presenta la siguiente sintaxis:

```

<fmt:bundle basename="expression" prefix="expression">
  Bundle-Body
</fmt:bundle>

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

basename: Atributo obligatorio que permite identificar el paquete de recursos a utilizar. Debe tenerse en cuenta que el nombre a especificar no debe contener la configuración regional, es decir, en nuestro ejemplo se debe especificar solamente "properties.MisMensajes"

prefix: Atributo opcional que permite definir un prefijo para utilizar en las etiquetas <fmt:message> anidadas.

Una vez que se ha definido el paquete de recursos a utilizar, las propiedades se acceden con la etiqueta <fmt:message>. La misma presenta dos sintaxis posibles:

Sintaxis 1

```
<fmt:message key="expression" bundle="expression" var="name" scope="scope"/>
```

Sintaxis 2

```
<fmt:message key="expression" bundle="expression" var="name" scope="scope">
  <fmt:param value="expression"/>
  ...
</fmt:message>
```

key: Atributo obligatorio que permite especificar la propiedad del paquete de recursos a mostrar.

bundle: Sirve para definir un paquete de recursos específico en donde buscar el mensaje asociado a la propiedad indicada en el atributo "key". Su valor debe ser la variable definida a través de la etiqueta <fmt:setBundle> o el prefijo de la etiqueta <fmt:bundle>.

var: Nombre de la variable que contiene el mensaje de texto generado por la etiqueta. Al usar este atributo no se imprime el mensaje dentro de la página JSP.

scope: Ámbito de visibilidad de la variable definida. Los valores posibles son los ya estudiados anteriormente. Se utiliza la etiqueta <fmt:param> para proporcionar los valores de los parámetros del mensaje de texto. Cabe destacar que es importante el orden en que se define cada una de estas etiquetas dentro de <fmt:message>.


Veamos un ejemplo,

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
  <fmt:setLocale value="en_US" />
  <fmt:setBundle basename="properties.MisMensajes" var="etq" />
  <h1>
    <fmt:message key="saludo" bundle="${etq}">
      <fmt:param value="Mariela" />
    </fmt:message>
    <fmt:message key="mensaje" bundle="${etq}" />
  </h1>
  <br><br>
  <fmt:setLocale value="es_AR" />
  <fmt:bundle basename="properties.MisMensajes">
    <h1>
      <fmt:message key="saludo">
        <fmt:param value="Mariela" />
      </fmt:message>
      <fmt:message key="mensaje" />
    </h1>
  </fmt:bundle>
</body>
</html>
```

SQL Tag Library

Esta cuarta librería JSTL permite interactuar con bases de datos relacionales.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

Está recomendada para prototipos rápidos o aplicaciones pequeñas. El empleo de acceso directo a la base de datos desde la capa de presentación es altamente desaconsejado en sistemas en producción o en aplicaciones medianas o grandes. El beneficio de mantener separada la lógica de negocio y acceso a los datos de la presentación se ve reflejado en un mantenimiento más sencillo y una mayor adaptabilidad de la aplicación a cambios futuros.

Fuente de datos

Las fuentes de datos (datasources) son fábricas para la obtención de conexiones a bases de datos. Los servidores de aplicaciones con Java 2 Enterprise Edition (J2EE) suelen proporcionar compatibilidad integrada para fuentes de datos que son puestas a disposición de las aplicaciones J2EE a través de Java Naming y Directory Interface (JNDI). Las etiquetas de la librería SQL confían en estas fuentes para obtener conexión. Varias etiquetas incluyen un atributo opcional "dataSource" para especificar explícitamente la conexión que puede ser una instancia de la interfaz `javax.sql.DataSource` o con un nombre JNDI.

Para obtener una instancia de la clase `javax.sql.DataSource` se utiliza la etiqueta `<sql:setDataSource>`, que tiene dos posibles sintaxis:

Sintaxis 1

```
<sql:setDataSource dataSource="expression" var="name" scope="scope"/>
```

Sintaxis 2

```
<sql:setDataSource url="expression" driver="expression" user="expression" password="expression" var="name" scope="scope"/>
```

En la sintaxis 1, el atributo "dataSource" es obligatorio. Para acceder a una fuente de datos se utiliza un nombre JNDI.

Mientras que en la sintaxis 2, el atributo "url" es obligatorio, donde debe especificarse una URL JDBC. A su vez, el atributo "driver" indica el nombre de la clase que implementa el controlador de base de datos, mientras que los atributos "user" y "password" proporcionan la credencial para el acceso a la base de datos.

Para cualquiera de las sintaxis de la etiqueta `<sql:setDataSource>`, son válidos los atributos "var" y "scope" cuyo significado es igual a todas las etiquetas hasta aquí estudiadas.

Consultas y actualizaciones

Una vez establecida la conexión, se puede utilizar la etiqueta `<sql:query>` para ejecutar consultas mientras que las actualizaciones se realizan mediante la etiqueta `<sql:update>`. Las consultas y actualizaciones se especifican como cualquier sentencia SQL, que puede ser parametrizada utilizando la interfaz `java.sql.PreparedStatement` JDBC. Para asignar los valores de los parámetros se utilizan las etiquetas `<sql:param>` y `<sql:dateParam>`.

Existen tres posibles sintaxis para la etiqueta `<sql:query>`:

Sintaxis 1

```
<sql:query sql="expression" dataSource="expression" var="name" scope="scope" maxRows="expression" startRow="expression"/>
```

Sintaxis 2


```
<sql:query sql="expression" dataSource="expression" var="name" scope="scope" maxRows="expression" startRow="expression">  
<sql:param value="expression"/>  
...  
</sql:query>
```

Sintaxis 3

```
<sql:query dataSource="expression" var="name" scope="scope" maxRows="expression" startRow="expression">  
SQL statement  
<sql:param value="expression"/>  
...  
</sql:query>
```

Tanto en la sintaxis 1 y 2, los atributos "sql" y "var" son obligatorios, mientras que en la sintaxis 3 solo el atributo "var" es obligatorio.

Los atributos "var" y "scope" permiten almacenar el resultado de la consulta bajo un determinado alcance. El atributo "maxRows" se utiliza para limitar la cantidad de filas devueltas por la consulta, mientras que el atributo "startRow" permite definir desde qué fila se comenzará a construir el resultado ignorando las filas anteriores.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

La variable generada es una instancia de la interfaz `javax.servlet.jsp.jstl.sql.Result`; este objeto proporciona las propiedades para el acceso a las filas, los nombres de las columnas, etc. A continuación se presenta un resumen de cada propiedad:

Propiedad	Definición
<code>rows</code>	Un vector de objetos <code>SortedMap</code> donde cada uno de ellos es un map de nombres de columna de una fila del resultado
<code>rowsByIndex</code>	Una matriz donde cada vector se corresponde con una fila del resultado
<code>columnNames</code>	Un vector con los nombres de columna del resultado basándose en el mismo orden de la propiedad <code>rowsByIndex</code>
<code>rowCount</code>	El número total de filas del resultado de la consulta
<code>limitedByMaxRows</code>	Su valor es verdadero si la consulta fue limitada por el atributo <code>"maxRows"</code>

Dentro de una etiqueta `<sql:query>` se puede especificar la sentencia SQL en el cuerpo de la misma o en el atributo `"sql"`. La instrucción SQL puede ser parametrizada con el caracter `"?"`. Para cada parámetro, debe haber una etiqueta `<sql:param>` o `<sql:dateParam>` que deben estar incluidas en el cuerpo de la etiqueta `<sql:query>`.

La etiqueta `<sql:param>` contiene un único atributo `"value"` que sirve para especificar el valor del parámetro. Cuando el valor del parámetro sea una cadena de caracteres, se puede omitir el atributo `"value"` y proporcionar dicho valor en el cuerpo de la etiqueta.

Para especificar parámetros que sean fechas, horas o ambas, se utiliza la etiqueta `<sql:dateParam>` con la siguiente sintaxis:

`<sql:dateParam value="expression" type="type"/>`

Siendo el atributo `"value"` una instancia de la clase `java.util.Date`, mientras que el atributo `"type"` debe ser `"date"`, `"time"` o `"timestamp"`.

Al igual que la etiqueta `<sql:query>`, la etiqueta `<sql:update>` presenta tres posibles sintaxis:

Sintaxis 1

`<sql:update sql="expression" dataSource="expression" var="name" scope="scope"/>`

Sintaxis 2

**`<sql:update sql="expression" dataSource="expression" var="name" scope="scope">`
`<sql:param value="expression"/>`
`...`
`</sql:update>`**

Sintaxis 3

`<sql:update dataSource="expression" var="name" scope="scope">`
`SQL statement`
`<sql:param value="expression"/>`
`...`
`</sql:update>`

Tanto el atributo `"sql"` como `"dataSource"` tienen la misma semántica que la etiqueta `<sql:query>`. Pero en este caso, la variable que se genera bajo un determinado alcance, es una instancia de la clase `java.lang.Integer` que contiene la cantidad de filas afectadas por la ejecución de la actualización.

Transacciones


Se utiliza la etiqueta `<sql:transaction>` que presenta la siguiente sintaxis:

`<sql:transaction dataSource="expression" isolation="isolationLevel">`
`<sql:query .../>` or **`<sql:update .../>`
`...`
`</sql:transaction>`**

Ningún atributo es obligatorio. El atributo `"isolation"` se utiliza para especificar el nivel de aislamiento de la transacción que puede ser: `"read_committed"`, `"read_uncommitted"`, `"repeatable_read"` o `"serializable"`.

Como era de esperarse, todas las consultas y actualizaciones deben utilizar el mismo `"dataSource"`, por lo que este atributo no puede ser especificado en las etiquetas `<sql:query>` y `<sql:update>` si están dentro de la etiqueta `<sql:transaction>`.

Veamos un ejemplo completo,

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>


<!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=UTF-8">
<title>Ejemplo</title>
</head>
<body>
<h1>Herramienta de SQL</h1>
<form name="sql" action="index.jsp" method="post" target="_top">
<textarea name="query" rows="10" cols="109"></textarea>
<br><br>
<input type="submit" value="Ejecutar">
</form>
<hr style="border: 1px solid #000000">
<c:set var="query" value="${param.query}" scope="request" />
<c:catch var="exception">
<c:if test="${!empty query}">
<sql:setDataSource var="dataSource" driver="com.microsoft.sqlserver.jdbc.SQLServerDriver"
url="jdbc:sqlserver://bilbo;databaseName=das;" user="sa" password="sa" />

<c:choose>
<c:when test="${fn:indexOf(fn:toLowerCase(query), 'select') >= 0}">
<sql:query var="result" dataSource="${dataSource}">
${query}
</sql:query>

<c:choose>
<c:when test="${result.rowCount == 0}">
<p>La consulta no retornó filas</p>
</c:when>
<c:otherwise>
<c:forEach items="${result.rowsByIndex}" var="row" varStatus="s">
<c:if test="${s.first}">
<table border="1" cellpadding="1" cellspacing="0">
<tr>
<c:forEach items="${result.columnNames}" var="col">
<th><c:out value="${col}" /></th>
</c:forEach>
</tr>
</table>
</c:if>
<tr>
<c:forEach items="${row}" var="value">
<td><c:out value="${value}" /></td>
</c:forEach>
</tr>
<c:if test="${s.last}">
</table>
</c:if>
</c:forEach>
</c:otherwise>
</c:choose>
</c:when>
<c:otherwise>
<sql:transaction dataSource="${dataSource}" isolation="read_committed">
<sql:update var="updateCount">
${query}
</sql:update>
<p><c:out value="${updateCount}" /> filas afectadas</p>
</sql:transaction>
</c:otherwise>
</c:choose>

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE ARQUITECTURA MODELO 1	VERSIÓN: 1.2 VIGENCIA: 06-08-2009

```

    </c:if>
  </c:catch>
  <c:if test="${!empty exception}">
    <c:out value="${exception}"/>
  </c:if>
</body>
</html>

```