

Programación en Internet

Tema 4. Servlets y JSP

Contenido

1. Introducción
2. Servlets
3. JSP

Referencias

Libro:

Capítulos 30-31. Internet & WWW: How to Program.
Capítulo 9. Apto 10. Programación de Aplicaciones Web
Capítulos 19-20. Core Web Programming.
Teach Yourself JSP 2.0 with Apache Tomcat in 24 Hours

Sites:

<http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html>
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>
<http://www.kinabalo.com/manual.htm>
<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>



Introducción

■ ¿ Qué son los servlets ?

- Tecnología Java que añade funcionalidad a un servidor web.
- Un Servlet es un objeto Java que atiende peticiones de un cliente (p.e. un navegador). Aunque los servlets no están limitados al protocolo HTTP, los servlets más conocidos y utilizados son los HTTP Servlets.
- Tecnología Similar: CGI (Common Gateway Interface): C, C++, PERL, Pythom. (más información de CGI en <http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>)
- Ventajas de los Servlets frente a GCI
 - Eficiencia: Los servlets no tiene que cargase por cada llamada.
 - Potencia: Java dispone de un amplio API que añade gran funcionalidad.
 - Portabilidad: Las clases Java son multiplataforma.
 - Barato: Casi toda la tecnología Java es 'free'.

Introducción – Ejemplo Servlet

```
// servletHolaMundo.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class servletHolaMundo extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hola Mundo!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hola Mundo!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Servlets:

Código Java que genera HTML

Introducción

■ ¿ Qué son los JSP ?

- JSP es una tecnología de *scripting* del lado del servidor que permite incluir código Java embebido con HTML
- Una página JSP (.jsp) es en realidad un servlet, que se encarga de generar y compilar el contenedor de servlet.
- Tecnología Similar a : PHP y ASP.
- Ventajas de JSP
 - Al ser un servlet, todas las ventajas de éstos (eficiencia, potencia, portabilidad y barato)
 - Pero, con respecto a los servlets, los JSP son más cómodos de implementar.

Introducción – Ejemplo JSP

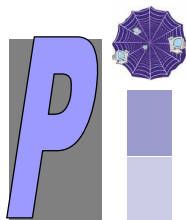
```
<!-- holaPepe.jsp -->

<%
  String persona = new String("Pepe");
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title> Saludando a Pepe! </title>
</head>
<body>
<h1>
  Hola <%=persona%>!
</h1>
</body>
</html>
```

JSP:

Código HTML con sentencias
Java embebidas



Programación en Internet

Tema 4. Servlets y JSP

Contenido

1. Introducción

2. Servlets

- Ciclo de vida de un Servlet
- Servlet para WWW: HTTP Servlets
- Un Primer ejemplo de Servlet
- Configuración e Invocación
- Objeto HttpServletResponse
- Objeto HttpServletRequest
- Resumen

3. JSP



Universidad
de Huelva

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA,
SISTEMAS INFORMÁTICOS Y AUTOMÁTICA

Servlets

- Ciclo de vida: Un servlet genérico pasa por tres etapas

Inicialization: Creación e inicialización de los recursos del servlet.

Todo servlet implementa la interface *javax.servlet.Servlet*, que define el método *init()*

Service: Atiende las peticiones de los clientes y envía las respuestas.

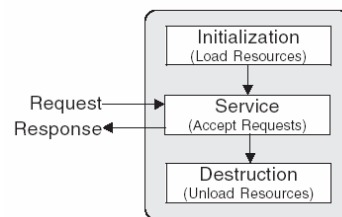
La interface Servlet anterior define un método *service()* con dos parámetros:

Request: Petición del cliente al servidor

Response: Respuesta del servidor para el cliente.

Destruction: Borra el objeto servlet cuando no es necesario.

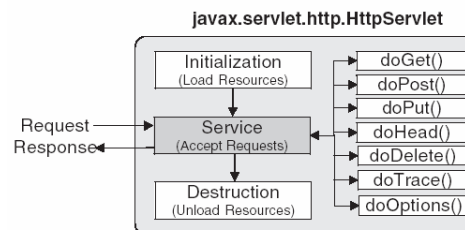
La interface Servlet define un método *destroy()* con ese objetivo.



Servlets para World Wide Web: HTTP Servlets

- Para la utilización de HTTP Servlets se utiliza la clase *HttpServlet*. *HttpServlet* está incluida en *javax.servlet.http.HttpServlet*
- Esta clase implementa la interface Servlets, añadiéndole funcionalidad propia del protocolo HTTP a los métodos *init()*, *service()* y *destroy()*.
- El método *service()* discrimina el tipo de petición del cliente y para ello define 7 métodos abstractos que el desarrollador de servlets debe implementar. Los tipos de petición y el método asociado para atenderla son:

Petición	Método
GET	doGet()
POST	doPost()
PUT	doPut()
HEAD	doHead()
OPTIONS	doOptions()
DELETE	doDelete()
TRACE	doTrace()



- Los dos parámetros de estos métodos son: *HttpServletRequest* (petición del cliente) y *HttpServletResponse* (respuesta para el cliente).

Servlets: Un ejemplo

```
// servletHolaMundo.java

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

public class servletHolaMundo extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hola Mundo!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hola Mundo!</h1>");
        out.println("</body>");
        out.println("</html>");

    }
}
```

Servlets: Configuración e Invocación

■ Configuración

- Estructura de directorios

WEB-INF/		
classes/	-----	(clases servlets desarrollados)
lib/	-----	(Librerías .jar necesarias)
(descriptor de la aplicación web) web.xml	-----	(fichero de configuración)

- Fichero web.xml: Documento XML. Contenido básico:

```
<web-app> ... </web-app>: Etiqueta Principal
<display-name> ... </display-name> : Nombre que mostrara el contenedor
<description> ... </description> : Descripción de la aplicación
<servlet> ... </servlet> : Existe una para cada servlet de la aplicación, con:
    <servlet-name> ... </servlet-name> : Nombre del servlet
    <servlet-class> ... </servlet-class> : Clase que ejecutará el servlet
<servlet-mapping> ... </servlet-mapping> : Una para cada servlet con:
    <servlet-name> ... </servlet-name> : Nombre del servlet
    <url-pattern> ... </url-pattern> : Ruta URL para invocar el servlet
```

Servlets: Configuración e Invocación

■ Para el ejemplo anterior:

□ servletHolaMundo.java → compilamos → servletHolaMundo.class

□ /home/pr_xx/web_app/WEB-INF/

classes/ servletHolaMundo.class

lib/ (no es necesaria ninguna librería)

web.xml →

```
<web-app version="2.4">
  <display-name>Servlet HolaMundo</display-name>
  <description>
    Presenta un saludo al mundo
  </description>
  <servlet>
    <servlet-name> holaMundo </servlet-name>
    <servlet-class> servletHolaMundo </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name> holaMundo </servlet-name>
    <url-pattern> /servlets/saludo </url-pattern>
  </servlet-mapping>
</web-app>
```

Nota:

Es posible crear un fichero WAR con la aplicación comprimida.

```
$ jar cvf mifichero.war dir_WEBINF
```

Servlets: Configuración e Invocación

■ Invocación

http://nombre_servidor_ruta/servlets/saludo

■ En nuestro ejemplo: http://progInternet/pr_xx/servlets/saludo

Es necesario haber desplagado (deploy) la aplicación al contenedor de servlets.

Suponiendo la configuración de prácticas:

http://progInternet:8080/manager/deploy?path=/%7Epr_xx& (continúa en la siguiente línea)

war=file:/home/pr_xx/web-app&update=true

```
...
out.println("<html>");
out.println("<head>");
out.println("<title>Hola Mundo!</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Hola Mundo!</h1>");
out.println("</body>");
out.println("</html>");
...
```



Servlets: Objeto HttpServletResponse

- Este objeto permite enviar la respuesta al cliente.

Algunos métodos son:

getWriter(): devuelve un objeto tipo `PrintWriter` para el envío de texto plano.

Un objeto `PrintWriter` dispone de los métodos: `print()`, `println()` o `write()`.

getOutputStream(): devuelve un objeto tipo `ServletOutputStream` que permite enviar un flujo de bytes.

sendRedirect(*url*): La respuesta es la url indicada.

setContentType(*tipo*): Establece el tipo de respuesta. El más común `text/html`

addHeader(*nomb*, *valor*)

containsHeader(*nomb*)

setHeader(*nomb*, *valor*)

setIntHeader(*nomb*, *valorInt*)

setDateHeader(*nomb*, *fecha*)

addIntHeader(*nomb*, *valorInt*)

addDateHeader(*nomb*, *fecha*)

} Envío de cabeceras HTTP al cliente

addCookie(*cookie*): Envía una cookie al cliente, Donde *cookie* es un objeto `javax.servlet.http.Cookie` que debe ser creado,

Servlets: Objeto HttpServletRequest

- En este objeto se recibe la petición del cliente.

Algunos métodos:

getParameter(*NombParam*): Devuelve el valor del parámetro especificado o null si el parámetro no existe.

getParameterNames(*NombParam*): Devuelve un array con los valores del parámetro indicado. Este método es útil para elementos que tienen más de una valor (p.e. checkbox o combobox de un formulario).

getParameterNames(): Devuelve un objeto `Enumeration` (`java.util.Enumeration`) de todos los parámetros enviados en la petición.

getInputStream(): Devuelve un objeto `ServletInputStream` que permite recibir otro tipo de información (como ficheros) desde el cliente.

getHeader(*nomb*)

getHeaders(*nomb*)

getHeaderNames()

getIntHeader(*nomb*)

getDateHeader(*nomb*)

} Devuelve la información de las cabeceras HTTP del cliente

getCookies(): Devuelve un array de objetos `Cookie` enviados desde el cliente

getSession(*crear*): Devuelve un objeto `HttpSession`, creando una sesión.

Servlets: Ejemplo Parámetros Petición GET

- Petición get `http://[host][:port]/[ruta]/nomServlet?var1=val1`

clase: `ej1Parametros` ([contenido ej1Parametros.java](#))

servlet: `/parametros1` ([contenido web.xml](#))

petición:

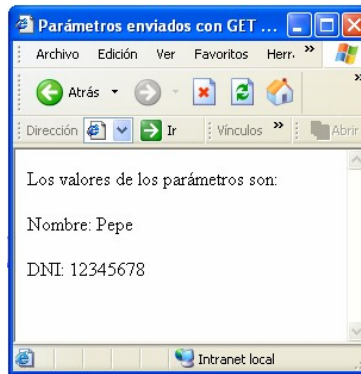
`http://progInternet/~pr_xx/servlets/parametros1?nombre=Pepe&dni=12345678`

```
String nombre = request.getParameter("nombre");
String dni = request.getParameter("dni");

response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>"); out.println("<head>");
out.println("<title>Parámetros enviados con GET </title>");
out.println("</head>"); out.println("<body>");
out.println("<p>Los valores de los parámetros son:</p>");

out.print("<p>Nombre: ");
if (nombre == null) out.println("<i>No definido</i>");
else out.println(nombre);

out.print("<p>DNI: ");
if (dni == null) out.println("<i>No definido</i>");
else out.println(dni); .../...
```



Servlets: Ejemplo Parámetros Petición POST

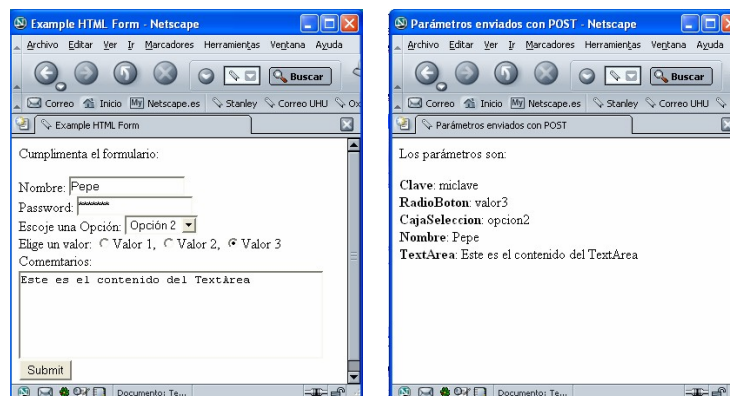
- Tratamiento de los datos enviados desde un formulario, petición POST

Formulario: `formServlet.html` ([contenido formServlet.html](#))

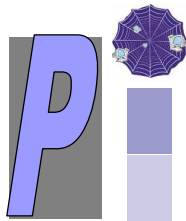
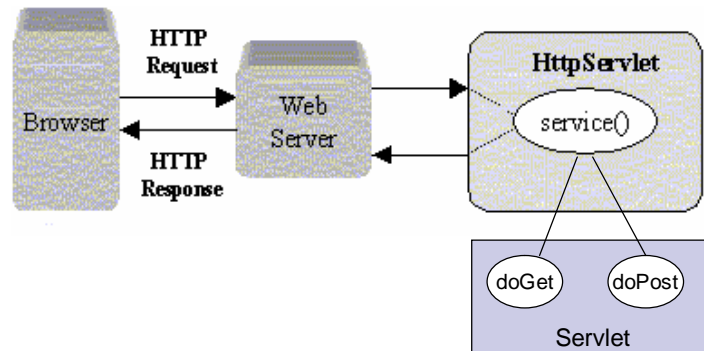
clase: `ej2Parametros` ([contenido ej2Parametros.java](#))

servlet: `/parametros2` ([contenido web.xml](#))

petición: `http://progInternet/~pr_xx/formServlet.html`



Servlets. Resumen



Programación en Internet

Tema 4. Servlets y JSP

Contenido

1. Introducción

2. Servlets

3. JSP

- Aspectos básicos
- Elementos de Scripting
- EL – Expression Language
- Directivas
- Acciones
- Objetos
- Configuración
- Traza de usuarios: Sesiones
- Acceso a BD
- JavaBeans
- Librerías de etiquetas: JSLT



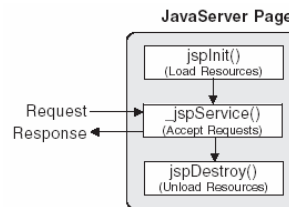
Universidad
de Huelva

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA,
SISTEMAS INFORMÁTICOS Y AUTOMÁTICA

JSP: Aspectos Básicos

- Un JSP es un documento xHTML con código Java embebido
- Formalmente, una página JSP es una implementación de la clase **javax.servlet.Servlet**, que describe como crear un objeto (response) respuesta **HttpServletResponse** a partir de un objeto (request) petición **HttpServletRequest**.
- Un JSP es transformado a un Servlet por el contenedor de Servlet (Tomcat)
- Ciclo de vida.

Inicialización: `jspInit()`
Servicio: `jspService()`
Destrucción: `jspDestroy()`



JSP: Elementos de Scripting

- Declaraciones: variables y objetos del servlet, son externas a `jspService()`.
`<%! declaration; [declaration;]+ ... %>`
`<jsp:declaration>`
code fragment [declaration;]+ ...
`</jsp:declaration>`
Ejemplos
`<%! int i = 0; %>`
`<%! int a, b, c; %>`
`<%! Circle a = new Circle(2.0); %>`
- Expresiones: expresión que será evaluada y el resultado insertado en la salida.
`<%= expression %>`
`<jsp:expression>`
expression
`</jsp:expression>`
Ejemplos
`<%= 4+3+valor %>`
`<%= balance.haber() %>`
`<a href="fichero.<%= ext %>" > ...`
- Scriptlets. Fragmento de código java que se incluirá en `jspService()`.
`<% code fragment %>`
`<jsp:scriptlet>`
code fragment
`</jsp:scriptlet>`
Ejemplo
`<% String name = null;`
`if (request.getParameter("name") == null) { %>`
`<p>Error: Falta parámetro</p>`
`<%} else {`
`obj.procesa(request.getParameter("name"))`
`} %>`

JSP: Lenguaje de Expresiones (EL – Expression Language)

- Expresión EL (Expression Language). Lenguaje de expresiones (JSP 2.0)

<code>\${Expression}</code>	Ejemplo
	<code>\${10*var+2}</code>
	<code>\${12 mod 2}</code>
	<code>\${a >= b}</code>
	<code>\${!empty param.nombClte}</code>
	<code>\${sessionScope.numArticulos}</code>

- Comentarios. Aclaraciones en el código

```
<!-- comentarios -->  
<%-- comentarios --%>
```

JSP: Directivas (I)

- Directivas: Mensajes al contenedor de Servlets

- Include: Inserta un fichero de texto o código Java en la salida.

```
<%@ include file="relativeURL" %>  
o  
<jsp:directive.include file="relativeURL" />
```

Ejemplo

```
<!-- fichero include.jsp -->  
<html>  
<head><title>Ej Include</title></head>  
<body>  
Hoy es: <%@ include file="date.jsp" %>  
</body>  
</html>
```

```
<!-- Fichero date.jsp -->  
<%@ page import="java.util.*" %>  
<%= (new java.util.Date() ).toLocaleString() %>
```

JSP: Directivas (II)

■ Directivas: Mensajes al contenedor de Servlets

- page: Define los atributos de la página JSP.

```
<%@ page atrib="valor" %>      o      <jsp:directive.page listAtributosPagina />
```

Atributos

```
language="java"
```

```
extends="package.class"
```

```
import="{package.class | package.*}, ..."
```

```
session="true|false"
```

```
buffer="none|8kb|sizekb"
```

```
autoFlush="true|false"
```

```
isThreadSafe="true|false"
```

```
info="text"
```

```
errorPage="relativeURL"
```

```
contentType="mimeType [; charset=characterSet ]" | "text/html ; charset=ISO-8859-1"
```

```
isErrorPage="true|false"
```

```
pageEncoding="characterSet | ISO-8859-1"
```

```
isELIgnored="true|false"
```

JSP: Directivas (III)

■ Directivas: Mensajes al contenedor de Servlets

- taglib: Define una librería de etiquetas y un prefijo para usarlo en la página JSP

```
<%@ taglib uri="http:..." prefix="..." %>
```

```
Ejemplo: <%@ taglib uri="http://java.sun.com/jstl/core_rc" prefix="c" %>
```

```
...
```

```
<c:set var="saludo" value="Hola Mundo!" />
```

- tag: define las propiedades de una etiqueta de usuario

```
<%@ tag ... %>
```

- attribute: Define atributos para las etiquetas.

```
<%@ attribute name="attribute-name" ... %>
```

- variable: Define variables estableciendo su ámbito para otros JSP

```
<%@ variable name-given="..." scope="..." ... %>
```

JSP: Principales Acciones (actions)

■ Acciones: Asocian código dinámico con una simple etiqueta.

- `<jsp:include>` : Incluye un Servlet o JSP en otro.

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

- `<jsp:forward>` : Redirige una petición a otro HTML, JSP o servlet.

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

- `<jsp:useBean>` : Instancia o referencia un bean asignando nombre y ámbito.

```
<jsp:useBean id="miBean" class="ejemplos.ejtBean" />
```

- `<jsp:getProperty>` : Obtiene el valor de una propiedad de un bean instanciado

```
<jsp:getProperty name="miBean" property="nombreCliente"/>
```

- `<jsp:setProperty>` : Asigna el valor de una propiedad de un bean

```
<jsp:setProperty name="miBean" property="nombreCliente" value="Pepe" />
```

JSP: Objetos

■ Objetos creados por el contenedor de Servlets

- **request** : Instancia de `javax.servlet.ServletRequest`. Encapsula la petición del cliente. Es pasado al JSP por el contenedor de servlets, como un parámetro de `_jspService()`.
- **response**: Instancia de `javax.servlet.ServletResponse`. Encapsula la respuesta generada por el JSP para enviar al cliente. Es pasado al JSP por el contenedor de servlets, como parámetro de `_jspService()`.
- **out** : Instancia de `javax.servlet.jsp.JspWriter`, es un objeto `PrintWriter` usado para devolver la respuesta al cliente.
- **session** : Instancia de `javax.servlet.http.HttpSession`. Representa la sesión creada para las peticiones de un cliente. Las sesiones se crean automáticamente.
- **application** : Instancia de `javax.servlet.ServletContext`. Representa el contexto dentro del cual el JSP se está ejecutando.
- **pageContext** : Instancia de `javax.servlet.jsp.PageContext`. Encapsula el contexto de la página para un JSP específico.
- **config** : Objeto que permite recuperar la información (atributos, etc) del fichero de configuración `web.xml`
- **page** : Instancia de `java.lang.Object`. Representa a la instancia de la clase del JSP; es decir, el propio JSP.
- **exception** : Instancia de `java.lang.Throwable`. Representa un objeto excepción que provocará la invocación de una página de error. Este objeto está disponible sólo para la página de error (`isErrorPage=true`).

JSP: Objetos

■ Ejemplo: Recepción de parámetros desde un formulario

El formulario: **formJSP.html**

```
...
<body>
  ...
  <form method="get" action="atenderFormulario.jsp">
    <label for="campo1">Campo 1:</label>
    <input type="text" size="10" name="campo1"><br />
    <label for="campo2">Campo 2:</label>
    <input type="text" size="10" name="campo2"><br />
    <input type="submit" value="Enviar">
  </form>
  ...
</body>
...
```

JSP: Objetos

■ Ejemplo: Recepción de parámetros desde un formulario

El JSP: **atenderFormulario.jsp**

```
<%
String c1=request.getParameter("campo1");
String c2=request.getParameter("campo2");
%>
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title>Atender formJSP</title> </head>
<body>
<% out.println("<h1>Resultados del formulario</h1>"); %>
<p>Los valores introducidos en el formulario son:</p>
<% if (c1.equals(c2)) {%>
  <p>Has introducido el mismo valor para Campo 1 y Campo 2: <%=c1%><p>
<%} else {%>
  <p>Campo 1: <%=c1%></p>
  <p>Campo 2: <%=c2%></p>
<%}%>
</body>
</html>
```

JSP: Configuración

- Los JSP de una aplicación pueden ser configurados, como los servlets, mediante el descriptor de aplicación web.xml

A diferencia de los servlets no es obligatorio su uso, aunque si aconsejable.

Para ello se utiliza el elemento `<jsp-config>`, el cual puede aparecer una o varias veces en el fichero web.xml

Los subelementos de `<jsp-config>` son:

- El elemento `<taglib>` : Para configurar librería de etiquetas
- El elemento `<jsp-property-group>` : Para establecer propiedades de un grupo de JSP que cumplan un patrón

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

JSP: Configuración

- Subelementos de `jsp-property-group`

el-enabled : Establece si el lenguaje de Expresión (EL) está disponible para usarlo en los JSP. Funcionalidad análoga al atributo `isELEnabled` de la Directiva `page`.

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

scripting-enabled : Análogo a `isScriptingEnabled` de la Directiva `page`, establece la posibilidad de usar scripting en el JSP.

page-encoding : Como `pageEncoding` de `page`, determina el juego de caracteres de los JSP, y provoca un error en el caso de que otro juego de caracteres sea utilizado desde el atributo `page-encoding`.

include-prelude : Permite especificar un contenido que será presentado antes del contenido de los JSP.

include-coda : Permite especificar un contenido que se mostrará después del contenido generado por los JSP.

is-xml : Establece si los JSP son documentos XML.

JSP: Configuración

■ Ejemplo: Añadir cabecera y pie en todas las páginas JSP

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <include-prelude>/cabecera.jsp</include-prelude>
    <include-coda>/pie.jsp</include-coda>
  </jsp-property-group>
</jsp-config>
```

- El contenido del fichero *cabecera.jsp* será incluido antes del contenido que genera cada JSP.
- El contenido del fichero *pie.jsp* será incluido después del contenido que genera cada JSP.

JSP: Traza de usuarios: Sesiones

■ Posibilidades

- Reescritura de URL
La información se pasa en la propia URL que se va reescribiendo en cada petición.
- Cookies.
Pares *nombre=valor* que se almacenan en el cliente y se envían al servidor en cada petición.
- Sesiones.
Periodo de tiempo establecido en el que un conjunto de objetos *nombre=valor* son mantenidos entre cliente y servidor.

■ Sesiones: Interface HttpSession

- Los JSP disponen de un objeto **session** de tipo **HttpSession**.
- Este objeto permite manipular una sesión entre cliente y servidor.
- Añadir objetos que permiten que la información de usuario perdure entre varias peticiones.

JSP: Traza de usuarios: Sesiones

■ Principales métodos del objeto HttpSession

Métodos	
java.lang.String	getId() : Devuelve ID de la sesión
boolean	isNew() : Devuelve true si la session es nueva.
void	invalidate() : Invalida la session.
void	setAttribute(java.lang.String name, java.lang.Object value) : Crea un objeto para esta session.
java.lang.Object	getAttribute(java.lang.String name) : Devuelve el valor del objeto pasado como atributo.
java.util.Enumeration	getAttributeNames() : Devuelve un objeto Enumeration con los nombres de los objetos de la session.
void	removeAttribute(java.lang.String name) : Elimina un objeto de la sesión.
int	getMaxInactiveInterval() : devuelve el tiempo (seg) que durará la session.
void	setMaxInactiveInterval(int interval) : Asigna el periodo (seg) que sera valida la session.

JSP: Traza de usuarios: Sesiones

■ Ejemplo: menu.jsp

```
<% synchronized(session) {
    if (session.isNew()) {
        session.setAttribute("numArticulos",new Integer(0));
    }
}%>
<html> <head> <title>Menu MiniTiendaVirtual: Informatica TV</title>
<script>
function seguro() {
    return confirm("Hay seleccionado <%=session.getAttribute("numArticulos")%> producto(s).
    Seguro que deseas eliminar la selección?");
}
</script> </head><body>

<p style="text-align: center; font-size: 16pt; color:red"> Menu </p>

<p style="text-align: center; font-size: 14pt">
<a target="PPAL" href="ppal.html">Inicio</a> -
<a target="PPAL" href="pag1.html">Pág1</a> -
<a target="PPAL" href="pag2.html">Pág2</a> -
<a target="PPAL" href="pag3.html">Pág3</a> -
<a target="PPAL" href="articulos.jsp">Ver selección</a> -
<a target="PPAL" href="borrar.jsp" onClick="return seguro()">Borrar selección</a> </p>
</body> </html>
```

JSP: Traza de usuarios: Sesiones

■ Ejemplo: sumarCarrito.jsp

```
<% String codigo=request.getParameter("cod");
    int n = ((Integer) session.getAttribute("numArticulos")).intValue();
    n++;
    session.setAttribute("Articulo"+n, codigo);
    session.setAttribute("numArticulos", new Integer(n));
%>
<html>

<head> <title>Sumar al carrito</title>
<script>
function volver() {
    top.MENU.location.reload();
    setTimeout("history.back()",3000);
}
</script> </head>
<body onLoad="volver()">
<h1>Ha añadido al carrito el producto: </h1>
<p>  </p>

<p>En 3 seg. volverá a la página anterior</p>
<p><a href="javascript:history.back()">Si no es así haga clic aquí</a></p>

</body>
</html>
```

JSP: Traza de usuarios: Sesiones

■ Ejemplo: carrito.jsp

```
<%
String codigo=new String();
int n = ((Integer) session.getAttribute("numArticulos")).intValue();
%>
<html>
<head><title>Sumar al carrito</title></head>

<body>
<p>Sesión: <%=session.getId()%> </p>

<h1>Ha seleccionado <%=session.getAttribute("numArticulos")%> producto(s): </h1>
<hr />

<% for (int i=1;j<=n; i++) {
    codigo = (String) session.getAttribute("Articulo"+i);
%>
<p><%=codigo%>

</p>
<hr />
<% } %>

</body>
</html>
```

JSP: Acceso a BD (JDBC)

■ JSP – JDBC – PostgreSQL

- Pasos: Es necesario importar el paquete `java.sql.*`

1) Cargar el Driver. Posteriormente será manipulado con el *DriverManager*

```
Class.forName("org.postgresql.Driver");
```

(Es necesario disponer del Driver, puedes descargarlo en <http://jdbc.postgresql.org/>)

2) Establecer la conexión. Crear un objeto *Connection* desde el *DriverManager*.

```
Connection bd;
```

```
bd = DriverManager.getConnection("jdbc:postgresql:BaseDatos","Usu", "Pass");
```

Donde: *BaseDatos* es la base de datos PostgreSQL

Usu es el usuario y *Pass* la password

Este método lanza una **SQLException** si se produce un error en la conexión

JSP: Acceso a BD (JDBC)

■ JSP – JDBC – PostgreSQL

3) Crear una sentencia JDBC con el objeto *Statement* desde el objeto *Connection*.

```
Statement st;
```

```
st = db.createStatement();
```

El método lanza una **SQLException** si se produce un error en la base de datos.

4) Preparar la sentencia SQL

- Mediante un String:

```
String sql = "SELECT * FROM PERSONAS";
```

```
String sql = "SELECT * FROM "+tabla+" WHERE nomb='"+vnomb+"'";
```

o - Mediante un objeto *PreparedStatement* del objeto *Connection*.

```
PreparedStatement ps;
```

```
ps = bd.prepareStatement("SELECT * FROM ? WHERE n= ? AND f< ? ");
```

```
ps.setString(1, tabla);
```

```
ps.setInt(2, 10);
```

```
ps.setFloat(3, 3.14);
```

El método lanza una **SQLException** si se produce un error en la base de datos.

JSP: Acceso a BD (JDBC)

■ JSP – JDBC – PostgreSQL

5) Ejecutar la sentencia

- A partir del objeto *Statement*, si la SQL se ha definido con un String
- o - A partir del objeto *PreparedStatement*

Dependiendo de la sentencia a ejecutar se utilizan, básicamente, los métodos:

`execute()` : Para cualquier sentencia SQL.

Este método lanza una **SQLException** si se produce un error

`executeQuery()` : Para una consulta SELECT. Devuelve un objeto *ResultSet*.

Lanza **SQLException** si se produce un error o no se obtiene un objeto *ResultSet*.

`executeUpdate()`: Para una consulta que no devuelva resultado de la Base Datos

Devuelve un entero que indica las filas insertadas, actualizadas o borradas, ó 0.

Lanza **SQLException** si se produce un error o se obtiene un objeto *ResultSet*.

```
Ej1.   ResultSet rs = st.executeQuery("SELECT * FORM Articulos");
Ej2.   st.execute(sql); // donde sql es un String con una sentencia SQL
Ej3.   ResultSet rs = ps.executeQuery();
Ej4.   ps.executeUpdate();
```

JSP: Acceso a BD (JDBC)

■ JSP – JDBC – PostgreSQL

6) Acceder a la información obtenida (*ResultSet*)

Un objeto *ResultSet* mantiene un cursor apuntando a los datos obtenidos.

Ej: `ResultSet rs = st.executeQuery("SELECT * FORM Articulos");`

Principales métodos:

`getString(pos)` : Devuelve el String almacenado en la columna pos.

También, se puede usar el nombre de la columna.

`getInt(pos)` : Devuelve el Integer almacenado en la columna pos.

`next()` : Cambia a la siguiente fila, devuelve verdadero si existe o falso en caso contrario.

```
Ej.
while ( rs.next() ) {
    c1 = rs.getString(1);
    c2 = rs.getInt(2);
    c3 = rs.getFloat(3);
}
```

Nota 1: Es muy buena práctica liberar los recursos con `close()`. [Connection/ Statement / *ResultSet*]

Nota 2: Para eliminar los espacios en blanco de una cadena se puede usar `trim()`;

JSP: Acceso a BD (JDBC)

■ Ejemplo

```
<%@ page import="java.sql.*" %>
<%
Connection db=null; Statement st=null; ResultSet rs=null;
try {
    Class.forName("org.postgresql.Driver");
    db = DriverManager.getConnection("jdbc:postgresql:prueba","web", "web");
    st = db.createStatement();
    String sql ="SELECT * FROM personas";
    rs = st.executeQuery(sql);
    db.close(); st.close();
}catch(SQLException e) {}
%>
<html> <head> <title>Conexión a Base de datos</title> </head> <body>
<h1>Base de datos Prueba</h1>
<% while (rs.next()) {%>
<p>
Nombre: <%=rs.getString(1).trim%><br />
Edad: <%=rs.getString(2).trim%><br />
<a href="http://www.emp.es/~<%=rs.getString(1).trim%>">WebSite</a><br />
E-mail:
<a href="mailto:<%=rs.getString(1).trim%>@emp.es"><%=rs.getString(1).trim%>@emp.es</a>
</p>
<%} %> <% rs.close(); %>
</body> </html>
```

JSP: JavaBeans

- JavaBeans es la tecnología de componentes de la plataforma Java. Los componentes (Beans) son clases java reutilizables y que pueden ser compartidos fácilmente por varias aplicaciones Java.
- En general, un bean es una clase que obedece a ciertas reglas:
 - Un bean debe tener un constructor por defecto (sin argumentos).
 - Un bean tiene que tener persistencia, es decir, implementar la interface Serializable.
 - Un bean tiene que tener introspección (instrospection). Básicamente con métodos set... y get... para cada uno de sus atributos.

```
public void setNombrePropiedad(TipoPropiedad valor);
public TipoPropiedad getNombrePropiedad( );
```

JSP: Beans

- Crear un Bean

```
// ejBean.java – Ejemplo de un componente Bean

package ejBean;

public class miBean implements java.io.Serializable {
    String unAtributo="Hola Mundo!";

    public miBeans() {
    }

    public String getUnAtributo(){
        return unAtributo;
    }

    public void setUnAtributo(String valor){
        unAtributo = valor;
    }
}
```

JSP: Beans

- Crear un JSP que use el Bean

Usar las acciones: <jsp:usebean ... >, <jsp:getProperty ... > y <jsp:setProperty ...>

```
<html>
<head> <title>Ejemplo de beans</title> </head>

<body>
<jsp:useBean id="mibean" class="ej.miBeans" />

<p>
Saludo: <jsp:getProperty name="mibean" property="unAtributo" />
</p>

<jsp:setProperty name="mibean" property="unAtributo" value="Adios" />

<p> Y como te marchas
pues <jsp:getProperty name="mibean" property="unAtributo" /> hasta pronto
</p>
</body>
</html>
```

JSP: Librerías de etiquetas estándar (JSTL)

■ Etiquetas de Usuario

JSP permite a los usuarios crear nuevas etiquetas (custom tags) que agrupen acciones utilizadas en varios JSP.

Esto requiere:

- 1) Crear, en Java, la funcionalidad de las etiquetas (implementando la interface Tag)
- 2) Crear el descriptor de la librería de etiquetas tld (Tag Library Descriptor)

■ JSTL (JSP Standard Tag Library). [<http://java.sun.com/products/jsp/jstl/>]

La librería estándar es una colección de etiquetas que encapsulan funcionalidad de uso muy frecuente en los JSP.

Beneficios:

- Reducir el tiempo de desarrollo de una página JSP.
- Separar la programación Java (lógica del negocio) del diseño web (presentación).

JSP: Librería de etiquetas estandar (JSTL)

■ Funcionalidad encapsulada en JSTL

Áreas:

core	:	Entrada y Salida, condicionales y bucles
xml	:	Procesamiento de documentos XML
sql	:	Acceso y utilización de Bases de Datos
fmt	:	Capacidades de formateo de Internacionalización
fn	:	Funciones

■ Configuración:

Incluir los ficheros standard.jar y jstl.jar en el directorio WEB-INF/lib.

[<http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html>]

■ Utilización

```
<@ taglib prefix="xxx" uri="url_al_tld" />
...
<XXX:accion ....> ....
```

JSP: Librería de etiquetas estandar (JSTL)

■ Utilización

```
core:      <@ taglib prefix="c" uri=http://java.sun.com/jstl/core_rt" />
xml:      <@ taglib prefix="x" uri=http://java.sun.com/jstl/xml_rt" />
fmt:      <@ taglib prefix="fmt" uri=http://java.sun.com/jstl/fmt_rt" />
sql:      <@ taglib prefix="sql" uri=http://java.sun.com/jstl/sql_rt" />
fn:       <@ taglib prefix="fn" uri=http://java.sun.com/jstl/functions_rt" />
```

Ejemplo

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<html>
<head> <title>Ejemplo con JSTL</title> </head>

<body>
  <c:forEach var="i" begin="1" end="10" step="1">
    <c:out value="{i}" /> <br />
  </c:forEach>
</body>
</html>
```

JSP: Librería de etiquetas estandar (JSTL)

■ Principales etiquetas de core.

Entrada/Salida

```
<c:set var="nombre" value="valor" />
<c:out value="valor" />
```

Condicionales

```
<c:if test="exp_test"> ... </c:if>

<c:choose>
  <c:when test="exp_test1"> ... </c:when>
  <c:when test="exp_test2"> ... </c:when>
  ...
  <c:otherwise> ... </c:otherwise>
</c:choose>
```

Iterativas

```
<c:forEach var="nombre" begin="ini" end="fin" step="paso">
  ...
</c:forEach>
```

JSP: Librería de etiquetas estandar (JSTL)

■ Ejemplo core

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
```

<p>Asignado el valor del parámetro número a la variable num.

```
<c:set var="num" value="${param.numero}"/> </p>
```

<p>Asignando la cadena "Hola, Mundo!" a la variable hola.

```
<c:set var="hola" value="Hola, Mundo!"/> </p>
```

<p>Mostrando una EL expresión: <c:out value="\${1+5}" /></p>

<p>Mostrando variable hola: <c:out value="\${hola}" /> </p>

```
<c:if test="${num>1 }">
```

<p>num mayor que 10.

```
</c:if>
```

```
<c:if test="${num<1 }">
```

<p>num menor que 10.

```
</c:if>
```

```
<p>
```

```
<c:forEach var="i" begin="1" end="10" step="1">
```

```
<c:out value="${i}" /> <br />
```

```
</c:forEach>
```

```
</p>
```

JSP: Librería de etiquetas estandar (JSTL)

■ Principales etiquetas de sql.

Origen de datos (dataSource)

```
<sql:setDataSource scope="ambito" var="nombre"
  url=" ..." driver=" ..."
  user=" ..." password=" ..." />
```

Consulta

```
<sql:query var="nombre" dataSource="varDataSource" scope=" ..." >
  Sentencia SQL, p.e. SELECT * FROM articulos
</sql:query>
```

Otras SQL

```
<sql:update var="nombre" dataSource="varDataSource" scope=" ..." >
  Sentencia SQL, p.e. INSERT INTO prueba VALUES ('Paul', 20)
</sql:update>
```

JSP: Librería de etiquetas estandar (JSTL)

■ Ejemplo SQL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql_rt" %>

<sql:setDataSource scope="session" var="dataSource"
    url="jdbc:postgresql://localhost/tvirtual" driver="org.postgresql.Driver"
    user="Jose Luis" password="jlam"/>

<sql:update var="numero" dataSource="${dataSource}" scope="session">
    INSERT INTO articulos VALUES ('P-XXXXXX','nuevo', 20, 0)
</sql:update>

<p>Insertados:${numero}</p>
```

... continúa ...

JSP: Librería de etiquetas estandar (JSTL)

■ Ejemplo SQL

...

```
<sql:query var="articulos" dataSource="${dataSource}" scope="session">
    SELECT * FROM articulos
</sql:query>

<table>
<c:forEach var="row" items="${articulos.rows}">
<tr>
<td><c:out value="${row.codigo}"/></td>
<td><c:out value="${row.descripcion}"/></td>
<td><c:out value="${row.precio}"/></td>
</tr>
</c:forEach>
</table>
```

... continúa ...

...

JSP: Librería de etiquetas estandar (JSTL)

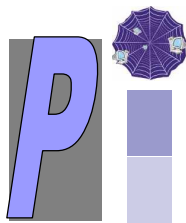
■ Ejemplo SQL

...

```
<sql:update var="numero" dataSource="${dataSource}" scope="session">
  DELETE FROM articulos WHERE codigo='P-XXXXXX'
</sql:update>
<p>Borrados:${numero}</p>
```

```
<sql:query var="articulos" dataSource="${dataSource}" scope="session">
  SELECT * FROM articulos
</sql:query>
```

```
<table>
<c:forEach var="row" items="${articulos.rows}">
<tr>
<td><c:out value="${row.codigo}"/></td>
<td><c:out value="${row.descripcion}"/></td>
<td><c:out value="${row.precio}"/></td>
</tr>
</c:forEach>
</table>
```



Programación en Internet

Tema 4. Servlets y JSP

Contenido

1. Introducción
2. Servlets
3. JSP

Referencias

Libro:

Capítulos 30-31. Internet & WWW: How to Program.
Capítulo 9. Apto 10. Programación de Aplicaciones Web
Capítulos 19-20. Core Web Programming.
Teach Yourself JSP 2.0 with Apache Tomcat in 24 Hours

Sites:

<http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html>
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>
<http://www.kinabalo.com/manual.htm>
<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>



Universidad
de Huelva

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA,
SISTEMAS INFORMÁTICOS Y AUTOMÁTICA