Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en Red

Módulo Profesional: IAW

*Unidad de Trabajo 5.- **Programación web utilizando lenguaje de script servidor. Programación JavaServer Pages (JSP).***

Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada


I ES
SAN JUAN BOSCO

Índice de Contenidos

JavaServer Pages (JSP)

Class SERVLET – *Web.xml*

Formularios en HTML

Otros métodos utilizados para enviar información

Procesamiento de peticiones. *REQUEST*

Manejo de sesiones con JSP

Manejo de cookies con JSP

Abstract/Resumen:

Las JavaServer Pages (JSP) nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático. Simplemente escribimos el HTML regular de la forma normal, usando cualquier herramienta de construcción de paginas Web que usemos normalmente. Encerramos el código de las partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empiezan con "<%" y terminan con "%>".

Los JSPs son en realidad servlets: un JSP se compila a un programa en Java la primera vez que se invoca, y del programa en Java se crea una clase que se empieza a ejecutar en el servidor como un servlet. La principal diferencia entre los servlets y los JSPs es el enfoque de la programación: un JSP es una página Web con etiquetas especiales y código Java incrustado, mientras que un servlet es un programa que recibe peticiones y genera a partir de ellas una página web.

1.- JavaServer Pages (JSP)
1.1.- Introducción

- Las JavaServer Pages (JSP) nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático. Simplemente escribimos el HTML regular de la forma normal, usando cualquier herramienta de construcción de páginas Web que usemos normalmente.

- Encerramos el código de las partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empiezan con "<%" y terminan con "%>".

- Por ejemplo, aquí tenemos una sección de una página JSP que resulta en algo así como "Gracias por programar en JSP" para una URL como:

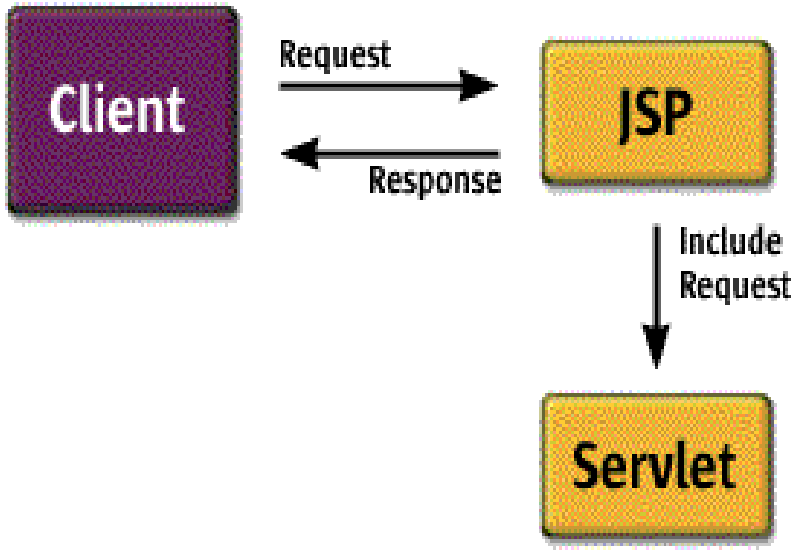
http://host/OrderConfirmation.jsp?title= Gracias+por+programar+en+JSP

OrderConfirmation.jsp

```
<i><%= request.getParameter("title") %></i>
```

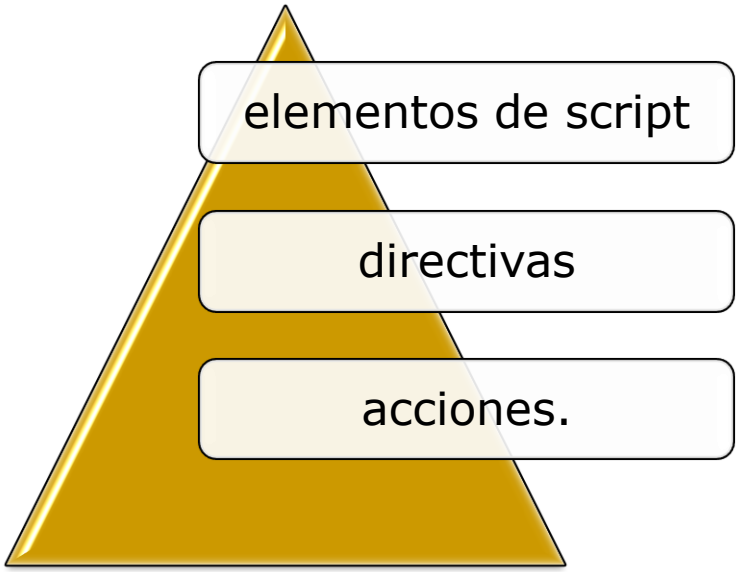
1.- JavaServer Pages (JSP)
1.1.- Introducción

- Normalmente daremos a nuestro fichero una extensión .jsp, y normalmente lo instalaremos en el mismo sitio que una página Web normal.
- Aunque lo que escribamos frecuentemente se parezca a un fichero HTML, detrás de la escena, la página JSP se convierte en un servlet normal, donde el HTML estático simplemente se imprime en el stream de salida estándard asociado con el método service del servlet.
- Esto normalmente sólo se hace la primera vez que se solicita la página, y los desarrolladores pueden solicitar la página ellos mismos cuando la instalan si quieren estar seguros de que el primer usuario real no tenga un retardo momentáneo cuando la página JSP sea traducida a un servlet y el servlet sea compilado y cargado.



1.- JavaServer Pages (JSP)
1.1.- Introducción

- Además del HTML normal, hay tres tipos de construcciones JSP que embeberemos en una página:



- *Los elementos de **script** nos permiten especificar código Java que se convertirá en parte del servlet resultante, las **directivas** Las directivas JSP nos permiten configurar alguna información que pueda ser usada en nuestra página JSP, por ejemplo cosas como importar clases, definir una página de error, incluir una página JSP en otra, …, y acciones que son construcciones de síntaxis XML para controlar el comportamiento del motor de Servlets. Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, o generar HTML para el plug-in Java.*

1.- JavaServer Pages (JSP)
1.2.- Sumario de Sintaxis

Elemento JSP	Síntaxis	Interpretación	Notas
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>.
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>.
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 son (con los valores por defecto en negrita): import="package.class" contentType="MIME-Type" isThreadSafe="true|false" session="true|false" buffer="sizekb|none" autoflush="true|false" extends="package.class" info="message" errorPage="url" isErrorPage="true|false" language="java"

1.- JavaServer Pages (JSP)
1.2.- Sumario de Sintaxis

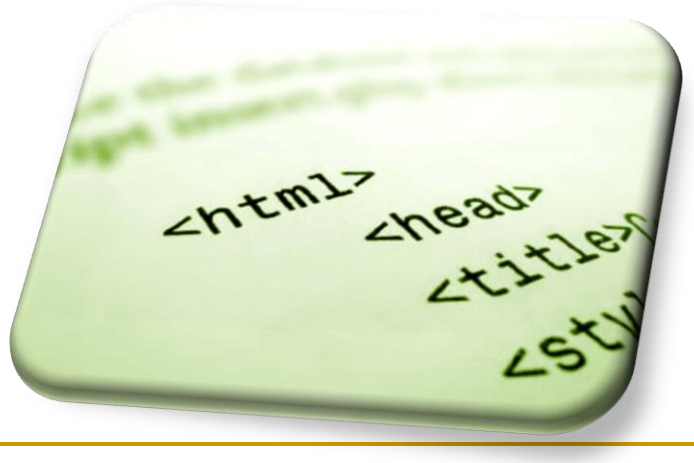
Elemento JSP	Síntaxis	Interpretación	Notas
Directiva include JSP	<%@ include file="url" %>	Un fichero del sistema local se incluirá cuando la página se traduzca a un Servlet.	El equivalente XML es: <jsp:directive.include file="url"\>. La URL debe ser relativa. Usamos la acción jsp:include para incluir un fichero en el momento de la petición en vez del momento de la tradución.
Comentario JSP	<%-- comment --%>	Comentario ignorado cuando se traduce la página JSP en un servlet.	Si queremos un comentario en el HTML resultante, usamos la síntaxis 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).
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"

1.- JavaServer Pages (JSP)
1.2.- Sumario de Sintaxis

Elemento JSP	Síntaxis	Interpretación	Notas
Acción jsp:setProperty	<jsp:setProperty att=val*/>	Selecciona las propiedades del bean, bien directamenteo 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.	
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.	

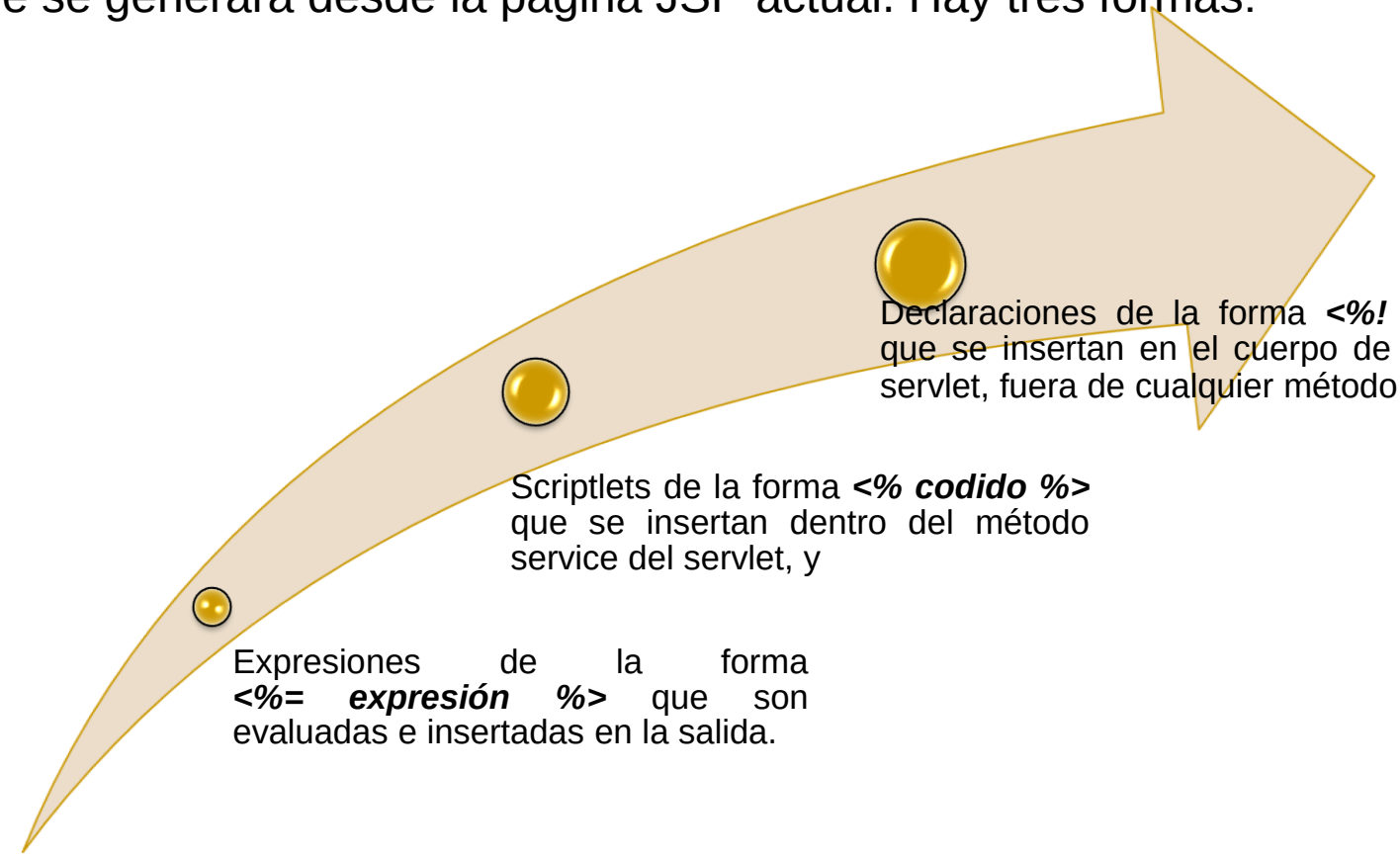
1.- JavaServer Pages (JSP)
1.3.- Plantilla de Texto: HTML estático

- En muchos casos, un gran porcentaje de nuestras páginas JSP consistirá en HTML estático, conocido como plantilla de texto.
- En casi todos lo aspectos, este HTML se parece al HTML normal, sigue la mismas reglas de sintaxis, y simplemente "pasa a través" del cliente por el servlet creado para manejar la página.
- No sólo el aspecto del HTML es normal, puede ser creado con cualquier herramienta que usemos para generar páginas Web.
- La única excepción a la regla de que "la plantilla de texto se pasa tal y como es" es que, si queremos tener "<%" en la salida, necesitamos poner "<\%" en la plantilla de texto.



1.- JavaServer Pages (JSP)
1.4.- Elementos de Script JSP

Los elementos de script nos permiten insertar código Java dentro del servlet que se generará desde la página JSP actual. Hay tres formas:



Declaraciones de la forma ***<%! codigo %>*** que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente.

Scriptlets de la forma ***<% codido %>*** que se insertan dentro del método service del servlet, y

Expresiones de la forma ***<%= expresión %>*** que son evaluadas e insertadas en la salida.

1.- JavaServer Pages (JSP)
1.4.1.- Expresiones JSP

Una expresión JSP se usa para insertar valores Java directamente en la salida. Tiene la siguiente forma:

<%= expresión Java %>

La expresión Java es evaluada, convertida a un ***String***, e insertada en la página. Por ejemplo, esto muestra la fecha y hora en que se solicitó la página: *<H1>Fecha/Hora: </H1> <%= new java.util.Date() %>*

Por otra parte, hay un gran número de variables predefinidas que podemos usar, que se definirán más adelante:

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

Aquí tenemos un ejemplo: *Nombre del equipo: <%= request.getRemoteHost() %>*

Finalmente, observa que los autores de XML pueden usar una sintaxis alternativa para las expresiones JSP:

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

1.- JavaServer Pages (JSP)
1.4.2.- Scriptlets JSP

Si queremos hacer algo más complejo que insertar una simple expresión, los scriptlets JSP nos permiten insertar código arbitrario dentro del método servlet que será construido al generar la página. Los Scriptlets tienen la siguiente forma:

<% Código Java %>

```
<%
out.println(“Mi primer código JSP”);
%>
```

Observa que 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 scriplets. Por ejemplo, el siguiente fragmento JSP, contiene una mezcla de texto y scritplets:

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

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

Si queremos usar los caracteres "%>" dentro de un scriptlet, debemos poner "%\>". Finalmente, observa que el equivalente XML de <% Código %> es :

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

1.- JavaServer Pages (JSP)
1.4.3.- Declaraciones JSP

Una declaración JSP nos permite definir métodos o campos que serán insertados dentro del cuerpo principal de la clase servlet (fuera del método service que procesa la petición).

<%! Código Java%>

Como las declaraciones no generan ninguna salida, normalmente se usan en conjunción con expresiones JSP o escriptlets. Por ejemplo, aquí tenemos un fragmento de JSP que imprime el número de veces que se ha solicitado la página actual desde que el servidor se arrancó (o la clase del servelt se modificó o se recargó):

```
<%! private int accessCount = 0; %>
Número de accesos a la página:
<%= ++accessCount %>
```

Como con los scriptlet, si queremos usar los caracteres "%>", ponemos "%\>". Finalmente, observa que el equivalente XML de <%! Código %> es:

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

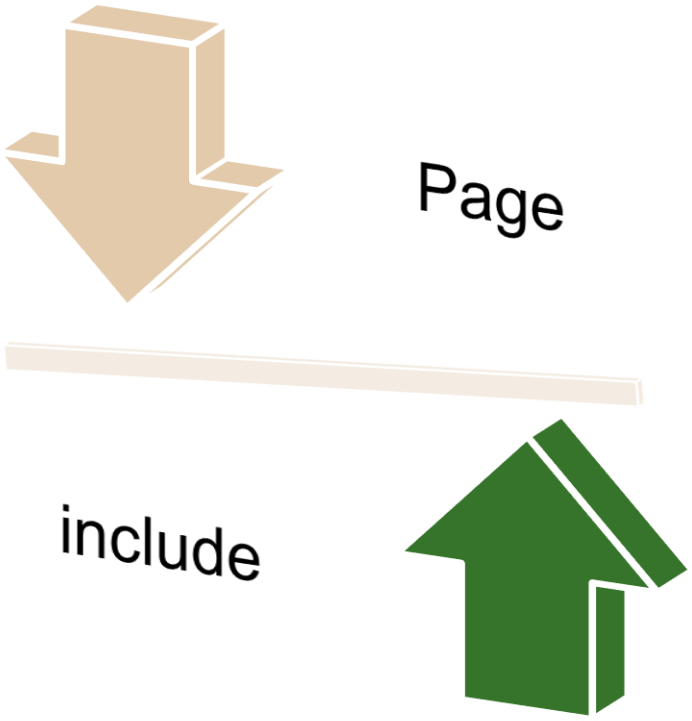
1.- JavaServer Pages (JSP)
1.5.- Directivas JSP

Una directiva JSP afecta a la estructura general de la clase servlet.

Normalmente tienen la siguiente forma:

<%@ directive attribute="value" %>

Hay dos tipos principales de directivas:



1.- JavaServer Pages (JSP)
1.5.1.- Directiva page

La ***directiva page*** nos permite definir uno o más de los siguientes atributos sensibles a las mayúsculas:

- **import="package.class“**
  import="package.class1,...,package.classN".

  Esto nos permite especificar los paquetes que deberían ser importados. Por ejemplo:
- **<%@ page import="java.util.*" %>** El atributo import es el único que puede aparecer múltiples veces.
- **contentType="MIME-Type"**

  contentType="MIME-Type; charset=Character-Set"

  Esto especifica el tipo MIME de la salida. El valor por defecto es text/html. Por ejemplo, la directiva: <%@ page contentType="text/plain" %> tiene el mismo valor que el scriptlet <% response.setContentType("text/plain"); %>

1.- JavaServer Pages (JSP)
1.5.1.- Directiva page

- **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 del autor sincroniza las variables de ejemplar. Un valor de false indica que el servlet debería implementar SingleThreadModel, con peticiones enviadas serialmente o con peticiones simultáneas siendo entregadas por ejemplares separados del servelt.

- **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.

1.- JavaServer Pages (JSP)
1.5.1.- Directiva page

- **buffer="sizekb|none".** Esto especifica el tamaño del buffer para el JspWriter out. El valor por defecto es específico del servidor, debería ser de al menos 8kb.
- **autoflush="true|false".** Un valor de true (por defecto) indica que el buffer debería desacargase cuando esté lleno. Un valor de false, raramente utilizado, indica que se debe lanzar una excepción cuando el buffe se sobrecargue. Un valor de false es ilegal cuando usamos buffer="none".
- **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="message".** Define un string que puede usarse para ser recuperado mediante el método getServletInfo.
- **errorPage="url".** Especifica una página JSP que se debería procesar si se lanzará cualquier Throwable pero no fuera capturado en la página actual.

1.- JavaServer Pages (JSP)
1.5.1.- Directiva page

- **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="java".** En algunos momentos, esto está pensado para especificar el lenguaje a utilizar. Por ahora, no debemos precuparnos por él ya que java es tanto el valor por defecto como la única opción legal.

Ejemplo:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.io.*,java.lang.*,java.util.*,java.sql.*,info_sesion.Sesion,Carrito.*" %>
```

1.- JavaServer Pages (JSP)
1.5.2.- Directiva include

Esta directiva nos permite incluir ficheros en el momento en que la página JSP es traducida a un servlet.

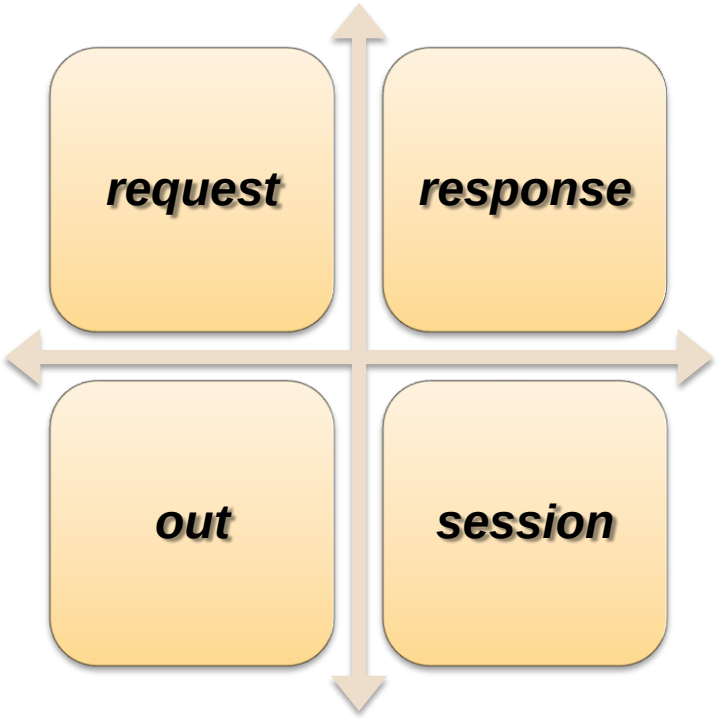
<%@ include file="url relativa" %>

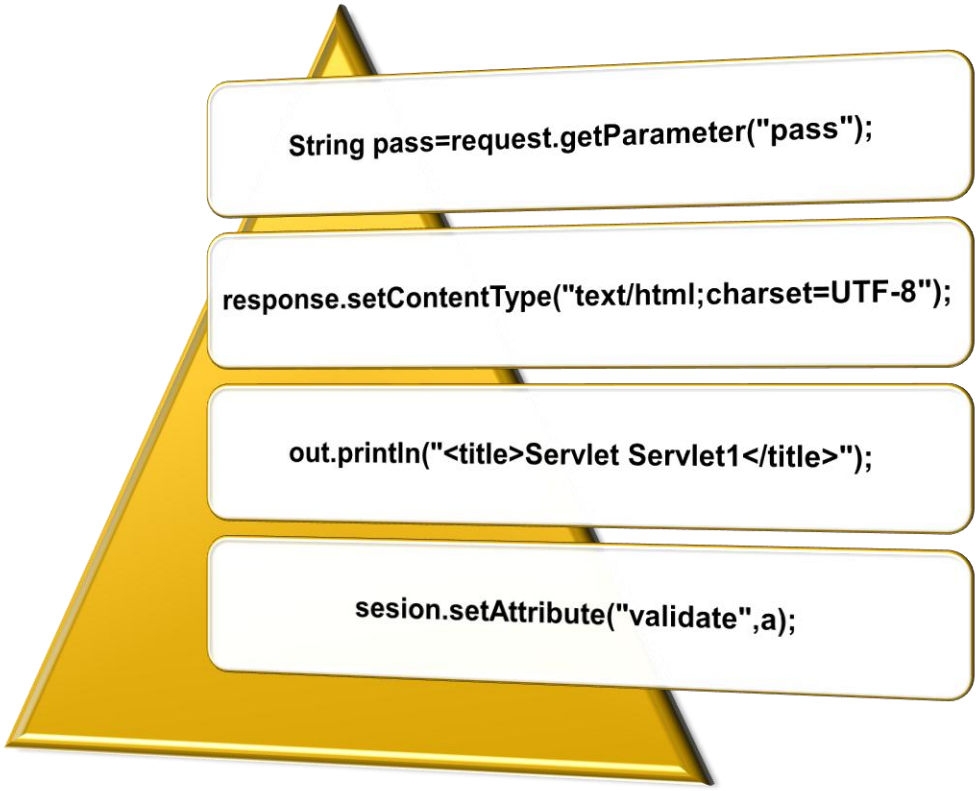
La URL especificada normalmente se interpreta como relativa a la página JSP a la que se refiere, pero, al igual que las URLs relativas en general, podemos decirle al sistema que interpreta la URL reativa al directorio home del servidor Web empezando la URL con una barra invertida. Los contenidos del fichero incluido son analizados como texto normal JSP, y así pueden incluir HTML estático, elementos de script, directivas y acciones.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.io.*,java.lang.*,java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN”
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
</HEAD>
<BODY>
<%@ include file="/navbar.html" %>
<!-- Part specific to this page ... -->
</BODY>
</HTML>
```

1.- JavaServer Pages (JSP)
1.6.- Variables Predefinidas

Tenemos ocho variables definidas automáticamente, algunas veces llamadas objetos implícitos. Las variables disponibles son: request, response, out, session, application, config, pageContext, y page.





2.- Class SERVLET – Web.xml

```
import java.io.*;
import java.net.*;
 import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldExample extends HttpServlet
{
        protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");
            PrintWriter out = response.getWriter();
                    /* TODO poner la página de salida
                     out.println("<html>");
                     out.println("<head>");
                     out.println("<title>Servlet Servlet</title>");
                     out.println("</head>");
                     out.println("<body>");
                     out.println("<h1>Hola mundo " + request.getContextPath () + "</h1>");
                     out.println("</body>");
                     out.println("</html>");
                      */
            out.close();
        }
        // Métodos: doGet, doPost, getServletInfo
```

2.- Class SERVLET – Web.xml

Hay una pequeña característica de Tomcat que está relacionada con web.xml. Tomcat permite al usuario definir los valores por defecto de web.xml para todos los contextos poniendo un fichero web.xml por defecto en el directorio conf. Cuando construimos un nuevo contexto, Tomcat usa el fichero web.xml por defecto como la configuración base y el fichero web.xml específico de la aplicación (el localizado en el WEB-INF/web.xml de la aplicación).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
   <display-name>Servlet 2.4 Examples</display-name>
   <description>
    Servlet 2.4 Examples.
   </description>
      <servlet>
            <servlet-name>HelloWorldExample</servlet-name>
           <servlet-class>paquete1.paquete2.HelloWorldExample</servlet-class>
      </servlet>
      <servlet-mapping>
           <servlet-name>HelloWorldExample</servlet-name>
          <url-pattern>/servlets/HelloWorldExamplet</url-pattern>
   </servlet-mapping>
</web-app>
```

3.- Formularios en HTML

Su utilidad se basa fundamentalmente en el envió de datos a scriprt, CGI, servlets ..etc almacenados en un contenedor/repositorio del servidor web. Podríamos decir que los formularios son el interface mediante el cual una persona introduce datos en un programa web.

Las etiquetas que definen un formulario son, a "grosso modo":

<form action="elemento_destino" method="metodo">…</form>

Básicamente, podemos ver que el contenido del formulario viene delimitado por las etiquetas de apertura y cierre <form> y </form>. Y sólo cabe destacar los parámetros de la etiqueta de apertura , que son:

- action="elemento_destino" : donde elemento_destino es la dirección del script o CGI que va a recibir los parámetros que se introduzcan en el formulario.
- method="método" : mediante este parámetro especificamos el método mediante el cual se mandan los datos del formulario al script o CGI. Existen 2 métodos, GET y POST.

  GET: el conjunto de datos del formulario se agrega al URL especificado por el atributo action (con un signo de interrogación ("?") como separador) y este nuevo URL se envía al agente procesador.

  POST: Con el método HTTP post, el conjunto de datos del formulario se incluye en el cuerpo del formulario y se envía al agente procesador.

3.- Formularios en HTML

Etiqueta select

Se definen mediante las etiquetas:

```
<select name="nombre" [size="filas" multiple]>
<option value="01">valor1
<option deleted value="02">valor2
<option">valor3
(…)
</select>
```

Se utilizan para realizar selecciones de un menú, que definiremos por tantas opciones como etiquetas <option> utilicemos. Donde nombre es el nombre del parámetro, filas es el número de filas del menú que queremos mostrar (por defecto 1) y múltiple indica si el menú permitirá activar múltiples opciones o no. El atributo value es opcional e indica el valor que se va a enviar al servidor.

3.- Formularios en HTML

Etiqueta textarea

Se define mediante la etiqueta:

<textarea name="nombre" rows="filas" cols="columnas">

[texto predefinido]
</textarea>

En donde nombre es el nombre del parámetro, y rows y cols es el número de filas y columnas (en caracteres), respectivamente, que ocupará el campo de entrada. Todos estos parámetros son obligatorios. Cabe mencionar que entre las etiquetas de apertura y cierre se puede insertar el texto predefinido que se quiera (si se quiere) que se mostrará como texto por defecto.

3.- Formularios en HTML

Etiqueta input

La etiqueta <INPUT> va a definir la mayoría de los diferentes elementos que va a contener el formulario. Sus atributos y valores son:

TYPE = " tipo ", donde tipo puede ser uno cualquiera de los elementos que veremos a continuación:

text, que sirve para introducir una caja de texto simple, y admite los parámetros:

name=" nombre", que asigna de forma unívoca un nombre identificador a la variable que se introduzca en la caja de texto.

maxlenght=" n ", que fija el número máximo de caracteres que se pueden itroducir en la caja de texto.

size=" n ", que establece el tamaño de la caja de texto en pantalla.

value=" texto ", que establece el valor por defecto del texto que aparecerá en inicialmente en la caja de texto.

disabled, que desactiva la caja de texto, por lo que el usuario no podrá escribir nada en ella.

accept = " lista de content-type", Indica el tipo de contenido que aceptará el servidor.

Sus posibles valores son:

text/html , application/msexcel , application/msword , application/pdf , image/jpg , image/gif ....etc.

readonly, que establece que el texto no puede ser modificado por el usuario.

tabindex = " n ", que especifica el orden de tabulador que tendrá el campo respecto todos los elementos que incluye el formulario.

alt= " texto ", que asigna una pequeña descripción al elemento.

3.- Formularios en HTML

radio, que define un conjunto de elementos de formulario de tipo circular, en los que el usuario debe optar por uno solo de ellos, que se marca con el ratón o tabulador. Admite los parámetros:

name=" nombre", que asigna un nombre identificador único a la variable definida por el elemento. Este identificador debe ser el mismo para todos los elementos radio de un grupo.

value =" valor ", que define un valor posible de la variable para cada uno de los radio botones.

checked, que marca por defecto uno de los radio botones del grupo.

disabled, que desactiva el radio botón, por lo que el usuario no podrá marcarlo.

tabindex = " n ", que especifica el orden de tabulador que tendrá el campo respecto todos los elementos que incluye el formulario.

Ejemplo.-

```
<form action="mailto:yo@miservidor.com" method="post" enctype="text/plain" name="miform">
  marca tu equipo favorito:
 <input type="Radio" name="equipo" value="madrid" checked>Real Madrid
       <input type="Radio" name="equipo" value="atletico">Atlético de Madrid
       <input type="Radio" name="equipo" value="barcelona">Barcelona  </form>
```

3.- Formularios en HTML

checkbox, que define una o más casillas de verificación, pudiendo marcar el usuario las que desee del conjunto total. Si pinchamos una casilla con el ratón o la activamos con el tabulador y le damos a la barra espaciadora la casilla se marca; si volvemos a hacerlo, la casilla se desmarca. Sus parámetros complementarios son:

name=" nombre", que asigna un nombre identificador único a la variable definida por el elemento. Este identificador debe ser el mismo para todos los elementos conjunto de casillas.

value =" valor ", que define un valor posible de la variable para cada uno de casillas de verificación.

checked, que marca por defecto una o más de las casillas del grupo.

disabled, que desactiva la casilla de verificación, por lo que el usuario no podrá marcarla.

tabindex = " n ", que especifica el orden de tabulador que tendrá el campo respecto todos los elementos que incluye el formulario.

Ejemplo.-

```
<form action="mailto:yo@miservidor.com" method="post" enctype="text/plain" name="miform">
  marca tu música favorita:
  <input type="checkbox" name="musica" value="rock" checked>Rock
  <input type="checkbox" name="musica" value="pop" checked>Pop
  <input type="checkbox" name="musica" value="heavy">Heavy
  <input type="checkbox" name="musica" value="tecno">Tecno
</form>
```

3.- Formularios en HTML

button, que define un botón estándar. Este botón puede ser usado para diferentes acciones, pero normalmente mediante JavaScript, con el evento "OnClick". Sus parámetros son:

name=" nombre", que asigna un nombre al botón, que nos puede servir para acciones con lenguaje de script.

value=" valor ", que define el texto que va a figurar en el botón.

disabled, que desactiva el botón, de tal forma que no se produce ninguna acción cuando se pulsa, pués permanece inactivo.

tabindex = " n ", que especifica el orden de tabulador que tendrá el campo respecto todos los elementos que incluye el formulario.

Ejemplo.-

```
<form action="mailto:yo@miservidor.com" method="post" enctype="text/plain"
name="miform">
  <input type="Button" name="boton" value="pulsame">
</form>
```

3.- Formularios en HTML

password, que define una caja de texto para contener una clave o password, por lo que el texto que introduzca el usuario aparecerá como asteriscos, por motivos de seguridad. Sus parámetros opcionales son los mismos que los del tipo text.

Ejemplo.-

```
<form action="mailto:yo@miservidor.com" method="post"
 enctype="text/plain" name="miform">
  <input type="password" size="15" maxlength="10">
</form>
```

3.- Formularios en HTML

hidden, que define un campo invisible, por lo que no se ve en pantalla. Aunque parece así definido que no tiene utilidad, sus usos son varios e importantes, y los veremos más tarde. Sus atributos son:

name=" nombre", que asigna un nombre identificador único al campo oculto.

value=" valor ", que va a ser el valor fijo que se le va a pasar al programa del servidor, ya que el usuario no puede modificarlo. En realidad este valor no tiene porqué ser fijo, ya que lo vamos a poder modificar mediante código de script, lo que nos va a permitir ir pasando una serie de variables ocultas de una página a otra.

Ejemplo.-

```
<form action="mailto:yo@miservidor.com" method="post“ name="miform">
<input type="hidden" name="contraseña" value="123ABC">
.....
</form>
```

3.- Formularios en HTML

submit, que incorpora al formulario un botón de envío de datos. Cuando el usuario pulsa este botón los datos que ha introducido en los diferentes campos del formulario son enviados al programa del servidor o a la dirección de correo indicada en action. Sus atributos son:

value=" valor ", que define el texto que va a aparecer en el botón de envío.

disabled, que desactiva el botón, de tal forma que no se produce ninguna acción cuando se pulsa, pués permanece inactivo.

tabindex = " n ", que especifica el orden de tabulador que tendrá el campo respecto todos los elementos que incluye el formulario.

reset, que define un botón que al ser pulsado por el usuario borra todos los datos que hubiera introducido en cualquiera de los campos del formulario. Sus atributos son los mismos que los de SUBMIT.

4.- Otros métodos utilizados para enviar información

Además de los formularios, mediante los caracteres de **?** y **&** colocándolos en un enlace también podemos enviar información.

```
<a href="recoge.jsp?valor=1&valor2=2"> Enviar </A>
```

```
<a href="recoge.jsp?valor=<%=expresion_java%>&valor2=2"> Enviar </A>
```

```
out.print("<a href=\"recoger.jsp?valor="+(expresion_java)+"&valor2=3.14\"> Enviar </a>");
```

De este modo nos aparece un enlace *Enviar* en el cual llama a *recoge.jsp* enviando dos datos *(nombre/valor).*

La cuestión que surge es como el programa jsp puede recuperar estos datos para poder procesarlos y generar un resultado

5.- Procesamiento de peticiones. REQUEST

- Para recoger los datos que llegan desde la barra del navegador, java utiliza una variable predefinida denominada *request* con la que utiliza el metodo *getParameter*.

- Este metodo, *getParameter* requiere el nombre de la variable, de la cual obtiene la variable que contiene.

String aux= request.getParameter("campo");

con este código obtenemos el contenido de la variable campo y se lo asignamos a la variable aux de tipo String.

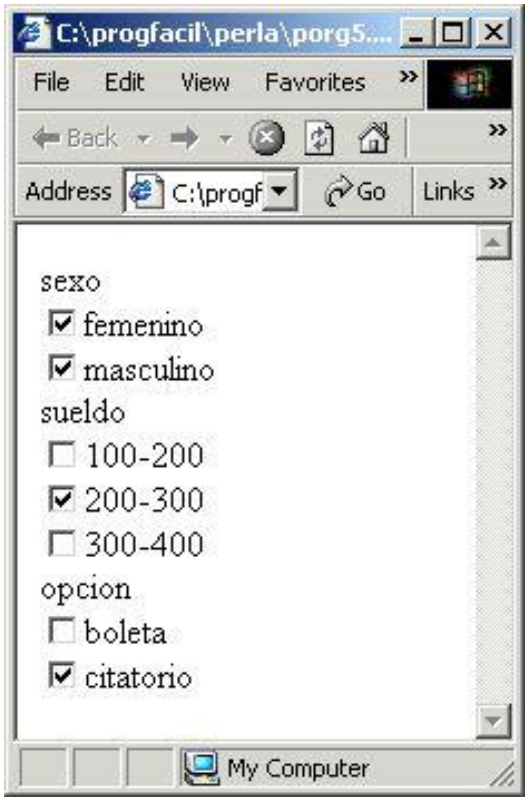
- Una cuestión a tener en cuenta es que todos los datos que se van a enviar por la barra del navegador van a ser de tipo texto, por lo que si se quiere pasar un número, y después utilizar su valor para hacer alguna operación debemos de transfórmalo, por ejemplo:

int valor= Integer.valueOf(request.getParameter("valor")).intValue();

lo que hacemos es del mismo modo que el anterior recoger el valor de la variable/nombre valor y lo transformamos a entero, guardándolo en la variable valor, de tipo int para posteriormente utilizar el valor numerico en la pagina jsp, estos ejemplos son los más típicos, recuerda que puedes pasarlos al formato que más te convenga, por ejemplo pasarlo a double, float.

5.- Procesamiento de peticiones. REQUEST

```
<%
String campo= request.getParameter("campo");
int valor= Integer.valueOf(request.getParameter("valor")).intValue();
int valor2= Integer.valueOf(request.getParameter("valor2")).intValue();
out.println(campo);
%>
<br>
<% //salto de linea out.println(valor);%>
<br>
<%out.println(valor2);%>
```



5.- Procesamiento de peticiones. REQUEST

En el caso de tener un campo/atributo que puede devolver más de un valor precisamos:

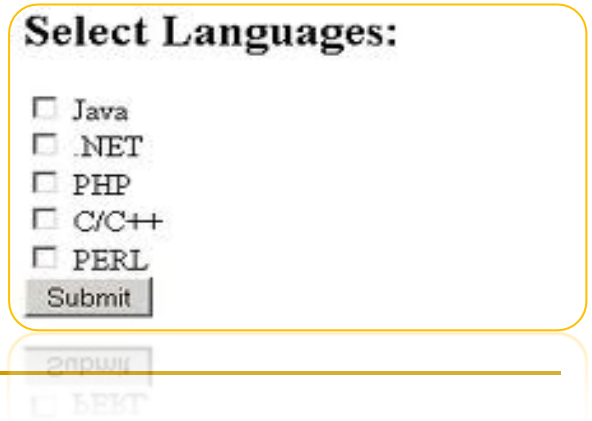
request.getParametresValues()

<input type="checkbox" id="aviso" name="aviso" value="0" /> <label for="aviso">Primero</label>
<input type="checkbox" id="aviso" name="aviso" value="1" /> <label for="aviso">Segundo</label>
<input type="checkbox" id="aviso" name="aviso" value="2" /> <label for="aviso">Tercero</label>
...
<input type="checkbox" id="aviso" name="aviso" value="6" /> <label for="aviso">Séptimo</label>

Luego cuando lo recibes, request.getParameterValues("aviso") te devuelve un String[] así que puedes acceder a ello normalmente con algo como:

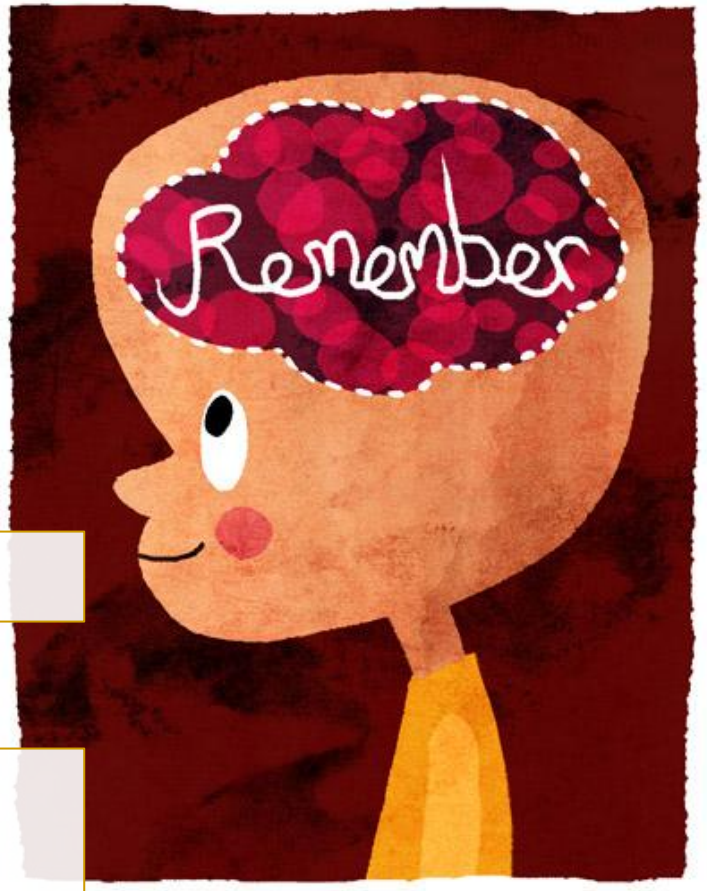
String[] titulos = request.getParameterValues(“aviso");

for (int i = 0; i< titulos.length; i++)
{
····System.out.println(titulos[i]);
}



6.- Manejo de sesiones con JSP

Una sesión es una serie de comunicaciones entre un cliente y un servidor en la que se realiza un intercambio de información. Por medio de una sesión se puede hacer un seguimiento de un usuario a través de la aplicación.



El tiempo de vida de una sesión comienza cuando un usuario se conecta por primera vez a un sitio web pero su finalización puede estar relacionada con tres circunstancias:

- Cuando se abandona el sitio web.
- Cuando se alcanza un tiempo de inactividad que es previamente establecido, en este caso la sesión es automáticamente eliminada. Si el usuario siguiera navegando se crearía una nueva sesión.
- Se ha cerrado o reiniciado el servidor.

6.- Manejo de sesiones con JSP

- Una posible aplicación de las sesiones es en el comercio electrónico. En este caso una sesión permite ir eligiendo una serie de productos e irlos añadiendo a nuestro “carrito” y así hasta finalizar la compra. Sin el uso de sesiones no se podría hacer porque al ir navegando de una página a otra se iría perdiendo toda la información.
- También se utilizan para la identificación de usuarios, en la que se deben de introducir un login y un password. Después de haber hecho esto el usuario tendrá una serie de permisos sobre las páginas que va a visitar, de tal forma que si un usuario intenta pasar a una página si haberse identificado, el sistema comprobará que no se ha identificado y sería redireccionado a la página de identificación. Para poder realizarse estas operaciones es necesario almacenar en unas tipo sesión la información necesaria para saber que el usuario se ha identificado correctamente.

Para obtener la sesión de un usuario se utiliza el método getSession() que devuelve una interfaz de tipo HttpSession.

```
<%
HttpSession sesion=request.getSession();
%>
```

6.- Manejo de sesiones con JSP

Guardar objetos en una sesión:

Para guardar un objeto en una sesión se utiliza el método setAttribute(), Este método utiliza dos argumentos:

SetAttribute(java.lang.String name, java.lang.Object value)

Un ejemplo de cómo guardar una cadena de texto en la sesión:

```
<%@page import=”java.util.*” session=”true” %>

<%
HttpSession sesion=request.getSession();
sesion.setAttribute(“trabajo”,”Paginas de JSP”);
%>
```

6.- Manejo de sesiones con JSP

Si se quiere pasar un parámetro que no sea un objeto es necesario realizar una conversión:

```
<%@page import=”java.util.*” session=”true” %>
<%
HttpSession sesion=request.getSession();
Integer edad=new Integer(26);
sesion.setAttribute(“edad”,edad);
%>
```

Para introducir un objeto de la clase X:

```
<%@page import=”java.util.*” session=”true” %>
 <%
HttpSession sesion=request.getSession();
sesion.setAttribute(“carrito_compra”,new Carrito());
%>
```

6.- Manejo de sesiones con JSP

Recuperar objetos de una sesión

Para poder realizar este paso se utiliza el método getAttribute()

getAttribute(java.lang,String nombre)

Cuando este método devuelve el objeto no establece en ningún momento de qué tipo de objeto se trata(String, Vector...)

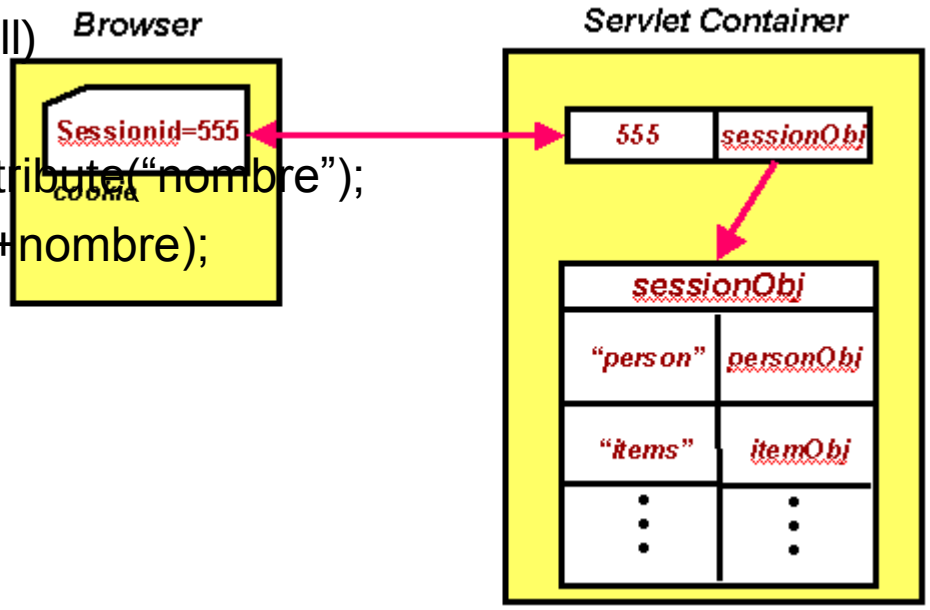
Por ello si se conoce previamente el tipo de objeto que puede devolver tras ser recuperado de la sesión es necesario realizar un casting, para convertir el objeto de tipo genérico al objeto exacto que se va a usar. Para realizar esta operación se añade el tipo de objeto al lado de tipo HttpSession que utiliza el método getAttribute() para obtener el objeto que devuelve:

```
<%
HttpSession sesion=request.getSession();
String nombre=(String)sesion.getAttribute(“nombre”);
out.println(“Contenido de nombre: “+nombre);
%>
```

6.- Manejo de sesiones con JSP

- Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el método getAttribute(), el valor devuelto será null. Por ello habrá que prestar especial atención ya que si se realiza el casting de un valor null el contenedor JSP devolverá un error. Lo mejor en estos casos es adelantarse a los posibles errores que pueda haber.

```
<%
if(sesion.getAttribute("nombre")!=null)
{
String nombre=(String)sesion.getAttribute("nombre");
out.println("Contenido de nombre: "+nombre);
}
%>
```



6.- Manejo de sesiones con JSP

- En el caso de tipos primitivos deben de ser convertidos a objetos previamente a su integración sesión de tal forma que su proceso de extracción viene a ser similar:

```
<%
 HttpSession sesion=request.getSession();
Integer edad=(Integer)sesion.getAttribute(“edad”);
out.println(“Edad: “+edad.intValue());
 %>
```

- En esta ocasión el objeto devuelto y convertido a Integer no es del todo válido ya que es necesario obtener de él el valor entero. Para ello se utiliza el método intValue() que devuelve el valor que realmente se había guardado previamente en la sesión.

6.- Manejo de sesiones con JSP

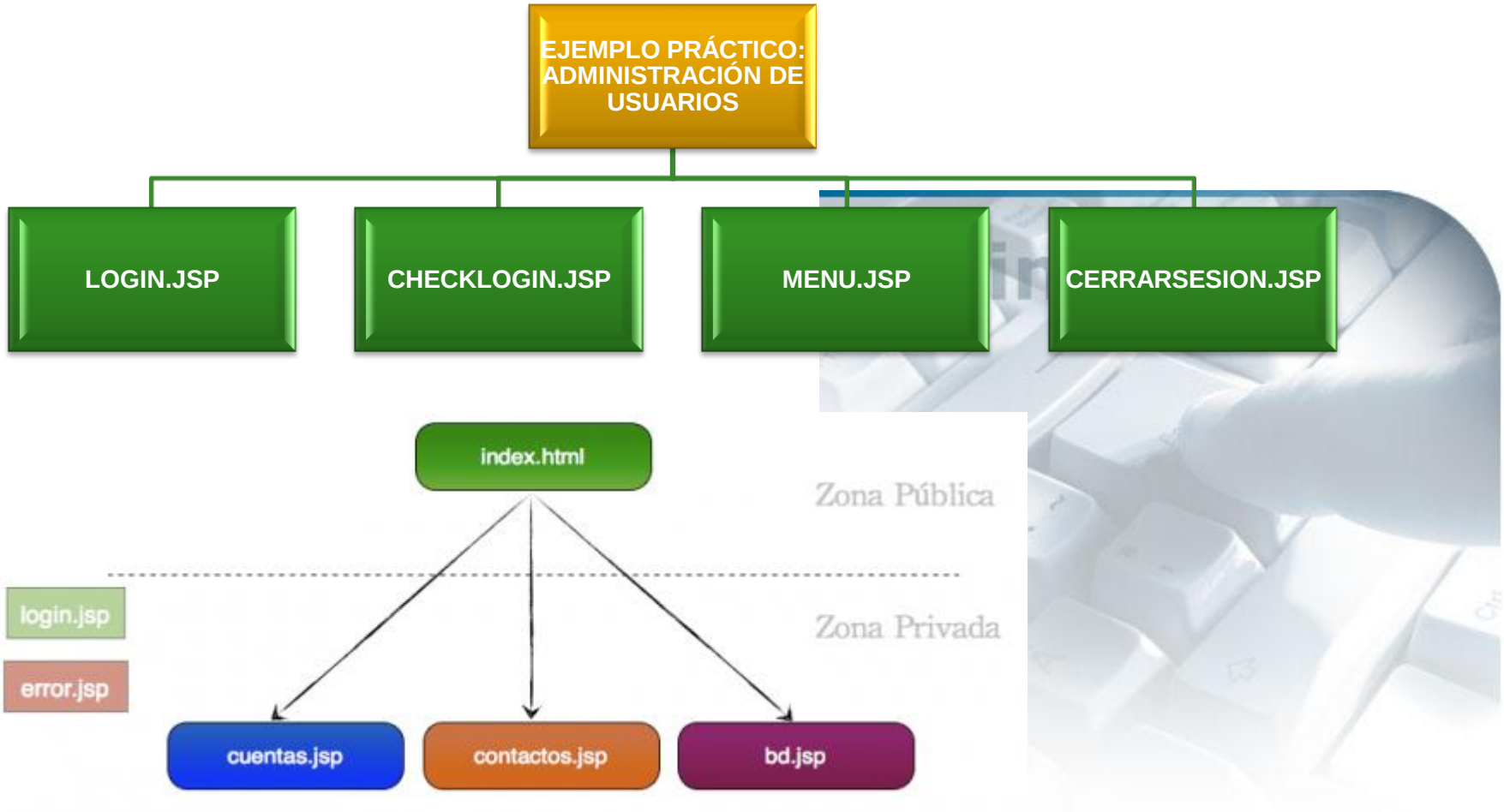
Principales métodos:

- **Object getAttribute(String nombreAtributo):** devuelve el objeto almacenado en la sesión actual, y cuya referencia se corresponde con el nombre de atributo indicado por parámetro como un objeto de la clase String. Sea cual sea la clase del objeto almacenado en la sesión, este método siempre devolverá un objeto de la clase genérica Object, a la hora de recuperarlo se deberá realizar la transformación de clases correspondiente. Este método devuelve null si el objeto indicado no existe.
- **Enumeration getAttributeNames():** este método devuelve en un objeto Enumeration del paquete java.util, que contiene los nombres de todos los objetos almacenados en la sesión actual.
- **long getCreationTime():** devuelve la fecha y hora en la que fue creada la sesión, medido en milisegundos desde el 1 de enero de 1970.
- **String getId():** devuelve una cadena que se corresponde con el identificador único asignado a la sesión. Luego se ve que este valor se corresponde con el valor de el *cookie* JSESSIONID utilizada para poder realizar el mantenimiento de la sesión de un usuario determinado, y en el caso de no utilizar *cookies se corresponde con la información que se añade* al final de cada enlace cuando se utiliza el mecanismo de reescritura de URLs.
- **long getLastAccesesedTime():** devuelve en milisegundos la fecha y hora de la última vez que el cliente realizó una petición asociada con la sesión actual.

6.- Manejo de sesiones con JSP

- **int getMaxInactiveInterval():** devuelve el máximo intervalo de tiempo, en segundos, en el que una sesión permanece activa entre dos peticiones distintas de un mismo cliente, es decir, es el tiempo de espera máximo en el que pertenece activa una sesión sin que el cliente realice ninguna petición relacionada con la sesión actual. El valor por defecto que puede permanecer inactiva unas sesión es de 30 segundos. Una vez transcurrido este tiempo el contenedor de *servlets (servlet container) destruirá la sesión, liberando de la memoria todos los objetos que* contiene la sesión que ha caducado.
- **void invalidate():** este método destruye la sesión de forma explícita, y libera de memoria todos los objetos (atributos) que contiene la sesión.
- **boolean isNew():** devuelve verdadero si la sesión se acaba de crear en la petición actual o el cliente todavía no ha aceptado la sesión (puede rechazar el *cookie de inicio de sesión). Este* método devolverá falso si la petición que ha realizado el cliente ya pertenece a la sesión actual, es decir, la sesión ya ha sido creada previamente,
- **void removeAttribute(String nombreAtributo):** elimina el objeto almacenado en la sesión cuyo nombre se pasa por parámetro. Si el nombre del objeto indicado no se corresponde con ninguno de los almacenados en la sesión, este método no realizará ninguna acción.
- **void setAttribute(String nombre, Object valor):** almacena un objeto en la sesión utilizando como referencia el nombre indicado como parámetro a través de un objeto String.
- **void setMaxInactiveInterval(int intervalo):** establece, en segundos, el máximo tiempo queuna sesión puede permanecer inactiva antes de ser destruida por el contenedor de *servlets.*

6.- Manejo de sesiones con JSP



6.- Manejo de sesiones con JSP

login.jsp

```
<%@page contentType="text/html; charset=iso-8859-1"
session="true" language="java" import="java.util.*" %>
<html>
<head><title>Proceso de login</title>
</head>
<body>
<b>Proceso de identificación</B>
<p>
    <% if(request.getParameter("error")!=null)
     {
     out.println(request.getParameter("error"));
     } %>
          <form action="checklogin.jsp" methop="post">
          usuario: <input type="text" name="usuario" size=20><br>
          clave: <input type="text" name="clave" size=20><br>
          <input type="submit" value="enivar"><br>
          </form>
</body>
</html>
```

6.- Manejo de sesiones con JSP

checklogin.jsp

```
<%@ page session="true" %>
<%
	String usuario ;
	String clave ;
	if (request.getParameter("usuario") != null)
		usuario = request.getParameter("usuario");
	if (request.getParameter("clave") != null)
		clave = request.getParameter("clave");
	if (usuario.equals("spiderman") && clave.equals("librojsp")) {
		HttpSession sesionOk = request.getSession();
		sesionOk.setAttribute("usuario",usuario);
%>
		// javascript para redireccionar menu.jsp
<%
	} else {
%>
		// error  y javascript para redireccionar a login.jsp
<%
	}
%>
```

6.- Manejo de sesiones con JSP

menu.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
      String usuario;
      HttpSession sesionOk = request.getSession();
      if (sesionOk.getAttribute("usuario") == null) {
%>
      // Error javascript para redireccionar a un hmt/jsp de error
<%
      }  else  {
      usuario = (String)sesionOk.getAttribute("usuario");
      }
%>
<html>
.............
<b>Menú de administración</b><br>
.............
<p>
<li> <a href="cerrarsesion.jsp">Cerrar sesión</a>
</body>
</html>
```

6.- Manejo de sesiones con JSP

cerrarsesion.jsp

La última opción que incorpora el menú es la de “Cerrar sesión”, que será de gran utilidad cuando se haya finalizado el trabajo y queremos estar seguro que nadie realiza ninguna acción con nuestro usuario y clave.

Al pulsar este enlace, se recupera de nuevo la sesión y mediante el método invalidate() se da por finalizada la sesión.

```
<%@ page session="true" %>
<%
HttpSession sesionOk = request.getSession();
sesionOk.invalidate();
%>
<jsp:forward page="login.jsp"/>
```

7.- Manejo de cookies con JSP

- Las cookies constituyen una potente herramienta empleada por los servidores Web para almacenar y recuperar información acerca de sus visitantes. Mediante el uso de cookies se permite al servidor Web recordar algunos datos concernientes al usuario, como sus preferencias para la visualización de las páginas de ese servidor, nombre y contraseña, productos que más le interesan, etc.

- ***Una cookie no es más que un fichero de texto que algunos servidores piden a nuestro navegador que escriba en nuestro disco duro, con información acerca de lo que hemos estado haciendo por sus páginas.***

7.- Manejo de cookies con JSP

Crear un cookie

- Un cookie almacenado en el ordenador de un usuario está compuesto por un nombre y un valor asociado al mismo. Además, asociada a este cookie pueden existir una serie de atributos que definen datos como su tiempo de vida, alcance, dominio, etc.
- Cabe reseñar que los cookies, no son más que ficheros de texto, que no pueden superar un tamaño de 4Kb, además los navegadores tan sólo pueden aceptar 20 cookies de un mismo servidor web (300 cookies en total).
- Para crear un objeto de tipo Cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar. El siguiente ejemplo crearía un objeto Cookie que contiene el nombre “nombre” y el valor “objetos” (String).

```
<%
Cookie miCookie=new Cookie(“nombre”,”objetos”);
%>
```

7.- Manejo de cookies con JSP

- También se pueden guardar valores o datos que provengan de páginas anteriores y que hayan sido introducidas a través de un formulario:

```
<%
Cookie miCookie=null;
String ciudad= request.getParameter("formCiudad");
miCookie=new Cookie("ciudadFavorita",ciudad);
%>
```

- Una vez que se ha creado un cookie, es necesario establecer una serie de atributos para poder ser utilizado. El primero de esos atributos es el que se conoce como tiempo de vida.
- Por defecto, cuando creamos un cookie, se mantiene mientras dura la ejecución del navegador. Si el usuario cierra el navegador, los cookies que no tengan establecido un tiempo de vida serán destruidos.

7.- Manejo de cookies con JSP

- Por tanto, si se quiere que un cookie dure más tiempo y esté disponible para otras situaciones es necesario establecer un valor de tiempo (en segundos) que será la duración o tiempo de vida del cookie. Para establecer este atributo se utiliza el método setMaxAge(). El siguiente ejemplo establece un tiempo de 31 días de vida para el cookie “unCookie”:

```
<%
unCookie.setMaxAge(60*60*24*31);
%>
```

Si se utiliza un valor positivo, el cookie será destruido después de haber pasado ese tiempo, si el valor es negativo el cookie no será almacenado y se borrará cuando el usuario cierre el navegador. Por último si el valor que se establece como tiempo es cero, el cookie será borrado.

7.- Manejo de cookies con JSP

- Otros de los atributos que se incluye cuando se crea un cookie es el path desde el que será visto, es decir, si el valor del path es “/” (raíz), quiere decir que en todo el site se podrá utilizar ese cookie, pero si el valor es “/datos” quiere decir que el valor del cookie sólo será visible dentro del directorio “datos”. Este atributo se establece mediante el método setPath().

<%

unCookie.setPath(“/”);

%>

Una vez que se ha creado el objeto Cookie, y se ha establecido todos los atributos necesarios es el momento de crear realmente, ya que hasta ahora sólo se tenía un objeto que representa ese cookie.

<%

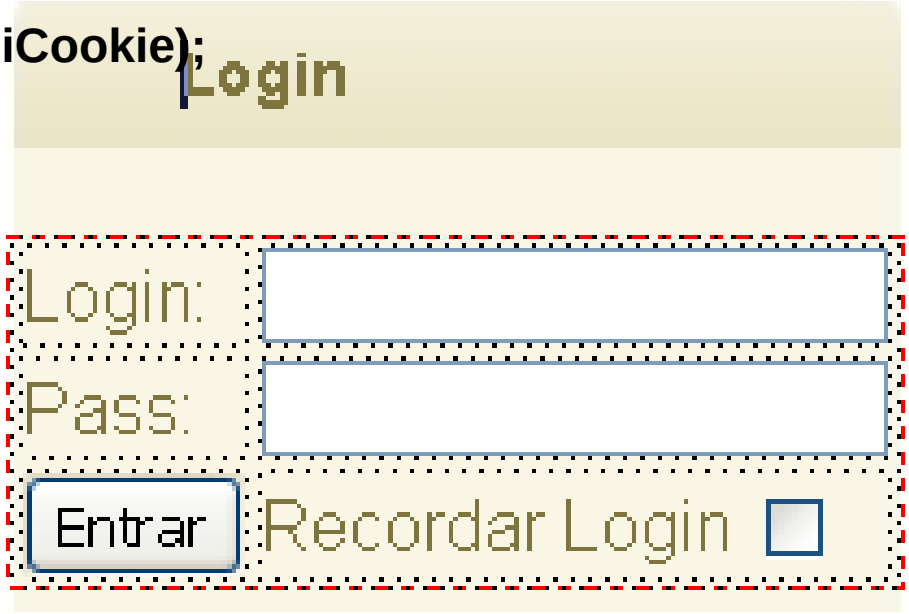
response.addCookie(unCookie);

%>

Una vez ejecutada esta línea es cuando el cookie existe en el disco del cliente que ha accedido a la página JSP.

7.- Manejo de cookies con JSP

```
// Tras la validación correcta y que quiera recordar el login
                if(rec!=null)
                {
                   miCookie=new Cookie("login",login);
                   miCookie.setMaxAge(60*60*24*31);
                   miCookie.setPath("/");
                   response.addCookie(miCookie);
                }
```



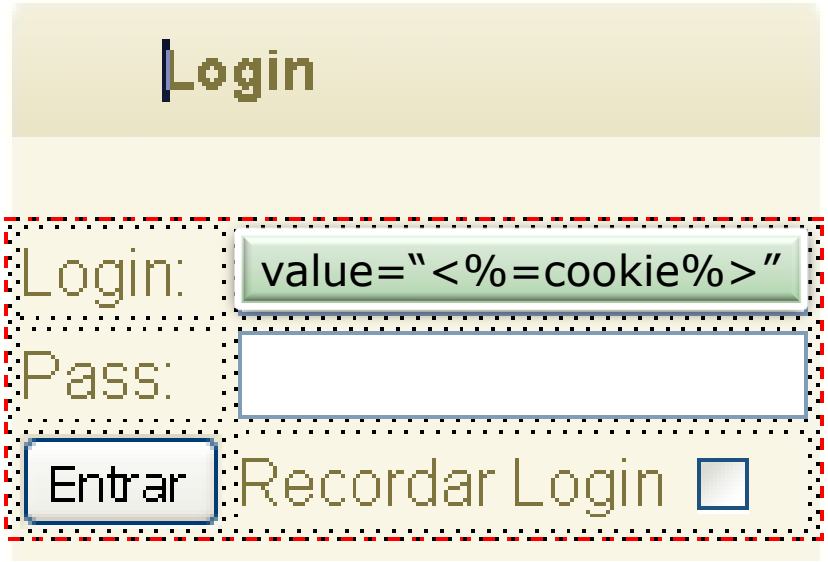
7.- Manejo de cookies con JSP

Recuperar un cookie

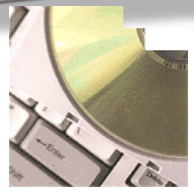
```
<%
Cookie [] todosLosCookies=request.getCookies();
Cookie unCookie=null;
/* El siguiente paso es crear un bucle que vaya leyendo todos los cookies. */
for(int i=0;i<todosLosCookies.length;i++)
{
    Cookie unCookie=todosLosCookies[i];
    if(unCookie.getName().equals("nombre"))
    break;
}
out.println("Nombre: "+unCookie.getName()+"<BR>");
out.println("Valor: "+unCookie.getValue()+"<BR>");
out.println("Path: "+unCookie.getPath()+"<BR>");
out.println("Tiempo de vida:"+unCookie.getMaxAge()+"<BR>");
out.println("Dominio: "+unCookie.getDomain()+"<BR>");
%>
```

7.- Manejo de cookies con JSP

```
String cookie=new String();
 int encontrado=-1;
 String log=new String();
 Cookie [] todosLosCookies=request.getCookies();
 Cookie unCookie=null;
 for(int i=0;(i<todosLosCookies.length) && (encontrado==-1);i++)
    {
       unCookie=todosLosCookies[i];
       if(unCookie.getName().equals("lc
       {
          encontrado=i;
       }
     }
     if(encontrado!=-1)
     {
        cookie=unCookie.getValue();
     }
```



Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada



FORMACION PROFESIONAL