



CC68J APLICACIONES EMPRESARIALES CON JEE

JSP: JAVA SERVER PAGES

Dejando los Servlets un poco de lado...

Profesores:
■ Andrés Farías



v1.0

Objetivos

- Introducción general y sintaxis.
- Conocer 3 tipos de elementos:
 - ✓ Expresiones JSP.
 - ✓ Directivas JSP.
 - ✓ Acciones JSP.
- EL: Simple Expression Language.
- Páginas de errores.

v1.1

Java Server Pages

INTRODUCCIÓN Y SINTAXIS

v1.1

Introducción

¿Qué es y para qué sirve?

- JSP es una tecnología que permite combinar código HTML estático con código generado dinámicamente en un mismo archivo.
- Ventajas:
 - ✓ Separación de datos estáticos/dinámicos.
 - ✓ Independencia de formato/plataforma.
 - ✓ Sencillez (sabiendo servlets)
- Las JSP nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático.
- Simplemente escribimos el HTML regular de la forma normal y encerramos el código de las partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empezarán con "<%>" y terminan con "%>".

Introducción

Las JSP

- Las JSP se almacenan en archivos de extensión **.jsp**.
- Un JSP se traduce en un Servlet que se encarga de responder las peticiones asociadas a la página JSP.
- Una JSP tiene 3 tipos de elementos:
 - ✓ Elementos script (scriptlets)
 - ✓ Directivas
 - ✓ Acciones

Introducción

JSP 2.0

- Algunas nuevas características de JSP 2.0 frente a 1.2 orientadas a simplificar su desarrollo son:
 - ✓ Simple Expression Language (EL)
 - ✓ Fragmentos JSP
 - ✓ Ficheros de Tags
 - ✓ Manejadores de Etiquetas Simplificados

Sintaxis

Scriptlets JSP

- **Expresión JSP** `<%= expression %>`
 - ✓ La Expresión es evaluada y situada en la salida.
 - ✓ El equivalente XML es:


```
<jsp:expression> expression </jsp:expression>
```
 - ✓ Las variables predefinidas son `request`, `response`, `out`, `session`, `application`, `config`, y `pageContext`.
- **Scriptlet JSP** `<% code %>`
 - ✓ El código se inserta en el método `service`.
 - ✓ El equivalente XML es:


```
<jsp:scriptlet>code</jsp:scriptlet>
```

Sintaxis

Directivas JSP

- **Declaración JSP** `<%! code %>`

El código se inserta en el cuerpo de la clase del servlet, fuera del método `service`.

El equivalente XML es:

```
<jsp:declaration> code </jsp:declaration>
```
- **Directiva page JSP** `<%@ page att="val" %>`

Dirige al motor servlet sobre la configuración general.

El equivalente XML es:

```
<jsp:directive.page att="val">
```

Los atributos legales:

```
import="package.class"
contentType="MIME-Type"
isThreadSafe="true|false"
```

Sintaxis

Directivas JSP

```
session="true|false"
buffer="sizekb|none"
autoFlush="true|false"
extends="package.class"
info="message"
errorPage="url"
isErrorPage="true|false"
language="java"
```

- **Directiva include JSP** `<%@ include file="url" %>`

Un fichero del sistema local se incluirá cuando la página se traduzca a un Servlet.

El equivalente XML es: `<jsp:directive.include file="url">`

Sintaxis

Acciones

Comentario JSP

```
<%-- comment --%>
```

Comentario ignorado cuando se traduce la página JSP en un servlet.

Si queremos un comentario en el HTML resultante, usamos la sintaxis de comentario normal del HTML `<!-- comment -->`.

Acción `<jsp:include>`

```
<jsp:include page="relative URL" flush="true"/>
```

Incluye un fichero en el momento en que la página es solicitada.

Aviso: en algunos servidores, el fichero incluido debe ser un fichero HTML o JSP, según determine el servidor (normalmente basado en la extensión del fichero).

Sintaxis

Acciones sobre beans

Acción `<jsp:useBean>`

```
<jsp:useBean att=val*>
```

```
<jsp:useBean att=val*>
```

```
...
```

```
</jsp:useBean>
```

Encuentra o construye un Java Bean.

Los posibles atributos son:

```
id="name"
scope="page|request|session|application"
class="package.class"
type="package.class"
beanName="package.class"
```

Sintaxis

Acciones sobre beans

Acción `<jsp:setProperty>`

```
<jsp:setProperty att=val*>
```

Selecciona las propiedades del bean, bien directamente o designando el valor que viene desde un parámetro de la petición.

Los atributos legales son:

```
name="beanName"
property="propertyName|*"
param="parameterName"
value="val"
```

Acción `<jsp:getProperty>`

```
<jsp:getProperty name="propertyName" value="val"/>
```

Recupera y saca las propiedades del Bean.

Sintaxis

Acciones sobre beans

- Acción `jsp:forward`

```
<jsp:forward page="relative URL"/>
```

Reenvía la petición a otra página.

- Acción `jsp:plugin`

```
<jsp:plugin attribute="value"*> ... </jsp:plugin>
```

Genera etiquetas OBJECT o EMBED, apropiadas al tipo de navegador, pidiendo que se ejecute un applet usando el Java Plugin.

SCRIPTLETS JSP

Expresiones

Sintaxis

- Sintaxis: `<%= <JavaExpresión> %>`

- Se evalúan y se insertan en la salida.

- Se tiene acceso a variables:

- request, el `HttpServletRequest`
- response, el `HttpServletResponse`
- session, el `HttpSession` asociado con el request (si existe)
- out, el `PrintWriter` (una versión con buffer del tipo `JspWriter`) usada para enviar la salida al cliente.

- ✓ Ejemplo:

```
Your hostname: <%= request.getRemoteHost() %>
```

Expresiones

Sintaxis XML

- El equivalente en XML es usar una sintaxis alternativa para las expresiones JSP:

```
<jsp:expression>
  <JavaExpression>
</jsp:expression>
```

- Los elementos XML, al contrario que los del HTML, son sensibles a las mayúsculas.

Scriptlets

Sintaxis

- Sintaxis: `<% <JavaInstruction> %>` que se insertan dentro del método **service** del servlet.
- Tienen acceso a las mismas variables que las expresiones.
- El código dentro de un scriptlet se insertará **exactamente** como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias **print**. Esto significa que los scriptlets no necesitan completar las sentencias Java, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets.

Scriptlets

Sintaxis

- La sentencia:

```
<% if (Math.random() < 0.5) { %>
  Have a <B>nice</B> day!
<% } else { %>
  Have a <B>lousy</B> day!
<% } %>
```

- Se traducirá en:

```
if (Math.random() < 0.5) {
  out.println("Have a <B>nice</B> day!");
} else {
  out.println("Have a <B>lousy</B> day!");
}
```

Scriptlets

Sintaxis XML

- El equivalente XML de `<% código %>` es:

```
<jsp:scriptlet>
  <JavaInstruction>
</jsp:scriptlet>
```

- Si se quiere poder usar los caracteres "<%" dentro de un scriptlet, hay que usar "<%\"

Declaraciones

Sintaxis y uso

- Sintaxis: `<%! código %>` que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente, como la declaración de un miembro Java.
- Permite insertar métodos, variables...
- No generan salida alguna. Se usan combinadas con scriptlets.

```
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```

Declaraciones

Sintaxis XML

- Como con los scriptlet, si queremos usar los caracteres "<%", ponemos "<%\"
- El equivalente XML de `<%! Código %>` es:

```
<jsp:declaration>
  Código
</jsp:declaration>
```

Configurando el Servlet

DIRECTIVAS JSP

Directivas

Sintaxis y conceptos generales

- Afecta a la estructura general de la clase servlet. Normalmente tienen la siguiente forma:

```
<%@ directive attribute="value" %>
```

- También podemos combinar múltiples selecciones de atributos para una sola directiva:

```
<%@ directive attribute1="value1"
      attribute2="value2"
      ...
      attributeN="valueN" %>
```

- Se reconocen las siguientes directivas:

- ✓ Page
- ✓ include

Directivas

Sintaxis XML

- La sintaxis XML para definir directivas es:

```
<jsp:directive.TipoDirectiva atributo=valor />
```

- Por ejemplo, el equivalente XML de:

```
<%@ page import="java.util.*" %>
```

es:

```
<jsp:directive.page import="java.util.*" />
```


Directiva: page

Atributos: import & contentType

■ Import:

- ✓ `import="package.class" o import="package.class1,...,package.classN".`
- ✓ Esto permite especificar los paquetes que deberían ser importados.
- ✓ La directiva `import` es la única que puede aparecer múltiples veces.

■ Tipos de media:

- ✓ `ContentType = "MIME-Type" o contentType = "MIME-Type; charset = Character-Set"`
- ✓ Esto especifica el tipo MIME de la salida.
- ✓ El valor por defecto es `text/html`. Tiene el mismo valor que el scriptlet usando `response.setContentType`.

Directiva: page

Atributos: Threads & Sesiones

■ Threads

- ✓ `isThreadSafe="true|false".`
- ✓ Un valor de `true` (por defecto) indica un procesamiento del servlet normal, donde múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet, bajo la suposición que el autor sincroniza los recursos compartidos.
- ✓ Un valor de `false` indica que el servlet debería implementar `SingleThreadModel`.

■ Sesión:

- ✓ `session="true|false".`
- ✓ Un valor de `true` (por defecto) indica que la variable predefinida `session` (del tipo `HttpSession`) debería unir a la sesión existente si existe una, si no existe se debería crear una nueva sesión para unirla.
- ✓ Un valor de `false` indica que no se usarán sesiones, y los intentos de acceder a la variable `session` resultarán en errores en el momento en que la página JSP sea traducida a un servlet.

Directiva: page

Atributos: buffer & autoflush

■ Buffer:

- ✓ `buffer="sizekb|none".`
- ✓ Esto especifica el tamaño del buffer para el `JspWriter out`. El valor por defecto es específico del servidor y debería ser de al menos 8kb.

■ Autoflush:

- ✓ `autoflush="true|false".`
- ✓ Un valor de `true` (por defecto) indica que el buffer debería descargarse cuando esté lleno.
- ✓ Un valor de `false`, raramente utilizado, indica que se debe lanzar una excepción cuando el buffer se sobrecargue.
- ✓ Un valor de `false` es ilegal cuando usamos `buffer="none".`

Directiva: page**Atributos: Extends, Info & ErrorPage**

- **Extends:**
 - ✓ `extends="package.class"`.
 - ✓ Esto indica la superclase del servlet que se va a generar.
 - ✓ Debemos usarla con extrema precaución, ya que el servidor podría utilizar una superclase personalizada.
- **Info:**
 - ✓ `info="message"`.
 - ✓ Define un string que puede usarse para ser recuperado mediante el método `getServletInfo`.
- **ErrorPage:**
 - ✓ `errorPage="url"`.
 - ✓ Especifica una página JSP que se debería procesar si se lanzará cualquier **Throwable** pero no fuera capturado en la página actual.

Directiva: page**Atributos: isErrorPage & Language**

- **isErrorPage:**
 - ✓ `isErrorPage="true|false"`.
 - ✓ Indica si la página actual actúa o no como página de error de otra página JSP.
 - ✓ El valor por defecto es false.
- **Language:**
 - ✓ `language="java"`.
 - ✓ En algunos momentos, esto está pensado para especificar el lenguaje a utilizar.
 - ✓ Por ahora, no debemos preocuparnos por él ya que java es tanto el valor por defecto como la única opción legal.

Directiva: include**Atributos: isErrorPage & Language**

- Permite incluir ficheros en el momento en que la página JSP es traducida a un servlet.


```
<%@ include file="<urlRelativa>" %>
```
- Los contenidos del fichero incluido son analizados como texto normal JSP y así pueden incluir HTML estático, elementos de script, directivas y acciones.
- **Uso típico:**
 - ✓ Barras de navegación.
 - ✓ Encabezados / Pie de páginas.

Para usar en las acciones...

VARIABLES PREDEFINIDAS

Variables predefinidas

La petición: request

- Nombre: **request**.
- Este es el **HttpServletRequest** asociado con la petición, y nos permite:
 - ✓ Mirar los parámetros de la petición (mediante **getParameter**).
 - ✓ El tipo de petición (**GET**, **POST**, **HEAD**, etc.).
 - ✓ Y las cabeceras HTTP entrantes (**cookies**, **Referer**, etc.).
- Estrictamente hablando, se permite que la petición sea una subclase de **ServletRequest** distinta de **HttpServletRequest**, si el protocolo de la petición es distinto del HTTP. Esto casi nunca se lleva a la práctica.

Variables predefinidas

La respuesta: response

- Nombre: **response**.
- Este es el **HttpServletResponse** asociado con la respuesta al cliente.
- Como el stream de salida tiene un buffer, es legal seleccionar los códigos de estado y cabeceras de respuesta, aunque no está permitido en los servlets normales una vez que la salida ha sido enviada al cliente.

Variables predefinidas

La salida: out

- Nombre: **out**.
- Este es el **PrintWriter** usado para enviar la salida al cliente.
- Sin embargo, para poder hacer útil el objeto **response** esta es una versión con buffer de **PrintWriter** llamada **JspWriter**.
- Podemos ajustar el tamaño del buffer, o incluso desactivar el buffer, usando el atributo **buffer** de la directiva **page**.
- Se usa casi exclusivamente en scriptlets ya que las expresiones JSP obtienen un lugar en el stream de salida, y por eso raramente se refieren explícitamente a **out**.

Variables predefinidas

La sesión: session

- Nombre: **session**.
- Este es el objeto **HttpSession** asociado con la petición.
- Las sesiones se crean automáticamente, por esto esta variable se une incluso si no hubiera una sesión de referencia entrante.
- La única excepción es usar el atributo **session** de la directiva **page** para desactivar las sesiones, en cuyo caso los intentos de referenciar la variable **session** causarán un error en el momento de traducir la página JSP a un servlet.

Variables predefinidas

Varios: application, Config & PageContext

- Nombre: **application**.
 - ✓ El **ServletContext** obtenido mediante el método **getServletConfig().getContext()**.
- Nombre: **config**.
 - ✓ El objeto **ServletConfig**.
- Nombre: **pageContext**.
 - ✓ JSP presenta una nueva clase llamada **PageContext** para encapsular características de uso específicas del servidor como **JspWriters** de alto rendimiento.
 - ✓ La idea es que si tenemos acceso a ellas a través de esta clase en vez directamente, nuestro código seguirá funcionando en motores **servlet/JSP** "normales".
- Nombre **page**.
 - ✓ Esto es sólo un sinónimo de **this**, y no es muy útil en Java.
 - ✓ Fue creado como situación para el día que los lenguajes de script puedan incluir otros lenguajes distintos de Java.

ACCIONES JSP

Acciones

Conceptos Generales

- Usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets.
- Podemos insertar un fichero dinámicamente, utilizar componentes JavaBeans, reenviar al usuario a otra página, o generar HTML para el plug-in Java.

Acciones

include

- **jsp:include** nos permite insertar ficheros en una página que está siendo generada.
- La sintaxis se parece a esto:

```
<jsp:include page="relative URL" flush="true" />
```
- Al contrario que la directiva **include**, que inserta el fichero en el momento de la conversión a un Servlet, ésta inserta el fichero cuando la página es solicitada.
 - ✓ Se pierde eficiencia, e imposibilita a la página incluida contener código JSP general pero se obtiene gran flexibilidad.
 - ✓ Uso: Noticias...

Acciones: useBean

Sintaxis

- `jsp:useBean` permite cargar y utilizar un **JavaBean** en la página JSP y así utilizar la reusabilidad de las clases Java sin sacrificar la conveniencia de añadir JSP sobre servlets solitarios.

```
<jsp:useBean id="name" class="package.class" />
```

- Esto normalmente significa "usa un objeto de la clase especificada por el atributo **class**, y únelo (*bind*) a una variable con el nombre especificado por **id**."
- Ahora podemos modificar sus propiedades mediante `jsp:setProperty`, o usando un scriptlet y llamando a un método de **id**.
- Para recoger una propiedad se usa `jsp:getProperty`.

Acciones: useBean

Uso

- La forma más sencilla de usar un Bean es usar:

```
<jsp:useBean id="name" class="package.class" />
```

- Pero si hacemos

```
<jsp:useBean ...> Body </jsp:useBean>
```

La porción **Body** sólo se debería ejecutar cuando el bean es instanciado por primera vez, no cuando un bean existente se encuentre y se utilice.

- No todas las sentencias `jsp:useBean` resultan en la instanciación de un Bean.

Acciones: useBean

Atributos

- **id**
 - ✓ Da un nombre a la variable que referencia al bean.
 - ✓ Se usará un objeto bean anterior en lugar de instanciar uno nuevo si se puede encontrar uno con el mismo **id** y **scope**.
- **class**
 - ✓ Designa el nombre completo de la clase del bean.
- **scope**
 - ✓ Indica el contexto en el que el bean debería estar disponible.
 - ✓ Hay cuatro posibles valores: **page**, **request**, **session**, y **application**.
- **type**
 - ✓ Especifica el tipo de la variable a la que se referirá el objeto.
- **beanName**
 - ✓ Da el nombre del bean, como lo suministraríamos en el método `newInstance` de Beans.
 - ✓ Esta permitido suministrar un **type** y un **beanName**, y omitir el atributo **class**.

Acciones: setProperty

Uso

- Se utiliza para obtener valores de propiedades de los beans que se han referenciado anteriormente.
- 2 usos:
 - ✓ Después de un `useBean`.

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName"
    property="someProperty" ... />
```

Se ejecuta siempre que haya una solicitud.

Acciones: setProperty

Uso (2/2)

- Dentro de un `useBean`

```
<jsp:useBean id="myName" ... >
...
    <jsp:setProperty name="myName"
        property="someProperty" ... />
</jsp:useBean>
```

- Sólo se ejecuta cuando haya que instanciar un bean.

Acciones: setProperty

Atributos (1/2)

- name**
 - ✓ Este atributo requerido designa el bean cuya propiedad va a ser seleccionada. El elemento `jsp:useBean` debe aparecer antes del elemento `jsp:setProperty`.
- property**
 - ✓ Este atributo requerido indica la propiedad que queremos seleccionar. Sin embargo, hay un caso especial: un valor de `*` significa que todos los parámetros de la petición cuyos nombres correspondan con nombres de propiedades del Bean serán pasados a los métodos de selección apropiados.
- value**
 - ✓ Este atributo opcional especifica el valor para la propiedad. Los valores string son convertidos automáticamente a lo que corresponda mediante el método estándar `valueOf`. No se pueden usar `value` y `param` juntos, pero si está permitido no usar ninguna.

Acciones: setProperty

Atributos (2/2)

param

- ✓ Este parámetro opcional designa el parámetro de la petición del que se debería derivar la propiedad.
- ✓ Si la petición actual no tiene dicho parámetro, no se hace nada: el sistema no pasa null al método seleccionador de la propiedad.
- ✓ Así, podemos dejar que el bean suministre los valores por defecto, sobrescribiéndolos sólo cuando el parámetro dice que lo haga.

```
<jsp:setProperty
    name="orderBean"
    property="numberOfItems"
    param="numItems" />
```

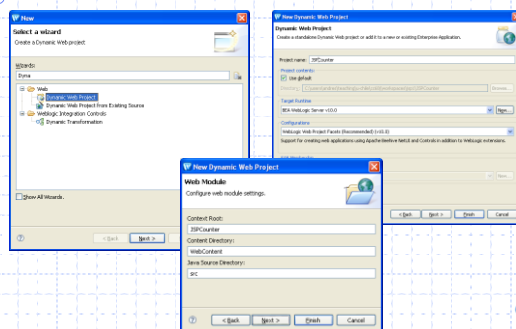
- ✓ Si no indicamos nada, el servidor revisa todos los parámetros de la petición e intenta encontrar alguno que concuerde con la propiedad indicada.

Con BEA Workshop

UN EJEMPLO SIMPLE DE JSP

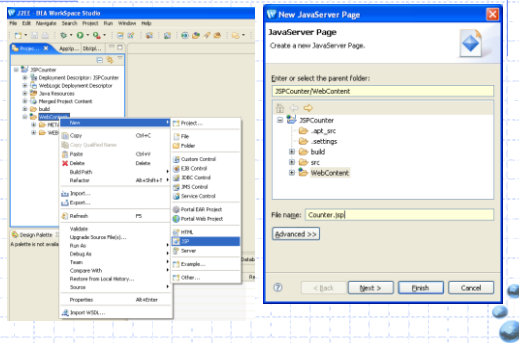
Ejemplo simple de JSP

Creación del proyecto web dinámico



Ejemplo simple de JSP

Wizard de creación de JSP



Ejemplo simple de JSP

Wizard de creación de JSP

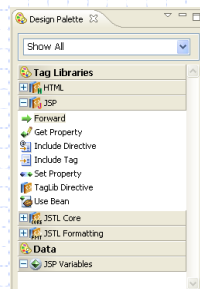


- Editor estándar con dos vistas: source y diseño.
- La vista (view) Outline refleja la estructura de la JSP.

Ejemplo simple de JSP

La uista de Paleta de diseño

- Permite acceder rápidamente a los principales elementos de una JSP (scriptlets, directivas y acciones).
- Se agregan a la JSP con drag & drop.
- Algunos elementos tienen asociados pequeños asistentes que facilitan el ingreso de parámetros.



Ejemplo simple de JSP

La vista de Paleta de diseño

- Se declara una variable de instancia en el Servlet para mantener la cuenta de accesos.
 - ✓ Se realiza mediante un elemento JSP de scriptlet.
 - ✓ `<%! private int counter = 0; %>`
- Se despliega el valor de esta variable, incrementándola simultáneamente.
 - ✓ Se realiza mediante un elemento JSP de scriptlet.
 - ✓ `<%= ++counter %>`
- La simpleza de la página supera ampliamente el trabajo necesario para generar el Servlet equivalente.



EL: SIMPLE EXPRESSION LANGUAGE

Simple Expression Language (SLE)

Introducción

- El *Simple Expression Language* está inspirada en los lenguajes de expresiones de ECMAScript y XPath.
 - ✓ Simplifican el uso de expresiones en JSPs.
 - ✓ Permite la ejecución de expresiones fuera de los elementos de scripting de JSP.
 - ✓ Fue introducida con JSTL 1.0 como un mecanismo alternativo al uso de expresiones en Java para asignar valores a atributos.
- Desde JSP 2.0 el Servlet Container entiende expresiones en EL.
- EL es mucho más tolerante sobre variables sin valor (null) y realiza conversiones automáticas de datos.
- Se puede habilitar (por defecto) o deshabilitar el uso de expresiones EL:
 - ✓ `<%@ page isScriptingEnabled="true|false" isELEnabled="true|false"%>`

Sintaxis EL

Expresiones: literales

- Una expresión en EL contiene variables y operadores

- `${expr}`, donde `expr` es:

- ✓ Literales:

- true o false
- Integer
- Floating point
- String
- null

- Ejemplos:

```
${false} <!-- evalúase to false -->
${3*8}
```

Sintaxis EL

Expresiones: operadores

- `${expr}`, donde `expr` es:

- ✓ Operadores:

- Aritméticos: +, -, *, /, div, %, mod, -
- Lógicos: and, &&, or, ||, !, not
- Relacionales: ==, eq, !=, ne, <, lt, <=, le, >=, ge
- Vacío, `empty`, valida si una variable es `null` o una colección no tiene elementos.
- Llamadas a función, donde `func` es el nombre de la función y `args` es 0 o más argumentos separados por comas.
- Condicional: A ? B : C, como en C y Java

- Ejemplos:

```
${ (6 * 5) + 5 } <!-- evalúa a 35 -->
${empty name}
```

Sintaxis EL

Expresiones: objetos implícitos

- `${expr}`, donde `expr` es:

- ✓ Objetos implícitos:

- `pageContext`
 - contexto del JSP, puede usarse para acceder a objetos implícitos como `request`, `response`, `session`, `out`, `ServletContext`, etc.
 - Ejemplo: `${pageContext.response}`
- `param`
 - `${param.name} <-> request.getParameter (name)`
- `paramValues`
 - `${paramValues.name} <-> request.getParameterValues(name)`
- `header`
 - `${header.name} <-> request.getHeader(name)`
- `headerValues`
 - `${headerValues.name} <-> request.getHeaderValues(name)`

Sintaxis EL

Expresiones: objetos implícitos

- `${expr}`, donde `expr` es:
 - ✓ **Objetos implícitos:**
 - `cookie`
 - `${cookie.name.value}` → devuelve el valor de la primera cookie con el nombre dado
 - `initParam`
 - `initParam.name` → `ServletContext.getInitParameter(String name)`
 - `pageScope` → puede acceder a objetos dentro del contexto de página
 - `${pageScope.objectName}`
 - `${pageScope.objectName.attributeName}`
 - `requestScope`
 - `${requestScope.objectName}`
 - `${requestScope.objectName.attributeName}`
 - `sessionScope`
 - `${sessionScope.name}`
 - `applicationScope`

Ejemplos EL

Expresiones simples y objetos implícitos

- `${amount + 5}`:
 - ✓ Añade 5 a una variable `amount` y despliega la suma.
 - ✓ Equivalente a `<%= amount + 5 =%>`
 - ✓ Más robusto: si `amount` no existe, despliega 5!
- `${order.amount + 5}`:
 - ✓ Añade 5 a la propiedad `amount` del bean `order`.
 - ✓ Lo mismo sería hacer: `${order['amount'] + 5}`
- Para asignar un valor dinámico a un atributo podemos hacer:


```
<input type="text" name="ip"
  value="${pageContext.request.remoteAddr}">
```

Ejemplos EL

Operador ?

```
<select name="artist">
  <option value="1" ${param.artist == 1 ? 'selected' : ''}>
    Vanessa Paradis
  </option>

  <option value="2" ${param.artist == 2 ? 'selected' : ''}>
    Van Tiersen
  </option>

  <option value="3" ${param.artist == 3 ? 'selected' : ''}>
    Dave Matthews Band
  </option>
</select>
```

PÁGINAS DE ERRORES

Páginas de Error JSP

Las clases de soporte

- Errores en servlets and JSPs desde su versión 2.0 vienen dadas en la variable `javax.servlet.error.exception`.
- La propiedad `errorData` del objeto implícito `pageContext` expone información sobre el problema.
- `javax.servlet.jsp.ErrorData` tiene las siguientes propiedades:
 - ✓ `requestURI` → la URI para la que falló la petición
 - ✓ `servletName` → el nombre del servlet o JSP que falló
 - ✓ `statusCode` → el código del fallo
 - ✓ `throwable` → la excepción que causó la invocación de la página de error.

Páginas de Error JSP

Ejemplo de Uso

```
<%@ page isErrorPage="true" contentType="text/html" %>
<%@ taglib prefix="log" uri="http://jakarta.apache.org/taglibs/log-1.0" %>
```

Disculpa, pero las cosas no funcionaron como estaban planeadas.
Registré tanto como supe del problema, asíque esté tranquilo que mi
Maestro mirará el error tan pronto como esté sobrio.

```
<jsp:useBean id="now" class="java.util.Date" />
<log:fatal>
  -----
  ${now}
  Petición que falló: ${pageContext.errorData.requestURI}
  código de status: ${pageContext.errorData.statusCode}
  Exception: ${pageContext.errorData.throwable}
  -----
</log:fatal>
```

Páginas de Error JSP

Declaración de la página de error

- En el **web.xml** se puede indicar que esta es la página de error mediante los siguientes elementos:

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

<error-page>
  <exception-code>500</exception-code>
  <location>/error.jsp</location>
</error-page>
...
```

web.xml en JSP 2.0

- Aunque se sigue utilizando **web.xml** hay dos diferencias principales:
 - ✓ Las reglas de **web.xml** están definidas en un XML Schema
 - ✓ Las configuraciones específicas de JSP han sido movidas a un nuevo elemento XML
- Usar Schemas en vez de DTDs nos permite colocar los elementos XML de primer nivel en cualquier orden y realizar aún mas verificaciones sobre la sintaxis de **web.xml**
- Todos los elementos específicos a la configuración de un JSP se encuentran agrupados dentro del nuevo elemento **<jsp-config>**.

Elemento jsp-config

- Su subelemento **<jsp-property-group>** es muy interesante.
- Permite aplicar configuraciones a un grupo de JSPs que conforman con un patrón de url específico

```
...
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</jsp-config>
...
```

Elemento jsp-config

Elemento	Descripción
<el-ignored>	Indica si ignorar o no las expresiones EL dentro de un JSP correspondiente al patrón URL indicado. Por defecto es false.
<scripting-invalid>	Si es true no se pueden introducir scriptlets en el JSP.
<page-encoding>	Indica la codificación de caracteres para un conjunto de JSPs
<include-coda>	Indica el path correspondiente a un fichero a añadir al final de cada página JSP.
<include-prelude>	Lo mismo pero al comienzo de cada JSP.
<is-xml>	Si se asigna a true la sintaxis de los JSPs es en XML.

EJERCICIO PROPUESTO

Ejercicio práctico JSP

- Crear una aplicación web empresarial J2EE (basta con el WAR) para implementar una aplicación que permita convertir de una moneda a otra.
- Debe aceptar como input una cantidad de pesos chilenos y desplegarlo en:
 - ✓ Dólares
 - ✓ Euros
 - ✓ UF's

