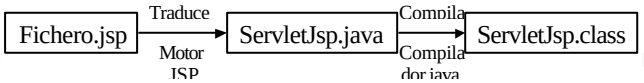
Introducción a la tecnología JSP
Aspectos avanzados de JSP

Introducción a JSP

- JSP es una especificación de *Sun Microsystems*
- Sirve para crear y gestionar **páginas web dinámicas**
- Permite mezclar en una página código HTML para generar la parte estática, con contenido dinámico generado a partir de marcas especiales <% .... %>
- El contenido dinámico se obtiene, en esencia, gracias a la posibilidad de incrustar dentro de la página código Java de diferentes formas
- Su objetivo final es separar la interfaz (presentación visual) de la implementación (lógica de ejecución)

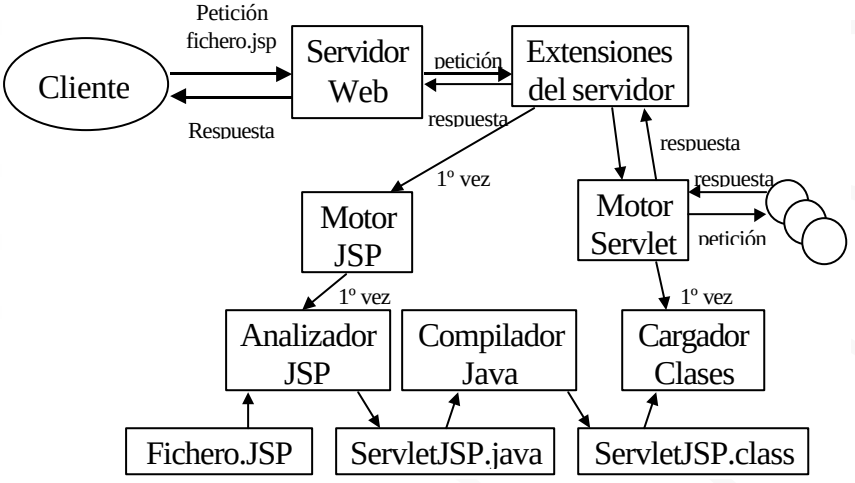
Introducción a JSP



Página JSP ----> Servlet

- La página JSP se convierte en un servlet
- La conversión la realiza en la máquina servidora el *motor o contenedor JSP*, la primera vez que se solicita la página JSP
- Este servlet generado procesa cualquier petición para esa página JSP
- Si se modifica el código de la página JSP, entonces se regenera y recompila automáticamente el servlet y se recarga la próxima vez que sea solicitada

Funcionamiento



Primer ejemplo de JSP

Ejemplo de página dinámica que dice Hola y escribe la fecha actual (fichero ej1_hola.jsp)

```
<%@ page info="Un ejemplo Hola Mundo"
   import="java.util.Date" %>
<HTML>
<head> <title> Hola, Mundo </title> </head>
<body> <h1> ¡Hola, Mundo! </h1>
La fecha de hoy es:   <%= new Date().toString()
   %>
</body>
</HTML>
```

- En esta página se mezcla código HTML con código Java incrustado con unas marcas especiales
- En este caso es una expresión, que se sustituye en la página por el resultado de evaluarla
- En otros casos es un trozo de código Java que simplemente se ejecuta

Puesta en marcha con el motor Tomcat

- Todas las páginas JSP deben guardarse en un subdirectorio (por ejemplo JSP), **dentro de Tomcat\webapps**

  Tomcat\webapps\JSP\ ej1_hola.jsp
- Se pueden organizar las páginas en directorios dentro del directorio definido para la aplicación.

  Ej: *Tomcat\webapps\JSP\ ejemplossimples\ej1_hola.jsp*
- Es necesario crear el **directorio Web-inf dentro del directorio generado** (Tomcat\webapps\JSP\Web-inf)
- Es necesario añadir información de **configuración al fichero Tomcat\conf\server.xml** (al final de los Context que haya en el fichero) indicándole cual va a ser el directorio base de la aplicación:

```
<Context path="/JSP" docBase=”JSP"
              debug="0" reloadable="true"/>
```

Puesta en marcha con el motor Tomcat

- Se accede a la página, por ejemplo desde un navegador en *http://localhost:8080/JSP/ejemplossimples\ej1_hola.jsp*
- Importante: se distingue entre mayúsculas y minúsculas en directorios y ficheros de configuración

Cuando se llama por primera vez al fichero JSP, se genera un servlet con las siguientes operaciones

- jspInit()
  - Inicializa el servlet generado
  - Sólo se llama en la primera petición
- jspService(petición,respuesta)
  - Maneja las peticiones. Se invoca en cada petición, incluso en la primera
- jspDestroy()
  - Invocada por el motor para eliminar el servlet

Servlet generado para una página JSP

- Si la página JSP está guardada en Tomcat\webapps\JSP, este código, generado automáticamente, se guarda en el directorio .... Tomcat\work\localhost\JSP

Para el fichero ej1_hola.jsp se genera:

- La clase **`public class ej1_0005fhola$jsp`**
- El método **`public final void _jspx_init()`**
- El método **`public void _jspService`**`(HttpServletRequest request, HttpServletResponse response)`

Servlet generado para una página JSP

En el método **`_jspService`** se introduce automáticamente el contenido dinámico de la página JSP.

- El código html se transforma en una llamada al objeto out donde se vuelca el contenido En el ejemplo:

```
out.write("\r\n\r\n<HTML>\r\n<head> <title>
 Hola, Mundo </title> </head>\r\n\r\n<body>
 <h1> ¡Hola, Mundo! </h1> \r\nLa fecha de hoy
 es: ");
```

- El código dinámico se traduce en función del contenido

Ej: El código jsp **`<%= new Date().toString() %>`** se traduce por **`out.print( new Date().toString() );`**

Servlet generado a partir del ejemplo JSP

En la primera invocación de esta página se genera automáticamente el siguiente servlet:

```
public class ej1_0005fhola$jsp extends HttpJspBase
  {
.....
public final void _jspx_init() throws
  JasperException {  ...
public void _jspService(HttpServletRequest request,
  HttpServletResponse  response)
        throws IOException, ServletException {
....
        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String  _value = null;
```


Servlet generado a partir del ejemplo JSP

```
try {
            if (_jspx_inited == false) {
                _jspx_init();
                _jspx_inited = true;
            }
            _jspxFactory =
  JspFactory.getDefaultFactory();

  response.setContentType("text/HTML;charset=ISO-
  8859-1");
            pageContext =
  _jspxFactory.getPageContext(this, request,
  response,"", true, 8192, true);

            application =
  pageContext.getServletContext();
            config =
  pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
```

Servlet generado a partir del ejemplo JSP

```
// HTML
// begin [file="C:\\web\\jakarta-
   tomcat3.2.2\\webapps\\marian\\hola.jsp";from=(0
   ,40);to=(6,20)]
    out.write("\r\n\r\n<HTML>\r\n<head> <title>
   Hola, Mundo </title> </head>\r\n\r\n<body> <h1>
   ¡Hola, Mundo! </h1> \r\nLa fecha de hoy es: ");
// end
// begin [file="C:\\web\\jakarta-
   tomcat3.2.2\\webapps\\marian\\hola.jsp";from=(6
   ,23);to=(6,46)]
    out.print( new Date().toString() );
// end
// HTML // begin [file="C:\\web\\jakarta-
   tomcat3.2.2\\webapps\\marian\\hola.jsp";from=(6
   ,48);to=(11,0)]

   out.write("\r\n\r\n</body>\r\n</HTML>\r\n\r\n")
   ;
// end
```


Objetos implícitos

- JSP utiliza los objetos implícitos, basados en la API de servlets.
- Estos objetos están disponibles para su uso en páginas JSP y son los siguientes:

Objeto request

  - Representa la petición lanzada en la invocación de service(). Proporciona entre otras cosas los parámetros recibidos del cliente, el tipo de petición (GET/POST)

- Objeto response
  - Instancia de HttpServletResponse que representa la respuesta del servidor a la petición. Ámbito de página

Otros: pageContext, session,application,out,config,page

Ámbitos

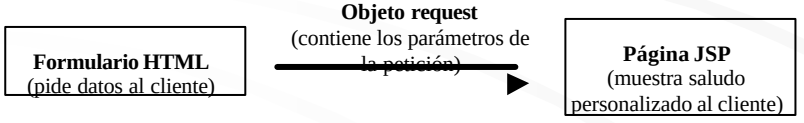
- Define dónde y durante cuanto tiempo están accesibles los objetos (Objetos implícitos, JavaBeans, etc)

Tipos de ámbitos

- de página. El objeto es accesible por el servlet que representa la página
- de petición
- de sesión. El objeto es accesible durante toda la sesión, desde los servlets a los que se accede
- de aplicación. El objeto es accesible por el servlet que representa la página

Ejemplo de uso de objetos implícitos

Aplicación que pide el nombre al usuario y le devuelve un saludo . Utiliza un fichero HTML como formulario que pide los datos al cliente y se los pasa a una página JSP que muestra el saludo con éstos datos. El paso de los datos del formulario al JSP se realiza a través de un objeto especial: objeto request



Ejemplo de uso de objetos implícitos

Este es el fichero HTML que pide los datos al cliente

```
<HTML>
  <head>
     <title> Formulario de petición de nombre
  </title>
     </head>
      <body>
          <h1> Formulario de petición de nombre
  </h1>
     <!-- Se envía el formulario al JSP
  "saludo.jsp" -->
          <form method="post" action="saludo.jsp"
  >
          <p> Por favor, introduce tu nombre:
          <input type="text" name="nombre">
           </p>
          <p> <input type="submit" value="enviar
  información"> </form> </body>
</HTML>
```


Ejemplo de uso de objetos implícitos

Fichero JSP que opera dinámicamente con los datos del cliente y muestra los resultados

```
<HTML>
  <head>
    <title> Saludo al cliente </title>
  </head>
<body>
  <h1> Saludo al cliente</h1>
<!-- Los parámetros que le pasa el cliente en la
  petición se obtienen del objeto implícito
  request -->
<%
  String nombre = request.getParameter("nombre");
  out.println("Encantado de conocerle,  " +
  nombre);
%>
<!-- Al evaluarse el código hay que escribir
  explícitamente en la salida (objeto implícito
  out) --> </body> </HTML>
```

Elementos de una página JSP

- **Código HTML**

Además de código HTML la página JSP puede incluir marcadores que se agrupan en tres categorías:

- **Directivas.**
  - Afectan a toda la estructura del servlet generado
- **Elementos de Scripting** (guiones)
  - Permiten insertar código Java en la página JSP
- **Acciones**
  - Afectan al comportamiento en tiempo de ejecución del JSP

Directivas JSP

- Utilizadas para definir y manipular una serie de atributos dependientes de la página que afectan a todo el JSP.
- Las directivas existentes son las siguientes:
  - Page
  - Include
  - Taglib

Directiva Page

Sintaxis

<%@ page ATRIBUTOS %>

- Donde ATRIBUTOS son parejas:
  nombre=”valor”
- Ejemplo:

<%@ page language=”Java” import=”Java.rmi.*,java.util.*” session=”true” buffer=”12kb” %>

- Existe una lista de atributos que pueden ser usados

Directiva Page

Algunos de los atributos más usados

- import
  - Lista de paquetes o clases, separados por coma, que serán importados para utilizarse dentro del código java.
- session
  - Especifica si la página participa en una sesión HTTP. Si se inicializa a true, está disponible el objeto implícito sesión.
- buffer
  - Especifica el tamaño de un buffer de salida de tipo stream, para el cliente.
- autoflush, info, errorPage, isErrorPage, ...

Directiva Include

- Indica al motor JSP que incluya el contenido del fichero correspondiente en el JSP, insertándolo en el lugar de la directiva del JSP.
- El contenido del fichero incluido es analizado en el momento de la traducción del fichero JSP y se incluye una copia del mismo dentro del servlet generado.
- Una vez incluido, si se modifica el fichero incluido no se verá reflejado en el servlet
- El tipo de fichero a incluir puede ser un
  - fichero HTML (estático)
  - fichero jsp (dinámico)

Sintaxis

<%@ include file=”Nombre del fichero” %>

Ejemplo de uso de la Directiva Include

Ejemplo:Página JSP que incluye el contenido de dos ficheros (una página HTML y una página JSP)

```
<HTML>
  <head>
   <title> Página de prueba de directivas de
  compilación  </title>
  </head>
  <body>
      <h1> Página de prueba de directivas de
  compilación  </h1>
      <%@ include file=”/fichero.html” %>
      <%@ include file=”/fichero.jsp” %>
   </body>
</HTML>
```

Ejemplo de uso de la Directiva Include

- Siendo, por ejemplo el fichero HTML el siguiente:

```
<HTML>
<head> <title> Hola, Mundo </title> </head>
<body> <h1> ¡Hola, Mundo! </h1>
</body>
</HTML>
```

- y el fichero JSP el siguiente:

```
<%@ page info="Un ejemplo Hola Mundo"
   import="java.util.Date" %>
La fecha de hoy es:   <%= new Date().toString() %>
```


Directiva Taglib

- Permite extender los marcadores de JSP con etiquetas o marcas generadas por el propio usuario (etiquetas personalizadas).
- Se hace referencia a una biblioteca de etiquetas que contiene código Java compilado definiendo las etiquetas que van a ser usadas, y que han sido definidas por el usuario

Sintaxis

<%@ taglib uri=”taglibraryURI” prefix=”tagPrefix” %>

Elementos de una página JSP

- **Código HTML**
- **Directivas**
  - page
  - include
  - taglib
- **Elementos de Scripting** (guiones)
- **Acciones** (marcas estandar)

Elementos Scripting

- Permiten la inserción de **Declaraciones**, **Código Java arbitrario** (scriptlets) y **Expresiones** dentro de una página JSP
  - Declaraciones
  - Código Java arbitrario
  - Expresiones

Declaraciones

- Usadas para definir variables y métodos **con ámbito de clase** para el servlet generado
- Estas variables o métodos declarados pasarán a ser variables de instancia de la clase servlet generada

Esto significa que serán globales a todo el servlet generado para la página

- Sintaxis

  <%! Declaración %>

- Ejemplo:

  <%! int contador >

Ejemplo de uso de Declaraciones

Uso de un contador que indica el número de veces que se accede a una página.

```
<HTML>  <head>
<title> Página de control de declaraciones
  </title>  </head>
<body>
  <h1> Página de control de declaraciones </h1>
  <%! int i=0 ; %> <!-- Esto es una declaración
  (una variable de instancia en este caso) -->
  <%
   i++;
  %> <!-- Esto es un scriptlet (código Java) que
  se ejecuta-->
  HOLA MUNDO
  <%= "Esto ha sido un JSP accedido " + i + ”
  veces" %>
<!-- Esto es una expresión que se evalúa y se
  sustituye en la página por su resultado
</body></HTML>
```

Scriptlets

- Un scriptlet es un bloque de código Java insertado en la página y ejecutado durante el procesamiento de la respuesta
- El código introducido se inserta directamente en el método _jspService() del servlet generado para la página

Sintaxis

<% código Java %>

- Ejemplo

```
<% int i,j;
    for (i=0;i<3;i++) {
       j=j+1;
     }
 %>
```


Ejemplo de uso de Scriptlets

- Página JSP que usa código Java para repetir 10 veces un saludo

```
<HTML>  <head>
    <title> Página de ejemplo de scriptlet
  </title>  </head>
<body>  <h1> Página de ejemplo de scriptlet </h1>
  <%
   for (int i=0; i<10; i++;){
     out.println("<b> Hola a todos. Esto es un
  ejemplo de scriptlet " + i + "</b><br>");
     System.out.println("Esto va al stream
  System.out" + i );
      //Esto último va a la consola del Java, no
  al cliente.
      //out a secas es para la respuesta al
  cliente.
     }
  %>
</body> </HTML>
```

Expresiones

- Notación abreviada que envía el valor de una expresión Java al cliente.
- La expresión se traduce por la llamada al método println del objeto out dentro del método _jspService(), con lo que en cada petición, la expresión es evaluada y el resultado **se convierte a un String y se visualiza**

Sintaxis

<%= Expresión Java a evaluar %>

- Ejemplo

<%= "Esta expresión muestra el valor de un contador " + contador %>

Nota: será necesario que previamente contador haya tomado un valor a través de un scriptlet

Ejemplo de uso de Expresiones

En esta página JSP la expresión consiste en crear un objeto y llamar a uno de sus métodos. El resultado es un string que se muestra al cliente

```
<HTML>
  <head>
    <title> Página de ejemplo de expresiones
  </title>
  </head>
<body>
  <h1> Página de ejemplo de expresiones </h1>
  Hola a todos, son las <%= new Date().toString()
  %>
</body>
</HTML>
```

Ejercicios

1. Con los ejemplos existentes en el directorio JSP\ejemplossimples
   - Ejecutarlos, visualizar el contenido de los ficheros, visualizar el contenido de alguno de los servlets generados para ellos
- Modificar el fichero incluido y comprobar que dichas modificaciones no aparecen reflejadas por la directiva

1. Utilizar el formulario de saludo visto con servlets, ycrear una página JSP que devuelva el saludo al usuario
2. Ejecutar la sumadora situada en el directorio sumadora
3. Crear una aplicación que funcione como calculadora con todas las operaciones.
4. Crear una aplicación que funciones como euroconversor

Elementos de una página JSP

- **Código HTML**
- **Directivas**
  - page
  - include
  - taglib
- **Elementos de Scripting** (guiones)
  - Declaraciones
  - Código Java arbitrario
  - Expresiones
- **Acciones**
  - **Acciones estándar**
  - **Acciones personalizadas**

Acciones estándar

- Son marcas estándar, con formato XML, que afectan al comportamiento en tiempo de ejecución del JSP y la respuesta se devuelve al cliente.
- En la traducción de JSP al servlet, la marca se reemplaza por cierto código Java que define a dicha marca. Una marca por tanto define un cierto código Java (es como una macro)
- Constan de un prefijo y un sufijo además de una serie de atributos. El prefijo es siempre "jsp" en las acciones estándar

Sintaxis

<jsp:sufijo atributos/>

- Ejemplo

<jsp:include page="mijsp.jsp" "flush="true" />

Tipos de acciones estándar existentes

- <jsp:include>
- <jsp:forward>
- <jsp:param>

- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>

- <jsp:plugin>

Acción jsp:include

- Permite incluir un recurso especificado por la URL, en la petición JSP en **tiempo de ejecución**.
- Cuando se realiza la traducción de JSP al servlet, dentro del método _jspService() se genera el código que comprueba si existe el recurso (página) y si no se crea, invocándolo a continuación.
- Cuando se ejecuta el servlet, se invoca al recurso que realiza la operación y devuelve el resultado al servlet
- El elemento incluido puede acceder al objeto request de la página padre, y además de los parámetros normales, a los que se añadan con <jsp:param>

Acción jsp:include

Sintaxis

```
<jsp:include page="URL" flush="true">
    <jsp:param name="nombre clave" value="valor"
      /> (no obligatorios) ....
</jsp:include>
```

Diferencia acción include- directiva include

Es importante distinguir entre directiva include y acción include

- Directiva <%@ include file="Nombre fichero" /> se añade el código al servlet que se genera para la página en tiempo de compilación y se incluye el contenido EXISTENTE EN EL MOMENTO INICIAL.
- Acción <jsp:include> no se añade código al servlet, sino que se invoca al objeto en tiempo de ejecución y se ejecuta el contenido EXISTENTE EN EL MOMENTO DE LA PETICIÓN

Ejemplo de uso de la acción include

```
<HTML>
  <head>
     <title> Inclusión de fichero </title>
     </head>
      <body>
          <h1> Inclusión de ficheros </h1>
         <%@ include file="incluido.jsp" %>
      </body>
</HTML>
```

- Fichero incluido (*"incluido.jsp"*)

```
<%@ page import="java.util.Date" %>
<%= "Fecha actual es " + new Date() %>
```

Tipos de acciones existentes

- <jsp:include>
- <jsp:param>
- <jsp:forward>
- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:plugin>

Acción jsp:param

- Se usa como submarca dentro de cualquier otra marca
- Sirve para pasar parámetros a un objeto

Sintaxis

```
<jsp:....      >
    <jsp:param name="nombre clave"  value="valor"
      /> (no obligatorios)
....
</jsp:....     >
```

Tipos de acciones existentes

- <jsp:include>
- <jsp:param>
- <jsp:forward>
- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:plugin>

Acción jsp:forward

- Esta marca permite que la petición sea redirigida a otra página JSP, a otro servlet o a otro recurso estático
- Muy útil cuando se quiere separar la aplicación en diferentes vistas, dependiendo de la petición interceptada.
- Cuando se ejecuta el servlet se redirige hacia otro servlet y no se vuelve al servlet original

Sintaxis

```
<jsp:forward page="URL" >
    <jsp:param name="nombre clave" value="valor"
      /> (no obligatorios)
....
</jsp:forward>
```

Ejemplo de uso de la acción forward

Formulario HTML que pide nombre y password y los envía a una página jsp que lo analiza *(forward.jsp)*

```
<HTML>
  <head>  <title> Ejemplo de uso del forward
  </title> </head>
  <body>
   <h1> Ejemplo de uso del forward </h1>
  <form method="post" action="forward.jsp">
          <input type="text" name="userName">
          <br> y clave:
          <input type="password"
  name="password">
          </p>
          <p><input type="submit" name="log
  in">
          </form>
</body>
</HTML>
```


Ejemplo de uso de la acción forward

- Página JSP que lo ejecuta
  - No tiene nada de HTML
  - En función de los valores de los parámetros de la petición redirige a una segunda página JSP (si es un usuario y una clave determinadas) o bien recarga la página inicial (incluyéndola)
  - Mezcla código Java puro con acciones estándard

Ejemplo de uso de la acción forward

```
<% if
  ((request.getParameter(“userName”).equa
  ls(“Ricardo”)) &&
  (request.getParameter(“password”).equal
  s(“xyzzy”))) {
%>
   <jsp:forward page=“saludoforward.jsp”
  />
<% } else {  %>
<%@ include file=“forward.html”%>

<% } %>
```


Ejemplo de uso de la acción forward

El programa *saludoforward.jsp* podría ser el siguiente:

```
<HTML>
  <head>
    <title> Saludo al cliente </title>
  </head>
<body>
  <h1> Saludo al cliente</h1>
<%
  out.println(“Bienvenido a la nueva
  aplicación");
%>
</body> </HTML>
```

Tipos de acciones existentes

- <jsp:include>
- <jsp:forward>

- <jsp:param>

- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>

- <jsp:plugin>

Acción jsp:useBean

- Esta marca sirve para instanciar un JavaBean si no existe, o localizar una instancia ya existente, para su uso desde la página.
- Los JavaBeans son objetos Java que cumplen ciertas características en cuanto a su diseño
- Se utilizan para reducir al máximo el código Java insertado en una página JSP. En lugar de meterlo directamente en el fichero JSP se mete en un objeto y éste se llama desde el JSP
- Permite separar la lógica de ejecución (en el JavaBean) de la presentación (en el servlet generado)
  - Se encapsula el código Java en un objeto (JavaBean) y se instancia y usa con el JSP.

Acción jsp:useBean

- Los JavaBeans se caracterizan porque a sus atributos (llamados propiedades) se acceden (por convenio) a través de los métodos setNombreAtributo y getNombreAtributo
  - Ojo, si el nombre va en minúsculas el método lleva la inicial del nombre en mayúsculas para "nombre" se pone "getNombre".
- Si se usa un JavaBean en una página habrá que definir la clase correspondiente, creando los métodos set y get para los atributos definidos
- Dentro del servlet generado se puede llamar a métodos de un JavaBean que se encarguen de realizar ciertas operaciones y el servlet muestra el resultado de las mismas
- Ventaja del traslado de la lógica a un JavaBean
  - **Separación de interfaz de la implementación**

Acción jsp:useBean

Sintaxis

<jsp:useBean id="nombre" scope="nombreámbito" ***detalles*** **/>**

Características de los atributos de esta acción:

- En id se define el nombre asignado al JavaBean (identificador asociado)
- El ámbito se refiere a dónde puede referenciarse el Javabean. Permite compartir objetos en una sesión
  - "page","request","session" y "application"
- Los detalles pueden ser:
  - class="Nombre de la clase del JavaBean" (es lo que más se usa)
  - otros

Acción jsp:setPropertiy

- Esta marca se utiliza junto con la marca useBean para asignar valor a las propiedades del Bean
- En el método _jspService() del servlet generado se invoca al método set de la propiedad deseada.

Acción jsp:setPropertiy

- Sintaxis

  <jsp:setProperty name="identificador del Bean" detalles de la propiedad />

- Donde los detalles pueden ser
  - property="*" (se cogen como propiedades y valores todos los parámetros del objeto request)
  - property="Nombre" (se coge un parámetro con el mismo nombre del objeto request)
  - property="Nombre" param="NombreParámetro" (si se desean nombres distintos)
  - property="Nombre" value="valor parámetro" (se asignan propiedades arbitrarias con valores concretos)

Acción jsp:getPropertiy

- Se utiliza para obtener el valor de las propiedades de un Bean.
- Dentro del método _jspService() del servlet generado se accede al valor de una propiedad, **lo convierte a string y lo imprime en la salida del cliente (objeto out)**.

Sintaxis

<jsp:getProperty> name=“nombre del Bean” property=“Nombre de la propiedad” />

Ejemplo de uso de JavaBean (Formulario cliente)

(fichero *beans.html*)

```
<HTML>  <head>
 <title> Página de prueba del uso de
  beans </title> </head>  <body>
  <h1> Página de prueba del uso de beans
  </h1>
 <form method="post" action="beans.jsp" >
  Se envía el formulario al servicio cuyo
  fichero es “beans.jsp”
 <p> Por favor, introduce tu nombre:
```

Ejemplo de uso de JavaBean (Formulario cliente)

```
<input type="text" name="nombre">
          <br> ¿Cuál es tu lenguaje de
  programación favorito?
          <select name="lenguaje">
            <option value="Java"> Java
            <option value="C++"> C++
            <option value="Perl"> Perl
          </select>
          </p>
          <p> <input type="submit"
  value="enviar información">
        </form>  </body> </HTML>
```


Ejemplo de uso de JavaBean (Fichero jsp)

- Fichero *beans.jsp* que usa un Bean para elaborar los resultados y los muestra

```
<jsp:useBean id="lenguajeBean"
  scope="page" class="LenguajeBean">
```

usa un Bean generado a partir de la clase denominada "LenguajeBean" con ámbito de página

```
 <jsp:setProperty name="lenguajeBean"
  property="*"/>
```

Las propiedades del Bean las toma del objeto petición

```
</jsp:useBean>
<HTML> <head>
<title> Resultado de prueba del uso de
  beans </title> </head>
```

Ejemplo de uso de JavaBean (Fichero jsp)

```
<body> <h1> Resultado de prueba del uso
  de beans </h1>
<p> Hola
<jsp:getProperty name="lenguajeBean"
  property="nombre" />. </p>
```

coge el valor de la propiedad indicada y lo imprime para lo cual se ejecuta un método del Bean con el nombre de la propiedad

```
<p> Tu lenguaje favorito es
<jsp:getProperty name="lenguajeBean"
  property="lenguaje" />. </p>
<p> Mis comentarios acerca del lenguaje
<p> <jsp:getProperty name="lenguajeBean"
  property="comentariosLenguaje" />. </p>
        </body> </HTML>
```


Ej. de uso de JavaBean (Definición de la clase)

Clase LenguajeBean (*LenguajeBean.java*) elabora la respuesta

```
public class LenguajeBean {
  private String nombre;
  private String lenguaje;
  public LenguajeBean() {}
  public void setNombre(String nombre)  {
```

Coloca el valor a la propiedad "Nombre"

```
    this.nombre=nombre;  }
  public String getNombre() {
```

Consigue el valor de la propiedad "Nombre"

```
    return nombre;  }
```

Ej. de uso de JavaBean (Definición de la clase)

```
public void setLenguaje(String lenguaje)
  {    Coloca el valor a la propiedad "lenguaje"
    this.lenguaje=lenguaje;
  }
public String getLenguaje() {
Consigue el valor de la propiedad "Lenguaje"
return lenguaje; }
public String getcomentariosLenguaje ()
  {Consigue el valor de la propiedad "comentariosLenguaje"
if (lenguaje.equals("Java")) {
      return "El rey de los lenguajes
  Orientados a objetos";   }
```


Ej. de uso de JavaBean (Definición de la clase)

```
else if (lenguaje.equals("C++")) {
      return "Demasiado complejo";
    }
else if (lenguaje.equals("Perl")) {
      return "OK si te gusta el código
  incomprensible";  }
    else {
        return "Lo siento, no conozco el
  lenguaje " + lenguaje ;
    }
  }
}
```

Puesta en marcha de la aplicación ejemplo

Acciones a realizar

- Salvar *beans.html* y *beans.jsp* en el directorio de trabajo personal *(webapps\JSP\lenguaje)*
- Compilar el bean en el directorio *clases* de WEB-INF *(webapps\JSP\WEB-INF\classes)*

  javac ..\WEB-INF\classes\lenguajeLenguajeBean.java
- Desde el visualizador, apuntar a

  http://localhost:8080/JSP/lenguaje/beans.html

Ejercicios

1. Ejecutar el ejemplo del Bean que evalúa el lenguaje preferido del cliente
2. Crear un Bean que funcione como sumadora.
3. Usar un Bean para crear un euroconversor
4. Ampliar la sumadora a calculadora con más operaciones

- Modifica la página para incluir también el fichero "incluido.jsp" como directiva
- Crear un fichero HTML estático y añádelo también como acción y como directiva.
- Ejecutar la página resultante y comprueba resultados
- Cambiar la página HTML o el JSP y vuelve a ejecutar comprobando resultados.

Resumen

- **Código HTML**
- **Directivas <%@**
  - page **<%@ page ATRIBUTOS %>**
  - include **<%@ include file="Nombre del fichero" %>**
  - taglib **<%@ taglib uri ="taglibraryURI" prefix="tagPrefix" %>**
- **Elementos de Scripting** (guiones) <%
  - **Declaraciones** **<% ! Declaración %>**
  - **Código Java arbitrario (scriptlets)** **<% código Java %>**
  - **Expresiones** **<%= Expresión Java a evaluar %>**
- **Acciones** estándar *(formato XML) <jsp:*
  - <jsp:useBean> <jsp:setProperty> <jsp:getProperty>
  - <jsp:include> <jsp:param>
  - <jsp:forward> <jsp:param>
  - <jsp:plugin>
- Acciones personalizadas *(formato XML) <etiq:*

Etiquetas personalizadas

Las acciones personalizadas están definidas por el programador de la aplicación web mediante el uso de etiquetas personalizadas.

Una etiqueta personalizada permite ocultar bajo ella un conjunto de acciones (definidas con instrucciones java) evitando que las mismas se incluyan en el fichero JSP.

Así pues, para incluir lógica de programa en una aplicación web, es posible realizarlo de tres modos

- Incluyendo el correspondiente código Java en una página JSP
- Incluyéndolo en un Java Bean que se llama desde la página JSP
- Incluyéndolo en una etiqueta personalizada

Características

Ventajas que proporcionan

- Permiten reutilizar código
  - Usando bibliotecas de etiquetas con las funcionalidades más extendidas
- Permiten separar las funciones del diseñador web (que usa HTML y XML) de las del programador web (que usa Java)
  - Permiten invocar funcionalidad sin necesidad de introducir código Java en la página JSP
- Son más potentes que los JavaBeans

(página siguiente)

Características

- Son más potentes que los JavaBeans
  - El uso exclusivo de Beans no es suficiente para evitar todo tipo de código Java en una página JSP
  - Las marcas propias pueden lograr directamente lo que los beans pueden sólo lograr en conjunción con los scriptlets

Ej: Una etiqueta que compruebe que un usuario existe (u otra condición) y se redirija a una página de éxito o de error en función de la comprobación.

- El uso exclusivo de Beans no es suficiente.
  - Podríamos usar un bean para comprobar que el usuario existe.
  - El hecho de redirigir a una página o a otra en función de si existe o no debe hacerse en una página JSP (puesto que desde un bean no se puede reenviar a una página JSP) y debe utilizarse código Java para realizar la comparación (if…)

Ejemplo de uso

- Si se desea solicitar una lista de libros de una base de datos con JSP podría hacerse utilizando un Bean que devuelva la lista de libros.

```
<jsp:useBean id="biblio" class="Libros" />
<% ResultSet res=getLibros("Libros");%)
```

- Posteriomente se recorre la lista y se muestra el atributo "título de cada libro". Esto no se puede hacer en el Bean porque en él no se puede acceder al objeto out para devolver el resultado al cliente, por lo que hay que incluirlo en el fichero JSP

```
<% while (res.next()){%>
  <%=res.getString("titulo") %>
<% }%>
```


Ejemplo de uso

- La inclusión de las acciones personalizadas permite ocultar todo el código Java al diseñador web, permitiendo que éste utilice etiquetas como si de cualquier otra etiqueta HTML se tratase

```
<%@ taglib uri="/bilbliolib" prefix="etiq" %>
<etiq:getLibro id="libros" />
<ul>
<etiq:bucle name="libros" bucledId="biblio" >
  <li> <jsp:getProperty name="libros" property="titulo" />
</etiq:bucle>
```

Ventajas que aportan

- Reducen o eliminan código Java en las páginas JSP
- Sintaxis muy simple, como código HTML
- Pueden ser personalizadas a través de atributos y determinadas de forma estática o dinámica
- Tienen acceso a todos los objetos implícitos de las páginas JSP
- Se pueden anidar, consiguiendo interacciones complejas
- Facilitan la reutilización de código porque pueden ser usadas en cualquier aplicación web

Definición

1. Definición de la estructura en un fichero de **definición de etiquetas** (biblioteca de etiquetas)
   - Fichero con extensión .tld "Tag Library Descriptor (TLD)" es un fichero XML que describe la biblioteca
   - Los .tld se guardan en un directorio TLDS dentro de Web-inf de la aplicación
   - La biblioteca (.tld) debe estar a su vez definida en el fichero web.xml, que existirá en el directorio Web-inf de la aplicación
2. Definición de la funcionalidad asociada a la etiqueta (lo que queremos que haga cuando se use la etiqueta) . Se define a través de un **controlador de la etiqueta** (clase que implementa la etiqueta)
   - Es un JavaBean, con propiedades que coinciden con los atributos XML de la etiqueta.

Uso en una página JSP

- Cuando se desee usar la etiqueta, se incluye en el fichero JSP la directiva JSP *taglib* para importar las etiquetas

<%@ taglib uri=”taglibraryURI” prefix=”tagPrefix” %>

Ejemplo de uso en un fichero JSP

- Caso simple, sin atributos ni cuerpo, que saca código HTML y contenido dinámico

```
<%@ taglib uri=“/hola” prefix=“ejemplos” %>
// La Biblioteca /hola contiene la definición de la etiqueta
<html>
  <head>
    <title> Primer ejemplo de uso de etiquetas personalizadas
    </title>
  </head>
  <body>
     Esto es una salida estática
   <p />
   <i>
     <ejemplos:hola> </ejemplos:hola>
     // usa una etiqueta personalizada sin atributos ni cuerpo
   </i>
   Esto es una salida estática de nuevo
  </body> </html>
```

Partes de un fichero TLD

- Descriptor de la biblioteca TLD *Tag Library Descriptor*
  - Información acerca de la biblioteca (cabecera)
  - Definición de la estructura de cada etiqueta

Definición de la estructura de una etiqueta

- En la definición de una etiqueta personalizada debe aparecer al menos:

`<tag>`
  - *Indica comienzo de definición*

`<name> Hola </name>`
  - *Nombre de la etiqueta*

`<tagclass> tagPaquete.HolaTag </tagclass>`
  - *Nombre de la clase que implementa la funcionalidad*

`</tag>`
  - *Fin de la etiqueta*

- Además puede llevar

  atributos anidaciones cuerpo

Ejemplo de definición de estructura de una etiqueta

TLD *Tag Library Descriptor*

Información acerca de la biblioteca (cabecera)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib >
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname> ejemplos </shortname>
  <info> Biblioteca de ejemplos sencillos </info>
```

Información de cada etiqueta

```
<tag>
    <name> Hola </name>
    <tagclass> tagPaquete.HolaTag </tagclass>
    <info> Ejemplo simple </info>
</tag>
</taglib>
```


Definición de la funcionalidad de una etiqueta

- **La funcionalidad de una etiqueta está definida en una clase controladora (clase Java)**
- **Permite instanciar un objeto que será invocado por el servlet generado para la página JSP que usa la etiqueta**
- Constituye un **javaBean** que hereda de **TagSupport** con dos métodos básicos:
  - doStartTag() llamado cuando se abre la etiqueta
  - doEndTag() llamado cuando se cierra
  - Si la etiqueta tiene atributos hay que definir los métodos set y get para los mismos
  - Otros métodos son doInitBody, doAfterBody(), si tiene cuerpo, etc.
- Desde el servlet para la página se invoca a estos métodos

Ejemplo de definición de la clase controladora

- Para el ejemplo anterior <ejemplos:hola>

```
package tagPaquete;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class HolaTag extends TagSupport {
public int doStartTag() throws JspTagException {
// este método será llamado cuando el motor JSP
  encuentre el
// comienzo de una marca implementada por esta
  clase
return EVAL_BODY_INCLUDE;
}
```


Ejemplo de definición de la clase controladora

```
public int doEndTag() throws JspTagException {
// este método será llamado cuando el motor
  encuentre el final
// de una marca implementada por esta clase
String dateString = new Date().toString();
try  {
  pageContext.getOut().write("Hola mundo.<br/>");
  pageContext.getOut().write("Mi nombre es" +
  getClass().getName() + "y son las " +
  dateString + "<p/>");
 }  catch (IOException ex) {
    throw new JspTagException
        ("Error fatal: la marca hola no puede
  escribir en la salida del JSP");
 }
  return EVAL_PAGE;
}
}  // clase HolaTag
```

Otro ejemplo de definición en la tld

```
<tag>
    <name>hola</name>
    <tagclass>tagPaquete2.HolaTag</tagclass>
    <bodycontent> empty</bodycontent>
    <info>Esta es una etiqueta muy simple de saludo
    </info>
    <atribute>
        <name>nombre</name>
        <required>false</required>
        <rtextvalue>false</rtexpvalue>
    </atribute>
</tag>
</taglib>
```


Otro ejemplo de definición de la clase controladora

```
/**
 * Esta es una etiqueta muy simple que se utiliza solamente para  mostrar
 * la creación de los métodos implicados en el control de las etiquetas.
 * La etiqueta lo único que hace es añadir el saludo al canal de salida
 * hacia el cliente, sustituyendo a la etiqueta que se coloca en la
 * pagina JSP.
 */
package tagPaquete2;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HolaTag extends TagSupport {
  // Atributo de la etiqueta definido en el fichero TLD
  private String nombre = null;
```

Otro ejemplo de definición de la clase controladora

```
// Constructor sin argumentos, porque se la clase controladora de
// una etiqueta sigue el modelo JavaBean.
// Si no se utilizan constructores con argumentos, la declaración
// de este constructor sin argumentos no es necesaria
public void HolaTag() {}

// Métodos set() y get() correspondientes al atributo "nombre" que
// se ha definido en el fichero TLD para esta etiqueta
public void setNombre( String _nombre ) {
 nombre = _nombre;
 }
public String getNombre() {
 return( nombre );
 }
```


Otro ejemplo de definición de la clase controladora

```
// Método que se invoca cuando se encuentra la marca que indica la
// presencia de la etiqueta en la página JSP
public int doStartTag() throws JspTagException {
 try {
  JspWriter out = pageContext.getOut();
  out.println( "<table border=1>" );
   if( nombre != null ) {
    out.println( "<tr><td> Hola " +nombre+ " </td></tr>" );
    }
   else {  out.println( "<tr><td>Hola Mundo JSP</td></tr>" );       }
 } catch( Exception e ) {  throw new JspTagException( e.getMessage() );
 }
 return( SKIP_BODY );    }
```

Otro ejemplo de definición de la clase controladora

```
// Método que se invoca cuando se encuentra la marca de cierre de
  // la etiqueta
  public int doEndTag() throws JspTagException {
    try {
      // Se utiliza el pageContext para obtener el objeto de salida
      // sobre el que colocar la etiqueta HTML de cierre de la tabla
      pageContext.getOut().print( "</table>" );
    } catch( Exception e ) {
      throw new JspTagException( e.getMessage() );
      }
return( SKIP_BODY );
    }
```


Otro ejemplo de definición de la clase controladora

```
  public void release() {
    // Llama al método release() del padre, para que se devuelvan
    // todos los recursos utilizados al sistema. Esta es una buena
    // práctica, sobre todo cuando se están utilizando jararquías
    // de etiquetas
    super.release();
    }
  }
```

Otro ejemplo de uso de la etiqueta

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN">
<%-- etiqhola1.jsp
  Ejemplo para mostrar el uso de la etiqueta HolaTag, que se encarga
  de sustituir las apariciones de "hola" en este fichero por el
  correspondiente saludo generado desde la clase controladora de la
  librería
--%>
<%-- referenciamos la librería de etiquetas del capítulo --%>
<%@ taglib uri="/WEB-INF/tlds/ejemplo2.tld" prefix=”etiq" %>
<html>
<head>
<title>Ejemplo Etiquetas, Etiqueta HolaTag</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
    1">
```


Otro ejemplo de uso de la etiqueta

```
<html>
<head></head>
<body>
<h2>Ejemplo de biblioteca de etiquetas: <i>HolaTag</i></h2>
En esta página se muestra el uso de esta etiqueta, que se invoca en
primer lugar especificando un valor para el atributo <i>nombre</i>
y luego sin ningún argumento.
<p>
<hr><center>
  <etiq:hola nombre="Agustín"/>
</center><hr>
  <etiq:hola />
</body>
</html>
```

Ciclo de vida de las etiquetas

- La clase que implementa la etiqueta se crea cuando el motor JSP encuentra la etiqueta durante la traducción de la página JSP al servlet y se fija el contexto de visibilidad
- Se invocan sus métodos set de cada uno de sus atributos
- Se invoca el método doStartTag(). En este método se incluyen las inicializaciones. Al final del método se incluye
  - SKIP_BODY
  - EVAL_BODY_INCLUDE
  - EVAL_BODY_BUFFERED
- Se invoca el método setBodyContent()
  - Cualquier salida de la etiqueta se deriva al objeto BodyContent

Ciclo de vida de las etiquetas

- Si se invoca el método doInitBody() permite realizar acciones antes de ser evaluado el cuerpo. La salida se guarda en el buffer BodyContent
- Se invoca el método doAfterBody(). Después de que el cuerpo haya sido evaluado. Se suelen realizar acciones basadas en la evaluación del cuerpo de la etiqueta. Al final del método se puede determinar por dónde seguirá el ciclo de vida, devolviendo una de las siguientes constantes:
  - EVAL_BODY_AGAIN
  - SKIP_BODY
- Se invoca el método doEndTag()
- Se invoca el método release()

Ejercicios

- Ejecutar los ficheros
  - ejtag1_hola.jsp (Etiqueta sin atributos ni cuerpo)
  - ejtag2_hola.jsp (Etiqueta con atributos)
  - ejtag3_hola.jsp (etiqueta con atributos y cuerpo)
- Comprobar el contenido del fichero web.xml
- Comprobar el contenido del fichero etiquetas.tld
- Hacer una etiqueta denominada suma, que efectúe la suma de los dos números pasados como parámetros

Inclusión de la funcionalidad en una aplicación web

Así pues, la funcionalidad de una aplicación puede ser itegrada de tres modos:

- Como código Java dentro de las páginas JSP
  - No separa la interfaz de la implementación
- Con el uso de JavaBeans llamados desde las páginas JSP
  - Separa la interfaz de la implementación en gran medida
- Con el uso de etiquetas personalizadas
  - Evitan la necesidad de inclusión de cualquier código Java en la página JSP

- Un buen diseño pasa por evitar en la medida de lo posible la primera de las tres opciones

Mantenimiento de sesiones

- Dado que el JSP se convierte en un servlet, todo lo visto en servlets es válido con JSP.
- Uso de Interfaz HttpSession para la gestión de sesiones
- El objeto "session" se crea automáticamente en JSP (puede suprimirse con page)
- El objeto "session" es un diccionario al que se pueden asociar objetos (atributos) y darles un nombre (también eliminar)
  - `session.setAttribute("mipaquete.sesion.nombre", miObjeto);`
- Se puede recuperar la información desde otras peticiones posteriores de la misma sesión
  - `Clase unObjeto = session.getAttribute("mipaquete.sesion.nombre);`
- También se puede eliminar un objeto asociado
  - `session.removeAttribute("mipaquete.sesion.nombre");`

Proceso de autentificación de usuarios

- Todo lo visto en servlets es aplicable en JSP
- Mecanismos
  - Declarativo. Usa un mecanismo proporcionado por el motor de servlets
    - Indicar restricciones de seguridad
    - Indicar el modo de realizar la autentificación
    - Indicar los usuarios definidos en el sistema
  - Por programa
    - Generar formulario html para nombre y clave
    - Generar página JSP que accede a Base de Datos, comprueba clave y se fijan los parámetros de seguridad en el JavaBean, cuyo ámbito se restringe a la sesión que acaba de establecer el usuario

XML y JSP trabajando juntos

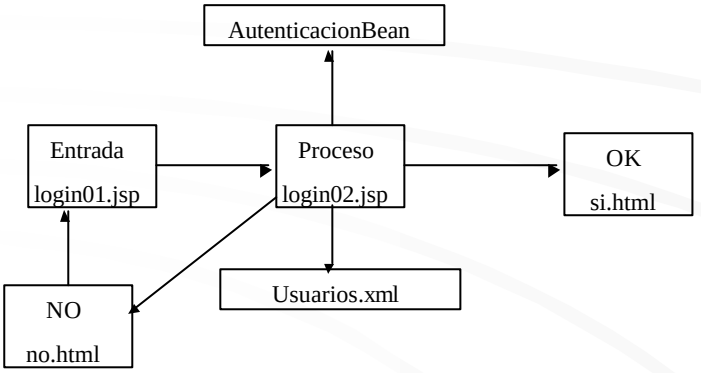
- Para utilizar un documento XML en una aplicación es preciso analizarlo
- Existen dos modelos de analizadores aceptados:
  - DOM (Modelo de objetos de documento)
  - SAX (API simple para XML)
- Analizador de XML tipo DOM
  - El analizador crea una estructura de datos en memoria en forma de árbol, donde se coloca todos los elementos del documento
  - Ofrece métodos para acceso a los nodos y a la información de los mismos a través de una API
- Analizador SAX
  - En lugar de crear un árbol, va leyendo el archivo XML y ejecuta acciones según las etiquetas encontradas.

XML y JSP trabajando juntos

- Si se desea usar un documento XML en una aplicación JSP habrá que
  - Incluir el código Java para la manipulación del documento XML
  - o bien incluir un JavaBean que lo manipule
  - o bien definir una etiqueta personalizada que envuelva la operación dentro de su clase
- En cualquier caso se trata de manejar XML con Java

Ej. de análisis XML con DOM en una página JSP

- Veamos cómo realizar la autenticación de usuarios en base a un fichero XML con los datos de autenticación



Ej. de análisis XML con DOM en una página JSP

Supongamos el siguiente contenido de la página xml de usuarios autorizados

```
<?xml version=`1.0` encoding=`iso-8859`?>
<usuariosAutorizados>
  <usuario nombre="Agustin" pwd="agustin"
  nivel="administrador">
  </usuario>
  <usuario nombre="Invitado" pwd="invitado"
  nivel="usuario">
  </usuario>
</usuariosAutorizados>
```

Ej. de análisis XML con DOM en una página JSP

Formulario de entrada (login01.jsp)

```
<form method="POST" accion="login02.jsp"
  name="autenticacion">
  <table border="0" cellpadding="0" cellspacing="0"
  width="200">
  <tr>
  <td width="50%">Nombre</td>
  <td width="50"><input type="text" name="nombre"
  size="16"></td>
  </tr>
  <tr>
  <td width="50%">Contraseña</td>
  <td width="50%"><input type="password" name="pwd"
  size="16"></td>
  </tr>
  <tr>
  <td width="50%"> colspan="2" align="center">
      <input type="submit" value="Entrar"
  size="16"></td>
  </tr> </table> </form>
```


Ej. de análisis XML con DOM en una página JSP

Autenticación (login02.jsp)

- Importación de los paquetes para el control del fichero XML

```
<%@ page import="javax.xml.parsers.*" %>
```

- Definición de las varibles para información de entorno

```
<%! String usuario=""; %>
<%! String pwd=""; %>
<%! String redireccionURL =""; %>
```

- Instanciación del JavaBean responsable de mantener información del usuario mientras dure la sesión

```
<jsp:useBean id="login" scope="session"
  class="login.Autenticacion"/>
```

Ej. de análisis XML con DOM en una página JSP

- Recuperación de los datos de identificación del usuario enviados desde el formulario HTML de la página previa

```
if (request.getParameter("nombre")!= null)
   {usuario=request.getParameter("nombre");  }
if (request.getParameter("pwd")!=null)
   { pwd= request.getParameter("pwd"); }
```

- Inicialización de variables para el procesamiento del documento XML

```
Document documento;
DocumentBuilderFactory.newInstance();
redireccionURL="no.html";
```


Ej. de análisis XML con DOM en una página JSP

- Generación del conjunto de nodos siguiendo el modelo DOM

```
// Apertura del archivo
URL url=new URL(archivoXml);
InputStream datosXML=url.openStream();
// Construcción del documento XML
DocumentBuilder
   builder=factory.newDocumentBuilder();
documento=builder.parse(datosXML);
//Generación de lista de nodos en base a la clave "usuario"
NodeList
   listaNodos=documento.getElementsByTagName("usuario
   ");
```

Ej. de análisis XML con DOM en una página JSP

- Comparación de los atributos introducidos con los existentes en el árbol

```
for (int i=0; i<listaNodos.getLength();i++) {
   Node actNodo=listaNodos.item(i);
// para este nodo recogemos el valor del "nombre" y contraseña
Element actElemento=(Element)listaNodos.item(i);
String actUsuario=actElemento.getAtribute("nombre");
String actPwd=actElemento.getAtribute("pwd");
// si el usuario es correcto le dejamos que siga
if (actUsuario.equals(usuario)&& actPwd.equals(pwd)){
   redireccionURL="si.html";
```


Ej. de análisis XML con DOM en una página JSP

```
//Actualizamos el bean para indicar que el usuario ya está autorizado
logon.setNombre(usuario);
logon.setAutorizado();
break bucle();
     }
}
```

- Código JavaScript que permite el reenvío de una página a otra, de forma que se pueda presentar al visitante la página adecuada dependiendo de que haya sido posible la autenticación o no

```
<script language="javascript">
   setTimeout("document.location=`<%= redireccionURL
   %>`",100)
</script>
```

Ej. de análisis XML con DOM en una página JSP

- Control de la sesión
  - El JavaBean Autenticacion.java lleva a cabo el control
  - Es posible comprobar en cada página web si el usuario está autorizado llamando al JavaBean
  - Los métodos set de las propiedades nombre y clave se invocan desde las páginas de control de la lógica de aplicación (login02.jsp) una vez que el usuario ha sido validado
  - Los métodos get devuelven objetos String con la identificación y la autorización del usuario

Ej. de análisis XML con DOM en una página JSP

```
package login;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Autenticacion {
  // Propiedad "nombre" del bean, que corresponde
  al nombre
  // del usuario para el que se implementa el
  bean
  String nombre = "";
  // Variable de clase que indica si el usuario
  para el
  // que se crea el bean en la sesión que ha
  establecido
  // con el servidor está identificado ante el
  sistema y se
  // le concede autorización
  boolean autorizado = false;
```

Ej. de análisis XML con DOM en una página JSP

```
// Métodos get() y set() de la propiedad
 "nombre"
 public String getNombre() {
   return( nombre );
   }
 public void setNombre( String _nombre ) {
   nombre = _nombre;
   }
 // Métodos get() y set() de la variable de
 clase que controla
 // la autenticación del usuario ante el sistema
 public void setAutorizado() {
   autorizado = true;
   }
 public boolean getAutorizado() {
   return( autorizado );   }  }
```