MANUAL DE JSP

Qué es JSP

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.

> **Bibliografía:** Esta descripción de JSP está extraída de un PDF en inglés muy completo que introduce la tecnología, que se puede encontrar el la página corporativa de Java de Sun Microsystems, en su sección de aprendizaje online. A su vez, dicho manual proviene del portal Java **jGuru**.
>
> jGuru: Introduction to JavaServer Pages technology

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.

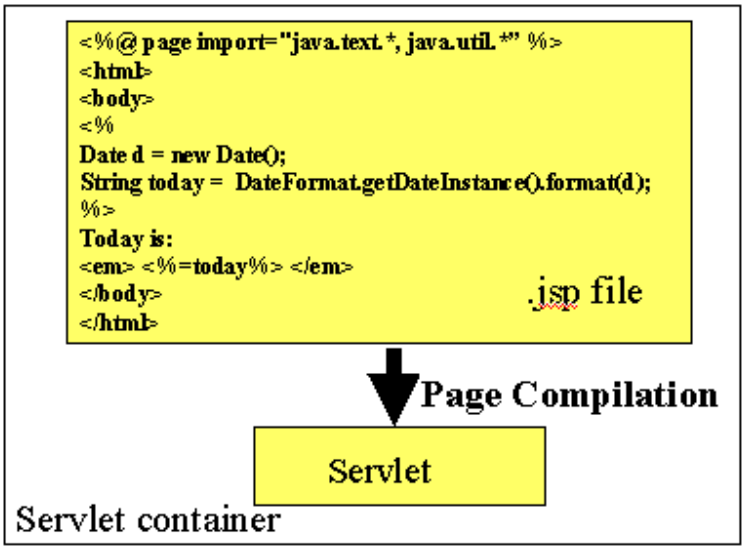
Motor JSP

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.

En JSP creamos páginas de manera parecida a como se crean en ASP o PHP -otras dos tecnologías de servidor-. Generamos archivos con extensión .jsp que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un servlet, implementado en un archivo class (Byte codes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página .jsp, aunque existe la opción de precompilar en código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

Ejemplo de página JSP

En la imagen siguiente se puede ver un ejemplo extremadamente simple de una página JSP y el esquema de conversión de esa página en un servlet.



Prerequisitos

Para aprender JSP, aparte de conocer HTML, será necesario comprender y tener algo de experiencia en la programación en Java, que es un lenguaje de programación Orientado a Objetos por completo. Una vez conocida la programación en Java se puede estudiar por encima el sistema de Servlets, lo que nos dará una mejor idea del funcionamiento interno del motor JSP.

Para aprender Java podemos consultar algunos enlaces del correspondiente directorio de nuestro buscador de enlaces.

Además, necesitaremos descargar e instalar Tomcat, el contenedor de servlets usado en la referencia oficial de implementación de JSP. Podemos acceder a un ejercicio para aprender a realizar esta instalación, disponible también en la referencia de aprendizaje de la página de Java.

Referencias JSP

Hemos creado una nueva sección en nuestro directorio dedicada por completo a las páginas JSP, que será muy interesante para todo aquel que desee profundizar en el tema.

Comparando JSP con ASP

JSP y ASP sirven para hacer, más o menos, el mismo tipo de aplicaciones web. Sin embargo, en el fondo tienen bastantes diferencias. Después de mi experiencia en el trabajo con JSP, un día un cliente me preguntó por qué no programaba la página en ASP en lugar de JSP, ya que había oido hablar que el sistema de Microsoft tenía unas características muy apropiadas para su modelo de negocio. A partir de esta sugerencia, y para que mi cliente quedase satisfecho con la tecnología JSP -que es la que prefiero utilizar-, preparé una lista de ventajas de utilizar páginas dinámicas Java frente a las de Microsoft.

Plataforma e independencia del servidor

JSP sigue la filosofía de la arquitectura JAVA de "escribe una vez ejecuta donde quieras". La implantación de ASP está limitada para arquitecturas basadas en tecnología Microsoft.

Así, JSP se puede ejecutar en los sistemas operativos y servidores web más populares, como por ejemplo Apache, Netscape o Microsoft IIS. Mientras que ASP sólo tiene soporte nativo para los servidores IIS y Personal Web Server, que son los dos servidores web para sistemas Microsoft, el primero con tecnología NT y el segundo para sistemas Windows 98 y similares.

Proceso de desarrollo abierto (open source)

El API JSP se beneficia de la extendida comunidad JAVA existente, por el contrario la tecnología ASP es específica de Microsoft que desarrolla sus procesos internamente.

TAGS

Mientras que tanto JSP como ASP usan una combinación de tags y scripts para crear paginas web dinámicas, la tecnología JSP permite a los desarrolladores crear nuevos tags. Así los desarrolladores pueden crear nuevos tags y no depender tanto de los scripts.

Reusabilidad entre plataformas.

Los componentes JSP son reusables en distintas plataformas (UNIX, Windows).

La ventaja Java

La tecnología JSP usa Java como lenguaje de Script mientras que ASP usa VBScript o Jscript. Java es un lenguaje mas potente y escalable que los lenguajes de Script. Las páginas JSP son compilados en Servlets por lo que actúan como una puerta a todos los servicios Java de Servidor y librerías Java para aplicaciones http. Java hace el trabajo del desarrollador más fácil p. e. ayuda a proteger el sistema contra las "caídas" mientras que las aplicaciones ASP sobre sistemas NT son más susceptibles a sufrirlas, también ayuda en el manejo de la memoria protegiendo contra fallos de memoria y el duro trabajo de buscar los fallos de perdida de punteros de memoria que pueden hacer mas lento el funcionamiento de una aplicación.

Mantenimiento

Las aplicaciones que usan JSP tiene un mantenimiento más fácil que las que usan ASP.

- Los lenguajes de Script están bien para pequeñas aplicaciones, pero no encajan bien para aplicaciones grandes. Java es un lenguaje estructurado y es más fácil de construir y mantenimientos grandes como aplicaciones modulares.
- La tecnología JSP hace mayor énfasis en los componentes que en los Scripts, esto hace que sea más fácil revisar el contenido sin que afecte a la lógica o revisar la lógica sin cambiar el contenido.
- La arquitectura EJB encapsula la lógica de p. e.: acceso a BD, seguridad, integridad transaccional y aislamiento de la aplicación.

- Debido a que la tecnología JSP es abierta y multiplataforma, los servidores web, plataformas y otros componentes pueden ser fácilmente actualizados o cambiados sin que afecte a las aplicaciones basadas en la tecnología JSP.

Conclusión

Las ventajas sobre utilizar la tecnología Java con respecto a la propietaria de Microsoft (ASP) son, como se ha podido ver, diversas e interesantes. Sin embargo, podemos apuntar una ventaja de la programación en ASP, pues resulta bastante más fácil de aprender que JSP, por lo menos si no se tiene una experiencia previa en programación. Esto es debido a que Java es un lenguaje muy potente, pero un poco más complicado de usar porque es orientado a objetos y la manera de escribir los programas es más rígida.

> **Referencias:** Para conocer más sobre ASP y JSP puedes acceder a las correspondientes categorías de enlaces de nuestro buscador.
>
> Directorio de enlaces ASP
> Directorio de enlaces JSP
>
> Además, para ASP tenemos una sección dedicada exclusivamente: ASP a Fondo

Conexión a un database server con JSP

En este artículo trataremos el tema de la conexión a un database desde una pagina Jsp, para esto, crearemos una clase (ConnectionCreator) , que será usada por nuestra página. En primer lugar nos conectaremos a un SQL Server, luego podremos ver como modificando un poco la clase tambien se puede usar para MySql.

Comenzamos entonces por escribir el código de la primiera versión de la clase, para luego esplicarla en detalle:

```
1   package notas;
2
3   import java.sql.DriverManager;
4   import java.sql.Connection;
5
6   public class ConnectionCreator {
7
8       public static java.sql.Connection getSqlServerConnection(String database,
9               String servername, int port, String username, String password ) {
10          try {
11              Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
12              String url ="jdbc:microsoft:sqlserver://" + servername + ":" + port +
13                          ";DatabaseName=" + database + ";user=" + username +
14                          ";password=" + password;
15              Connection conn = DriverManager.getConnection(url);
16              if (conn != null)
17                  System.out.println("---> CONNECTED TO SERVER : "+servername);
18              else
19                  System.out.println("---> UNABLE TO CONNECT TO SERVER : "+servername);
20
21              return conn;
22          }
23          catch(Exception e) {
24              System.out.println("ERROR = "+e);
25              return null;
26          }
```

Linea 1: definición del package que contendra la clase, necessario para luego importar la clase dentro la pagina.

Lineas 3 y 4: Importamos dos clases standard de Tomcat.

Linea 6 : Definición de la clase, como pública, con el nombre ConnectionCreator.

Linea 8: Definición del único método de la clase, getSqlServerConnection, que recibirá como parametros una lista de valores, para devolver luego, un objeto java.sql.Connection.

Los parametros que recibe la clase son: database (nombre del database), servername (nombre o ip del database server), port (puerta de conexión, en el caso de SQL generalmente es 1433), username y password.

Linea 10: try, es el comienzo del bloque de código que ejecutaremos

Linea 11: Se instancia el driver de SQL, que obviamente debe estar presente ya sea en las librerías de Tomcat o en las de nuestro sitio(carpeta \WEB-INF\lib). Estos drivers se descargan gratuitamente de Internet y vienen empaquetados bajo la forma de archivos JAR.

Lineas 12 a 14: Construimos una cadena de caracteres llamada url, que será utilizada luego para abrir la conexión. La estructura de este texto es estandard, solo hay que intercalar los parametros recibidos.

Linea 15: Finalmente llegamos al corazón de la clase, instanciamos un objeto Connection y, a través del DriverManager, tratamos de conectarnos al servidor, pasando como parametro la url construida anteriormente.

Lineas 16 a 19: No son extrictamente necesarias, pero sirven como ayuda al programador. Controlamos que la conexión haya sido exitosa y que esté abierta (por defecto siempre lo está) y mandamos a la consola de Tomcat el mensaje correspondiente. Lo mismo en el caso de que el proceso haya fallado.

Linea 21: la función devuelve el objeto.

Lineas 23 a 26: En el caso de que se produzca un error dentro la estructura try, el mismo es procesado por la estructura catch, que imprime el mensaje de error producido.

A esta altura el metodo ha terminado, y no nos queda sino ver cómo se utiliza dentro de la pagina JSP.

Veamos:

```
<%@ page language="java" %>
<%@ page import = "notas.ConnectionCreator"%>
<%@ page import = "java.sql.Connection"%>
<%
    Connection miConexion = ConnectionCreator.getSqlServerConnection("Orders", "127.0.0.1",1433,"sa", "");

    if (!miConexion.isClosed())
      out.print("FUNCIONA !");

   /* CUERPO DE LA PAGINA */

   miConexion.close(); /* no olvidarse de cerrar las conexiones. */

%>
```

Aqui simplemente creamos un objeto Connection y , a través de del metodo getSqlServerConnection de nuestra clase, nos conectamos al database.
Al final de la página, cerramos la conexion.

En el caso de querer conectar un servidor MySql, conviene agregar un segundo metodo a la clase, especifico para este tipo de server. Por supuesto que su nuestro sitio deberan existir tambien los drivers de MySql.

Veamos como sería el método getMySqlConnection

```
29
30      public static java.sql.Connection getMySqlConnection(String database,
31              String servername, int port, String username, String password ) {
32          try {
33              Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
34              String url ="jdbc:mysql://" + servername + ":" + port + "/" + database;
35              Connection conn = DriverManager.getConnection(url,username,password);
36              if (conn != null)
37                  System.out.println("---> CONNECTED TO SERVER : "+servername);
38              else
39                  System.out.println("---> UNABLE TO CONNECT TO SERVER : "+servername);
40
41              return conn;
42          }
43          catch(Exception e) {
44              System.out.println("ERROR = "+e);
45              return null;
46          }
47      }
48
```

Basicamente la lógica es la misma, cambia obviamente el driver inicializado, y la estructura de la url, ya que en este caso no contiene ni usuario ni password, datos estos que son pasados directamente al DriverManager.

Construcción dinámica de menús de selección utilizando JSP

Sucede seguido que tenemos que generar menús de selección dentro de páginas HTML. Muchas veces estos menús son idénticos o muy similares, y puede pasar que ocupan mucho lugar dentro de nuestra pagina web.

La idea es pasar a esta función los parámetros que nos interesan, y dejar que ella se ocupe del resto. Veamos el código de la página, con una breve explicación debajo de cada bloque de código.

```
<%@ page language="java"%>
<%!
  private void writeMenu(javax.servlet.jsp.JspWriter out, int startValue,
 int endValue, int selectedValue){
 try{
 // esta línea es opcional...
  out.println("<option value=\"-1\" selected>
 Selecciona una opcion</option>");

 // comienzo el ciclo con el rango de valores dado.
 for (int i=startValue; i<(endValue+1); i++){
 // si el valor actual corresponde al valor del ciclo,
 //lo hago aparecer como seleccionado
  if (i == selectedValue){
   out.println("<option value=\""+i+"\"
   selected>"+i+"</option>");
   }
```

```
   else{
    out.println("<option value=\""+i+"\">"+i+"</option>");
   }
   }
  }catch(java.io.IOException e1){
   System.out.println(e1);
   }
}
%>
```

Esta es la función que genera el menùs, recibe como parámetros el valor inicial, el valor final, el valor actual y el objeto JspWriter, que permite escribir dinámicamente sobre la página.

```
<%

java.util.GregorianCalendar cal = new java.util.GregorianCalendar();
    int day = cal.get(cal.DAY_OF_MONTH);
    int month = (cal.get(cal.MONTH)) + 1;
    int year = cal.get(cal.YEAR);

%>
```

Creamos variables de página, que nos servirán solamente para probar la función.

```
<html>
<head>
<title>Construcción dinámica de menús de selección</title>
</head>

<body>

<table>
   <h1>construcción dinámica de menús de selección</h1>
    <tr>
     <td>Dia:
     <select>
      <% writeMenu(out,1,31,day); %>
     </select>
     </td>

     <td>Mes:
     <select>
      <% writeMenu(out,1,12,month); %>
     </select>

     </td>
```

```
    <td>Año:
    <select>
     <% writeMenu(out,2000,2010,year); %>
    </select>

    </td>

    </tr>
   </table>
  </body>
</html>
```

Finalmente el html, muy simple, que muestra cómo se usa la función.
Una variante de este procedimiento sería por ejemplo leer datos de un database. Por ejemplo, si se quiere generar un clásico menú con la lista de países del mundo, se podría hacer una función de este tipo:

```
public void writeCountryMenu(Connection connection,
javax.servlet.jsp.JspWriter out, int currentValue){
  try{
    String sqlString = " SELECT id,Name from T_Countries order by orderview ";

    Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery(sqlString);
    while (rs.next()){
      String selected = "";
     int countryId = rs.getInt("id");
     if (countryId==currentValue)
      selected = "SELECTED";
     out.print("<option value=\""+countryId+"\" "+selected+">"+rs.getString("Name")+"</option>");
    }
    rs.close();
  stmt.close();

  }catch(Exception e){
    System.out.println(e);
  }
 }
```

En este caso obviamente tenemos que pasar una conexión a un database (puede ser MsSQL, MySql, etc.) en el cual tenemos la tabla T_countries, con los campos id, name y orderview.

Instalación de Tomcat para utilizar servlets o JSP

Este proceso es bastante sencillo, pero consta de una serie de pasos que se deben realizar al pie de la letra:

a) Instalar el servidor Tomcat y alguna versión del J2SDK (se recomienda 1.4.01 o la Enterprise Edition), indicándole el lugar donde se encuentra instalada la JVM (Java Virtual Machine), y de preferencia, dejando como puerto el 4848 para el acceso del servidor

b) Posteriormente se procede a configurarlo en la computadora. Para esto se crea una "variable de entorno". Si usas Windows 95/98/2000/XP, se crea modificando el archivo autoexec.bat ubicado en c:\ (esta como archivo oculto). Su modificación se hace agregándole la siguiente línea:

set classpath = "ruta"

Donde "ruta" es la ubicación de los archivos jsp-api.jar y servlet-api.jar que se encuentran en la carpeta: ruta_de_instalacion_del_servidor_tomcat/common/lib.

A continuación un ejemplo de línea a agregar al autoexec.bat, suponiendo que el servidor tomcat se instalo en c:\archivos de programa:

SET CLASSPATH= C:\Archivos de programa\Apache Software Foundation\Tomcat 5.0\common\lib\jsp-api.jar;C:\Archivos de programa\Apache Software Foundation\Tomcat 5.0\common\lib\jsp-api.jar

Recordando que también se le agrega al claspath la dirección del la ruta donde se encuentra la carpeta \bin del j2sdk (compilador de java).

c) Una vez echo lo anterior, para agregar el primer servlet al sitio, busca el archivo web.xml, el cual se encuentra en ruta_de_instalacion_del_servidor_tomcat\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\WEB-INF\ y se le agrega a la carpeta webapps el nuevo_servlet.class (que es el resultado de la compilacion del nuevo_servlet.java). Además hay que añadir estas líneas al archivo web.xml

```
<servlet>
   <servlet-name>nuevo_servlet</servlet-name>
   <servlet-class>nuevo_servlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>nuevo_servlet</servlet-name>
   <url-pattern>nuevo_servlet</url-pattern>
</servlet-mapping>
```

Pero como el archivo ya tienen escrito se acomoda de tal manera q queda asi:

```
<servlet>
   <servlet-name>org.apache.jsp.index_jsp</servlet-name>
   <servlet-class>org.apache.jsp.index_jsp</servlet-class>
</servlet>
<servlet>
   <servlet-name>nuevo_servlet</servlet-name>
   <servlet-class>nuevo_servlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
    <servlet-name>org.apache.jsp.index_jsp</servlet-name>
    <url-pattern>/index.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>nuevo_servlet</servlet-name>
    <url-pattern>nuevo_servlet</url-pattern>
</servlet-mapping>
```

d) Por último se prueba el servlet escribiendo en el explorador http:\\localhost:484\nuevo_servlet y listo, debería verse el resultado.

Uso de XML y XSL en JSPs

Utilizando JSP 2.0, podemos combinar JSP y XSL para formatear documentos XML en el servidor

No siempre es posible (ni conveniente) utilizar lo último de lo último. Si os fijáis en las grandes organizaciones, normalmente estandarizan una versión de Java, JSP, EJB, etc... y hasta que no pasa un tiempo y se consolidan las tecnologías (y aparecen parches) no se cambia de versión.

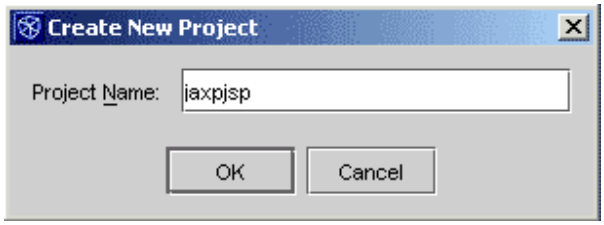
Entonces, es posible que tengáis que apañaros para hacer lo mismo ..... con versiones anteriores de JSP.

Veréis que es bastante sencillo y os vamos a mostrar como se hace paso a paso.... utilizando el patrón MVC.

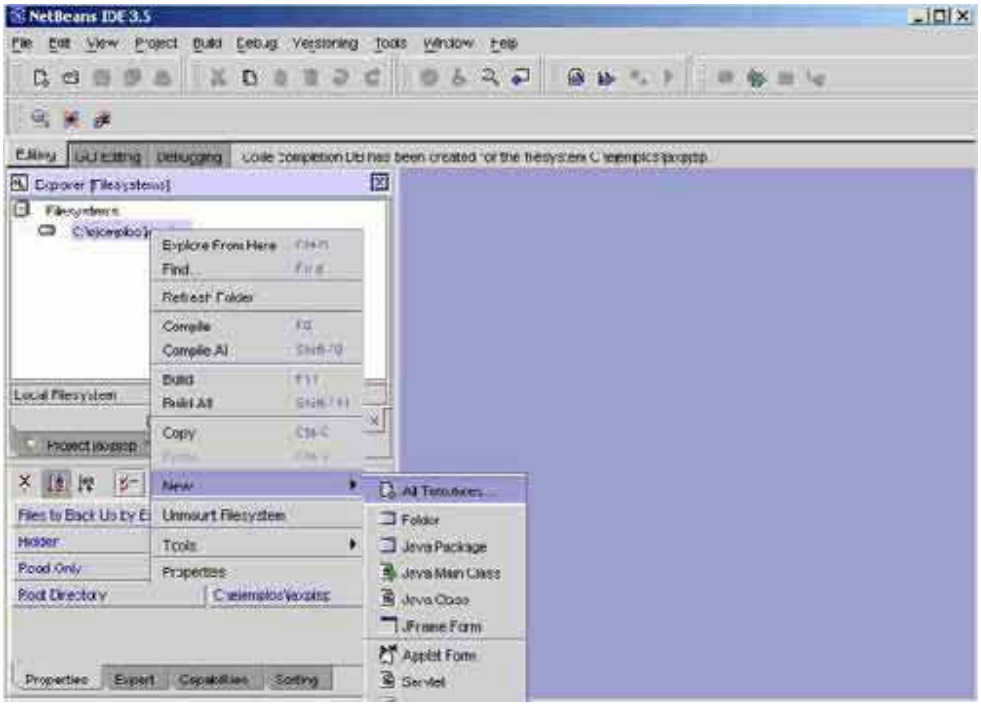
Pondremos los ejemplos con NetBeans. Mucha gente me pregunta por qué lo uso..... La razón es sencilla, me parece muy intuitivo... aunque hay otras opciones más potentes.

Crear el Proyecto

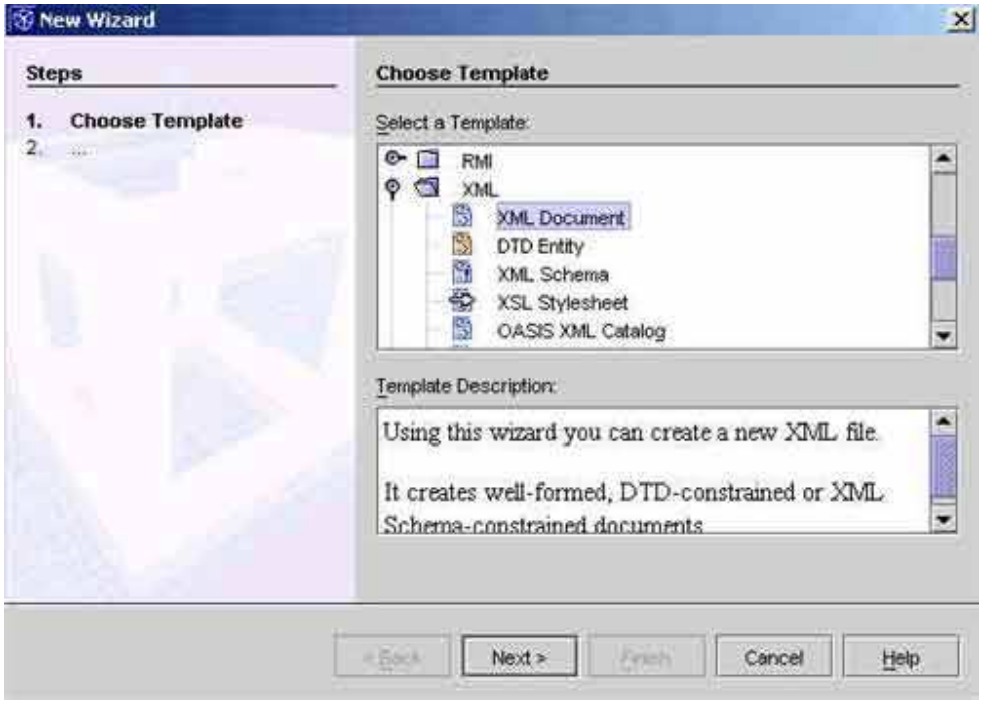
Creamos en NetBeans un proyecto



Vamos a crear unos documento XML y XSL con los generadores de código.



Seleccionamos dentro del grupo XML

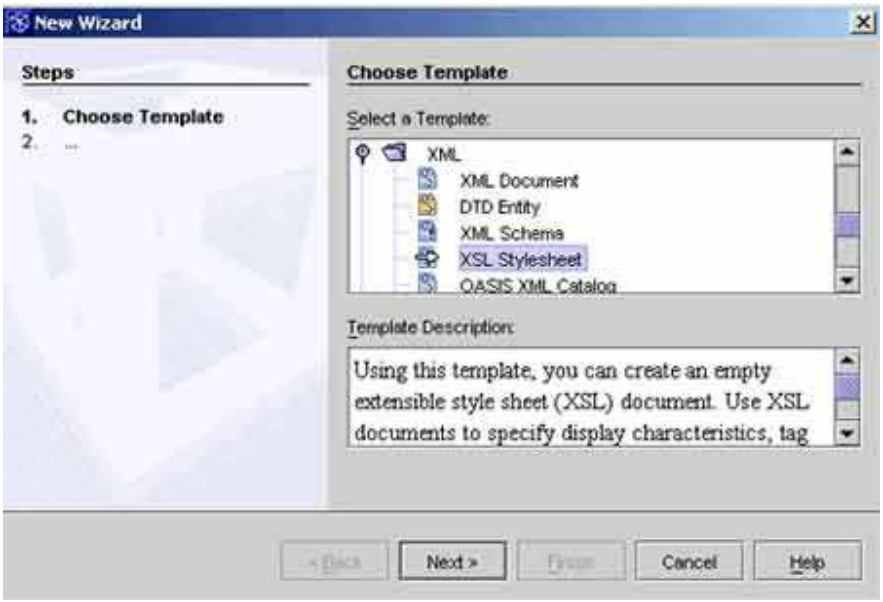


Asignamos un nombre

Vamos a utilizar este documento como base:

```
<?xml version="1.0" encoding="UTF-8"?>
<tutoriales>
 <tutorial>
   <autor>rcanales@autentia.com</autor>
   <nombre>JSP 2.0</nombre>
   <enlace>jspel</enlace>
   <descripcion>Nuevas caracteristicas de JSPs 2.0</descripcion>
 </tutorial>
 <tutorial>
   <autor>alejandropg@autentia.com</autor>
  <nombre>Struts y Eclipse</nombre>
  <enlace>struts</enlace>
  <descripcion>Configuración del entorno Struts en Eclipse</descripcion>
 </tutorial>
</tutoriales>
```

Repetimos pero ahora seleccionamos un documento XSL



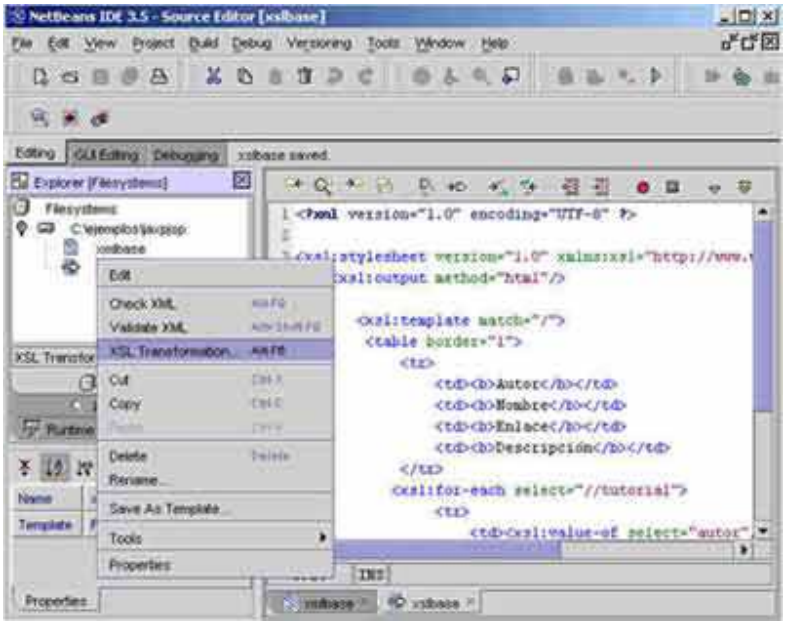
Escribimos nuestro XSL

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

  <xsl:template match="/">
   <table border="1">
   <tr>
    <td><b>Autor</b></td>
    <td><b>Nombre</b></td>
    <td><b>Enlace</b></td>
    <td><b>Descripción</b></td>
   </tr>
   <xsl:for-each select="//tutorial">
    <tr>
    <td><xsl:value-of select="autor"/></td>
    <td><xsl:value-of select="nombre"/></td>
    <td><xsl:value-of select="enlace"/></td>
    <td><xsl:value-of select="descripcion"/></td>
   </tr>
  </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```

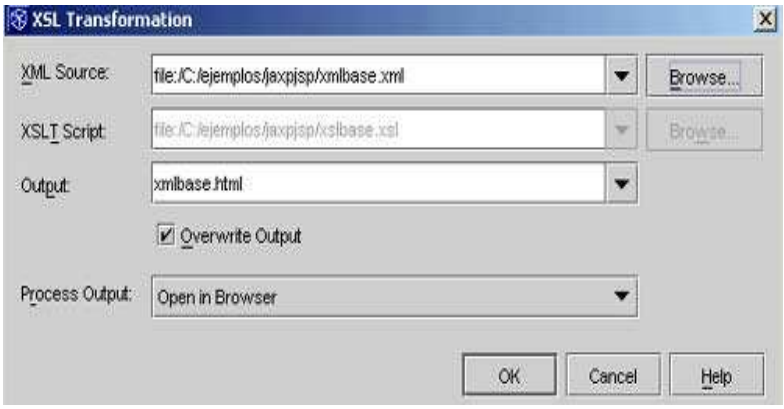
Probar la transformación en NetBeans

Vamos a usar las características de NetBeans para probar como quedaría....

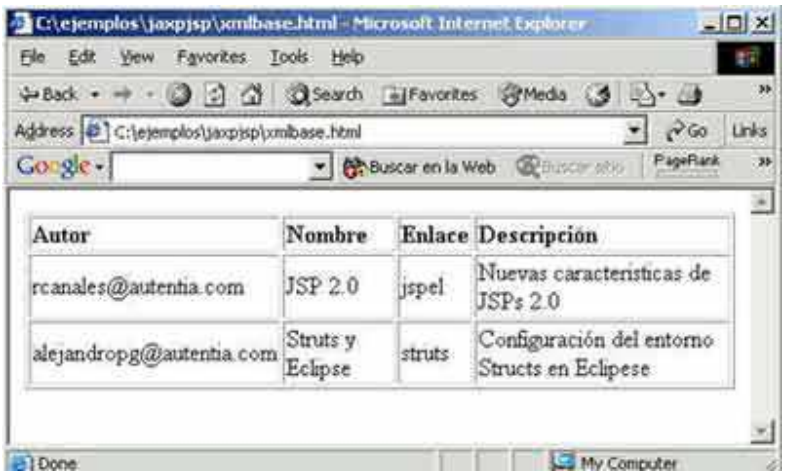
Seleccionamos, pinchando el botón derecho sobre el XML o XSL



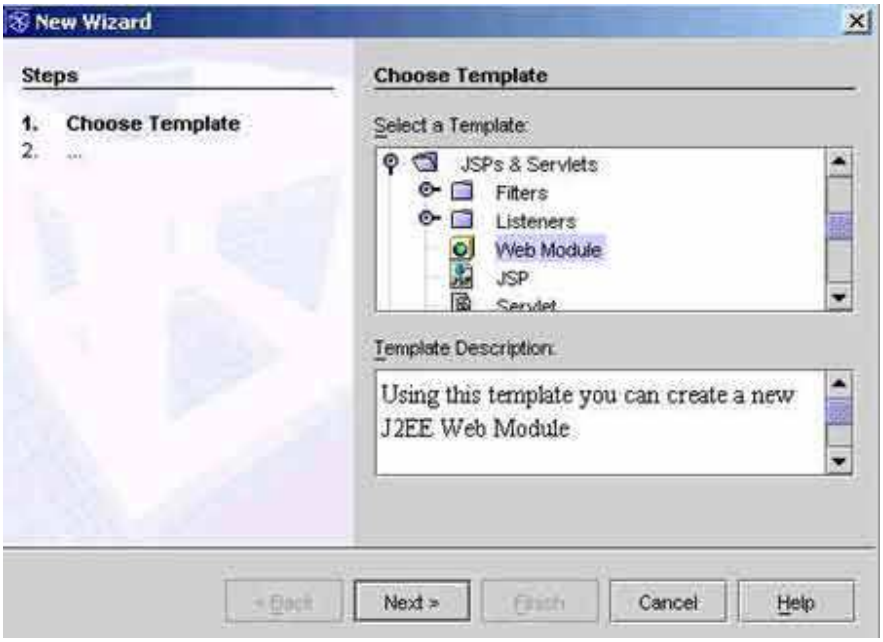
Seleccionamos los ficheros



Y vemos el resultado

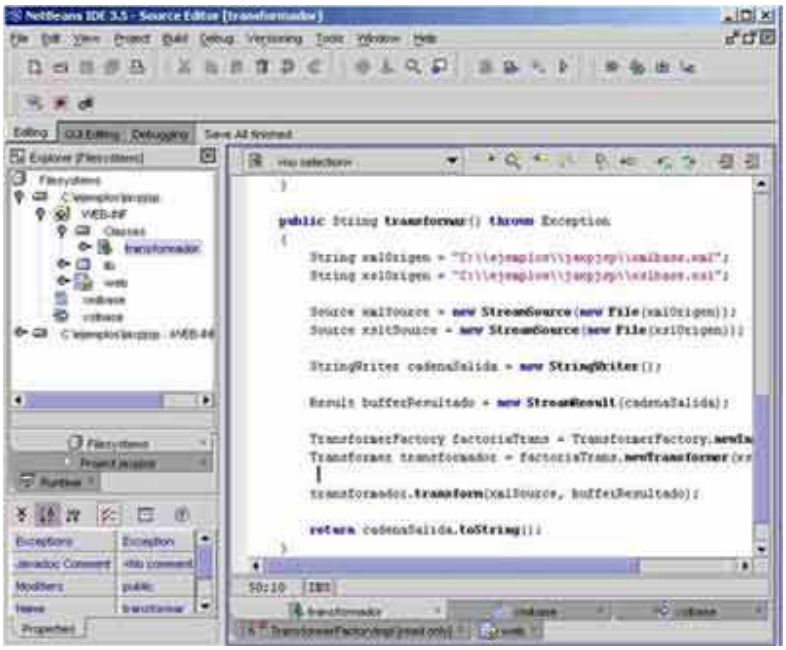


Activamos nuestro directorio como una WebApp



Introducir el código Java

Ahora, vamos a escribir una clase Java que sea capaz de hacer una transformación y retornárnosla como una cadena.



Escribir la clase Java de prueba

En este caso, vamos a utilizar un interfaz para motores de transformación denominado JAXP (ver documentación en SUN)

No tenemos que incluir nada extraordinario por la versión de Java que estamos usando.

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
/**
*
* @author Roberto Canales
*/
public class transformador
{
 void depura (String pCadena)
  {
  System.out.println("Mensaje: " + pCadena);
 }

  public static void main(String [] args) {
 transformador p = new transformador();

  try
 {
```

```
  p.depura("Comenzamos transformación");
   p.depura(p.transformar());
   p.depura("Terminamos");
 }
 catch(Exception e)
 {
  p.depura("Errores en aplicación");
  e.printStackTrace();
  }
}

public String transformar() throws Exception
{
  String xmlOrigen = "C:\\ejemplos\\jaxpjsp\\xmlbase.xml";
   String xslOrigen = "C:\\ejemplos\\jaxpjsp\\xslbase.xsl";

  Source xmlSource = new StreamSource(new File(xmlOrigen));
  Source xsltSource = new StreamSource(new File(xslOrigen));

  StringWriter cadenaSalida = new StringWriter();

  Result bufferResultado = new StreamResult(cadenaSalida);

  TransformerFactory factoriaTrans = TransformerFactory.newInstance();
  Transformer transformador = factoriaTrans.newTransformer(xsltSource);

  transformador.transform(xmlSource, bufferResultado);

  return cadenaSalida.toString();
}
}
```

Si ejecutamos este código, veremos en la pantalla de salida:

```
Mensaje: Comenzamos transformación
Mensaje: <table border="1">
<br>
<tr>

<td><b>Autor</b></td><td><b>Nombre</b> </td><td><b>Enlace</b></td><td><b>Descripción</b></td>

</tr>

<tr>

<td>rcanales@autentia.com</td><td>JSP 2.0</td><td>jspel</td><td>Nuevas caracteristicas de JSPs 2.0</td>
</tr>
<tr>
```

<td>alejandropg@autentia.com</td><td>Struts y Eclipse</td><td>struts</td><td>Configuración del entorno Structs en Eclipse</td>
</tr>
</table>
Mensaje: Terminamos

Es decir, funciona.....

Crear el MVC

Ahora, vamos a crear un servlet, que generará un XML y lo pasará (en MVC) a un JSP. Vamos a introducir directamente el XML en una cadena de caracteres.... dando por supuesto que se obtendría de otro modo (por ejemplo usando una Base de Datos XML)

Este JSP, puede ejecutar el código de la transformación con distintas técnicas:

- Scriplet (Código Java en el JSP)
- Un Java Bean
- Un TAG de usuario
- Usando JSTL de JSP 2.0

Los distintos métodos de comunicación entre estos elementos (Servlets, Beans, JSPs y Tags los podéis ver en otro de loa tutoriales de adictosaltrabajo)

Os mostramos como hacerlo con un JSP con Scriplet... el resto de los métodos es muy sencillo .

El servlet

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class controlador extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
  try
  {
    String elXML = "<tutoriales><tutorial><autor>rcanales@autentia.com</autor>

<nombre>JSP 2.0</nombre><enlace>jspel</enlace><descripcion>Nuevas caracteristicas de JSPs 2.0</descripcion>
```

```
</tutorial><tutorial><autor>alejandropg@autentia.com</autor><nombre>Struts y Eclipse</nombre>

<enlace>struts</enlace><descripcion>Configuración del entorno Structs en Eclipse</descripcion>

</tutorial></tutoriales>";

    request.setAttribute ("xml",elXML);
    getServletConfig().getServletContext().getRequestDispatcher("/presentacion.jsp").forward(request,
response);
   }
  catch (Exception ex)
   {
    ex.printStackTrace ();
  }
  }
}
```

EL JSP

```
<%@page contentType="text/html" import="java.io.*,javax.xml.transform.*,javax.xml.transform.sax.*,
javax.xml.transform.stream.*,org.xml.sax.*"%>
<html>
<head><title>JSP Page</title></head>
<body>

<center>
<H1>JSP, transformando XML con XSL</h1>
<br>

<%
  String xmlOrigen = (String)request.getAttribute("xml");
  String xslOrigen = "C:\\ejemplos\\jaxpjsp\\xslbase.xsl";

  Source xmlSource = new StreamSource(new StringBufferInputStream(xmlOrigen));
  Source xsltSource = new StreamSource(new File(xslOrigen));

   StringWriter cadenaSalida = new StringWriter();

  Result bufferResultado = new StreamResult(cadenaSalida);

  TransformerFactory factoriaTrans = TransformerFactory.newInstance();
  Transformer transformador = factoriaTrans.newTransformer(xsltSource);

  transformador.transform(xmlSource, bufferResultado);
   out.print(cadenaSalida.toString());
%>
</center>
```

</body>
</html>

La salida es:

Bueno, ya sabéis como hacerlo ..... y con Struts.... sería igual de sencillo.

Manejo de archivos con JSP

Los comentarios escritos dentro del código, explican paso a paso que cosa se esta ejecutando en ese momento.

- La función readFile, dado un nombre de archivo (con el "camino" incluido. Ej. D:\files\prueba.txt), devuelve un objeto de tipo StringBuffer con el contendio del mismo.
- El método saveFile , toma como primer parámetro el nombre del archivo que se desea generar o modificar, el segundo parámetro es el contenido que se desea introducir en dicho archivo. En fin el tercer parámetro es una variable de tipo boolean, que si recibe el valor "true" , aprega la información recibida al final del archivo (si este archivo existe). Si, en cambio, recibe "false" como parametro, borra el contenido del actual del archivo y lo reemplaza con el segundo parametro.
- Finalmente la tercera función sirve para buscar uno o mas valores dentro de un archivo de una cadena de texto y reemplazarlos por una serie de valores dados.

  Por ejemplo se podría usar de este modo:

  ```
  String[] busqueda = {"NOMBRE","APELLIDO"};
  String[] reemplazo = {"FERNANDO","ARTURI"};
  FileManager fm = new FileManager();
  String resultado = fm.replaceValues("D:\myFiles\prueba.txt", busqueda, reemplazo);
  ```

A continuación el código de la classe:

```
package notas;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.FileNotFoundException;

/**
* <p>Title: FileManager</p>
*
* <p>Description: Manejo de archivos de texto</p>
*
* <p>Copyright: Copyright (c) 2006</p>
*
* @author Fernando Arturi
* @version 1.0
*/

public class FileManager {

public void FileManager(){}

  /**
  * El metodo readFile lee un archivo de texto y retorna su contenido en
   * formato de StringBuffer
  * @param filename String
   * @return StringBuffer
  */

public StringBuffer readFile(String filename){
  StringBuffer sb = new StringBuffer();
  try{
      /**
     * Aqui creamos un objeto File que representa el archivo de texto que
      * queremos leer
      */
      File file = new File(filename);

     /**
      * Variable temporal que usaremos para leer cada una de las lineas del
      * archivo de texto
      */
     String line = null;
```

```
      /**
       * BufferReader - Es el encargado de leer el archivo de texto.
      * El constructor recibe como parametro un objeto FileReader, que
       * a s vez recibe el objeto File creado precedentemente.
       */

      BufferedReader br = new BufferedReader(new FileReader(file));

      /**
       * A traves de este ciclo el BufferReader lee todo el archivo, y lo va acumulando (sb.append) en un
StringBuffer
       */
       while ((line = br.readLine()) != null) {
         sb.append(line);
       }

      /**
      * Al final de la lectura cerramos el objeto
      */
      br.close();

    } catch (FileNotFoundException fnfe){
       /**
      * Si damos un nombre de archivo que no existe el sistema genera automaticamente un error.
       */
          System.out.println("No ha sido posible encontrar el archivo "+ filename);
   }
   catch (IOException ioe){
      /**
       * Se ha producido un error durante lalectura del archivo
      */
       System.out.println("Se ha producido un error durante la lectura del archivo "+ filename);
    }
   return sb;
}

/**
* Este metodo permite, dada una cadena de caracteres determinada,
* salvar la misma como un archivo de texto, o agregarla a un archivo ya existente
*
* @param filename String
* @param dataToWrite String
* @param append boolean
*/
public void saveFile(String filename, String dataToWrite, boolean append) {
   try {
```

```
      /**
       * Creaciòn del objeto FileWriter dado un nombre de archivo determinado
       * El segundo parametro (append) contiene un valore booleano que
       * indica si la informacion recibida debe ser agregada el final del
      * archivo o, en caso contrario, reemplazar la informaciòn ya
      * existente.
       */
      FileWriter fw = new FileWriter(filename, append);

      /**
       * Escritura de la informacion en el archivo
       */
       fw.write(dataToWrite);

      /**
       * Se cierra el archivo
       */
      fw.close();
    } catch (IOException ioe) {
       /**
       * Se ha producido un error durante la lectura/escritura del archivo
       */
      System.out.println(
         "Se ha producido un error durante la lectura del archivo " + filename);
    }
}


/**
* Esta funcion permite, dado un archivo en particular, buscar dentro el mismo
* determinados valures y cambiarlos por una serie de nuevos valores dados,
* generando un objeto de tipo String con el resultado
*
* @param path String
* @param valuesToSearch String[] Ejemplo {"NOMRE", "APELLIDO"}
* @param valuesToReplace String[] Ejemplo {"Fernando Augusto", "Arturi"}
* @return String
*/
public String replaceValues (String path, String [] valuesToSearch, String [] valuesToReplace){
String line;
StringBuffer textComplete = new StringBuffer();
String tempText = "";

/**
* Lectura del archivo de texto dado
*/
try {
```

```
    BufferedReader br = new BufferedReader(new FileReader(path));
   while ((line = br.readLine()) != null){
      textComplete.append(line);
   }
   br.close();

} catch (FileNotFoundException fnfe){
    /**
    * Si damos un nombre de archivo que no existe el sistema
    * genera automaticamente un error.
    */
    System.out.println("No ha sido posible encontrar el archivo "+ filename);
}

catch (IOException ioe) {
   /**
    * Se ha producido un error durante la lectura/escritura del archivo
   */
   System.out.println(
       "Se ha producido un error durante la lectura del archivo " + filename);
}

/**
* Una vez completada la fase de lectura del archivo, pasamos a la
* bùsqueda y reemplazo de los valores datos.
* Para esto generamos un ciclo que recorremos tantas veces como valores
* que tenemos que procesar.
*/
for (int i=0; i<valuesToSearch.length; i++){
    int position = textComplete.indexOf(valuesToSearch[i]);
   if (position>0 ){
      tempText = textComplete.substring(0,position);
       /**
       * busqueda y reemplazo de la cadena.
       */
      tempText = tempText + valuesToReplace[i] +
textComplete.substring(position+valuesToSearch[i].length(),textComplete.length());
       textComplete = new StringBuffer(tempText);
   }
  }
  return tempText;
}

}
```