

Proyecto de PAW

Matrículas

Contenido

- Tomcat y Apache
- Sax y Dom
- Jsp
- Jsp y JavaBeans
- Jsp y XSLT
- Proyecto

Apache y Tomcat

- Introducción
- Como se usa

APACHE Y TOMCAT

Introducción

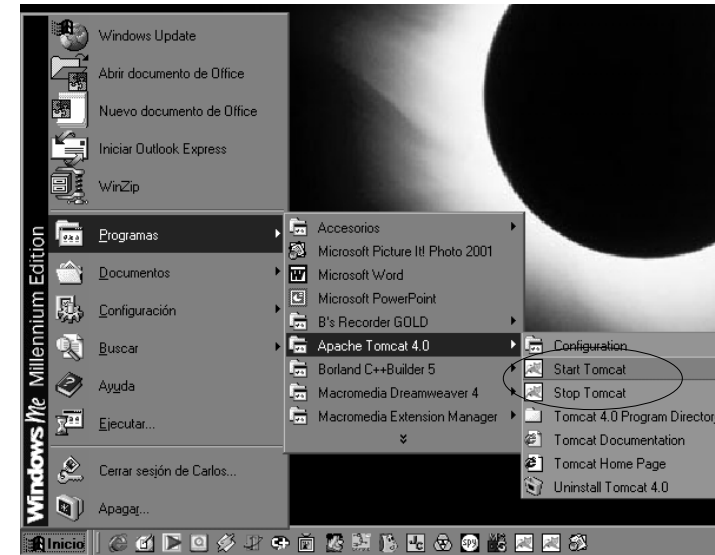
- Para la ejecución de Aplicaciones Web se necesita de un servidor. Este servidor se encarga de esperar peticiones HTTP de un cliente y servirle el contenido necesario.
- Además, si ejecutamos servlets y/o JSP necesitaremos un servidor de aplicaciones web.
- Por ello, se utiliza Apache para servir páginas estáticas, y Tomcat para dar soporte para servlets y páginas JSP.

APACHE Y TOMCAT

Cómo se usa

- Instalamos el Apache Tomcat 4.0.1. Por defecto se instalará en el directorio “c:\Archivos de Programa\Apache Tomcat 4.0”
- Si todo ha ido bien tomcat instalará dos accesos directos en nuestro menu inicio:

APACHE Y TOMCAT



APACHE Y TOMCAT

Como se usa

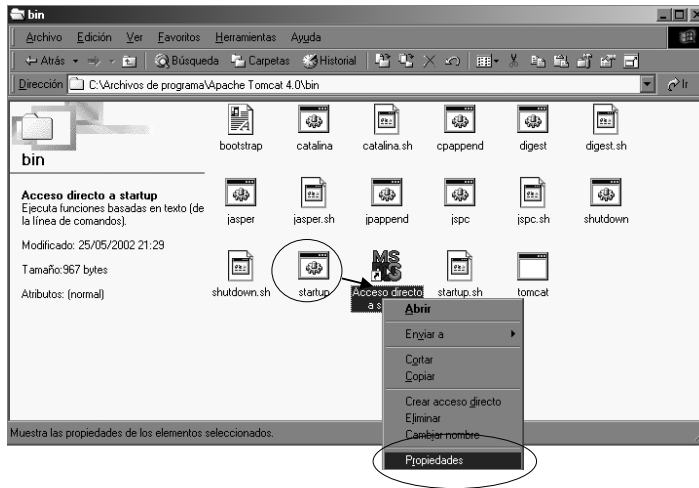
- Startup nos va a iniciar el servidor.
- Shutdown va a parar el servidor.
- De esta forma, cada vez que queramos ejecutar nuestras aplicaciones web lo primero que hay que hacer es arrancar el servidor para que pueda responder a las peticiones del usuario.

APACHE Y TOMCAT

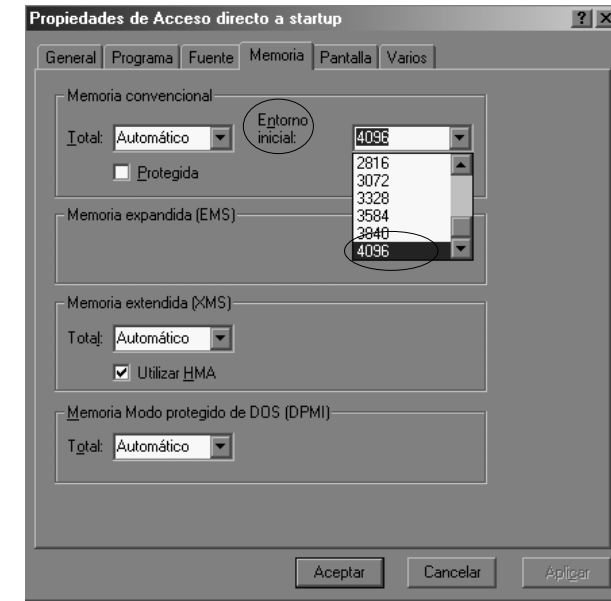
Como se usa

- Si no se generan estos accesos directos, o por alguna razón se pierden, se puede hacer lo siguiente:
 - Ir al Directorio_de_Tomcat\bin (por defecto “c:\Apache Tomcat 4.0\bin”)
 - Crear un acceso directo de startup.bat y shutdown.bat
 - Si al ejecutarlos nos da un problema de espacio de entorno, aumentamos la memoria de entorno en el acceso directo:

APACHE Y TOMCAT



APACHE Y TOMCAT



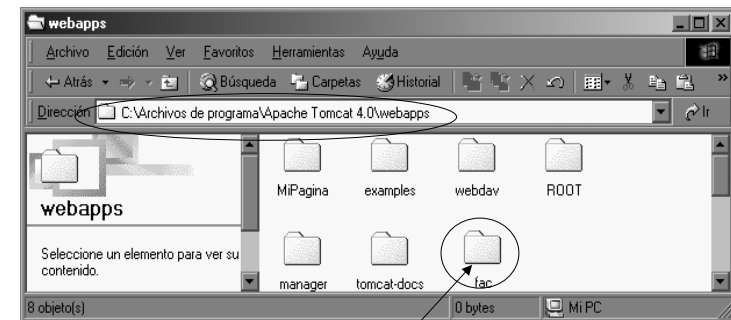
APACHE Y TOMCAT

Como se usa

- Ahora tenemos que ubicar nuestra aplicación para que nuestro servidor la pueda encontrar.
- Por defecto Apache Tomcat busca sus aplicaciones web en el directorio:
“c:\Archivos de Programa\webapps”
- Creamos entonces una carpeta con nuestra aplicación y la metemos dentro de “webapps”
- NOTA: dentro de nuestra carpeta tendremos una carpeta “WEB-INF”, y dentro de esta una carpeta “classes”, donde se van a ubicar nuestros servlets.

APACHE Y TOMCAT

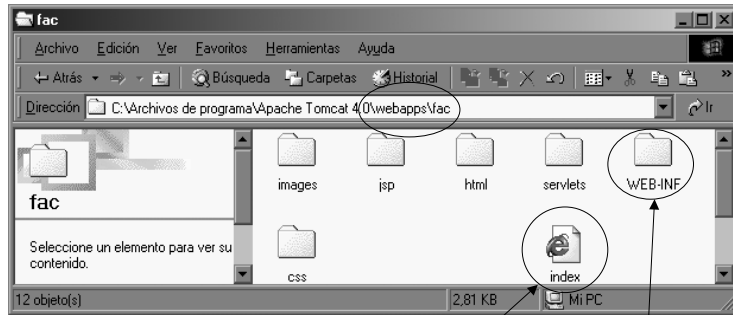
Webapps



Nuestra aplicación

APACHE Y TOMCAT

Nuestra aplicación

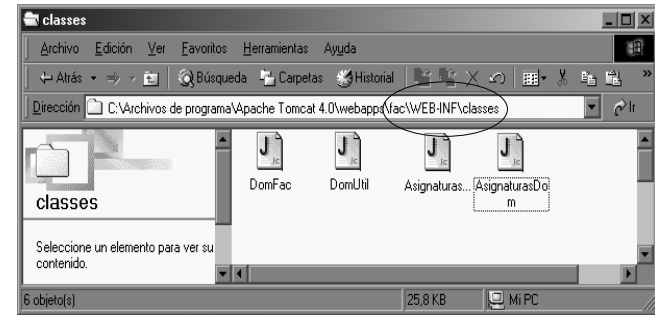


Página de bienvenida

WEB-INF

APACHE Y TOMCAT

classes



APACHE Y TOMCAT

SAX y DOM

- Introducción al SAX (Simple Api for Xml)
- Introducción al DOM (Document Object Model)
- Como utilizar Sax y Dom

SAX Y DOM

Introducción SAX (Simple Api for Xml)

- Vamos a utilizar SAX para comprobar que el documento XML con el que trabajamos es válido y esté bien construido.
- **SAX** es un interface dirigido a eventos, cuando ocurre un evento de análisis, (por ejemplo, una apertura de etiqueta) el analizador Sax invoca uno o más métodos.
- Sax consume menos memoria, es más eficiente y es más usado que DOM. Sin embargo en nuestra aplicación usamos DOM, porque SAX es secuencial.

SAX Y DOM

Introducción DOM

(Document Object Model)

- DOM va a reconstruir nuestro documento XML en un árbol de objetos en memoria.
- Nos va a permitir:
 - movernos por un árbol XML,
 - modificar,
 - consultar,
 - crear,
 - eliminar... sus nodos.
- De esta forma nos va ser posible manipular un documento XML a través de código.

SAX Y DOM

Cómo utilizar SAX y DOM

- Sax y Dom están incluidos en los jdk 1.4
- Es necesario importar las siguientes clases:
 - APIs JAXP necesarios:
 - `import javax.xml.parsers.*`
 - Tratamiento de excepciones en el análisis
 - `import org.xml.sax.SAXException;`
 - `import org.xml.sax.SAXParseException;`
 - La defición del W3C para un DOM y sus excepciones
 - `import org.w3c.dom.Document;`

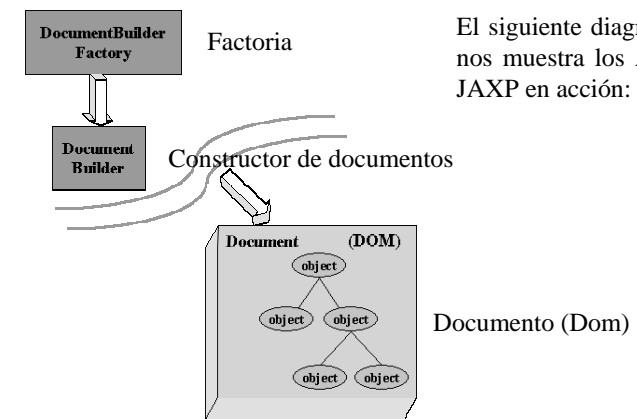
SAX Y DOM

Cómo utilizar SAX y DOM

- Lo primero es declarar el documento:
`Document documento;`
- Necesitamos además una factoría que nos ofrezca un constructor de documentos:
 - `DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();`
- Instanciamos el constructor de documentos:
 - `DocumentBuilder builder = factory.newDocumentBuilder();`
- Y lo utilizamos para analizar el fichero XML y generar nuestro documento:
 - `documento = builder.parse(ficheroXML);`

SAX Y DOM

SAX y DOM



SAX Y DOM

Cómo utilizar SAX y DOM

- Ahora que tenemos el documento utilizaremos la clase **org.w3c.dom.Node** para movernos por el árbol.
 - Node nodo
- Lo primero es hacer que el nodo apunte a la raíz del documento:
 - `nodo = documento.getDocumentElement();`
- A partir de aquí podemos navegar con nuestro nodo a través del árbol (ir al hijo, ir a un hermano, crear hijo...)

SAX Y DOM

Cómo utilizar SAX y DOM

- Ejemplos de navegación por el árbol:
 - Ir al primer hijo:
 - `nodo = nodo.getFirstChild()`
 - Ir al siguiente hijo:
 - `nodo = nodo.getNextSibling()`
 - Añadir un hijo:
 - `nodo.appendChild(otroNodo)`
donde `otroNodo` es el hijo a añadir.
 - » Ver la API de Java

SAX Y DOM

JSP (Java Server Pages)

- Introducción
- Funcionamiento
- Estructura de la página
- Directivas
- Scriptlets

JSP

Introducción

- Las páginas de JSP son documentos de texto que combinan sentencias en Java y etiquetas HTML.
- Están basadas en la tecnología de servlets. Prometen proporcionar una capacidad que es al menos tan poderosa como los Servlets y potencialmente mucho más fácil de usar.

JSP

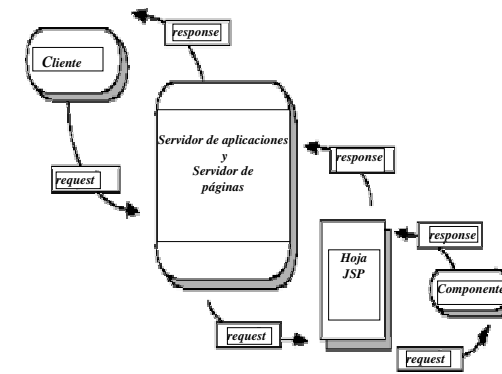
Funcionamiento

- Las páginas JSP, una vez escritas son analizadas y compiladas por nuestro Servidor de Aplicaciones (en nuestro caso Tomcat) de forma automática .
- Una vez analizadas, si el documento no contiene errores se genera un Servlet equivalente.
- De esta forma, cuando nuestra página es visitada, el servidor ejecuta el Servlet correspondiente (el cual Tomcat ubica en la carpeta “Directorio_de_Tomcat\work”).

JSP

Flujo de datos

La siguiente figura representa el flujo de datos entre el cliente y el servidor



JSP

Estructura de la página

- Una página JSP está compuesta por directivas, scriptlets (código Java) y etiquetas HTML.
- Las directivas nos van a servir para decir que este documento es una página JSP, importar clases,...
- Los scriptlets van a estar integrados dentro del código HTML y no son más que código en Java.
- La parte estática de la página será escrita en HTML, de forma que nos ahorramos numerosas sentencias del estilo `out.println("<HTML >")`; las cuales serían necesarias en un Servlet convencional.

JSP

Directivas

- Van entre las marcas `<% @ directiva %>`
- **Page:** se usa para definir atributos que se aplican a una página JSP entera.
- Aquí podemos ver la sintaxis de la directiva **page**. Los valores por defecto se muestran en **negrita**. Los corchetes ([...]) indican un término opcional. La barra vertical (|) proporciona una elección entre dos valores como true y false.

JSP

Directivas

- `<%@ page`
- `[language="java"`
- `[extends="package.class"`
- `[import= "{ package.class|package.*}, ..."`]
- `[session="true|false"`]
- `[buffer="none|8kb|sizekb"`]
- `[autoFlush="true|false"`]
- `[isThreadSafe="true|false"`]
- `[info="text"`]
- `[errorPage="URLrelativa"`]
- `[contentType="mimeType[;charset=characterSet]" |`
`text/html; charset=ISO-8859-1"`]
- `[isErrorPage="true|false"] %>`

JSP

Directivas

- En esta aplicación se utilizan los siguientes atributos:
- lenguaje = « java »
 - Este atributo define el lenguaje de script usado. En JSP 1.0 el único lenguaje permitido es Java.
- import = « { paquete.class | paquete.*},... »
 - Esta lista especifica los paquetes y/o clases que el fichero JSP debería importar. Las clases de los paquetes se ponen a disposición de los scriptlets, expresiones, declaraciones y etiquetas dentro del fichero JSP

JSP

Scriptlets

- Van entre las marcas `<% %>`.
- Etiquetas para los scriptlets:
 - `<%!.. %>` declara variables o métodos.
 - `<%=.. %>`. define una expresión del lenguaje script y fuerza el resultado a un String. No puede llevar “;”.
 - El scriptlet `<% %>` puede manejar declaraciones, expresiones o cualquier otro tipo de fragmento de código válido en el lenguaje script de la página.
 - Cuando escribimos un scriptlet, lo terminamos con `%>`. antes de cambiar a HTML, texto u otra etiqueta JSP

JSP

Scriptlets

- **Declaraciones:** (entre las etiquetas `<%! y %>`):
 - `<%! int i = 0; %>`
 - `<%! int a, b; double c; %>`
 - `<%! Circle a = new Circle(2.0); %>`
- **Expresiones** (entre las etiquetas `<%= y %>`):
 - `<%= Math.sqrt(2) %>`
 - `<%= items[i] %>`
 - `<%= a + b + c %><%= new java.util.Date() %>`
- **Scriptlets** (entre las etiquetas `<% y %>`)
 - `<% String name = null;`
`if (request.getParameter("name") == null) {%>`

JSP

JSP y JavaBeans

- Introducción
- Uso en JSP
- Escribir el Bean
- Utilizar el Bean en JSP

JSP y JAVABEANS

Introducción

- Los **Javabeans** son componentes reutilizables de **Java**.
- Estos componentes pueden unirse visualmente en otros componentes compuestos como applets, aplicaciones, servlets... para dar una funcionalidad concreta.
- Para su utilización no hay más que tener en cuenta unas guías de estilo.

JSP y JAVABEANS

Uso en JSP

- El protocolo **HTTP** tiene un funcionamiento muy simple: un cliente hace una petición de documento, el servidor responde y termina la transacción. No almacena el estado de cada petición, es un protocolo “sin estado”.
- Por esto, se van a utilizar los **JavaBeans** integrados en nuestro formulario para poder almacenar el estado de éste a lo largo de toda la sesión.
- Existen otras alternativas, como son los cookies, la reescritura de la URL ó campos ocultos.

JSP y JAVABEANS

Escribir el Bean

- Lo primero es escribir el código en Java del Bean siguiendo el siguiente esqueleto:
 - El constructor de la clase no tiene argumentos.
 - Por cada componente del formulario tendremos una variable privada con el mismo nombre.
 - private nombrePropiedad;
 - Cada variable tendrá dos métodos:
 - Un método accesor:
 - » public String getNombrePropiedad();
 - Y un método mutador:
 - » public void setNombrePropiedad (String name);

JSP y JAVABEANS

Escribir el Bean

- Ejemplo: si tenemos un formulario con un componente “text” llamada petición, el Beans sería el siguiente:

```
- public class EjemploBean{
-     private String peticion;
-     public EjemploBean() {
-         peticion = null;}
-     public void setPeticion( String nombre ) {
-         peticion = nombre; }
-     public String getPeticion() {
-         return peticion; }
- }
```

JSP y JAVABEANS

Utilizar el Bean en JSP

- Para integrar un Beans en nuestra página JSP ponemos:
 - `<jsp:useBean id="nombre_Bean" scope="ambito" class="nombre_clase" />`
- Para inicializar las variables:
 - `<jsp:setProperty name="nombre_Bean" property="*" />`
- Para cambiar el valor de una variable concreta:
 - `<jsp:setProperty name="nombre_Bean" property="nombre_variable" value="valor" />`
- Para obtener el valor de una variable:
 - `<jsp:getProperty name="nombre_Bean" property=" nombre_variable" >`

JSP y JAVABEANS

Utilizar el Bean en JSP

- Ejemplo: pide al usuario su nombre y lo muestra por pantalla. Utiliza EjemploBean.

```
- <%@ page import="EjemploBean"%>
- <jsp:useBean id="mybean" scope="application" class="EjemploBean" />
- <jsp:setProperty name="mybean" property="*" >
- <html>
- <head><title>Hola,que tal</title></head>
- <body> <h1>Me llamo Pepe. ¿Cómo te llamas? </h1>
- <form method="post">
-     <input type="text" name="peticion " size="25"><br>
-     <input type="submit" value="Submit">
-     <input type="reset" value="Reset"></form>
- <% if ( request.getParameter("peticion") != null ) {%>
- <h2>hola, <jsp:getProperty name="mybean" property=" peticion" >
- <% }%>
- </body></html>
```

JSP y JAVABEANS

JSP y XSLT

- Introducción
- Uso

JSP y XSLT

Introducción

- Hemos explicado como mostrar nuestro documento XML con código JAVA a través de SAX y DOM.
- Ahora vamos a ver otra alternativa, aplicar una hoja de transformación XSLT con código JSP.
- Para esto necesitamos un transformador XSLT. En nuestro proyecto hemos utilizado un transformador de JAKARTA, el transformador XALAN v.3.1.2.

JSP y XSLT

Uso

- Una vez instalado XALAN en nuestra máquina virtual de java, el código sería el siguiente:
 - //importamos las clases necesarias
 - <% @ page language="java" contentType="text/html" %>
 - <% @ page import="javax.xml.transform.*"%>
 - <HTML> <BODY>.....
 - <% //inicializamos los archivos xml y xslt
 - String **xmlFile** = "ARCHIVO_XML.xml";
 - String **xslFile** = "ARCHIVO_XSLT.xslt";
 - //iniciamos la factoría
 - TransformerFactory **factoria** = TransformerFactory.newInstance();
 - //instanciamos el transformador con el archivo XSL correspondiente
 - Transformer **transformador**= **factoria**.newTransformer(new StreamSource(**xslFile**));
 - //generamos la transformación
 - **transformador**.transform(new StreamSource(**xmlFile**), new StreamResult(**out**));%>
 - ...</BODY></HTML>

JSP y XSLT

PROYECTO

- Introducción
- XML
- Java
- JSP y HTML
- Demostración

PROYECTO

Introducción

- El objetivo del proyecto es hacer una aplicación para la matriculación de alumnos de cualquier universidad a través de la Web.
- La información sobre las asignaturas y carreras está en un documento **XML**, el cual será recorrido con tecnología **DOM**.
- El usuario irá seleccionando las asignaturas por facultades, optativas, ciclos, etc. Y se irán almacenando en un Bean.
- Por último rellenará sus datos personales y mandará imprimir la matrícula.

PROYECTO

XML

- Tendremos sólo un documento XML con todas las asignaturas registradas.
- Las asignaturas se irán clasificando en función de la facultad a la que pertenezcan, el plan, el ciclo, si son optativas, troncales,...
- Se distinguirán tres tipos de nodos:
 - Listas: listados de universidades, facultades...
 - Ramas: dan la opción de seleccionar entre distintas opciones (por ejemplo entre curso u optativas).
 - Asignaturas: listado de asignaturas.

PROYECTO

Java

- Utilizamos cuatro unidades:
 - Una con las funciones del DOM que vamos a utilizar.
 - Una para almacenar el camino recorrido en el árbol (en que nodo nos encontramos).
 - Una con nuestros Beans (que opción ha seleccionado el cliente, que asignaturas lleva marcadas...).
 - Y otra que genera código HTML dinámico de la página principal:
 - Genera las listas de selección para ir al siguiente nodo.
 - Muestra el camino recorrido en formato HTML.

PROYECTO

JSP y HTML

- Nuestra está estructurada en cinco marcos.
- La parte superior corresponde a la cabecera de la página y está hecha con páginas **HTML** y un **gif**.
- En la parte izquierda tendremos el índice generado con **JSP** ya que se encarga de establecer la carpeta de trabajo para toda la aplicación.
- La parte central está dividida en dos marcos:
 - Izquierdo: muestra la opción del índice en curso.
 - Derecho: muestra las asignaturas seleccionadas.

PROYECTO

JSP y HTML



PROYECTO

Demostración

<http://localhost:8080/fac>

PROYECTO