# JAVA WEB: JSP

Charly Cimino

# Java Web: *JSP*

## Charly Cimino



Versión 2023.09.16

# Aplicaciones web con Java

Los ejemplos y explicaciones de esta presentación fueron desarrollados y explicados a partir de las siguientes herramientas y sus respectivas versiones.

**JDK 17**

Por ser una versión **LTS**
(*Long Time Support*)

**NetBeans 17**

Instrucciones de instalación
https://youtu.be/2Et13pH2484

**Apache Tomcat**

Versión 10

Descargar de aquí

**En el siguiente repositorio encontrarás varios de los ejemplos de esta PPT (y más), para poder probarlos en tu máquina.**

# ¿Qué es JSP?

La tecnología **JSP** (Desde 2018: *Jakarta Server Pages* - Antes: *Java Server Pages*) nos permite crear contenido web dinámico con una sintaxis más cercana a XML, logrando que nuestra aplicación web tenga bien separadas la lógica de presentación de la lógica de negocio.

`http://localhost:8080/saludar?nombreCli=Juan`

**SaludarServlet.java**

```java
// Se omiten imports y package

@WebServlet(name = "saludar", urlPatterns = {"/saludar"})
public class ProcesarServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        String nombre = req.getParameter("nombreCli");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Saludando...</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hola " + nombre + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

**Sin JSP (HTML embebido en Java)**

**SaludarServlet.java**

```java
// Se omiten imports y package

@WebServlet(name = "saludar", urlPatterns = {"/saludar"})
public class ProcesarServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        req.getRequestDispatcher("saludo.jsp").forward(req, resp);
    }
}
```

**saludo.jsp**

```jsp
<html>
    <head>
        <title>Saludando</title>
    </head>
    <body>
        <h1>Hola ${param.nombreCli}</h1>
    </body>
</html>
```

**Con JSP (HTML separado de Java)**

# Composición de un JSP

La manera más fácil de entender a un **JSP** es como un documento **HTML** con incrustaciones dinámicas.

**saludo.jsp**

Es importante que el archivo tenga la extensión .jsp, de lo contrario, la expresión `${param.nombreCli}` se imprimiría literalmente.

```
<html>
    <head>
        <title>Saludando</title>
    </head>
<!-- Comentario HTML: Se ve en el browser del cliente -->
<%-- Comentario JSP: Solo se ve en el server --%>
    <body>
        <h1>Hola ${param.nombreCli}</h1>
    </body>
</html>
```

Se evalúa el valor del parámetro `nombreCli` recibido en la *request* y mapeado por el objeto `param`, usando ***EL*** (*Expression Language*), tras el envío de un formulario por parte del cliente.

Se detallará en próximas diapositivas.

# Ventajas de JSP

| Extiende a los *servlets* | Fácil de mantener | No hace falta recompilar y redesplegar | Menos código |
|---|---|---|---|
| Pueden usarse todas las características de un *servlet* en un **JSP**.<br><br>Los **JSP** son *servlets* con *sugar syntax**. | Es más simple mantener una *webapp* dada la separación entre la lógica de negocio (obtenida desde el modelo por un *servlet*) y la de presentación (en un **JSP**). | A diferencia de un *servlet*, si se modifica un documento **JSP**, no es necesario recompilar y redesplegar nuestro proyecto en el servidor web. | El uso de etiquetas específicas, objetos implícitos y expresiones en un **JSP** economiza las líneas de código necesarias y mejora su legibilidad. |

* De Wikipedia: ***Sugar syntax* (azúcar sintáctico)** es un término acuñado por Peter J. Landin en 1964 para referirse a los añadidos a la sintaxis de un lenguaje de programación diseñados para hacer algunas construcciones más fáciles de leer o expresar.

# Un JSP será un *servlet*

A través de NetBeans, puede hacerse click derecho sobre un documento **.jsp** y elegir la opción **"View Servlet"** para observar la traducción de ese **JSP** a *servlet* que se llevó a cabo cuando se ejecutó la aplicación.

```
32        try {
33          response.setContentType("text/html");
34          response.setHeader("X-Powered-By", "JSP/2.3");
35          pageContext = _jspxFactory.getPageContext(this, request, response,
36                      null, true, 8192, true);
37          _jspx_page_context = pageContext;
38          application = pageContext.getServletContext();
39          config = pageContext.getServletConfig();
40          session = pageContext.getSession();
41          out = pageContext.getOut();
42          _jspx_out = out;
43          _jspx_resourceInjector = (org.glassfish.jsp.api.ResourceInjector) application.getAttribute("com.sun.appserv.jsp.resource.injector");
44
45          out.write("\n");
46          out.write("<html>\n");
47          out.write("    <head>\n");
48          out.write("        <title>Saludando</title>\n");
49          out.write("    </head>\n");
50          out.write("    <body>\n");
51          out.write("        <h1>Hola ");
52          out.write((java.lang.String) org.apache.jasper.runtime.PageContextImpl.evaluateExpression("${param.nombreCli}", java.lang.String.class, (PageContext)_jspx_page_context, null));
53          out.write("</h1>        \n");
54          out.write("    </body>\n");
55          out.write("</html>\n");
56        } catch (Throwable t) {
57          if (!(t instanceof SkipPageException)){
58            out = _jspx_out;
59            if (out != null && out.getBufferSize() != 0)
60              out.clearBuffer();
61            if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
62            else throw new ServletException(t);
63          }
64        } finally {
65          _jspxFactory.releasePageContext(_jspx_page_context);
66        }
```

**Solo se muestra parte del método `_jspService()`, que se invoca cuando se debe procesar una solicitud.**

# Formas de expresión en JSP

A lo largo de la historia, han aparecido nuevas, mejores y más fáciles maneras de expresar contenido dinámico en los JSP. Vamos a ver todas ellas, con especial énfasis en las más modernas.

## *Scriptlets*

- Un *scriptlet* es un fragmento de código Java delimitado entre etiquetas `<% %>`.
- Incluye expresiones, delimitadas entre `<%= %>`, declaraciones, delimitadas entre `<%! %>`, y directivas, delimitadas entre `<%@ %>`
- Soportado desde la primera versión de JSP (1999). A día de hoy se lo considera obsoleto, pues incrusta código Java (lógica de negocio) en el propio JSP (lógica de presentación).
- Se detalla su sintaxis al final de esta PPT (Apéndice: Scriptlets y directivas JSP)

## EL (*Expression Language*)

- Pensado para los diseñadores de contenido web que tienen poco o nulo conocimiento de Java, permite un acceso más fácil y directo a los métodos básicos del lenguaje.
- Su sintaxis es similar a las expresiones colocadas entre *backsticks* `` en JavaScript: `${expresión}`
- Comenzó siendo parte de JSTL (descripto más adelante) pero a partir de la especificación 2.0 de JSP (2003) es parte de esta última. Desde 2019, con la versión 3.0.2, su paquete cambió de nombre, de `javax.el` a `jakarta.el`, como parte de la transición de Java EE (de Oracle) a Jakarta EE (de Eclipse Foundation).

## JSTL (Desde 2018: *Jakarta Standard Tag Library*) Antes: *JavaServer Pages Standard Tag Library*

- Reemplaza a los *scriptlets* y complementa a EL, añadiendo etiquetas personalizadas para ejecuciones condicionales (`<c:if>`), ciclos (`<c:foreach>`), internacionalización (`<fmt:…>`), y muchas otras.
- Se lanzó en mayo de 2005. Desde 2019, con la versión 1.2.3, su paquete cambió de nombre, de `javax.servlet.jsp.jstl` a `jakarta.servlet.jsp.jstl`, como parte de la transición de Java EE (de Oracle) a Jakarta EE (de Eclipse Foundation).

# EL: Expression Language

Simplifica la sintaxis utilizada hasta entonces para manejar *beans* (descriptos más adelante), acceder a las cabeceras y parámetros de la petición, manipular objetos implícitos, etc.

Contiene objetos implícitos, operadores y palabras reservadas.

## Sintaxis

```
${ expresión }
```

**saludo.jsp**

```jsp
<html>
    <head>
        <title>Saludando</title>
    </head>
    <body>
        <h1>Hola <%= request.getParameter("nombreCli") %> </h1>
    </body>
</html>
```

Con *scriptlet* (obsoleto)

**saludo.jsp**

```jsp
<html>
    <head>
        <title>Saludando</title>
    </head>
    <body>
        <h1>Hola ${param.nombreCli}</h1>
    </body>
</html>
```

Con *EL*

Documentación oficial: https://jakarta.ee/specifications/expression-language/5.0/jakarta-expression-language-spec-5.0

# Objetos implícitos en *EL*

Al igual que en los *scriptlets*, en *EL* hay 11 objetos implícitos que son creados por el contenedor web para acceder a ellos fácilmente desde los JSP, utilizando el **punto** (por ejemplo, `param.user`) o los **corchetes** (por ejemplo, `param["user"]`).

| Objeto | Descripción | Ejemplo |
|---|---|---|
| `pageScope` | Mapea los atributos de la página (*page*), para poder acceder a sus valores a través de sus nombres. | (Suponiendo que el *server* ligó un atributo `"user1"` con valor `"Pepe"` a nivel **página**)<br>`${ pageScope.user1 }` Mostraría `"Pepe"` (Poniendo solo `${user1}` también funciona) |
| `requestScope` | Mapea los atributos de la solicitud (*request*), para poder acceder a sus valores a través de sus nombres. | (Suponiendo que el *server* ligó un atributo `"user2"` con valor `"Luis"` a nivel **solicitud**)<br>`${ requestScope.user2 }` Mostraría `"Luis“` (Poniendo solo `${user2}` también funciona)<br>En un *servlet*, esto mismo se escribiría así: `req.getAttribute("user2");` |
| `sessionScope` | Mapea los atributos de la sesión (*session*), para poder acceder a sus valores a través de sus nombres. | (Suponiendo que el *server* ligó un atributo `"user3"` con valor `"Sara"` a nivel **sesión**)<br>`${ sessionScope.user3 }` Mostraría `"Sara"` (Poniendo solo `${user3}` también funciona)<br>En un *servlet*, esto mismo se escribiría así: `req.getSession().getAttribute("user3");` |
| `applicationScope` | Mapea los atributos de la aplicación (*application*), para poder acceder a sus valores a través de sus nombres. | (Suponiendo que el *server* ligó un atributo `"user4"` con valor `"Ana"` a nivel **aplicación**)<br>`${ applicationScope.user4 }` Mostraría `"Ana"` (Poniendo solo `${user4}` también funciona)<br>En un *servlet*, esto mismo se escribiría así: `getServletContext().getAttribute("user4");` |
| `param` | Mapea los parámetros (enviados por el cliente) de la solicitud (*request*), para poder acceder a sus valores a través de sus nombres.<br>(También existe el objeto `paramValues` que mapea los parámetros que tengan más de un valor a un array) | (Suponiendo que hay un parámetro `"nombreCli"` con valor `"Hugo"` en la *request*)<br>`${ param.nombreCli }` (Mostraría `"Hugo"`)<br>En un *servlet*, esto mismo se escribiría así: `req.getParameter("nombreCli");` |
| `header` | Mapea las cabeceras (*headers*) de la solicitud (*request*), para poder acceder a sus valores a través de sus nombres.<br>(También existe el objeto `headerValues` que mapea los parámetros que tengan más de un valor a un array) | (Suponiendo que llegó un *header* `"host"` con valor `"192.168.0.3:8080"`)<br>`${ header.host }` (Mostraría `"192.168.0.3:8080"`)<br>En un *servlet*, esto mismo se escribiría así: `req.getHeader("host");` |

**Documentación oficial:** https://docs.oracle.com/javaee/5/tutorial/doc/bnahq.html#bnaij

# Objetos implícitos en *EL* (continuación)

| Objeto | Descripción | Ejemplo |
|---|---|---|
| `cookie` | Mapea las *cookies* obtenidas en la solicitud para poder acceder a ellas a través de sus nombres. | (Suponiendo que hay una una *cookie* con valor `"color=Rojo"`)<br>`${ cookie.color.name }` (Mostraría `"color"`)<br>`${ cookie.color.value }` (Mostraría `"Rojo"`)<br>En un *servlet*, esto mismo implica buscar la *cookie* por nombre en el array devuelto por el método : `req.getCookies();` |
| `initParam` | Mapea los parámetros de inicialización de un *servlet* (entre etiquetas `<init-param>` dentro de `<servlet>` en el archivo **web.xml**) para poder acceder a sus valores a través de sus nombres. En este caso, el propio JSP es un *servlet*, y éstos rara vez tienen parámetros de inicio. | |
| `pageContext` | Provee acceso a los objetos `request`, `response`, `session` y `servletContext` | |
| `pageContext.request` | Es el mismo objeto (de tipo `HttpServletRequest`) que representa la solicitud (*request*) HTTP, enviada por el cliente, recibida por parámetro en los métodos `doGet()`, `doPost()` y otros, en un *servlet*. | (Suponiendo que la URL de acceso es `http://localhost:8080/mi-app/prueba`)<br>`${ pageContext.request.contextPath }` (Mostraría `/mi-app`)<br>En un *servlet*, esto mismo se escribiría así: `req.getContextPath();` |
| `pageContext.response` | Es el mismo objeto (de tipo `HttpServletResponse`) que representa la respuesta (*response*) HTTP, que enviará el servidor, recibida por parámetro en los métodos `doGet()`, `doPost()` y otros, en un *servlet*. | (Suponiendo que el código de estado actual de la respuesta es `200`)<br>`${ pageContext.response.status }` (Mostraría `200`)<br>En un *servlet*, esto mismo se escribiría así: `resp.getStatus();` |
| `pageContext.session` | Es el mismo objeto (de tipo `HttpSession`) que representa la sesión actual, obtenido a través del método `getSession()`, de `HttpServletRequest`, en un *servlet*. | (Suponiendo que el tiempo máximo de inactividad de la sesión actual es `60`)<br>`${ pageContext.session.maxInactiveInterval }` (Mostraría `60`)<br>En un *servlet*, esto mismo se escribiría así: `req.getSession().getMaxInactiveInterval();` |
| `pageContext.servletContext` | Es el mismo objeto (de tipo `ServletContext`) que representa el contexto actual, obtenido a través del método `getServletContext()` en un *servlet*. | (Suponiendo un parámetro `motorBD` con valor `MySQL` entre etiquetas `<context-param>` en el archivo **web.xml**)<br>`${pageContext.servletContext.getInitParameter("motorBD") }` (Mostraría `"MySQL"`)<br>En un *servlet*, esto mismo se escribiría así: `getServletContext().getInitParameter("motorBD");` |

# JavaBeans

Un **JavaBean** no es más que una clase de Java que debe seguir algunas convenciones, para que luego las librerías y los *frameworks* puedan instanciarlos y manipularlos de forma automática.

## Para ser un JavaBean debe...

- Tener sus atributos con visibilidad privada.
- Implementar la interfaz **`Serializable`**.
- Proveer un constructor sin parámetros.
- Dar acceso a los atributos a través de *getters* y *setters* públicos.

## No deja de ser JavaBean por...

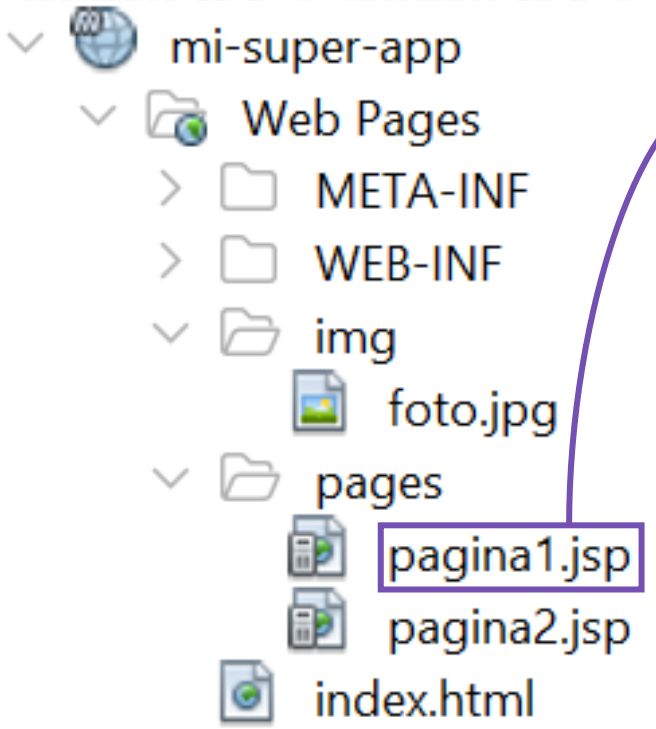
- Tener sobrecarga de constructores.
- No tener todos los *getters* y *setters*.
- Tener otros métodos que no sean *getters* y *setters*.

**PersonaBean.java**

```java
import java.io.Serializable;
public class PersonaBean implements Serializable {
    private int id;
    private String nombre;
    // Otros atributos...
    public PersonaBean() { } // Este constructor no puede faltar.
    public PersonaBean(int id, String nombre){
        setId(id);
        setNombre(nombre);
    }
    public void setNombre(String nombre) {
        if (nombre == null) throw new RuntimeException("¡Nombre nulo!");
        this.nombre = nombre;
    }
    public String getNombre() { return nombre; }
    // Otros getters y setters...
    // Otros métodos...
}
```

# Manejo de rutas

Las rutas relativas expresadas en un **.html** o **.jsp** varían su referencia inicial de acuerdo a lo que se le anteponga.




- Referencia a carpeta actual:
  - No anteponer nada
  - Anteponer `./`
- Referencia a la raíz: Anteponer `/`
- Subir un nivel: `../`
- Entrar a una carpeta: `/carpeta`

`http://localhost:8080/mi-super-app/pages/pagina1.jsp`

**pagina1.jsp**

```
<html>
    <head></head>
    <body>
        <a href="pagina2.jsp">Página 2</a> http://localhost:8080/mi-super-app/pages/pagina2.jsp
        <a href="./pagina2.jsp">Página 2</a> http://localhost:8080/mi-super-app/pages/pagina2.jsp
        <a href="/pagina2.jsp">Página 2</a> http://localhost:8080/pagina2.jsp
        <hr/>
        <a href="/">Volver a la Home</a> http://localhost:8080/
        <a href="/index.html">Volver a la Home</a> http://localhost:8080/index.html
        <a href="index.html">Volver a la Home</a> http://localhost:8080/mi-super-app/pages/index.html
        <a href="./index.html">Volver a la Home</a> http://localhost:8080/mi-super-app/pages/index.html
        <a href="../">Volver a la Home</a> http://localhost:8080/mi-super-app/
        <a href="../index.html">Volver a la Home</a> http://localhost:8080/mi-super-app/index.html
        <hr/>
        <img src="foto.jpg"> http://localhost:8080/mi-super-app/pages/foto.jpg
        <img src="/foto.jpg"> http://localhost:8080/foto.jpg
        <img src="img/foto.jpg"> http://localhost:8080/mi-super-app/pages/img/foto.jpg
        <img src="/img/foto.jpg"> http://localhost:8080/img/foto.jpg
        <img src="../img/foto.jpg"> http://localhost:8080/mi-super-app/img/foto.jpg
    </body>
</html>
```

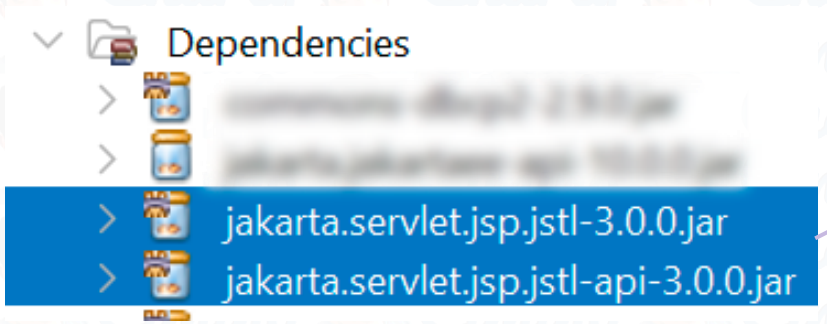
# JSTL: Jakarta Standard Tag Library

Librería que añade nuevas etiquetas y funciones a los archivos JSP, facilitando el desarrollo.

Para poder usar **JSTL**, es necesario descargar dependencias.

pom.xml

```xml
<dependencies>
    ...
    <dependency>
        <groupId>jakarta.servlet.jsp.jstl</groupId>
        <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
        <version>3.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>jakarta.servlet.jsp.jstl</artifactId>
        <version>3.0.0</version>
    </dependency>
    ...
</dependencies>
```




Tras guardar el archivo **pom.xml**

pom.xml

```xml
<dependencies>
    ...
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    ...
</dependencies>
```

**Esta es la versión de JSTL antes de migrar a Jakarta.**

**Hay muchos ejemplos en Internet (antiguos) que hacen uso de esta versión de JSTL.**

**No compatible con esta versión de Apache Tomcat**

# JSTL: Jakarta Standard Tag Library

JSTL añade nuevas etiquetas y funciones a los archivos JSP, divididas en cinco categorías

| Categoría | Descripción | URI | Prefijo | Etiquetas/Funciones | | |
|---|---|---|---|---|---|---|
| Etiquetas Núcleo (*Core Tags*) | Provee etiquetas de propósito general (declarar variables, controlar el flujo, manejar URLs, etc) | *jakarta.tags.core* | `c` | **(Cubiertas en próximas diapositivas)** | | |
| Etiquetas de función (*Function Tags*) | Provee funciones para manipular cadenas de caracteres. (Ver detalle en la documentación oficial) | *jakarta.tags.functions* | `fn` | `fn:contains()` | `fn:trim()` | `fn:substring()` |
| | | | | `fn:containsIgnoreCase()` | `fn:startsWith()` | `fn:substringAfter()` |
| | | | | `fn:endsWith()` | `fn:split()` | `fn:substringBefore()` |
| | | | | `fn:escapeXml()` | `fn:toLowerCase()` | `fn:length()` |
| | | | | `fn:indexOf()` | `fn:toUpperCase()` | `fn:replace()` |
| Etiquetas de formato (*Formatting Tags*) | Provee etiquetas para formatear mensajes, números, fechas y horas, etc. (Ver detalle en la documentación oficial) | *jakarta.tags.fmt* | `fmt` | `<fmt:parseNumber>` | `<fmt:parseDate>` | `<fmt:setBundle>` |
| | | | | `<fmt:timeZone>` | `<fmt:bundle>` | `<fmt:message>` |
| | | | | `<fmt:formatNumber>` | `<fmt:setTimeZone>` | `<fmt:formatDate>` |
| Etiquetas de XML (*XML Tags*) | Provee etiquetas para manipular documentos XML. (Ver detalle en la documentación oficial) | *jakarta.tags.xml* | `x` | `<x:out>` | `<x:choose>` | `<x:if>` |
| | | | | `<x:parse>` | `<x:when>` | `<x:transform>` |
| | | | | `<x:set>` | `<x:otherwise>` | `<x:param>` |
| Etiquetas de SQL (*SQL Tags*) | Provee etiquetas para soportar SQL. (Ver detalle en la documentación oficial) | *jakarta.tags.sql* | `sql` | `<sql:setDataSource>` | `<sql:update>` | `<sql:dateParam>` |
| | | | | `<sql:query>` | `<sql:param>` | `<sql:transaction>` |

```
<%@ taglib uri="verEnLaTabla" prefix="verEnLaTabla" %>
```

**Se debe declarar esta directiva en cada archivo .jsp que quiera utilizar alguna etiqueta JSTL. El `uri` y `prefix` dependen de la categoría.**

**Documentación oficial de JSTL (versión 3.0):** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0

# JSTL Core Tags

Provee etiquetas para declarar variables, controlar el flujo, manejar URLs, etc.

```
<%@ taglib uri="jakarta.tags.core" prefix="c" %>
```

**Se debe declarar esta directiva en cada archivo .jsp que la utilice.**

`<c:import>` `<c:param>` `<c:if>` `<c:choose>` `<c:forEach>` `<c:forTokens>`

**Etiquetas de la categoría *Core* con ejemplos en esta PPT**

| `<c:out>` | `<c:set>` | `<c:remove>` | `<c:catch>` | `<c:url>` | `<c:redirect>` |
|---|---|---|---|---|---|
| (Ver ejemplo) | (Ver ejemplo) | (Ver ejemplo) | (Ver ejemplo) | (Ver ejemplo) | (Ver ejemplo) |

**Otras etiquetas de la categoría *Core* (no cubiertas en esta PPT)**

# Importar recursos: Etiqueta `<c:import>`

Importa recursos estáticos o dinámicos a través del valor del atributo `url`

Muy útil para reutilizar partes comunes de un sitio web (*header*, *navbar*, *footer*, etc) o separar componentes complejos.

```
<c:import url="urlDelRecursoAImportar" />
```

Se le pueden pasar uno o más parámetros para flexibilizar la reutilización del componente a importar.

```
<c:import url="urlDelRecursoAImportar">
    <c:param name="nombreParam1" value="valorParam1" />
    <c:param name="nombreParam2" value="valorParam2" />
</c:import>
```

**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#cimport

# Importar recursos: Etiqueta <c:import> (Ejemplo)

**head.jsp**

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="css/estilo.css"/>
    <title>${param.tituloDePagina}</title>
</head>
```

**footer.html**

```
<footer>
    <p>Por Charly Cimino</p>
</footer>
```

**jstl_import.jsp**

```
<html>
    <c:import url="head.jsp">
        <c:param name="tituloDePagina" value="JSTL - Ejemplo &lt;c:import&gt;" />
    </c:import>
    <body>
        <p>El head fue obtenido a través de un import,
           con el valor del title por parámetro</p>
        <p>El siguiente JSON fue obtenido a través de un import</p>
        <pre><c:import url="https://jsonplaceholder.typicode.com/posts/1"/></pre>
        <p>El footer fue obtenido a través de un import</p>
        <c:import url="footer.html" />
    </body>
</html>
```

**otraPagina.jsp**

```
<html>
    <c:import url="head.jsp">
        <c:param name="tituloDePagina" value="Otra página" />
    </c:import>
    <body>
        <p>Otra página donde no hizo falta repetir el mismo
            código del head y el footer gracias al import</p>
        <c:import url="footer.html" />
    </body>
</html>
```

**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#cimport

# Control de flujo: Etiqueta <c:if>

Evaluará o no sus elementos de acuerdo a una expresión lógica (en el atributo `test`).

```
<c:if test="${expresiónLógica}">
    <%-- Elementos si test es true --%>
</c:if>
```

**Suponiendo un atributo `persona` de tipo `PersonaBean` ligado a la petición**

jstl_if.jsp

```
<html>
    <head>...</head>
    <body>
        <p>Hola ${persona.nombreCompleto}</p>
        <c:if test="${persona.nroHijos > 0}">
            <p>¡Saludos a tus hijos!</p>
        </c:if>
    </body>
</html>
```

**Esta etiqueta no provee un `else`. Se puede emular con `<c:choose>` (ver en próxima diapositiva)**

Documentación oficial: https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#cif

# Control de flujo: Etiqueta <c:choose>

Funciona análogamente a un **switch** en Java (pero en este caso, cada **case** es una expresión lógica)

Debe haber al menos una. Análogo al **case** en un **switch**.
Debe aparecer sí o sí antes de la etiqueta **<c:otherwhise>**

```
<c:choose>
    <c:when test="${expresiónLógica1}">
        <%-- Elementos si este when es true --%>
    </c:when>
    <c:when test="${expresiónLógica2}">
        <%-- Elementos si este when es true --%>
    </c:when>
    <c:otherwise>
        <%-- Elementos si todos los when fueron false --%>
    </c:otherwise>
</c:choose>
```

Es opcional. Análogo al **default** en un **switch**.
Debe ser la última acción dentro de la etiqueta **<c:choose>**

**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#cchoose

# Control de flujo: Etiqueta <c:choose> (Ejemplos)

**jstl_choose.jsp**

```
<html>
    <head>...</head><body>
    <p>Hola ${persona.nombreCompleto}</p>
    <c:choose>
       <c:when test="${persona.nroHijos >= 1}">
          <p>¡Saludos a tus hijos!</p>
       </c:when>
       <c:otherwise>
          <p>¡Saludos a tu familia!</p>
       </c:otherwise>
    </c:choose>
</body></html>
```

Un solo **</c:when>**

**jstl_choose.jsp**

```
<html>
    <head>...</head><body>
    <p>Hola ${persona.nombreCompleto}</p>
    <c:choose>
        <c:when test="${persona.nroHijos > 1}">
            <p>¡Saludos a tus hijos!</p>
        </c:when>
        <c:when test="${persona.nroHijos == 1}">
            <p>¡Saludos a tu hijo!</p>
        </c:when>
    </c:choose>
</body></html>
```

Múltiples **</c:when>**

Suponiendo un atributo **persona** de tipo **PersonaBean** ligado a la petición

**jstl_choose.jsp**

```
<html>
    <head>...</head><body>
    <p>Hola ${persona.nombreCompleto}</p>
    <c:choose>
        <c:when test="${persona.nroHijos > 1}">
            <p>¡Saludos a tus hijos!</p>
        </c:when>
        <c:when test="${persona.nroHijos == 1}">
            <p>¡Saludos a tu hijo!</p>
        </c:when>
        <c:otherwise>
            <p>¡Saludos a tu familia!</p>
        </c:otherwise>
    </c:choose>
</body></html>
```

Con **</c:otherwise>**

# Control de flujo: Etiqueta <c:foreach>

## Caso de uso 1

Análogo a un **for** en Java

```
<c:foreach var="i" begin="1" end="10" step="1">
    <%-- Elementos repetidos por cada valor de i --%>
</c:foreach>
```

**Suponiendo un atributo persona de tipo PersonaBean ligado a la petición**

jstl_foreach.jsp

```
<html>
    <head>...</head>
    <body>
        <p>Hola ${persona.nombreCompleto}</p>
        <c:forEach var="cont" begin="1" end="${persona.nroHijos}" step="1">
            <p>Saludos a tu hijo nº ${cont}</p>
        </c:forEach>
    </body>
</html>
```

**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#c:forEach

# Control de flujo: Etiqueta <c:foreach>

## Caso de uso 2

Análogo a un **foreach** en Java

```
<c:foreach items="${unaColeccion}" var="item">
    <%-- Elementos repetidos por cada item --%>
</c:foreach>
```

**Suponiendo un atributo `listaDePersonas` de tipo `List<PersonaBean>` ligado a la petición**

**jstl_foreach.jsp**

```
<html>
    <head>...</head>
    <body>
        <c:forEach items="${listaDePersonas}" var="per" varStatus="estado">
            <p>Persona nº ${ estado.count }: ${persona.nombreCompleto} </p>
        </c:forEach>
    </body>
</html>
```

Nombre de la variable que aloja temporalmente a cada ítem de la colección

(**Opcional**) Nombre de la variable que apunta a un objeto con información sobre la iteración

Otros campos de información de la iteración pueden ser **`index`**, **`first`**, **`last`**, etc. (Ver detalle en la documentación)

**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#c:forEach

# Control de flujo: Etiqueta <c:forTokens>

Permite iterar una cadena con valores delimitados por un separador

```
<c:forTokens items="${cadenaDeTokens}" items="token" delims="separador">
    <%-- Elementos repetidos por cada token --%>
</c:forTokens>
```

**Suponiendo un atributo `coloresCSV` de tipo `String` ligado a la petición**

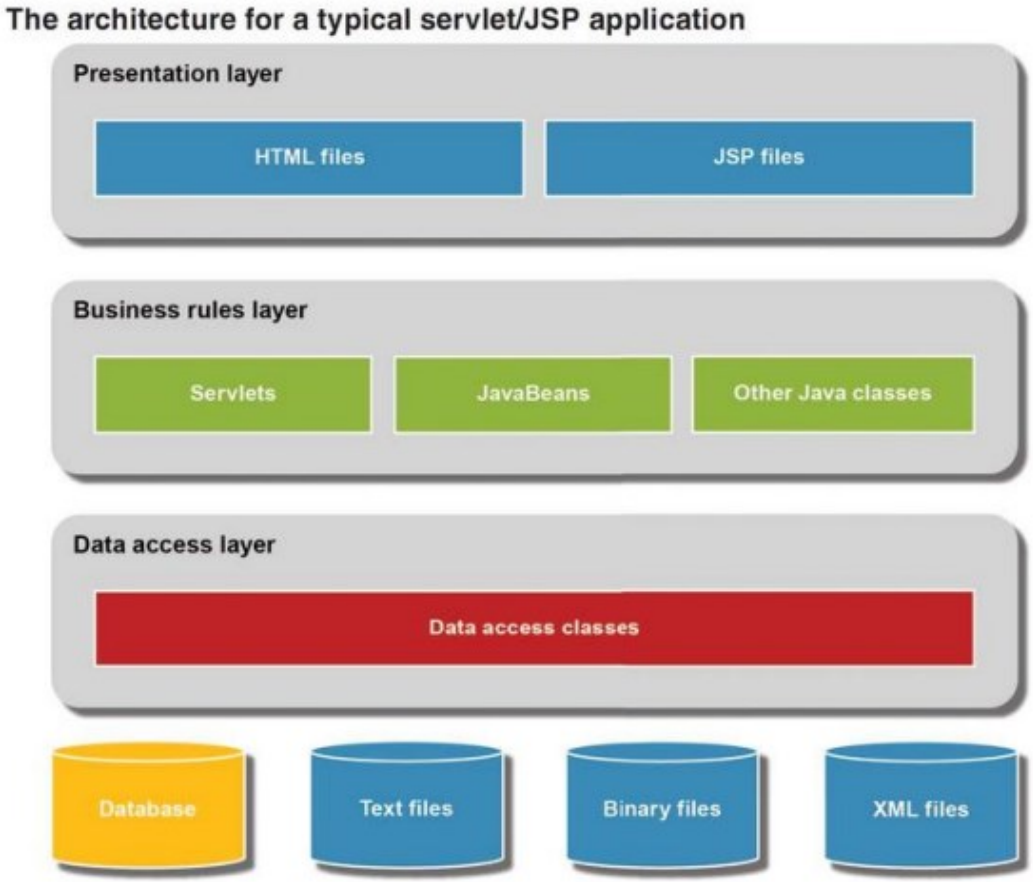
jstl_forTokens.jsp

```
<html>
    <head>...</head>
    <body>
        <p>Listado de colores que llegaron separados por comas:</p>
        <ul>
            <c:forTokens items="${coloresCSV}" var="color" delims=",">
                <li>${color}</li>
            </c:forTokens>
        </ul>
    </body>
</html>
```

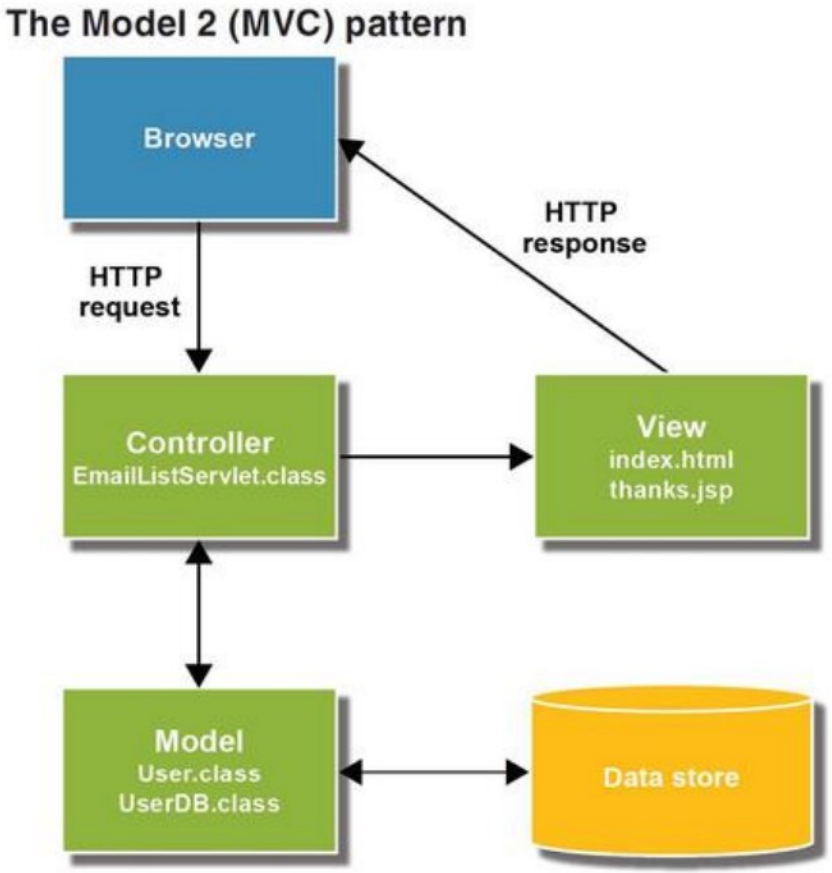
**Documentación oficial:** https://jakarta.ee/specifications/tags/3.0/jakarta-tags-spec-3.0#c:forTokens

# Arquitectura MVC con *Servlets* y JSP

En una arquitectura MVC (**Modelo-Vista-Controlador**), los archivos **.jsp** son las **vistas** (mientras que los *servlets* son los **controladores**)




Murach, J. and Urban, M. (2014) 'An Introduction to web programming with Java', in *Murach's Java Servlets and JSP: Training & reference*. Fresno, CA: Mike Murach & Associates, Inc., p. 17.




Murach, J. and Urban, M. (2014) 'How to structure a web application with the MVC pattern', in *Murach's Java Servlets and JSP: Training & reference*. Fresno, CA: Mike Murach & Associates, Inc., p. 33.

# Mini app de ejemplo

El siguiente repositorio permite acceder a una sencilla aplicación Java Web con *servlets* y JSP utilizando el patrón de arquitectura **MVC**.

Recetorium

## ¡Te damos la bienvenida a nuestro sitio!

¿Que deseás hacer?

Contribuir con una receta

Explorar recetas existentes

Charly Cimino

Recetorium

## Agregar nueva receta

Nombre:

Agregar foto:

Seleccionar archivo Sin archivos seleccionados

Instrucciones

Agregar receta Restablecer

Recetorium

## Lista de recetas

Nueva receta

| Nombre | Foto | Acciones | | |
|---|---|---|---|---|
| Tarta de manzana |  | Ver | Editar | Borrar |
| Sopa de pollo |  | Ver | Editar | Borrar |
| Ensalada César |  | Ver | Editar | Borrar |

Volver atrás

Recetorium

## Tarta de manzana

### Instrucciones

Pelar y cortar las manzanas en trozos. Mezclar con canela y azúcar. Cubrir con masa y hornear durante 45 minutos.

Volver atrás

Charly Cimino

# APÉNDICE: *Scriptlets* y directivas JSP

Considerados obsoletos, pero aún hay mucho código escrito con ellos.

# *Scriptlets*, expresiones y declaraciones JSP

Son etiquetas usadas para ejecutar código de Java embebido en el JSP.

**Esta metodología es obsoleta, pues estamos nuevamente mezclando lógica de negocio con lógica de presentación, que es lo que de un principio se quiere evitar.**

**saludo.jsp**

```
<html>
    <head></head>
    <body>
        <%-- Este es un comentario JSP: No es evaluado.
             Solo se ve en el server, no llega al cliente
        --%>
        <h1>
            <%
                String nombre;
                nombre = request.getParameter("nombreCli");
                out.print("Hola " + nombre);
            %>
        </h1>
          <p> 2 + 2 es <%= 2 + 2 %> </p>
    </body>
</html>
```

Las expresiones permiten imprimir el resultado de una expresión Java, evitando tener que recurrir a **`out.print()`**. Se encierran entre `<%= %>`

**ejemplo.jsp**

```
<html>
    <head></head>
    <body>
        <%!
            int contVisitas = 0;
            int doble(int n){
                return 2*n;
            }
        %>
        <p> El doble de 5 es <%= doble(5) %> </p>
        <p>
            Esta es la visita Nº <%
                contVisitas++;
                out.print(contVisitas);
            %>
        </p>
    </body>
</html>
```

Las declaraciones permiten definir variables o métodos en un JSP, que deben encerrarse entre `<%! %>`

Los *scriptlets* permiten incrustar código Java, que debe encerrarse entre `<% %>`

# Objetos implícitos en los *scriptlets*

Hay 9 objetos implícitos que son creados por el contenedor web para acceder a ellos fácilmente desde los JSP usando *scriptlets*.

| Objeto | Tipo | Descripción | Ejemplo |
|---|---|---|---|
| `out` | `JspWriter` | Permite escribir en el búfer de salida. Mientras que en un *servlet* debemos escribir una declaración como `PrintWriter out = response.getWriter();` aquí ya directamente contamos con el objeto `out` listo para usar. | Ver ejemplo |
| `request` | `HttpServletRequest` | Es el mismo objeto que representa la solicitud (*request*) HTTP, enviada por el cliente, recibida por parámetro en los métodos `doGet()`, `doPost()` y otros de un *servlet*. | Ver ejemplo |
| `response` | `HttpServletResponse` | Es el mismo objeto que representa la respuesta (*response*) HTTP, que enviará el servidor, recibida por parámetro en los métodos `doGet()`, `doPost()` y otros de un *servlet*. | Ver ejemplo |
| `config` | `ServletConfig` | Es el mismo objeto que se obtiene con el método `getServletConfig()` en un *servlet*. Permite obtener sus parámetros iniciales y otros datos. | Ver ejemplo |
| `application` | `ServletContext` | Es el mismo objeto que se obtiene con el método `getServletContext()` de un `ServletConfig` en un *servlet*. Permite obtener parámetros de inicialización del archivo de configuración (**web.xml**). También se puede utilizar para manipular atributos del ámbito de aplicación. | Ver ejemplo |
| `session` | `HttpSession` | Es el mismo objeto que se obtiene con el método `getSession()` de un `HttpServletRequest` en un *servlet*. Permite manipular atributos y otra información del ámbito de sesión. | Ver ejemplo |
| `pageContext` | `PageContext` | Permite manipular atributos en los diferentes ámbitos (página, solicitud, sesión o aplicación). | Ver ejemplo |
| `page` | `Object` | Hace referencia al propio JSP. No es muy utilizado. | Ver ejemplo |
| `exception` | `Throwable` | Representa la excepción capturada y manejada por este JSP, el cual debe ser marcado como página de error a través de una directiva JSP (visto más adelante). | Ver ejemplo |

# Directivas JSP

Son etiquetas usadas para indicarle al contenedor web cómo traducir una página JSP al *servlet* correspondiente.

## Sintaxis

```
<%@ tipoDirectiva atributo="valor" %>
```

## Tipos de directivas

- `page`
- `include`
- `taglib`

Se usó previamente para importar la librería de **JSTL**

# Directivas JSP page

Definen uno o más atributos que se aplican a una página JSP completa.

```
<%@ page atributo1="valor1" atributo2="valor2" atributoN="valorN" %>
```

| Atributos | Descripción | Ejemplo |
|---|---|---|
| `import` | Usado para importar clases, similar a como se hace en las de Java. | Ver ejemplo |
| `contentType` | Define el tipo MIME de la respuesta HTTP (tema explicado en la PPT de HTTP). Por defecto es `"text/html;charset=ISO-8859-1"` | Ver ejemplo |
| `extends` | Para lograr herencia entre JSPs. Rara vez usado. | |
| `info` | Establece información de la página JSP que se puede recuperar más tarde utilizando el método `getServletInfo()` en un *servlet*. | Ver ejemplo |
| `buffer` | Establece el tamaño del búfer en kilobytes de la salida generada por el JSP. Por defecto es **8**. | Ver ejemplo |
| `language` | Especifica el lenguaje de script utilizado en la página JSP. Por defecto es `"java"`. | |
| `isELIgnored` | Permite ignorar la sintaxis de *Expression Language*. Por defecto es `"false"`. | Ver ejemplo |
| `isThreadSafe` | Para manejar el comportamiento multiproceso del JSP. Por defecto es `"true"`. | Ver ejemplo |
| `autoFlush` | Especifica si la salida almacenada en el búfer debe vaciarse automáticamente cuando éste se llena o si en tal caso debe generarse una excepción de desbordamiento. Por defecto es `true` (vaciado automático) | `<%@ page autoFlush="true" %>` |
| `session` | Comprueba si la página JSP está en una sesión HTTP particular o no. Por defecto es `"true"`. | Ver ejemplo |
| `pageEncoding` | Define el esquema de codificación de caracteres. | `<%@ page pageEncoding="utf-8" %>` |
| `errorPage` | Se usa para definir la página de error. Si se produce una excepción en la página actual, se redirigirá a la página declarada en esta directiva. | Ver ejemplo |
| `isErrorPage` | Se usa para declarar que la página actual maneja errores. Habilita el objeto implícito `exception` visto en la diapositiva anterior. | Ver ejemplo |

# Directivas JSP include

Permiten incluir el contenido de cualquier otro recurso (JSP, HTML u otro archivo de texto) en el lugar donde se hace la declaración. Muy útil para reutilizar partes comunes de un sitio web (*header*, *navbar*, *footer*, etc).

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

**index.jsp**

```
<html>
    <head>
        <title>Prueba</title>
    </head>
    <body>
        <h1>El título</h1>
        <p>lorem ipsum...</p>

        <%@ include file="/footer.html" %>
    </body>
</html>
```

**footer.html**

```
<footer class="container-fluid">
    <div class="py-3 bg-dark">
        <p>Realizado por Prof. Carlos Cimino</p>
    </div>
</footer>
```

**Recomendado para incluir contenido estático (HTML, texto plano)**

**Para incluir contenido dinámico (otro .jsp) mejor usar la etiqueta de acción `<jsp:include>` (cubierto en siguientes diapositivas)**

# Etiquetas de acción

Permiten realizar tareas específicas, como controlar el flujo entre páginas o usar JavaBeans.
No se incluyen `<jsp:plugin>` y `<jsp:fallback>` por considerarse obsoletas.

| Etiqueta | Descripción | Ejemplo |
|---|---|---|
| `<jsp:forward>` | Para redireccionar a otro recurso con las mismas *request* y *response*.<br>Se pueden agregar parámetros adicionales a la *request* con etiquetas `<jsp:param>` anidadas. | **Sin parámetros**<br>`<jsp:forward page="presentar.jsp" />`<br>**Con parámetros**<br>`<jsp:forward page="presentar.jsp">`<br>`<jsp:param name="nombreCli" value="Pepe">`<br>`<jsp:param name="edad" value="20">`<br>`</jsp:forward>` |
| `<jsp:include>` | Para incluir otro recurso (estático o dinámico).<br>Si es dinámico, mejor usar esta etiqueta que una directiva.<br>Se pueden agregar parámetros adicionales a la *request* con etiquetas `<jsp:param>` anidadas. | **Sin parámetros**<br>`<jsp:include page="fechaDeHoy.jsp" />`<br>**Con parámetros**<br>`<jsp:include page="fechaDeHoy.jsp">`<br>`<jsp:param name="formato" value="dd/mm/aa">`<br>`<jsp:param name="color" value="#fac4c2">`<br>`</jsp:include>` |
| `<jsp:param>` | Para establecer un parámetro dentro de un `<jsp:forward>` o `<jsp:include>` | Ver ejemplos de `<jsp:forward>` y `<jsp:include>` |
| `<jsp:useBean>` | Crea o localiza un JavaBean. Si ya estaba creado, no lo vuelve a instanciar, de lo contrario, sí. | Ver ejemplo |
| `<jsp:setProperty>` | Establece un nuevo valor de atributo para un JavaBean | Ver ejemplo |
| `<jsp:getProperty>` | Obtiene el valor de un atributo para un JavaBean | |

# FIN DE LA PRESENTACIÓN

Encontrá más como estas en mi sitio web.