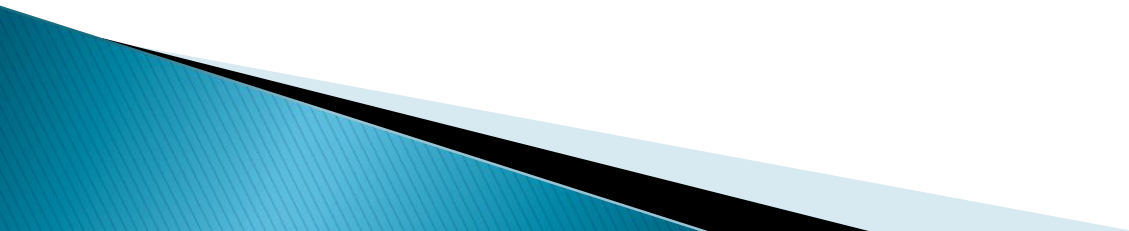


JSP

Java Server Pages



JSP: Introducción

- ▶ Una tecnología que permite combinar código HTML estático con código generado dinámicamente en un mismo fichero.
- ▶ Ventajas:
 - Separación de datos estáticos/dinámicos.
 - Independencia de formato/plataforma.
 - Sencillez (sabiendo servlets)

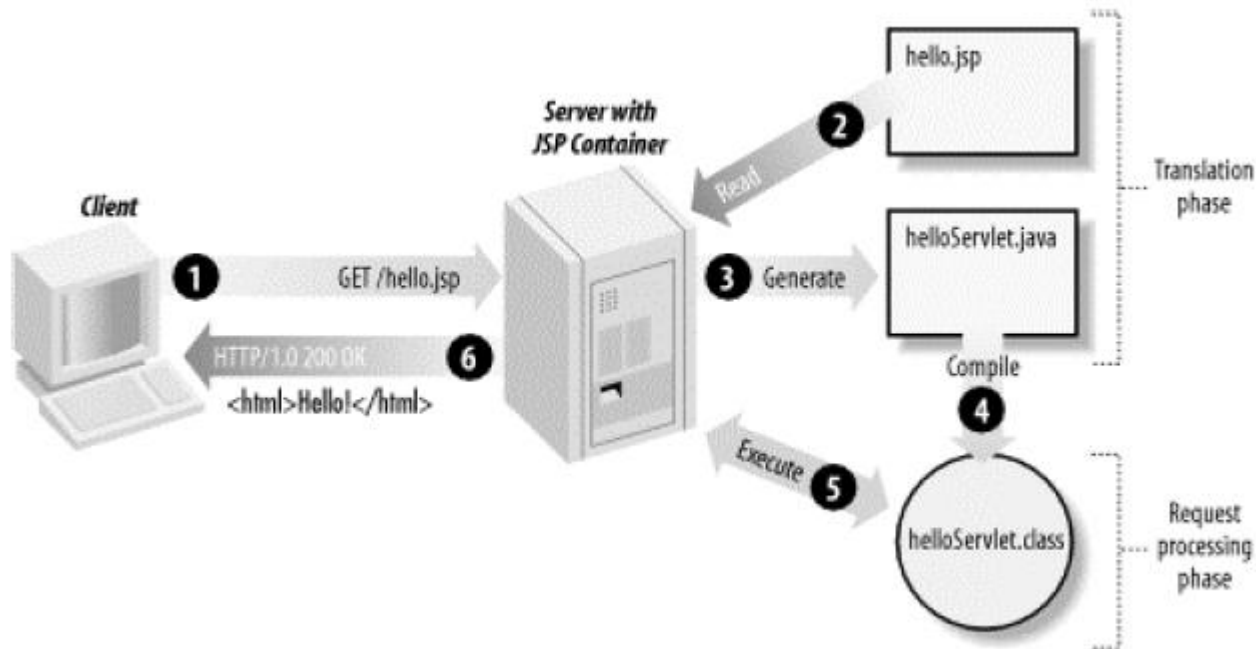
JSP: Introducción

- ▶ Comparaciones con otras tecnologías:
 - Vs ASP (Active Server Pages).
 - Vs Servlets.
- ▶ Los 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 empiezan con "<%" y terminan con "%>".

JSP: Introducción

- ▶ Damos extensión .jsp.
- ▶ Aunque el código parezca mas bien HTML, el servidor lo traduce a un servlet en la primera petición.
- ▶ 3 elementos en un JSP:
 - Elementos script (scriptlets)
 - Directivas
 - Acciones

JSP: Introducción



Elementos JSP

▶ Tres tipos de elementos en JSP:

◦ Directivas

- Permiten especificar información acerca de la página que permanece constante para todas las *request*
 - *Requisitos de buffering*
 - *Página de error para redirección, etc.*

◦ Acciones

- Permiten ejecutar determinadas acciones sobre información que se requiere en el momento de la petición de la jsp
 - Acciones estándar
 - Acciones propietarias (Tag libs)

◦ Scripting

- Permite insertar código java que serán ejecutadas en el momento de la petición.

Sintaxis de JSP

JSP: Sintaxis

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>.`

JSP: Sintaxis

Declaración JSP

```
<%! code %>
```

El código se inserta en el cuerpo de la clase del servlet, fuera del método service. Se utiliza para declarar variables y métodos.

El equivalente XML es:

```
<jsp:declaration> code </jsp:declaration>.
```

Directivas JSP

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

```
<%@ directive att="val" %>
```

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

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

JSP: Sintaxis

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 -->`.

Acciones jsp

```
<jsp:nombre-de-la-acción att=valor att2=valor2  
.../>
```

Usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets. Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, etc

JSP: Expresiones

- ▶ **EXPRESIONES:** `<%= expresión %>` 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.

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

JSP: Expresiones

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

```
<jsp:expression>  
Expresión Java  
</jsp:expression>
```

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

Ejercicio. Mi primera JSP

- ▶ A partir de trabajo 0.0 y de la Página index.html del ejemplo *Hola Mundo*
 - Renombrar index.html a index.jsp
 - Establecer index.jsp como página por defecto ... donde?
 - Añadir tras el saludo...

Hoy estamos a <%= (new java.util.Date()) %>

- Buscar lo que se haya generado a partir de la index.jsp en el directorio work de tomcat.

JSP: Scriptlets

- ▶ **SCRIPTLETS:** `<% código %>` 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 **write**.

Ejemplo:

```
<html>
<% for ( int i = 0 ; i < 10 ; i ++ )
        out.println("<br>" + i) ; %>
</html>
```

JSP: Scriptlets

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

- **Que se traducirá en:**

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

JSP: Scriptlets

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

```
<jsp:scriptlet>  
    Código  
</jsp:scriptlet>
```

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

JSP: Declaraciones

- ▶ **DECLARACIONES:** `<%! codigo %>` que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente.
 - Permite insertar métodos, variables...
 - No generan salida alguna. Se usan combinadas con scriptlets.

```
<%! private int accessCount = 0; %>
```

```
Accesses to page since server reboot:
```

```
<%= ++accessCount %>
```

JSP: Declaraciones

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

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

JSP: Directivas

- ▶ Afectan 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" %>
```

JSP: Directivas

▶ PAGE:

- **import**="package.class" o **import**="package.class1,...,package.classN". Esto permite especificar los paquetes que deberían ser importados. El atributo **import** es el único que puede aparecer múltiples veces.
- **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".
- **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`.

JSP: Directivas

- 10 `session="true|false"`. Un valor de `true` (por defecto) indica que la variable predefinida `session` (del tipo `HttpSession`) debería unirse 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.
- 10 `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.
- 10 `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"`.

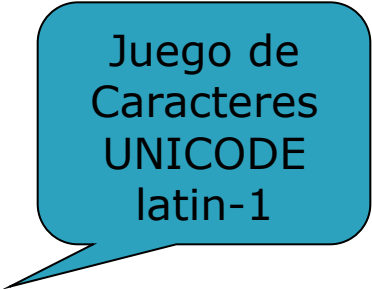
JSP: Directivas

- ⑩ **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.
- ⑩ **info**="message". Define un string que puede usarse para ser recuperado mediante el método `getServletInfo`.
- ⑩ **errorPage**="url". Especifica una página JSP que se debería procesar si se lanzará cualquier Throwable pero no fuera capturado en la página actual.
- ⑩ **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.
- ⑩ **language**="java". 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.

Ejercicio

- ▶ Modificar la jsp Hola Mundo para que en lugar de hacer referencia directamente a `java.util.Date` importe el paquete `java.util`
 - Añadimos al comienzo de la página la siguiente directiva **page**

```
<%@ page language="java"  
    import="java.util.*"  
    contentType="text/html"  
    pageEncoding="iso-8859-1" %>
```



Juego de
Caracteres
UNICODE
latin-1

JSP: Directivas

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

```
<%@ include file="url relativa" %>
```

- Los contenidos del fichero incluido son analizados como texto normal JSP y así pueden incluir HTML estático, elementos de script, directivas y acciones.
- Uso: Barras de navegación.

JSP: Directivas

- ▶ La sintaxis XML para definir directivas es:

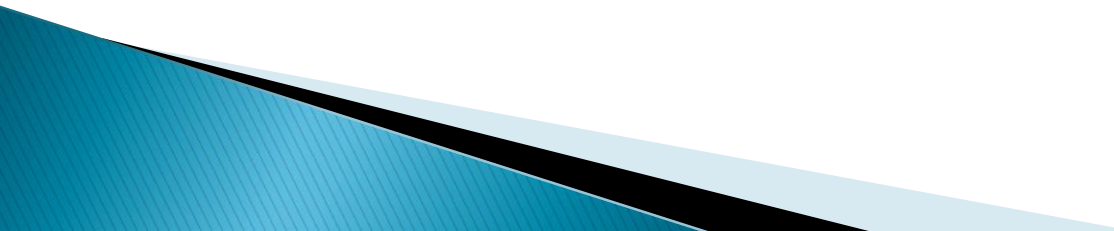
```
<jsp:directive.TipoDirectiva atributo=valor />
```

Por ejemplo, el equivalente XML de:

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

es:

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



JSP: Variables predefinidas

- ▶ **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.

JSP: Variables predefinidas

- ▶ **RESPONSE:** Este es el objeto de clase `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.

JSP: Variables predefinidas

- ▶ OUT: Este es el PrintWriter usado para envíar la salida al cliente. Sin embargo, para poder hacer útil el objeto response, ésta 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.

JSP: Variables predefinidas

- ▶ **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.

JSP: Variables predefinidas

- ▶ **APPLICATION:** El ServletContext obtenido mediante
`getServletConfig().getContext()`.
- ▶ **CONFIG:** El objeto ServletConfig.
- ▶ **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".

JSP: Variables predefinidas

- ▶ **PAGE**: Esto es sólo un sinónimo de `this`, y no es muy útil en Java. Fue creado como situación para el día que el los lenguajes de script puedan incluir otros lenguajes distintos de Java.

Taller práctico...

- ▶ Partiendo del proyecto con el servlet HolaMundoCordial, aprovechar la página index.html con el formulario pero modificarla para que en lugar de dirigirse a un servlet se dirija a **saluda.jsp**. Implementar saluda.jsp para que realice la misma funcionalidad que el servlet HolaMundoCordial.

Taller Práctico

Paso a paso ...

- ▶ ¿Qué debemos hacer?
 - Importar la `java.util.Vector` por medio de la directiva **page**
 - Modificar la `index.jsp` para que se redirija a `saluda.jsp` en lugar de al servlet `HolamundoCordial`
 - Implementar la lógica del servlet `HolaMundo` en la JSP.
En general...
 - Lo que es código java pasa a ser scriplet
 - Lo que se escribe de html pasa a ser html plano.

Acciones JSP

JSP: Acciones

- ▶ Usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets. Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, etc.
- ▶ Tipos de acciones
 - Estandar
 - A medida
 - jstl

JSP: Acciones: include

- a) ACCION 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, inserta el fichero cuando la página es solicitada.

JSP: Acciones: useBean

- b) ACCION jsp:useBean permite cargar y utilizar un JavaBean en la página JSP y así utilizar la reusabilidad de las clases Java.

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

Esto normalmente significa "usa un objeto de la clase especificada por class, y únelo 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`

JSP: Acciones: useBean

- ▶ La forma más sencilla de usar un Bean es usar:
`<jsp:useBean id="name" class="package.class" /`

JSP: Acciones: useBean

id Da un nombre a la variable que referenciará el 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 cualificado completo 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 `instantiate` de Beans. Esta permitido suministrar un `type` y un `beanName`, y omitir el atributo `class`.

JSP: Acciones: setProperty

- ▶ Para obtener valores de propiedades de los beans que se han referenciado anteriormente.
- ▶ 2 usos:
 - Despues de un useBean.

```
<jsp:useBean id="myName" ... />
```

```
...
```

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

Se ejecuta siempre que haya una solicitud.

JSP: Acciones: setProperty

- Dentro de un useBean

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

Solo se ejecuta cuando haya que instanciar un bean.

JSP: Acciones: setProperty

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.

JSP: Acciones: setProperty

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 parametros de la petición e intenta encontrar alguno que concuerde con la propiedad indicada.

Taller práctico

Insertando un javabean en la jsp

- ▶ Modificar la jsp para mediante la acción usebean ver el contador. ¿Dónde lo guardamos?

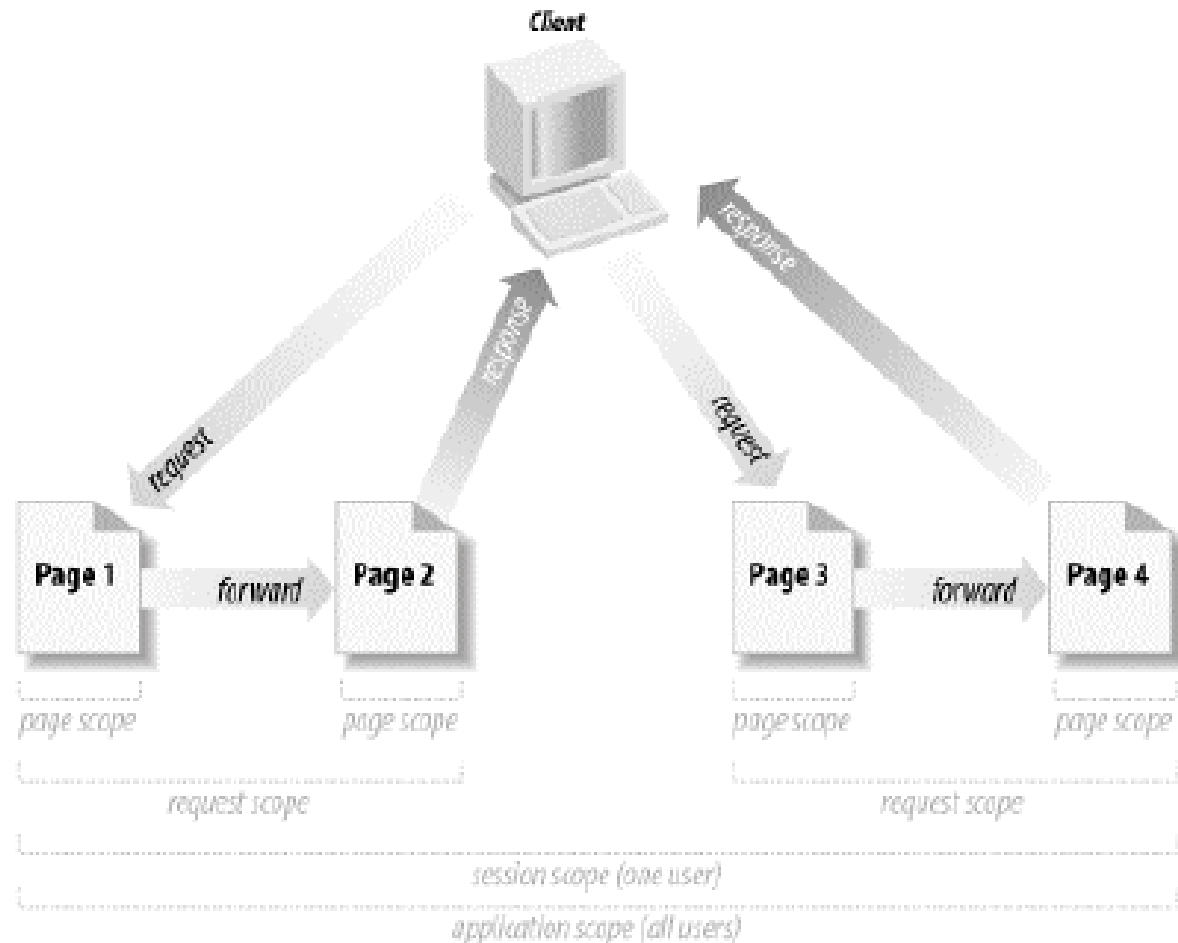
```
<jsp:useBean id="micontador"  
  class="com.dflanvin.beans.ContadorBean"  
  scope="?" />
```

Visitas:

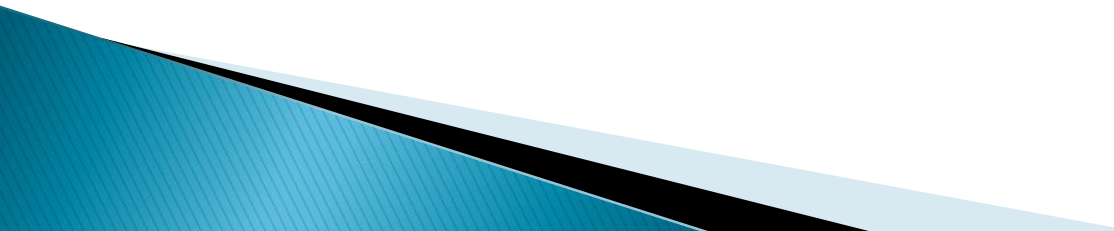
```
<jsp:getProperty name="micontador"  
  property="contador"/>
```

Comunicación entre jsp

Posibles Ámbitos



Desarrollo de TAG-LIBs

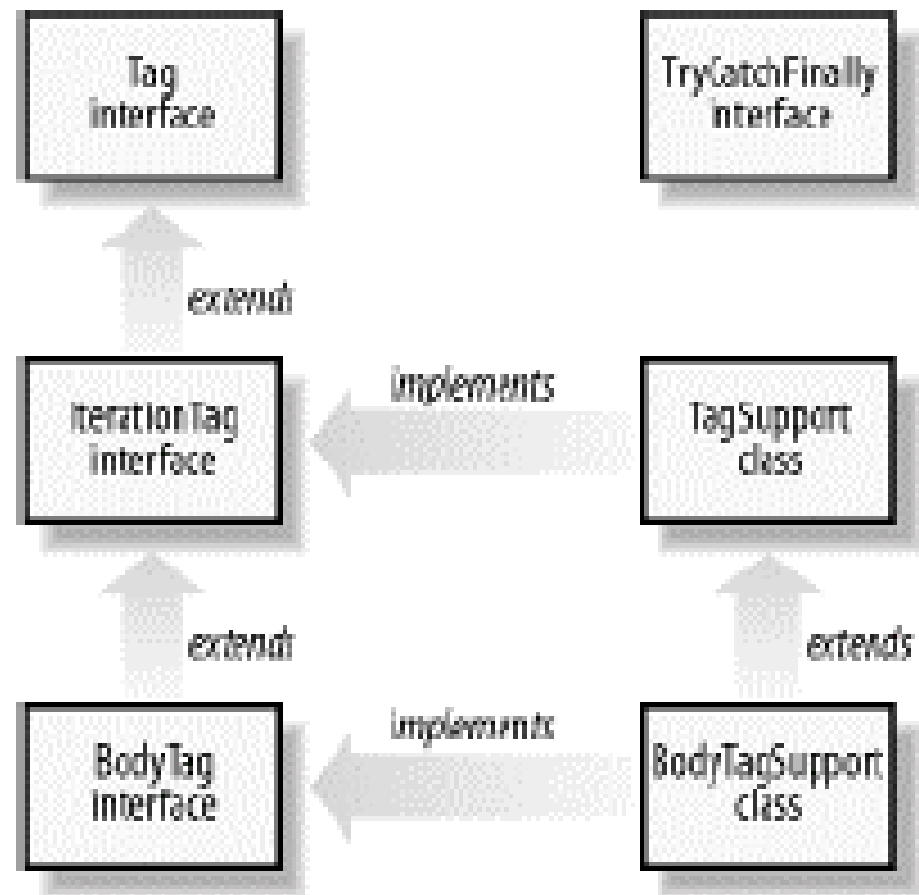
- ▶ Podemos desarrollar nuestras propias etiquetas de acción
 - ▶ Para ello:
 - Desarrollamos la clase que implementa la acción
 - Describimos la etiqueta en la TLD
 - Para usarla, la damos de alta en el web.xml (puesto que no es una etiqueta estándar).
- 

Desarrollo de TAG-LIBs

- ▶ Desarrollo de la clase (manejador del tag):
 - Los interfaces y clases necesarios están definidos en `javax.servlet.jsp.tagext`
 - Tres interfaces:
 - Y dos clases de soporte:
 - TagSupport
 - BodyTagSupport

Tag	Define los métodos necesarios para implementar cualquier acción
IterationTag	Extiende el interface Tag y define los métodos necesarios para iterar sobre los elementos del cuerpo del tag
BodyTag	Extiende el IterationTag y define los métodos para acceder al cuerpo del tag

Desarrollo de TAG-LIBs



Taller práctico:

Mi primer TAG

- ▶ Vamos a desarrollar un tag que añada la firma de la empresa.
 - Creamos una nueva clase com.dflanvin.tag.FirmaTag que extienda TagSupport
 - Añadimos un atributo privado **name** y su correspondiente método setName().
 - Implementamos el método doEndTag():

```
public int doEndTag( ) {  
    try {  
        pageContext.getOut( ).println("<br>" + name +  
        "<br>-----<br>Uniovi.es @ 2008");  
        pageContext.getOut().println(  
            "<br>+034985105094/FAX: +0349855094");  
    }  
    catch (IOException e) {} // Ignore it  
    return EVAL_PAGE;  
}
```

Taller práctico:

Mi primer TAG

- ▶ Una vez tenemos la clase, tenemos que describir el tag en su correspondiente ejemplo.tld que colocamos dentro de WEB-INF

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>ejemplo</short-name>
  <uri>es.uniovi.ejemplo</uri>
  <tag>
    <name>firma</name>
    <tag-class>com.dflanvin.tag.FirmaTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>name</name>
    </attribute>
  </tag>
</taglib>
```

Taller práctico:

Mi primer TAG

- ▶ Y finalmente lo damos de alta en el web.xml

```
<taglib>
```

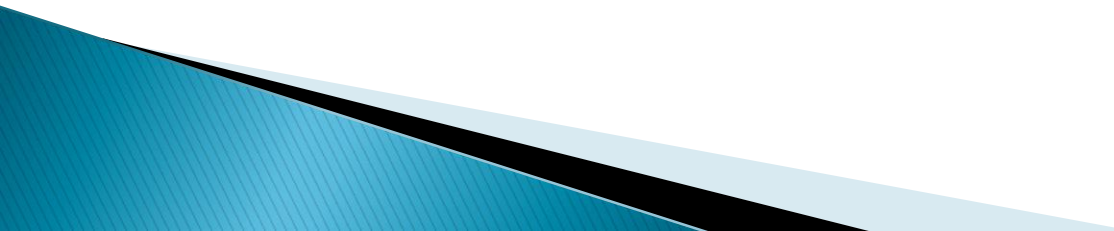
```
  <taglib-uri>es.uniovi.ejemplo</taglib-uri>
```

```
  <taglib-location>/WEB-INF/ejemplo.tld</taglib-location>
```

```
</taglib>
```

- ▶ Ya podemos insertar una llamada al tag al final de saluda.jsp

```
<ejemplo:firma name="Daniel F. Lanvin"/>
```



Taller práctico.

Juntándolo todo

- ▶ Vamos a hacer una aplicación que implemente un carrito de la compra aplicando el modelo de arquitectura 1.5, es decir, juntando jsps con servlets para separar la lógica de la aplicación. Tendremos los siguientes productos:

Título	Autor	Precio
Luz de agosto	William Faulkner	30
Grandes esperanzas	Charles Dickens	35
El amor en los tiempos del cólera	Gabriel García Márquez	20
Travesuras de una niña mala	Mario Vargas Llosa	25

Taller práctico.

Juntándolo todo

- ▶ Para ello necesitaremos:
 - **Index.jsp:** Guarda la información de la tabla en una hasmap colocada en el contexto, donde la clave sea el id del libro y el valor una referencia a `es.uniovi.si.LibroBean`, clase que deberemos implementar.
 - Redirige (`response.sendRedirect(...)`) a la página `listaLibros.jsp`
 - **listaLibros.jsp:** Muestra una tabla en la que se exponen los productos en venta. Además, debajo de la tabla mostrará un formulario con:
 - Un combo box donde se muestren la lista de libros
 - Un edit box donde podamos meter un número indicando la cantidad de copias que queremos.

Taller práctico.

Juntándolo todo

- Un servlet `AddCarrito` que:
 - Reciba por parámetro el id del producto a añadir y la cantidad de elementos a añadir.
 - Gestion su adición a una `hasmap` que se almacene... ¿Dónde? Bajo el nombre de **carrito**
 - Redirija a `muestraCarrito.jsp`
 - `muestraCarrito.jsp`
 - Se valdrá de una etiqueta `<muestraCarrito>` que deberemos implementar y que ocultará el código necesario para que en la página se muestre por cada libro, el número de copias, el precio de esas X copias y al final, el precio total de la compra. **Cada libro debe salir mostrando su título, no su id.**
 - Mostrará un enlace a `listaLibros.jsp`
- **NOTA:** Buscar en TagSupport algo que nos dé acceso a la sesión del usuario.

Referencias

- www.javasoft.com
- www.theserverside.com
- <http://paginaspersonales.deusto.es/dipina/>
- Java Server Pages [2nd Edition] OReilly