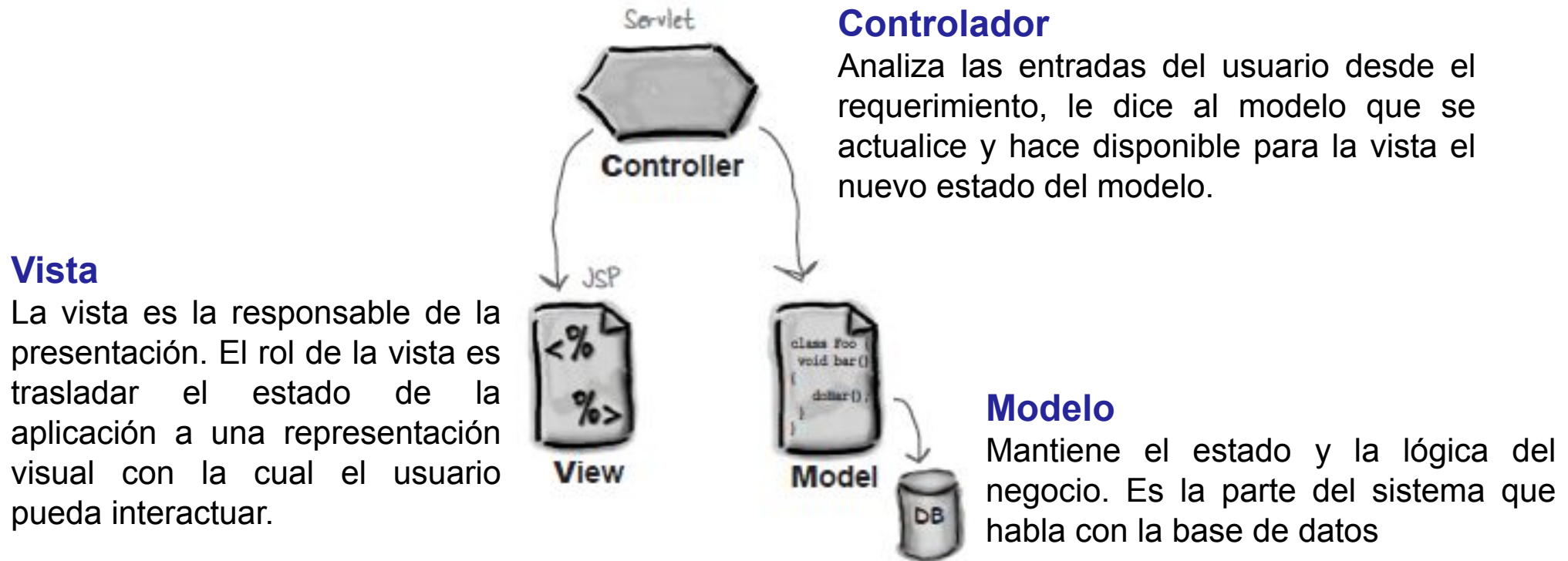


# JavaServer Pages

- 1 ¿Qué son las páginas JavaServer Pages (JSP)?
- 2 Ciclo de vida de una JSP
  - Fase de traducción
  - Fase de compilación
  - Fase de ejecución
- 3 La interface `javax.servlet.jsp.HttpJspPage`
- 4 Elementos para construcción de JavaServer Pages
  - Elementos de scripting
  - Directivas
  - Acciones estándares
  - El lenguaje de expresiones (JSP EL)
  - Java Standard Tag Library (JSTL)

# Servlets & JavaServer Pages

El patrón MVC (Model-View-Controller), es un modelo adoptado por muchas aplicaciones web e implementado por muchos frameworks de aplicaciones para lograr una clara separación entre la lógica de negocios y la lógica de presentación.



El MVC saca del servlet la lógica de negocio y la pone en un "modelo" -clases reutilizables simples de Java (POJO)-. El modelo es una combinación de los datos del negocio y los métodos (reglas) que operan sobre esos datos.

# JavaServer Pages

En las aplicaciones web actuales, una página web está compuesta por contenido estático o **no personalizado** + contenido dinámico o **personalizado**.

Contenido estático

Contenido dinámico

The screenshot shows the website **pagomiscuentas.com** in a Mozilla Firefox browser. The address bar shows the URL `https://paysrv2.pagomiscuentas.com/pmctas/userLogin.do?dispatch=executeAction`. The page features a blue header with the site name and a BANELCO logo. A left sidebar contains navigation links under 'Pago' and 'Consulta'. The main content area displays a welcome message, login options, and a table of 'Próximos Vencimientos' (Upcoming Payments). A date and login time are shown in the center. A 'Terminado' status bar is at the bottom.

Empresa	Importe a pagar	Vencimiento	Cuentas As
<input type="checkbox"/> Impuestos AFIP	Total	\$ 266,58	**/**/****
<b>Total de Pesos</b>		<b>0,00</b>	Seleccione una
<b>Total de Dolares</b>		<b>0,00</b>	

Las páginas que forman una aplicación web están compuestas por contenido estático y dinámico.


# JavaServer Pages

- Una **JavaServer Page (JSP)** es una componente web Java gerenciada por el Contenedor Web. El código fuente de una JSP es un archivo de texto que combina tags HTML con nuevos tags de scripting Java. Básicamente es un archivo HTML con código JAVA intercalado.
- El contenido de una JSP está compuesto por una parte estática (código HTML) que el Contenedor Web ignora y que envía como respuesta al cliente sin procesar y una parte dinámica que es evaluada y procesada por el Contenedor Web y es la que permite la personalización.
- Un archivo JSP combina sintaxis JSP y tags HTML y tiene extensión .jsp

Pagina JSP

```
<HTML>
<HEAD><TITLE>miprimerjsp.jsp</TITLE></HEAD>
<BODY>
<P><IMG border="0" src="../../imagenes/hola.jpg"
      width="129" height="97"></P>
<%= "Bienvenido"+request.getParameter("usr") %>
</BODY>
</HTML>
```

Contenido dinámico

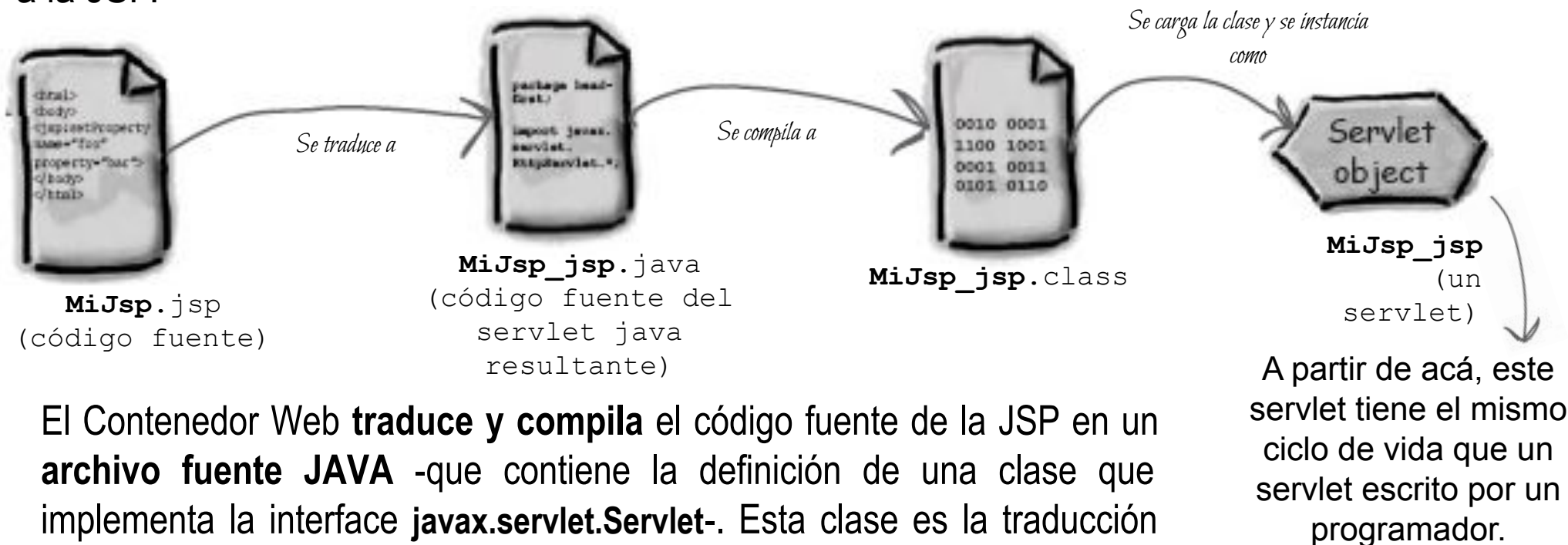


La versión actual de JSP es 2.1 La JSR 245 corresponde a la versión final de su especificación en el JCP.

# JavaServer Pages

## Las JSP se convierten en Servlet

Cada jsp se transforma en un servlet que ejecutará en un contenedor web. La traducción y compilación de la JSP la realiza el Contenedor antes de ser usada, esto puede suceder cuando se despliega la aplicación web o bajo demanda cuando se realiza la primera petición a la JSP.



El Contenedor Web **traduce y compila** el código fuente de la JSP en un **archivo fuente JAVA** -que contiene la definición de una clase que implementa la interface **javax.servlet.Servlet**-. Esta clase es la traducción de la JSP a un Servlet.

La traducción, compilación y carga en memoria es **automática cada vez que el fuente JSP se modifica**

# JavaServer Pages

## Ciclo de Vida

### 1 Traducción

Validación sintáctica de la JSP y archivos dependientes y construcción de la clase (servlet) que implementa la JSP.

`jspInit(ServletConfig)`

El Contenedor crea un objeto ServletConfig, que es posible recuperar en este método y que a su vez está disponible a través del objeto implícito config. Permite al servlet acceder a parámetros de inicialización, de la forma nombre-valor. El programador de la JSP puede sobrescribirlo.

### 2 Ejecución

Si NO existe la instancia del Servlet  
(se crea una instancia)

`jspDestroy()`



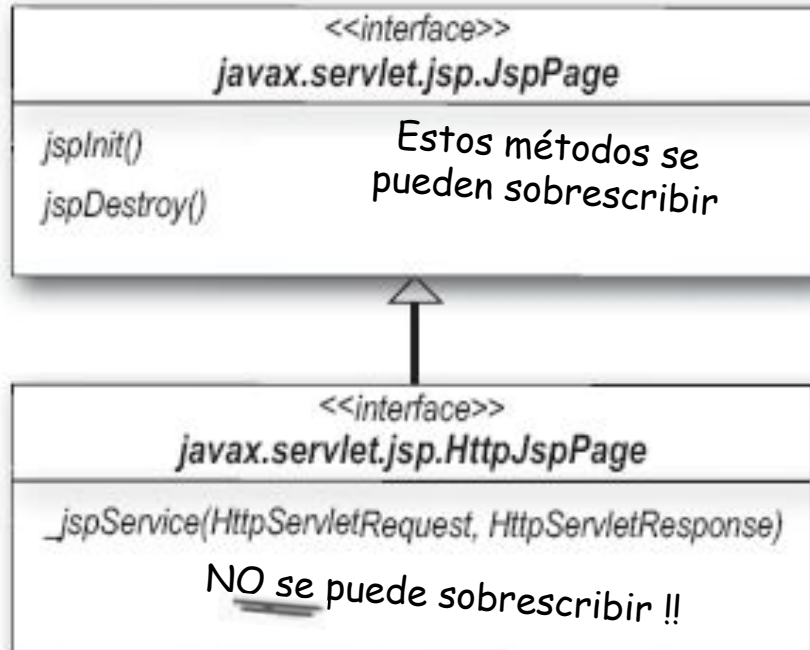
`_jspService()`

Atiende requerimientos entrantes de clientes. Es invocado por el contenedor web cada vez que recibe un requerimiento nuevo para la JSP. El Contenedor crea dos objetos que pasa como parámetro al servlet, del tipo HttpServletRequest y HttpServletResponse. Es el responsable de generar respuestas para GET y POST. El programador de la JSP no puede sobrescribir el `_jspService()`

# JavaServer Pages

## la interface `javax.servlet.jsp.HttpJspPage`

El contenedor web, a partir de una JSP crea una clase que implementa la interface **HttpJspPage**. Esta es la única parte de la API del “servlet generado” que necesitan conocer. La interface **JspPage**, a su vez, extiende la interface **Servlet**



**jspInit()**: el Contenedor web lo invoca para inicializar la JSP. Es similar al método `init()` de servlets. El autor de la JSP puede sobrescribirlo para inicializar la JSP.

**jspDestroy()**: el Contenedor lo invoca cuando la JSP está por ser removida. Es similar al método `destroy()` de servlets. El autor de la JSP puede sobrescribirlo y de esta manera liberar los recursos alocados por la JSP.

**\_jspService()**: Es similar al método **service()** de servlets. Es el método que el Contenedor invoca para atender las peticiones HTTP provenientes desde el cliente web.

La implementación de este método es generada por el Contenedor Web y **NO se puede sobrescribir**.



# ¿Qué podemos usar para escribir una JSP?

## Elementos de Scripting

Es el método más simple para escribir una JSP. Consiste en incluir código Java entre plantillas HTML. JSP no está limitada a elementos de *scripting* escritos en Java, sin embargo la especificación solamente habla de Java como lenguaje de *scripting*.

- Scriptlets `<% %>`
- Expresiones `<%= %>`
- Declaraciones `<%! %>`

## Directivas

`<%@ directiva {atributo="valor"}* %>`, siendo directiva: `page`, `include` o `taglib`

## Acciones

- Estándares
  - `<jsp:param>`
  - `<jsp:include>`
  - `<jsp:forward>`
  - `<jsp:useBean>`,
  - `<jsp:setProperty>`, `<jsp:getProperty>`
  - `<jsp:plugin>`
- Customizadas (Personalizadas)

**JSP EL** es una solución más simple y flexible que usar scripts embebidos.



# Elementos de Scripting Scriptlets

- Un **scriptlet** es un bloque de código Java, encerrado entre los símbolos `<%..%>`, intercalado entre plantillas de texto, que se ejecuta en el momento que el cliente peticiona la JSP.
- Los múltiples **scriptlets** de una JSP se combinan en el método `_jspService()` del servlet que implementa la JSP en el mismo orden en que aparecen en la JSP.
- El scriptlet JAVA contiene código Java normal. Para escribir **scriptlets** no es necesario declarar clases ni métodos.

```
<html>                                luck.jsp
<body>
<h1>Mi primera JSP</h1>
<% double num = Math.random();
  if (num > 0.95) { %>
    <h2>Día de suerte!!</h2>
    <p><%= num %></p>
  <%
  } else {
  %>
    <h2>Bueno, la vida sigue ... </h2>
    <p><%= num %></p>
  <%
  }
  %>
</body>
</html>
```

El código del scriptlet aparece dentro del método `_jspService()` del servlet generado automáticamente

```
public final class luck_jsp extends      luck_jsp.java
    org.apache.jasper.runtime.HttpJspBase {
public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
{
    . . .
    out.write("<html>\n");
    out.write("<body>\r\n");
    out.write("<h1> Mi Primera JSP </h1>\r\n");
    double num = Math.random();
    if (num > 0.95) {
        out.write("\n");
        out.write("<h2>Día de suerte!!</h2><p>");
        out.print( num );
        out.write("</p>\n");
    } else {
        out.write("\n");
        out.write("<h2>Bueno, la vida sigue ... </h2><p>");
        out.print( num );
        out.write("</p>\n");
    }
}
```

El contenido del *scriptlet* no se envía al cliente, lo que se devuelve es el resultado de su ejecución

# Elementos de Scripting

## Expresiones

- Proveen un método sencillo para enviar al cliente *Strings* generados dinámicamente.
- Las **expresiones**:
  - están encerradas entre los símbolos **<%= y %>**
  - se transforman en el argumento de un **out.print( ... )**
  - siempre envían **strings** al cliente, aunque el objeto obtenido de la evaluación de la expresión no lo sea. El resultado de una expresión, es convertido a **String** invocando al método **toString()** o **valueOf()**.

En este ejemplo se declara una variable en un **scriptlet** y luego usamos una **expresión** para mostrar su valor (previamente incrementado). ¿cómo queda el código del servlet?

scriptlet

contador.jsp

```
<html>
<body>
<%int cont=0;%>
El valor del contador es:
<%=++cont%>
</body>
</html>
```

expresión

```
public final class contador_jsp extends
    org.apache.jasper.runtime.HttpJspBase {

    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        ...
        int cont=0;
        out.write("El valor del contador es:\r\n");
        out.print(++cont);
        out.write("</body>\n");
        out.write("</html>");
        ...
    }
```

Una variable declarada en un scriptlet  
es siempre una variable local

El contenedor pone todo en un método  
service() genérico, no distingue  
doGet()/doPost()

¿qué imprime cuando se la peticona varias veces?

# Elementos de Scripting

## Expresiones (cont.)

*La primera vez que se requiere la página:*



*La segunda, tercera ... y cada vez:*



Declarar una variable en un scriptlets significa que es inicializada cada vez que recibe un requerimiento (y ejecuta el `_jspService()`). Con cada requerimiento se resetea a "0".

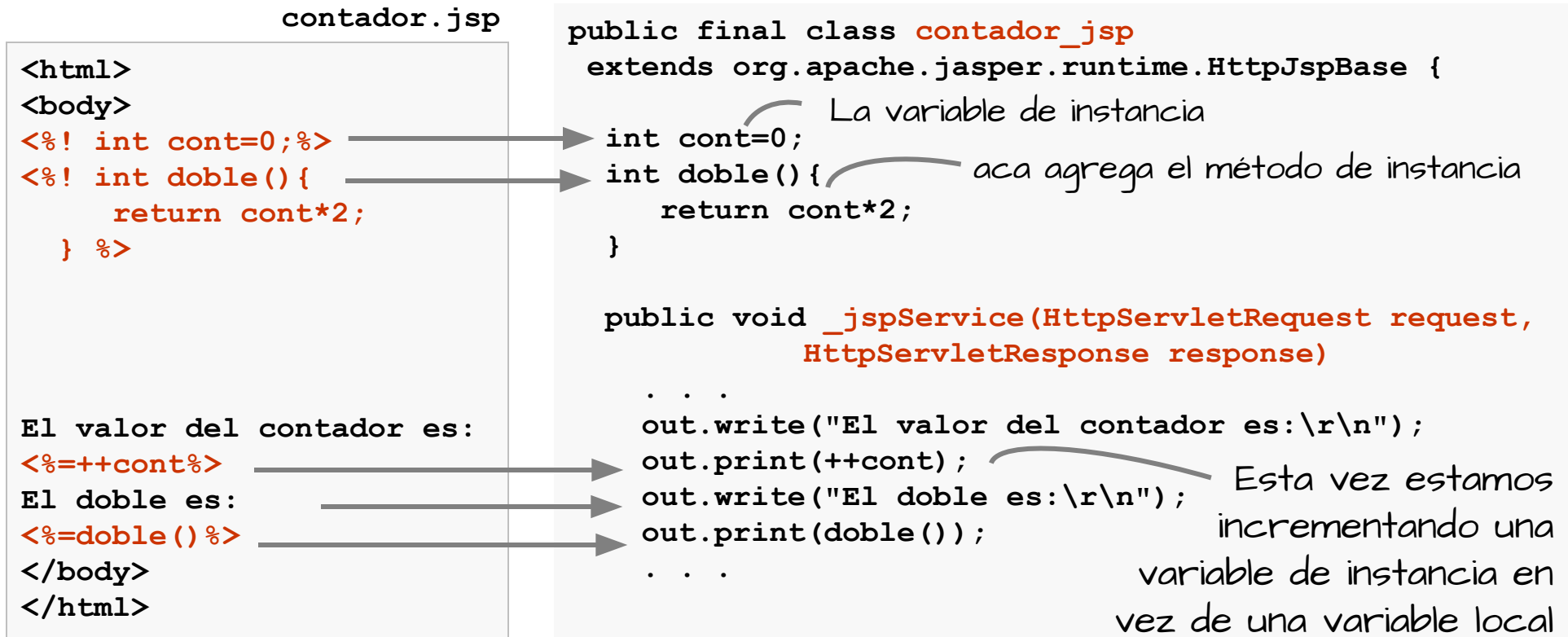
**¿Hay otro elemento JSP que me permita declarar una variable de instancia del servlet en vez de una variable local?**

Si!!, existe otro elemento llamado **declaración**. Las declaraciones permiten definir miembros de la clase del servlet generado.

# Elementos de Scripting


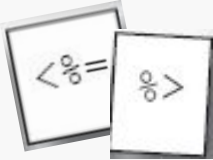

## Declaraciones

- Una declaración es un bloque de código Java usado para definir miembros (variables y métodos de instancia o clase) de la clase generada por el contenedor.
- Las declaraciones están encerradas entre los símbolos `<%!` y `%>`. Cualquier cosa contenida en estos tags, es agregada afuera del método `_jspService()`.
- Las declaraciones permiten sobrescribir los métodos `jspInit()` y `jspDestroy()`.



# Elementos de Scripting

## Analogía con Servlets

JSP	Servlet
<p>scriptlets</p> 	<p>Es el código que se escribe adentro del método <b>jsp_service()</b> donde pueden usarse estructuras de control, asignaciones y cualquier lógica.</p>
<p>expresiones</p> 	<p>Es el código que se escribe adentro del método <b>jsp_service()</b> invocando el método <code>println()</code> sobre el objeto <code>PrintWriter</code>.</p>
<p>declaraciones</p> 	<p>Es el código (métodos y declaraciones de variables) que está afuera del método <b>jsp_service()</b> pero que forma parte del cuerpo del servlet.</p>

# Objetos implícitos

Cuando un Contenedor Web traduce una JSP a Servlet, las primeras líneas del `_jspservice()` corresponden a una pila de declaraciones y asignaciones de objetos. Estos objetos conocidos como **objetos implícitos** mapean con objetos de la API de servlet/JSP y no es necesario declararlos ni instanciarlos, lo hace automáticamente el contenedor, de ahí su nombre.

API (tipo)

Objeto(variable)

JspWriter	_____	out
HttpServletResponse	_____	response
HttpServletRequest	_____	request
HttpSession	_____	session
ServletContext	_____	application
ServletConfig	_____	config
Throwable	_____	exception
PageContext	_____	pageContext

es usado para enviar contenido en una respuesta (representa al objeto `PrintWriter`).

representa el objeto `Throwable` que provocó la invocación de la página de error

Encapsula el contexto de una JSP, incluye todos los objetos implícitos y métodos para redireccionar e incluir recursos. Es útil en acciones customizadas

# Objetos Implícitos

## Ejemplo de uso

*En ningún lugar se declaran los objetos **request**, **session** y **out** en la JSP, están automáticamente disponibles.*

```
<HTML>
<%@ page import="java.util.HashMap"%>
<BODY>
<% out.print("Bienvenido a nuestro Supermercado");%>
<%
String nom=request.getParameter("nombre");
String ap=request.getParameter("apellido");
String[] prod=request.getParameterValues("productos");
session.setAttribute("firstname", nom);
session.setAttribute("lastname", ap);
HashMap<String,String> changuito=(HashMap<String,String>)session.getAttribute("chango");
//Agrego productos al changuito
%>
</BODY>
</HTML>
```



# Elementos de Scripting

## Análisis de un Servlet generado a partir de contador.jsp

```
public final class contador_jsp extends org.apache.jasper.runtime.HttpJspBase {
```

```
    int cont=0;  
    int doble(){  
        return cont*2;  
    }
```

El contenedor pone todas las declaraciones `<%! %>` que encuentra en la JSP primero y debajo las propias del contenedor

```
    public void _jspService(HttpServletRequest request, HttpServletResponse response)  
        throws java.io.IOException, ServletException {
```

```
        PageContext pageContext = null;  
        HttpSession session = null;  
        ServletContext application = null;  
        ServletConfig config = null;  
        JspWriter out = null;
```

El contenedor declara una pila de variables locales, incluyendo aquellas que representan a los "objetos implícitos"

contador.jsp

```
        try {  
            response.setContentType("text/html; charset=ISO-8859-1");  
            pageContext = _jspxFactory.getPageContext(this, ...);  
            application = pageContext.getServletContext();  
            config = pageContext.getServletConfig();  
            session = pageContext.getSession();  
            out = pageContext.getOut();
```

```
            out.write("<html>\n");
```

```
            ...
```

```
            out.write("El valor del contador es:\r\n");
```

```
            out.print(++cont);
```

```
            out.write("El doble es:\r\n");
```

```
            out.print(doble());
```

```
            ...
```

```
        } catch () { }
```

```
    }
```

Aca trata de inicializar a los "objetos implícitos"

Aca procesa JSP, HTML, scripts y expresiones

maneja errores ...

```
<html>  
<body>  
<%! int cont=0;%>  
<%! int doble(){  
    return cont*2;  
} %>
```

El valor del contador es:

```
<%=++cont%>
```

El doble es:

```
<%=doble()%>
```

```
</body>
```

```
</html>
```

# JavaServer Pages

## Directivas

Son instrucciones especiales de la JSP para el Contenedor Web. Se usan para definir atributos de la página, importar librerías de tags e incluir archivos.

No producen ninguna salida para el cliente web. El alcance de las directivas es todo el archivo JSP.

```
<%@ directiva    {atributo="valor"}* %>
```

Las directivas pueden ser:

**page**

```
<%@ page    {atributo="valor"}* %>
```

**include**

```
<%@ include    file="URLRelativa"%>
```

**taglib**

```
<%@ taglib    uri="URI"    prefix="prefijoDelTag"%>
```

# JavaServer Pages

## La directiva `<%@ page . . %>`

Esta directiva le brinda al Contendor información específica sobre la JSP. Establece atributos que afectan a toda la página. Una JSP puede tener múltiples directivas page. Durante la fase de traducción, todas estas directivas se aplican juntas a la página. De cada atributo sólo puede haber una ocurrencia. La excepción es el atributo **import**, que puede tener múltiples ocurrencias.

```
<%@ page {atributo="valor"}* %>
```

- Por defecto, la directiva page toma los valores de uso más común de JSP para construir páginas dinámicas.
- Cuando se crea una JSP, no siempre es necesario especificar algún atributo de la directiva page. Un ejemplo de uso es cuando se necesitan usar clases o interfaces para codificar elementos de scripting (import) o especificar tamaño del buffer (buffer).

```
<%@ page import="java.util.*" buffer="8k" autoflush="true" %>
<html><body>
<% if (Calendar.getInstance().get(Calendar.AM_PM)==Calendar.AM) {%>
¿Cómo estás esta mañana?
<%} else { %>
¿Cómo estás esta noche?
<% } %>
</body></html>
```

`hola.jsp`

# JavaServer Pages

## La directiva `<%@ page . . %>`

Los atributos de esta directiva son:

```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import= "{ package.class | package.* }, ..." ]
  [ session="true|false" ]
  [ buffer="none|nkb" ]
  [ autoFlush="true|false" ]
  [ errorPage="relativeURL" ]
  [ contentType="mimeType [;charset=characterSet ]"|"text/html ; charset=ISO-8859-1"]
  [ isErrorPage="true|false" ]
  [ isElEnabled = "true|false" ] “
%>
```

**language**: define el lenguaje de scripting a usar. en la página. Actualmente java es el único valor posible y es el de default.

**extends**: define el nombre de la superclase del servlet generado a partir de la jsp. No se recomienda usar , afecta portabilidad.

**import**: define el conjunto de clases y paquetes que importará la clase del servlet generado a partir de la jsp.

**session**: define si la jsp participa de una sesión HTTP. Puede ser true o false, por default es true.

**buffer**: define el tamaño del buffer usado para escribir salida. El valor es *none* o *nKB*. El tamaño del buffer por default es 8kb. Si es None, no se usa buffer y la salida se escribe directamente en el objeto PrintWriter. Si se especifica un tamaño, la salida se escribe en el buffer.

**autoFlush**: define si limpia el buffer cuando se llenó (autoflush=true) o dispara una excepción (autoflush=false)

**errorPage**: define una url a un recurso de la aplicación web, que manejará todos los errores generados.

**isErrorPage**: indica que la JSP es una página de error -tienen acceso al objeto implícito exception-

# JavaServer Pages

## La directiva `<%@include . . %>`

- Notifica al contenedor que incluya en la página JSP, el contenido del recurso especificado en el atributo file. El recurso puede ser estático o dinámico y debe ser parte de la aplicación web.
- La inclusión se realiza durante la traducción de la JSP. Es equivalente a incluir el código fuente en la JSP antes de la compilación.
- La inclusión de recursos del lado del servidor es una característica común en JSP. Ejemplo: el encabezamiento y el pie común en las páginas de un sitio.

`<%@ include file="URLRelativa"%>`

bienvenida.jsp

header.jsp

```
<%! private int cont=0;%>
<center>
<h1>Curso de J2EE - Facultad de Informática
</h1>
<br>
El sitio fue visitado <%= ++cont%>
</center>
```

footer.jsp

```
<center>
Copyrighth &copy; 2009
</center>
```

```
<HTML>
<HEAD>
</HEAD>
<%@ include file="header.jsp"%>
<BODY>
El contenido de la página es único.
El mismo header.jsp y footer.jsp es usado en
todas las páginas mediante la directiva
include
</BODY>
<%@ include file="footer.jsp"%>
</HTML>
```

# JavaServer Pages

## La directiva `<%@taglib ... %>`

- Esta directiva le indica al traductor de la JSP que se utilizará una librería de tags customizados en la página JSP. Esta directiva incluye la uri del TLD (Tag Library Descriptor) y un prefijo para los custom tags en la página.
- La sintaxis es la siguiente:

`<%@ taglib prefix="prefijo" uri="identificacion recurso" %>`

prueba.jsp

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

```
<%@ taglib prefix="test" uri="/WEB-INF/customtags/libreriaCT.tld" %>
```

```
<html>
<title>Un Tag Simple </title>
<html>
<body>
<h2><test:pageVisit/></h2>
```

Declara el uso de una librería de tags en una JSP. Luego, es posible usar los custom tag definidos en la librería.

```
</body>
</html>
```

Prefijo para esta JSP: el atributo prefix define el prefijo que se usará en la JSP para identificar a los tags definidos en una librería de tags específica.

# JavaServer Pages

## Acciones

- Las acciones permiten conectar tags que aparecen en la página JSP con código dinámico.
- La funcionalidad que pueden hacer las acciones es idéntica a la de los elementos de scripting, pero tienen la ventaja de abstraer cualquier código que normalmente estaría entremezclado con la JSP.
- Hay 2 tipos de acciones disponibles para usar con jsp: estándares y customizadas. Ambas siguen la misma sintaxis compatible con XML:

`< prefijo:elemento {atributo=valor}* />`

Conjunto de pares (atributo, valor) que permiten *customizar* la acción.

- Las acciones estándares están especificadas e implementadas en la especificación JSP. Están disponibles para usar en cualquier contenedor web.
- Las acciones customizadas son un mecanismo definido por la especificación que le permite a los programadores crear sus propias acciones. La funcionalidad no está implementada en la JSP, las acciones *custom* deben ser instaladas antes de ser usadas.



# Acciones Estándares

## `jsp:param`

- Este tag es usado para proveer información adicional en la forma clave/valor a los tags **`jsp:include`** y **`jsp:forward`**, si no se ubica dentro de estos tags, da un error durante la traducción de JSP a Servlet.
- Cuando se usa el subelemento **`jsp:param`**, la página incluída o redireccionada verá el requerimiento original con sus parámetros, más los nuevos parámetros con valores que tienen precedencia sobre los valores originales (cuando coinciden los *names*).


Sintaxis:

```
<jsp:param name="nomParametro" value="valParametro"/>
```

Ejemplo:

```
<jsp:param name="subtitulo" value="Taller TTPS"/>
```

Se modifica el requerimiento original agregándole un nuevo parámetro y valor



Si un requerimiento original ya tiene un parámetro `subtitulo={"Curso"}` y se especifica un `<jsp:param>` como el del ejemplo, el requerimiento tendrá el parámetro `subtitulo={"Taller TTPS", "Curso"}`

# Acciones Estándares

## `jsp:include`

Permite incluir un recurso estático o dinámico en una JSP (del mismo contexto de la página) en el momento en que la JSP se ejecuta.

El recurso se especifica mediante una URL.

- Si el recurso es estático, el contenido se inserta directamente.
- Si el recurso es una componente web, el requerimiento original es enviado a la JSP a ser incluida, la JSP se ejecuta y el resultado se incluye en la respuesta de la JSP original.

Sintaxis:

```
<jsp:include page="URLRelativa" flush="true"|"false"/>
```

```
<jsp:include page="URLRelativa" flush="true"|"false" >  
  <jsp:param name="nomParametro" value="valParametro"/>  
  ...  
</jsp:include>
```

Se pueden proveer parámetros al requerimiento para ser usados por el recurso destino.

URLRelativa fija

Ejemplos:

```
<jsp:include page="/utiles/date.jsp" >  
  <jsp:param name="mes" value="3"/>  
</jsp:include>
```

`flush="true"`: el buffer es vaciado antes de la inclusión.

`flush="false"`: el buffer no es vaciado antes de la inclusión.

```
<% String aDonde=nomPagina;%>  
<jsp:include page="<%=aDonde%>" />
```

*expresión*

# Acciones Estándares

## jsp:include

### La directive include

La directiva `include` se hace en **tiempo de traducción** al servlet, incluye la versión del recurso (*file*) en el momento de la traducción. Si el archivo incluido se actualiza, no se refleja.

```
<HTML>
<HEAD>
<%@ page language="java"...%>
</HEAD>
<%@ include file="banner.jsp"%>
<BODY>
El contenido de la página es único.
Los mismos banner.jsp y footer.jsp son reusados
usando la directiva include.
</BODY>
<%@ include file="footer.jsp"%>
</HTML>
```

El código de servlet resultante se generará incluyendo `header.jsp/footer.jsp` antes de compilar el servlet.

**Más performante**

### La acción estándar include

La acción `include` ocurre en **tiempo de ejecución** y garantiza que se use la última actualización del recurso que está incluyendo.

Se puede mandar parámetros, con la directiva no se puede.

```
<HTML>
<HEAD>
<%@ page language="java"... %>
</HEAD>
<jsp:include page="banner.jsp" flush="false">
  <jsp:param name="titulo" value="cursoJ2EE"/>
</jsp:include>
<BODY>
El contenido de la página es único.
Los mismos banner.jsp y footer.jsp son reusados
usando la acción include.
</BODY>
<jsp:include page="footer.jsp"%>
</HTML>
```

El código del servlet resultante invoca al método `include()` para incluir `header.jsp/footer.jsp` en cada requerimiento.

**Más consistente**

# Acciones Estándares

## @include vs. jsp:include

Se incluye el recurso dinámico **banner.jsp**, usando los 2 mecanismos:

```
<HTML><HEAD>
<TITLE>Un banner</TITLE>
</HEAD>
<BODY>
<%= "Estoy en el Banner"%>
<%= new java.util.Date() %>
</BODY></HTML>
```

La directiva @include

```
<HTML>
<HEAD>
<%@ page language="java" ... %>
</HEAD>
<%@ include file="banner.jsp"%>
<BODY>
...
</BODY>
</HTML>
```

↓ Servlet generado

```
public void _jspService(..){
    ..
    out.write("\r\n");
    out.print("Estoy en el Banner");
    out.write("\r\n");
    out.print(new java.util.Date());
}
```

La acción jsp:include

```
<HTML>
<HEAD>
<%@ page language="java" buffer="8k"%>
</HEAD>
<jsp:include page="banner.jsp" flush="true">
<BODY>
...
</BODY>
</HTML>
```

↓ Servlet generado

```
public void _jspService(..){
    ..
    JspRuntimeLibrary.include(request,
    response, "banner.jsp", out, true);
    ..
}
```

Cada contenedor podría tener su mecanismo, pero todos invocan al include()

# Acciones Estándares

## `jsp:forward`

- Este tag permite redireccionar el requerimiento actual a otra página JSP, a un servlet o a un recurso estático que pertenezca al mismo contexto que la página que hace el **forward**.
- Cuando se encuentra el tag `<jsp:forward>`, la ejecución de la JSP finaliza, se limpia el buffer (`clear()`) y se descarta, el requerimiento original es modificado si hay parámetros y se pasa el control al otro recurso.

Sintaxis:

```
<jsp:forward page="URLRelativa" />
```

```
<jsp:forward page="URLRelativa" >
```

```
    <jsp:param name="nomParametro" value="valParametro"/>
```

```
    ...
```

```
</jsp:forward>
```

Se pueden proveer parámetros al requerimiento para ser usados por el recurso destino.

**URLRelativa fija**

Ejemplos:

```
<jsp:forward page="/busquedas/resultado.jsp">
  <jsp:param name="cantidad" value=20/>
</jsp:forward>
```

```
<%String aDonde="/busquedas/"+nomPagina%>
<jsp:forward page="<%=aDonde%>" />
```

**Expresión**

**NOTA:** Hay que ser cuidadoso cuando se usa `<jsp:forward>` con salidas NO *buffered* (es decir, `<@ page buffered="none">`). Si la JSP hace un forward, cuando se intenta limpiar un buffer inexistente, causa una excepción de tipo **IllegalStateException. Error 500: Illegal to `clear()` when buffer `size==0`**

# JSP y JavaBeans

- JavaBeans son componentes de software escritas en Java, que siguen unas pocas convenciones de codificación. Técnicamente un JavaBean es una clase java que:
  - a) implementa la interface `java.io.Serializable`,
  - b) provee un constructor sin argumentos
  - c) dispone de métodos setters (setean valores a las propiedades del bean) y getters (recuperan valores de las propiedades del bean).
- Los JavaBeans son usados en JSP/Servlets para pasar información con algún alcance (`page`, `request`, `session`, `application`). El uso más común de JavaBeans es como objetos simples que mantiene datos del lado del servidor.
- JSP provee 3 acciones estándares para manipular JavaBeans: `<jsp: useBean>`, `<jsp: setProperty>` y `<jsp: getProperty>` las cuales permiten reemplazar código de scripts incluido en las JSP.
- El uso de JavaBeans en JSP, tiene como resultado un código JSP más simple, pero no es la única solución para eliminar scripting de las páginas JSP.

# JavaBeans Propiedades

Las propiedades de un bean describen los datos internos del bean, usualmente son variables de instancia de acceso protegido o privado que puede ser accedidas por métodos (getters y setters) de alcance público.

Las propiedades de un bean pueden ser de tipo de datos primitivos o referencias a objetos.

Constructor por defecto.

```
public class Usuario implements java.io.Serializable {  
  
    private String nombre;  
    private String password;  
    private boolean activo;  
    private Domicilio domicilio;  
    private String[] areas;  
  
    public Usuario() {  
    }  
  
    . . .  
}
```

Define que el bean puede serializarse

El constructor por defecto, permite la creación de un bean de una manera genérica, conociendo simplemente el nombre de la clase.



# JavaBeans

## Métodos para acceder a las propiedades

La especificación de JavaBeans establece que los métodos que acceden a las propiedades cumplen con los siguientes patrones de nombres:

```
public void set<NomPropiedad>(<tipo_de_la_Propiedad> valor){}
public <tipo_de_la_Propiedad> get<NomPropiedad>(){}
public boolean is<nomPropiedad>(){}
public void set<NomPropiedad>(int i, <tipo_de_la_Propiedad> valor);
public <tipo_de_la_Propiedad> get<NomPropiedad>(int i);
```

```
public class Usuario implements java.io.Serializable {
    private String nombre;
    private String password;
    private boolean activo;
    private String[] areas;
```

Propiedad  
indexada

métodos  
simétricos

```
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String string) {
        nombre = string;
    }
    public void String[] getAreas() {
        return areas;
    }
    public boolean isActive() {
        return activo;
    }
}
```

# Lenguaje de expresión (EL)

Una característica importante de la tecnología JSP a partir de versión 2.0 es el lenguaje de expresiones. El lenguaje de expresiones o **EL** facilita el acceso a los datos almacenados en componentes JavaBeans mediante una sintaxis simple.

Los objetos del modelo (ligados a algún alcance) son accedidos por nombre.

Este lenguaje provee:

- Sintaxis y accesos simples y precisos.
- Accesos a subpropiedades o propiedades anidadas de objetos java beans.
- Acceso a propiedades indexadas de objetos java beans.

El lenguaje **EL** permite acceder a un bean usando sintaxis simple como **`${user}`** para una variable simple o **`${user.nombre}`** para propiedades anidadas.

Usando el operador “.” se accede a propiedades anidadas cuando el objeto sigue las convenciones de JavaBeans.

El operador “[ ]” se usa para acceder a maps, listas, arreglos de objetos y a propiedades de objetos JavaBeans.

# Lenguaje de expresión (EL)

Las expresiones **EL** se colocan adentro de los caracteres **`${}`**. El lenguaje EL es case-sensitive.

Las expresiones **EL** pueden ser usadas:

(A) en **texto estático**. El valor de la expresión es computado e insertado en la salida que se envía al cliente. Supongamos que existe un objeto Usuario (con una propiedad nombre) ligado a la sesión:

```
<HTML>
<BODY>
Bienvenido, ${usr.nombre}
</BODY>
</HTML>
```

→

```
<HTML>
<BODY>
<% Usuario usr = (Usuario)session.getAttribute("usr");%>
Bienvenido, <%=usr.getNombre()%>
</BODY>
</HTML>
```

(B) en **atributos de acciones estándares o customizadas** que acepten expresiones.

```
<jsp:include page="encabezado.jsp">
  <jsp:param name="titulo" value="Bienvenido"/>
  <jsp:param name="subtitulo" value= "${usr.apellido} ${usr.nombre}"/>
</jsp:include>
```

Las expresiones EL se pueden  
combinar, no anidar

# Lenguaje de expresión (EL)

## Variables

Las variables son referencias a objetos accesibles por nombre. Pueden ser creadas por la aplicación y también existen variables creadas implícitamente por el EL.

Las variables específicas de la aplicación pueden crearse de diferente maneras, por ejemplo desde un servlet, se pueden almacenar objetos usando el método **setAttribute()** en los objetos **HttpServletRequest**, **HttpSession**, **ServletContext**.

Todo objeto disponible en uno de los alcances JSP puede accederse como una variable EL, usando la sintaxis **`${unaVariable}`**

```
<HTML>
<BODY>
Bienvenido, ${usr}
</BODY>
</HTML>
```

En este caso es equivalente a una expresión **`<%=usr%>`**  
Lo convierte a String (`toString()`) y lo envía a la salida.

### *¿Cómo se recuperan las variables ligadas a los diferentes alcances JSP?*

El contenedor web evalúa la variable que aparece en la expresión buscándola en los distintos alcances con el siguiente orden:

**request -> session -> application**

# Lenguaje de expresión (EL)

En el **EL** también existen ***variables implícitas***. Cuando el contendor web evalúa una expresión que coincide con uno de los siguientes objetos implícitos, retorna ese objeto.

Nombre de Variable	Descripción
pageScope	Una colección (java.util.Map) de todas las variables de alcance page
requestScope	Una colección (java.util.Map) de todas las variables de alcance request
sessionScope	Una colección (java.util.Map) de todas las variables de alcance session
applicationScope	Una colección (java.util.Map) de todas las variables de alcance application
param	Una colección (java.util.Map) de todos los valores de los parámetros del requerimiento HTTP. Un String por parámetro.
paramValues	Una colección (java.util.Map) de todos los valores de los parámetros del requerimiento HTTP. Un arreglo de Strings por parámetro.
header	Una colección (java.util.Map) de todos los valores de los headers del requerimiento HTTP. Un String por header.
headerValues	Una colección (java.util.Map) de todos los valores de los headers del requerimiento HTTP. Un arreglo de Strings por header.
cookie	Una colección (java.util.Map) de todos los valores de las cookies del requerimiento HTTP. Un objeto javax.servlet.http.Cookie por cookie.
initParam	Una colección (java.util.Map) de todos los valores de los parámetros de inicialización de la aplicación. Un String por parámetro.
pageContext	Una instancia de javax.servlet.jsp.PageContext proveyendo acceso a datos de la petición.

Para acceder a variables ligadas a los distintos alcances

# Lenguaje de expresión (EL)

## Ejemplos

- Soporte para acceso a propiedades simples

`${usr.nombre}`

`${sessionScope.usr.nombre}`

- Soporte para propiedades anidadas

Las propiedades de un javaBean, pueden ser accedidas usando el operador “.”

```
public class Usuario implements Serializable {
    private String nombre;
    private String password;
    private boolean activo;
    private String[] areas;
    private Domicilio domicilio;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String string) {
        nombre = string;
    }
    ...
}
```

`${usr.domicilio.calle}`

busca en los  
distintos  
alcances

`usr`, esta ligado al alcance Page

`${pageScope.usr.domicilio.calle}`

Usamos el  
objeto implícito

- Soporte para propiedades indexadas

Para acceder a un elemento de un objeto Map, List o de un arreglo se lo puede hacer mediante el operador “.” o el operador [].

`${usr.areas[0]}`

`${paramValues.cursos[0]}`

Equivalente con  
scriptlets

```
<%String[] cursos = request.getParameterValues("cursos"); %>
<%=cursos[0]%>
```

# Lenguaje de expresión (EL)

## Operadores

El lenguaje **EL** provee operadores aritméticos (+, -, \*, /, %), operadores lógicos (and, or, not) y operadores de comparación (>, >=, ==, etc.).

Ejemplos: `${param.dias/24}`  
`${3 div 2}`

## Deshabilitando el EL

Antes del JSP 2.0, los caracteres `${}` no habían sido reservados, lo que podría provocar problemas de compatibilidad con JSP 1.2 y anteriores. Para solucionarlo, se puede deshabilitar el EL en páginas individuales o en grupos de páginas, en el web.xml.

Directiva page de la página JSP

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

```
<jsp-config>  
  <jsp-property-group >  
    <url-pattern>*.jsp</url-pattern>  
    <el-ignored>true</el-ignored>  
  </jsp-property-group>  
</jsp-config>
```

WEB.XML