

# ÍNDICE

# Contenido

[JSPs 2](#_Toc102567285)

[1. Introducción 2](#_Toc102567286)

[2. Arquitectura Modelo-Vista-Controlador 2](#_Toc102567287)

[Modelos 3](#_Toc102567288)

[Vistas 3](#_Toc102567289)

[Controladores 3](#_Toc102567290)

[3. Funcionamiento JSP 4](#_Toc102567291)

[4. ¿Qué es una página JSP? 5](#_Toc102567292)

[Probando login.jsp 8](#_Toc102567293)

[5. ¿Cómo funciona una JSP en el servidor o contenedor de Servlets? 9](#_Toc102567294)

[6. Estructuras JSP de código 11](#_Toc102567295)

[Directivas: 11](#_Toc102567296)

[Directiva page 12](#_Toc102567297)

[Directiva include 14](#_Toc102567298)

[Elementos de scripting: 16](#_Toc102567299)

[7. Integración de Servlets y JSP 17](#_Toc102567300)

[Modelos de Funcionamiento 17](#_Toc102567301)

[Modelo de funcionamiento (I) 17](#_Toc102567302)

[Modelo de funcionamiento (II) 17](#_Toc102567303)

[Mecanismos de redirección de peticiones 18](#_Toc102567304)

[Redirecciones *sendRedirect* 18](#_Toc102567305)

[Redirecciones *forward* 18](#_Toc102567306)

[Redirecciones *forward* con parámetros 19](#_Toc102567307)

[Redirecciones *forward* con parámetros 19](#_Toc102567308)

[Envío de parámetros de formularios 19](#_Toc102567309)

# JSPs

# Introducción

JSP es un acrónimo de Java Server Pages, que en castellano vendría a decir algo como Páginas de Servidor Java. Es, pues, una tecnología orientada a crear páginas web con programación en Java.

Con JSP podemos crear aplicaciones web que se ejecuten en variados servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java. Por tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual.

El motor de las páginas JSP está basado en los servlets de Java -programas en Java destinados a ejecutarse en el servidor-, aunque el número de desarrolladores que pueden afrontar la programación de JSP es mucho mayor, dado que resulta mucho más sencillo aprender que los servlets.

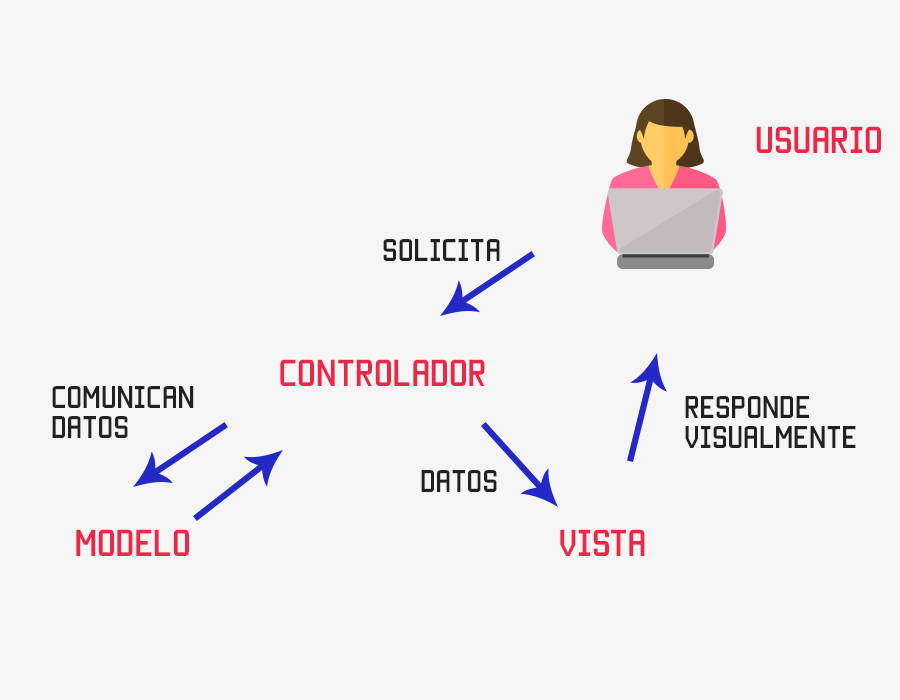
# Arquitectura Modelo-Vista-Controlador

MVC es una propuesta de arquitectura del software utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, lo que ofrece beneficios diversos.

MVC se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés. En este artículo estudiaremos con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando desarrollamos.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web.



## Modelos

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

## Vistas

Las vistas, como su nombre nos hacen entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

## Controladores

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

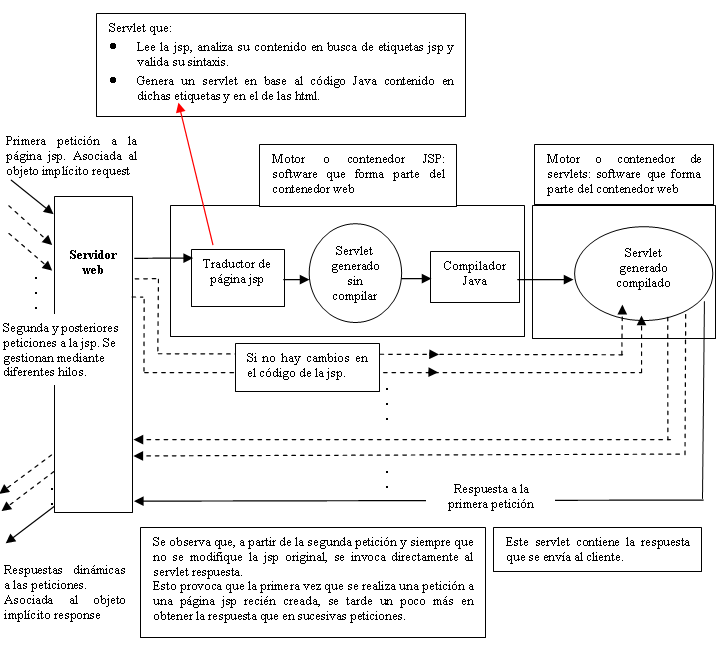
En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

# Funcionamiento JSP

Para poder acceder desde un cliente a una página jsp, es preciso que se encuentre almacenada en un servidor J2EE compatible.

contenedor web = contenedor servlet + contenedor jsp

Se utilizará, al igual que se hizo para el desarrollo de servlets, Apache Tomcat.



Etapas realizadas en el contenedor de jsps:

* Traducción de la jsp y generación de un código Java que es un servlet.
* Compilación del servlet asociado a la jsp.

Etapas realizadas en el contenedor de servlets:

* Instanciación del servlet y carga en memoria.

# ¿Qué es una página JSP?

Una JSP es un documento basado en tags HTMLs y tags propios de JSP. A continuación, vemos un ejemplo de una página llamada login.jsp:

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
<title>Login result</title>  
</head>  
<body>  
 <%  
 String user = request.getParameter("user");  
 String pass = request.getParameter("password");  
 if ("edu4java".equals(user) && "eli4java".equals(pass)) {  
 out.println("login ok");  
 } else {  
 out.println("invalid login");  
 }  
 %>  
</body>  
</html>

Aquí podemos ver que todo es HTML salvo lo que se encuentra dentro de los tags <% %> que es código java.

Esta página JSP hace lo mismos que el código del siguiente Servlet.

package com.edu4java.servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String user = request.getParameter("user");

String pass = request.getParameter("password");

if ("edu4java".equals(user) && "eli4java".equals(pass)) {

response(response, "login ok");

} else {

response(response, "invalid login");

}

}

private void response(HttpServletResponse resp, String msg)

throws IOException {

PrintWriter out = resp.getWriter();

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"" +

" \"http://www.w3.org/TR/html4/loose.dtd\">");

out.println("<html>");

out.println("<head>");

out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; " +

"charset=ISO-8859-1\">");

out.println("<title>Login result</title>");

out.println("</head>");

out.println("<body>");

out.println("<t1>" + msg + "</t1>");

out.println("</body>");

out.println("</html>");

}

}

Desde el punto de vista de un diseñador de páginas webs la jsp es infinitamente más sencilla que el servlet. Teniendo en cuenta que normalmente en las empresas los diseñadores web normalmente no tiene grandes conocimientos de java y viceversa se hace evidente las ventajas de JSP. Incluso para un programador java experimentado cuando la cantidad de html en la página es mucha el trabajo se puede volver una pesadilla. Imaginen páginas y páginas de out.println( ...

<%  
 String user = request.getParameter("user");  
 String pass = request.getParameter("password");  
 if ("edu4java".equals(user) && "eli4java".equals(pass)) {  
 out.println("login ok");  
 } else {  
 out.println("invalid login");  
 }  
 %>

Si nos centramos en el código java de la JSP podremos notar no hay declaración de las variables request y out. Estas variables son conocidas como Objetos Implícitos (Implicit Objects) y están predefinidas con nombres estándares para ser utilizadas por el programador en el código java de las JSPs.

A continuación, se describe algunos de estos Objetos Implícitos:

|  |  |
| --- | --- |
| Objeto | Descripción |
| request | es el objeto con la información recibida desde el browser |
| response | es el objeto que contendrá la información que enviaremos como respuesta al browser |
| out | es un objeto Writer que podemos utilizar para enviar Html (o cualquier clase de contenido) al browser |
| session | este objeto mantiene datos entre llamadas del mismo usuario. |
| application | este objeto mantiene datos durante toda la vida de la aplicación |

Los objetos **request y response** son los que se reciben como parámetros en los métodos doPost(HttpServletRequest request, HttpServletResponse response) y doGet(HttpServletRequest request, HttpServletResponse response). Los objetos**request, response y application** permiten almacenar información y los veremos en profundidad en un tutorial sobre Alcances o Scopes en una aplicación web.

## Probando login.jsp

Copiamos login.jsp a WebContent y modificamos login.html para que en vez de llamar a /login llame directamente a login.jsp

<html>  
<body>  
 <form action="login.jsp" method="post">  
 <table>  
 <tr>  
 <td>User</td>  
 <td><input name="user" /></td>  
 </tr>  
 <tr>  
 <td>password</td>  
 <td><input name="password" /></td>  
 </tr>  
 </table>  
 <input type="submit" />  
 </form>  
</body>  
</html>

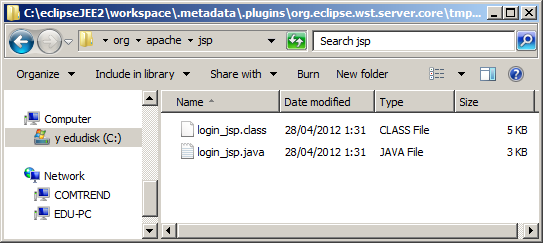
Botón derecho del ratón en login.html - Run as - Run on server.

Otra conveniencia de las JSPs es que no es necesario declararlas en web.xml.

# ¿Cómo funciona una JSP en el servidor o contenedor de Servlets?

Cuando desplegamos una JSP e iniciamos el servidor, el servidor crea el código java para un servlet a partir del contenido de la JSP y lo compila. En mi instalación podemos encontrar los archivos .java y .class de este servlet en:

C:\eclipse\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\first-jee\org\apache\jsp



No es necesario entender todo el código java generado pero es interesante darle un vistazo.

package org.apache.jsp;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

public final class login\_jsp extends org.apache.jasper.runtime.HttpJspBase

    implements org.apache.jasper.runtime.JspSourceDependent {

  private static final JspFactory \_jspxFactory = JspFactory.getDefaultFactory();

  private static java.util.List \_jspx\_dependants;

  private javax.el.ExpressionFactory \_el\_expressionfactory;

  private org.apache.AnnotationProcessor \_jsp\_annotationprocessor;

  public Object getDependants() {

    return \_jspx\_dependants;

  }

  public void \_jspInit() {

    \_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(getServletConfig()

.getServletContext()).getExpressionFactory();

    \_jsp\_annotationprocessor = (org.apache.AnnotationProcessor) getServletConfig()

.getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());

  }

  public void \_jspDestroy() {

  }

  public void \_jspService(HttpServletRequest request, HttpServletResponse response)

        throws java.io.IOException, ServletException {

    PageContext pageContext = null;

    HttpSession session = null;

    ServletContext application = null;

    ServletConfig config = null;

    JspWriter out = null;

    Object page = this;

    JspWriter \_jspx\_out = null;

    PageContext \_jspx\_page\_context = null;

    try {

      response.setContentType("text/html");

      pageContext = \_jspxFactory.getPageContext(this, request, response,

       null, true, 8192, true);

      \_jspx\_page\_context = pageContext;

      application = pageContext.getServletContext();

      config = pageContext.getServletConfig();

      session = pageContext.getSession();

      out = pageContext.getOut();

      \_jspx\_out = out;

      out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\""+

" \"http://www.w3.org/TR/html4/loose.dtd\">\r\n");

      out.write("<html>\r\n");

      out.write("<head>\r\n");

      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");

      out.write("<title>Login result</title>\r\n");

      out.write("</head>\r\n");

      out.write("<body>\r\n");

      out.write("\t");

String user = request.getParameter("user");

String pass = request.getParameter("password");

if ("edu4java".equals(user) && "eli4java".equals(pass)) {

out.println("login ok");

} else {

out.println("invalid login");

}

      out.write("\r\n");

      out.write("</body>\r\n");

      out.write("</html>");

    } catch (Throwable t) {

      if (!(t instanceof SkipPageException)){

        out = \_jspx\_out;

        if (out != null && out.getBufferSize() != 0)

          try { out.clearBuffer(); } catch (java.io.IOException e) {}

        if (\_jspx\_page\_context != null) \_jspx\_page\_context.handlePageException(t);

      }

    } finally {

      \_jspxFactory.releasePageContext(\_jspx\_page\_context);

    }

  }

}

Algunas curiosidades:

* La clase HttpJspBase extiende HttpServlet.
* \_jspService(HttpServletRequest request, HttpServletResponse response) reemplaza a doPost y doGet.
* Podemos ver las declaraciones de los objetos implícitos request, response, out, session, etc.
* El código html de nuestra JSP es enviado al browser usando out.write().

# Estructuras JSP de código

Dentro de los jsps distinguimos una serie de estructuras que hay que aprender a diferenciar, entre ellas encontramos las directivas y los elementos de scrpting.

## Directivas:

Controlan la estructura global del servlet respuesta y no generan salida para el cliente.

Por ejemplo, mediante la directiva page y el atributo import, se indican los paquetes no disponibles por defecto en la página jsp que el programador necesita para desarrollar los bloques de código Java de la jsp.

La sintaxis de una directiva genérica es la siguiente:

<%@ nombreDirectiva atributo1=”…” atributo2=”…”  …     %>

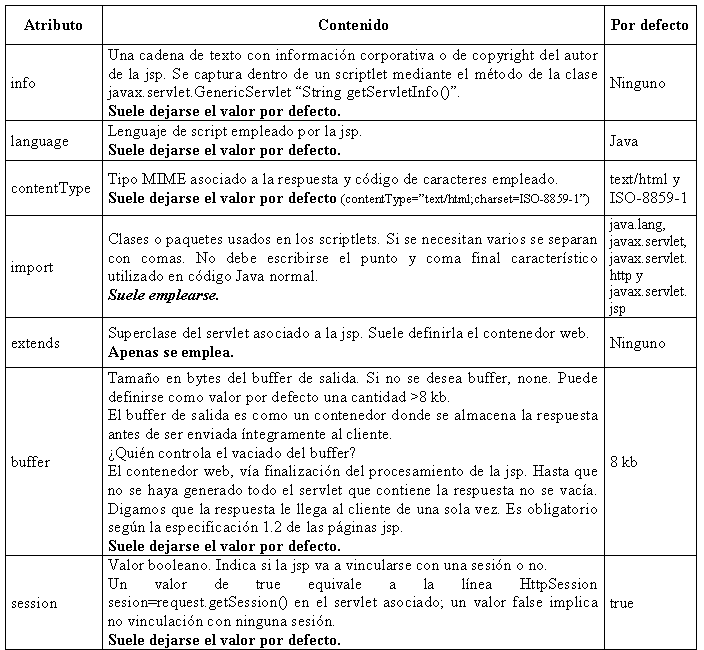
Las directivas más usadas son las siguientes:

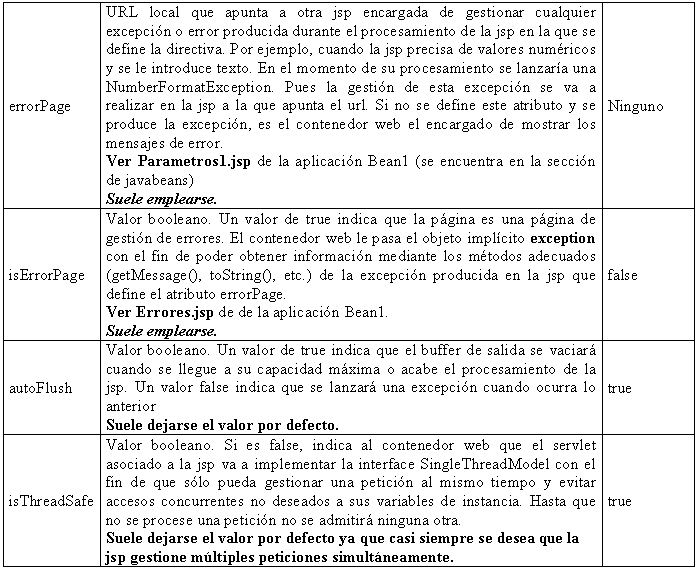
### Directiva page

Empleada para transmitir información de interés de la jsp al contenedor web. Por ejemplo, el lenguaje de script que va a emplear el programador, las clases y paquetes que va a necesitar para los scriptlets, cómo se van a gestionar los posibles errores en la introducción de datos, etc. Los nombres de los atributos son case-sensitive.

<%@ page atributo1=”…” atributo2=”…”  …       %>

Tabla de atributos, contenidos típicos y valores por defecto:

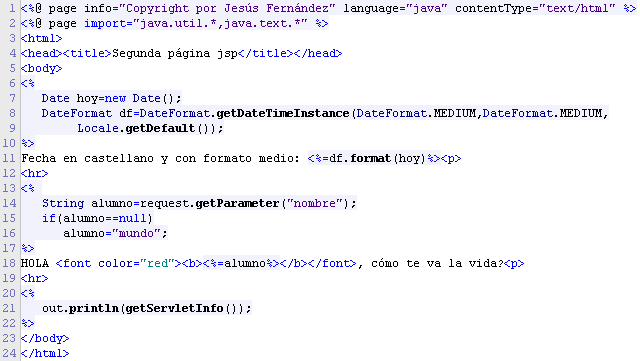




NOTA: los atributos principales y más habitualmente usados son:

* **import**. Permite importar clases Java en una jsp.
* **session**. Declara que una jsp se vincula a una sesión.
* **errorPage**. Declara que una jsp usa una página de gestión de errores.
* **isErrorPage**. Declara que una jsp es una página de gestión de errores.

Ejemplo: muestra el uso de la directiva page.



**NOTA: el orden en que aparecen las directivas page no es importante.**

Por ejemplo, puede colocarse la directiva import justo antes de </body>; incluso después de </html> y todo sigue funcionando. Por convenio, suelen situarse al principio de la jsp.

**En cambio, el orden en los elementos de scripting sí es importante.**

Por ejemplo, si la línea “Fecha en castellano …” se escribe antes que el scriptlet en el que se generan los objetos Date y DateFormat, se produce error.

### Directiva include

Permite la inclusión estática del contenido de otras jsps, páginas html, ficheros de texto no formateado, ficheros Java, ficheros XML, etc. No admite inclusión de servlets.

Se realiza durante la traducción de la jsp. En el momento en que el traductor de página detecta esta directiva, se incorpora el contenido del recurso incluido, en el método de servicio del servlet que se está gestando.

<%@ include file=”urlLocal”  %>

El contenido del atributo file es un url local (forma parte de la aplicación en la que está la jsp donde se define) que apunta al fichero que va a incluirse. No se admite ningún protocolo, ni nombre de host, ni número de puerto.

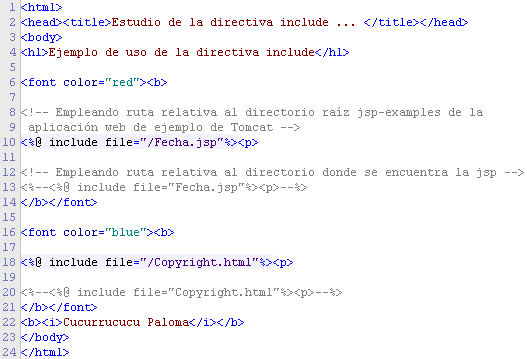
Este url puede definirse mediante:

* Ruta relativa al directorio raíz de la aplicación.
* Ruta relativa a la jsp donde se encuentra la directiva.

**Suele emplearse para incluir encabezados (headers) o pies de página (footers)** que contendrán todas las jsps de una aplicación web.

No hay limitación en el número de directivas include que pueden agregarse y sí es importante el orden en que aparecen.

Ejemplo: muestra el uso de la directiva include.





NOTA 1:

**No admite paso de parámetros al recurso incluido:**

<%@ include file=”incluir.jsp?saludo=hola” %>     es incorrecto

No admite que el atributo file sea evaluado mediante una expresión:

<% String esteUrl=”incluir.jsp”; %>

<%@ include file=”<%=esteUrl %>” %>     es incorrecto

NOTA 2: el orden importa. Si se modifica el orden de las líneas que incluyen la directiva include, cambia el orden en que se presenta la respuesta al cliente. Pueden colocarse las líneas que contienen la directiva include como primeras líneas.

## Elementos de scripting:

Permiten la inserción de código Java dentro de la página jsp.

Este código Java sirve de base para generar el servlet asociado a toda jsp, cuyas “tripas” contienen la respuesta html proporcionada al cliente.

Todos los elementos de scripting tienen acceso a una serie de objetos implícitos suministrados por el contenedor jsp que permiten aumentar la funcionalidad de las páginas jsp. Se estudiarán con detenimiento más adelante, aunque algunos como “request” y “out” ya han sido usados en el primer ejemplo de página jsp. Los elementos de scripting son los siguientes:

* Declaraciones: dentro de las etiquetas <%! código Java %>
* Expresiones: dentro de las etiquetas <%= código Java %>
* Scriptlets: dentro de las etiquetas <% código Java %>
* Comentarios: se tienen los siguientes tipos:
* De HTML: <!-- comentario -->
* De JSP: <%-- comentario --%>
* Del lenguaje de script Java: <%// comentario línea %> y <%/\* comentario varias líneas \*/%>
* Acciones:

Permiten trabajar con componentes complementarios a la página jsp como applets, otras páginas jsp, javabeans, etc.

Utilizan nomenclatura xml. Recordar que si una etiqueta xml no tiene contenido no es preciso cerrarla, basta con escribirla del modo <prefijo:nombreEtiqueta … /> teniendo en cuenta que los puntos suspensivos serán sus atributos. Son las siguientes:

No asociadas a los javabeans:

<jsp:include> … </jsp:include> o <jsp:include … /> si sólo tiene atributos

<jsp:plugin> … </jsp:plugin>

<jsp:forward> … </jsp:forward> o <jsp:forward … /> si sólo tiene atributos

Asociadas a los javabeans:

<jsp:useBean … /> si sólo tiene atributos o <jsp:useBean …> … </jsp:useBean>

<jsp:setProperty … />

<jsp:getProperty … />

# Integración de Servlets y JSP

Una aplicación Web realiza tareas de procesado y presentación:

* + Los Servlets son adecuados para procesado.
  + Las páginas JSP son adecuadas presentación.

Una aplicación Web puede combinar Servlets y páginas JSP:

* + Procesado de parámetros de la petición: Servlets.
  + Acceso a bases de datos: Servlets.
  + Lógica de la aplicación: Servlets.
  + Presentación (vistas): JSP.

## Modelos de Funcionamiento

### Modelo de funcionamiento (I)

1. El cliente envía la petición HTTP a un servlet.
2. El servlet procesa la petición.
   * Si es necesario, se conecta a la base de datos.
3. El servlet redirige la petición a un JSP.
   * Si es necesario, añade *beans* como parámetros.
4. El JSP lee los parámetros y devuelve la respuesta formateada visualmente al usuario.

Modelo de funcionamiento (II)

Figura que ilustra lo expuesto en la
 sección anterior (redirección con forward de un servlet a un
 JSP)

## Mecanismos de redirección de peticiones

Hay dos formas de redirigir una petición a otro recurso:

* + Redirecciones HTTP (*sendRedirect*):
    - El servidor envía una respuesta al cliente con un código 3xx y la URI a la que este debe enviar la petición.
  + Redirecciones internas en el servidor (*forward*):
    - Se redirige la petición de un recurso a otro dentro de la misma aplicación Web.
    - El recurso de la última redirección devuelve al cliente la respuesta HTTP.
    - La redirección es transparente para el cliente.

### Redirecciones *sendRedirect*

* Fuerza el envío de una respuesta HTTP de redirección al cliente.
* El cliente envía la petición a la URI recibida en la respuesta.

// Redirección con URI absoluta

response.sendRedirect("http://www.ejemplo.es/");

// Redirección con URI relativa a la URI de la petición actual

response.sendRedirect("otra.html");

### Redirecciones *forward*

* Un Servlet o JSP reenvía la petición a otro recurso (Servlet, JSP, HTML) de la misma aplicación Web.
* El cliente no se entera de la redirección (p.e., el navegador muestra la URI original de la petición, no la redirigida).
* El control retorna al finalizar el método *forward*, por lo que conviene que sea lo último que se ejecuta.

#### Redirecciones forward

* *Forward* desde un Servlet:

RequestDispatcher rd = request.getRequestDispatcher("/vista.jsp");

rd.forward(request, response);

* *Forward* desde un JSP:

<jsp:forward page="/vista.jsp"/>

### Redirecciones *forward* con parámetros

* El objeto de la petición (*ServletRequest*) de los recursos origen y destino de la redirección es el mismo:
  + Se pueden añadir parámetros como atributos a la petición.

Noticia nuevaNoticia = (...)

request.setAttribute("noticia", nuevaNoticia);

RequestDispatcher rd = request.getRequestDispatcher("/vista.jsp");

rd.forward(request, response);

### Redirecciones *forward* con parámetros

* Recogida de parámetros desde un Servlet:
* Noticia nuevaNoticia = (Noticia) request.getAttribute("noticia");
* Recogida de parámetros desde un JSP:
* <jsp:useBean id="noticia" class="beans.Noticia"
* scope="request" />

## Envío de parámetros de formularios

### Envío de parámetros de formularios

* El envío depende del método HTTP y la codificación:
  + Método HTTP:
    - Método GET.
    - Método POST.
  + Codificación:
    - application/x-www-form-urlencoded
    - multipart/form-data

### Envío de parámetros de formularios

* Codificación URL-encoded:
  + Lista de parámetros separados por ``SPMamp;''.
  + Para cada parámetro se especifica nombre ``='' valor.
  + Los caracteres especiales (no letras/dígitos ASCII) se codifican en hexadecimal por su código UTF-8.
  + Con método GET o POST.
  + No se usa para campos de tipo *file*.

usuario=juan&clave=juanpw&ssid=7fgxc&enviar=enviar

nombre=juan%20l%C3%B3pez%20l%C3%B3pez

### Envío de parámetros de formularios

* Codificación URL-encoded con GET:
  + Los parámetros se codifican en la ruta (*path*) de la petición HTTP.
  + Sólo apto para operaciones idempotentes.

GET /jaf/cgi-bin/html2xhtml.cgi?tipo=auto&html=default.html HTTP/1.1

Host: www.ejemplo.es

(...)

### Envío de parámetros de formularios

* Codificación URL-encoded con POST:
  + Los parámetros se codifican en el cuerpo de la petición HTTP.

POST /jaf/cgi-bin/html2xhtml.cgi HTTP/1.1

(...)

Content-Length: 27

Content-Type: application/x-www-form-urlencoded

tipo=auto&html=default.html