

I. Introducción a las aplicaciones Web con JAVA.

Macario Polo,
Daniel Villafranca

CONTENIDO:

| | |
|---|----|
| 1. Introducción a la tecnología WEB con J2EE..... | 2 |
| Modelo cliente-servidor | 2 |
| Acceso a Datos | 4 |
| Formas de seguir la trayectoria de los usuarios (clientes)..... | 5 |
| 2. Introducción a JSP | 6 |
| Estructura de una página JSP | 6 |
| Procesamiento de la página JSP | 8 |
| Objetos implícitos | 9 |
| 3. EJBs | 11 |
| Beneficios..... | 11 |
| Componentes..... | 11 |
| Desarrollar EJBs..... | 11 |
| EJBs contra Servlets..... | 12 |
| ¿Cuándo usar EJBs? | 12 |
| 4. Algunos ejemplos..... | 13 |
| Elementos de Script..... | 13 |
| Manejar Formularios | 13 |
| Uso de JavaBeans con JSP..... | 15 |
| 6. Ejercicios prácticos | 17 |
| 5. Próximas sesiones | 18 |
| JSP y XML en la tecnología Web | 18 |
| JSTL y desarrollar Etiquetas JSP Personalizadas..... | 18 |
| Implementación de MVC mediante Struts | 18 |
| EJB y acceso a datos (JDBC e Hibernate) | 18 |
| Depuración de aplicaciones Web | 18 |
| La capa cliente y AJAX | 18 |
| La capa servidor y uso de Servicios Web..... | 18 |
| Anexo I. Lenguajes del lado cliente y servidor..... | 19 |
| Anexo II. Instalación de Tomcat para uso de JSP..... | 23 |
| Referencias..... | 25 |

1. Introducción a la tecnología WEB con J2EE

La Web dinámica se ha desarrollado desde un sistema de información distribuido hypermedia (HTML) basado en red que ofrecía información estática hasta un conjunto de portales y aplicaciones en Internet que ofrecen un conjunto variado de servicios.

Las soluciones de *primera generación* incluyeron **CGI**, que es un mecanismo para ejecutar programas externos en un servidor web. El problema con los scripts CGI es la escalabilidad; se crea un nuevo proceso para cada petición.

Las soluciones de *segunda generación* incluyeron vendedores de servidores Web que proporcionaban plug-ins y APIs para sus servidores. El problema es que sus soluciones eran específicas a sus productos servidores. Microsoft proporcionó las páginas activas del servidor (ASP) que hicieron más fácil crear el contenido dinámico. Sin embargo, su solución sólo trabajaba con Microsoft IIS o Personal Web Server. Otra tecnología de segunda generación son los **Servlets**. Los Servlets hacen más fácil escribir aplicaciones del lado del servidor usando la tecnología Java. El problema con los CGI o los Servlets, sin embargo, es que tenemos que seguir el ciclo de vida de escribir, compilar y desplegar .

Las páginas **JSP** son una solución de *tercera generación* que se pueden combinar fácilmente con algunas soluciones de la segunda generación, creando el contenido dinámico, y haciendo más fácil y más rápido construir las aplicaciones basadas en Web que trabajan con una variedad de otras tecnologías: servidores Web, navegadores Web, servidores de aplicación y otras herramientas de desarrollo.

La tecnología Java Server Pages™ (JSP) nos permite poner segmentos de código servlet directamente dentro de una página HTML estática. Cuando el navegador carga una página JSP, se ejecuta el código del servlet y el servidor de aplicaciones crea, compila, carga y ejecuta un servlet en segundo plano para ejecutar los segmentos de código servlet y devolver una página HTML o imprimir un informe XML.

Modelo cliente-servidor

Cuando se utiliza un servicio en Internet, como consultar una base de datos, transferir un fichero o participar en un foro de discusión, se establece un proceso en el que entran en juego dos partes. Por un lado, el usuario, quien ejecuta una aplicación en el ordenador local: el denominado *programa cliente*. Este programa cliente se encarga de ponerse en contacto con el ordenador remoto para solicitar el servicio deseado. El ordenador remoto por su parte responderá a lo solicitado mediante un programa que esta ejecutando. Este último se denomina *programa servidor*. Los términos *cliente* y *servidor* se utilizan tanto para referirse a los programas que cumplen estas funciones, como a los ordenadores donde son ejecutados esos programas.

El programa o los programas cliente que el usuario utiliza para acceder a los servicios de Internet realizan dos funciones distintas. Por una parte, se encargan de gestionar la comunicación con el ordenador servidor, de solicitar un servicio concreto y de recibir los datos enviados por éste; y por otra, es la herramienta que presenta al usuario los datos en pantalla y que le ofrece los comandos necesarios para utilizar las prestaciones que ofrece el servidor.

El navegador es una especie de aplicación capaz de interpretar las órdenes recibidas en forma de código HTML fundamentalmente y convertirlas en las páginas que son el resultado de dicha orden.

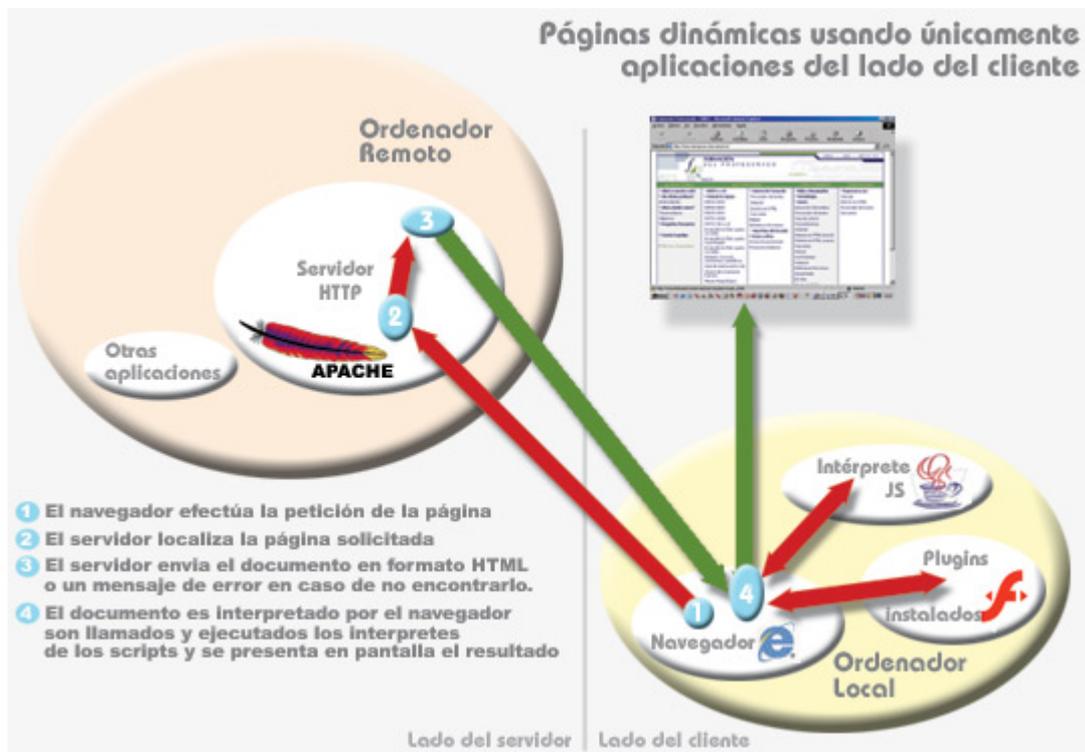


Fig. Lenguajes del lado cliente y del lado servidor

Cuando pinchamos sobre un enlace hipertexto, en realidad lo que pasa es que establecemos una petición de un archivo HTML residente en el servidor (un ordenador que se encuentra continuamente conectado a la red) el cual es enviado e interpretado por nuestro navegador (el cliente).

Sin embargo, si la página que pedimos no es un archivo HTML, el navegador es incapaz de interpretarla y lo único que es capaz de hacer es salvarla en forma de archivo. Es por ello que, si queremos emplear lenguajes accesorios para realizar un sitio web, es absolutamente necesario que sea el propio servidor quien los ejecute e interprete para luego enviarlos al cliente (navegador) en forma de archivo HTML totalmente legible por él.

De modo que, cuando pinchamos sobre un enlace a una página que contiene un script en un lenguaje comprensible únicamente por el servidor, lo que ocurre en realidad es que dicho script es ejecutado por el servidor y el resultado de esa ejecución da lugar a la generación de un archivo HTML que es enviado al cliente.

Así pues, podemos hablar de lenguajes de lado servidor que son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. Por otro lado, los lenguajes de lado cliente (entre los cuales no sólo se encuentra el HTML sino también el Java y el JavaScript los cuales son simplemente incluidos en el código HTML) son aquellos que pueden ser directamente "digeridos" por el navegador y no necesitan un pretratamiento.

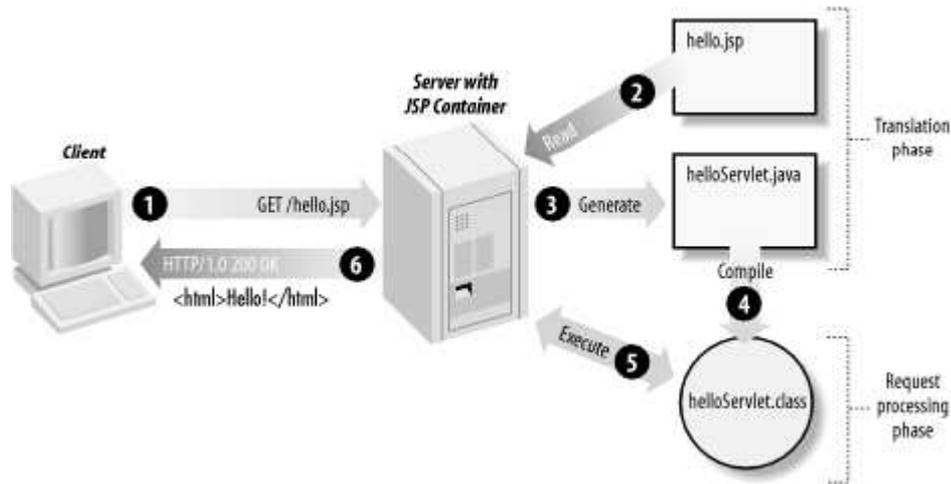


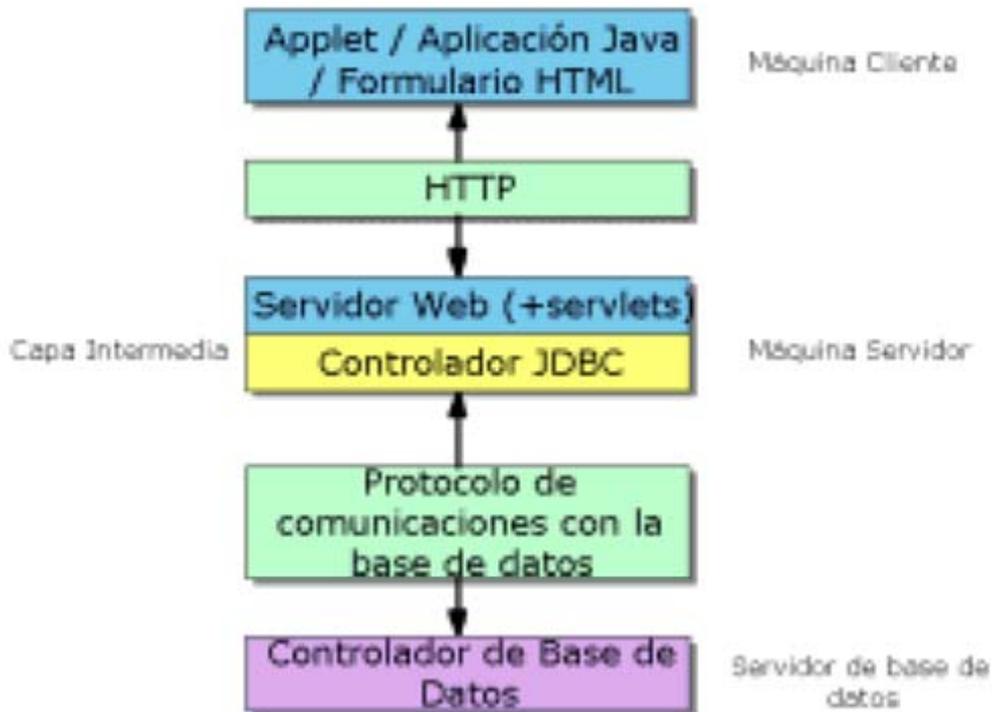
Fig. Ejecución en el modelo web cliente-servidor

En resumen, los programadores de aplicaciones JAVA mediante la especificación J2EE escriben componentes de aplicación J2EE. Un componente J2EE es una unidad de software funcional auto-contenida que se ensambla dentro de una aplicación J2EE y que se comunica con otros componentes de aplicación. La especificación J2EE define los siguientes componentes de aplicación:

- Componentes de Aplicación Cliente: Navegador y páginas *jsp* renderizadas
- Componentes JavaBeans Enterprise: *clases de negocio (EJBs)*
- Componentes Servlets y JavaServer Pages (también llamados componentes Web): *clases controladoras.*
- Applets: *pequeñas aplicaciones que se ejecutan en el cliente*

Acceso a Datos

Una de las tareas más importantes y más frecuentemente realizadas por los *servlets* es la conexión a *bases de datos* mediante *JDBC*. Esto es debido a que los *servlets* son un componente ideal para hacer las funciones de capa media en un sistema con una arquitectura de tres capas como la mostrada en la Figura:



Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los *drivers JDBC* no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de *driver*. En cualquier caso, tanto el *Servidor http* como el *Servidor de Base de Datos* pueden estar en la misma máquina, aunque en sistemas empresariales de cierta importancia esto no suele ocurrir con frecuencia.

JDBC (Java DataBase Connectivity) es una parte del API de *Java* que proporciona clases para conectarse con bases de datos. Dichas clases forman parte del package *java.sql*, disponible en el *jdk 1.1.7* y en *jdk 1.2*. El nombre **JDBC** es fonéticamente similar a **ODBC (Open DataBase Connectivity)**, que es el estándar más extendido para conectar PCs con bases de datos.

Formas de seguir la trayectoria de los usuarios (clientes)

Los *servlets* permiten seguir la trayectoria de un cliente, es decir, obtener y mantener una determinada información acerca del cliente. De esta forma se puede **tener identificado a un cliente** (usuario que está utilizando un browser) durante un determinado tiempo. Esto es muy importante si se quiere disponer de aplicaciones que impliquen la ejecución de varios *servlets* o la ejecución repetida de un mismo *servlet*. Un claro ejemplo de aplicación de esta técnica es el de los **comercios vía Internet** que permiten llevar un **carrito de la compra** en el que se van guardando aquellos productos solicitados por el cliente. El cliente puede ir navegando por las distintas secciones del comercio virtual, es decir realizando distintas conexiones **HTTP** y ejecutando diversos *servlets*, y a pesar de ello no se pierde la información contenida en el carrito de la compra y se sabe en todo momento que es un mismo cliente quien está haciendo esas conexiones diferentes.

El mantener información sobre un cliente a lo largo de un proceso que implica múltiples conexiones se puede realizar de tres formas distintas:

1. Mediante *cookies*
2. Mediante *seguimiento de sesiones (Session Tracking)*
3. Mediante la *reescritura* de URLs y paso de parámetros en formulario (*Request*)

2. Introducción a JSP

La tecnología JSP (*Java Server Pages*) es una especificación abierta desarrollada por Sun Microsystems como un alternativa a Active Server Pages (ASP) de Microsoft, y son un componente dominante de la especificación de Java 2 Enterprise Edition (J2EE). Muchos de los servidores de aplicaciones comercialmente disponibles (como BEA WebLogic, IBM WebSphere, Live JRun, Orion, etcétera) ya utilizan tecnología JSP.

Esta tecnología permite desarrollar páginas web con contenido dinámico y supone una evolución frente a la tecnología CGI, y los Servlets. Un fichero JSP puede contener etiquetas HTML normales, y elementos especiales para generar el contenido dinámico.

Al mismo tiempo permite una separación en n capas de la arquitectura de la aplicación web y se integra perfectamente con todas las API's empresariales de Java: JDBC, RMI (y CORBA), JNDI, EJB, JMS, JTA, ...

Estructura de una página JSP

Una página JSP es básicamente una página Web con HTML tradicional y código Java. La extensión de fichero de una página JSP es ".jsp" en vez de ".html" o ".htm", y eso le dice al servidor que esta página requiere un manejo especial que se conseguirá con una extensión del servidor o un plug-in.

Un ejemplo sencillo:

```
<%@ page language="java" contentType="text/html" %>
<html>
  <head>
    <title>Hola, mundo!!</title>
  </head>
  <body>
    <h1>Hola, mundo!</h1>
    Hoy es <%= new java.util.Date() %>.
  </body>
</html>
```

Directivas

Las directivas JSP son instrucciones procesadas por el motor JSP cuando la página JSP se traduce a un servlet. Las directivas usadas en este ejemplo le dicen al motor JSP que incluya ciertos paquetes y clases. Las directivas están encerradas entre etiquetas de directiva <%@ y %>.

```
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.PortableRemoteObject" %>
<%@ page import="Beans.*" %>
```

Declaraciones

Las declaraciones JSP nos permiten configurar variables para su uso posterior en expresiones o scriptlets. También podemos declarar variables dentro de expresiones o scriptlets en el momento de usarlas. El ámbito es toda la página JSP, no hay concepto de variables de ejemplar. Es decir, no tenemos que declarar variables de ejemplar para usar en más de una expresión o scriptlet. Las declaraciones van encerradas entre etiquetas de declaración `<%! y %>`. Podemos tener varias declaraciones. Por ejemplo,

```
<%! double bonus; String text; %> .
<%! String strMult, socsec; %>
<%! Integer integerMult; %>
<%! int multiplier; %>
<%! double bonus; %>
```

Scriptlets

Los scriptlets JSP nos permiten embeber segmentos de código java dentro de una página JSP. El código embebido se inserta directamente en el servlet generado que se ejecuta cuando se pide la página. Este scriptlet usa las variables declaradas en las directivas descritas arriba. Los Scriptlets van encerradas entre etiquetas `<% y %>`.

```
<%
strMult = request.getParameter("MULTIPLIER");
socsec = request.getParameter("SOCSEC");
integerMult = new Integer(strMult);
multiplier = integerMult.intValue();
bonus = 100.00;
%>
```

Variables Predefinidas

Un scriptlet puede usar las siguientes variables predefinidas: `session`, `request`, `response`, `out`, e `in`. Este ejemplo usa la variable predefinida `request`, que es un objeto `HttpServletRequest`. De igual forma, `response` es un objeto `HttpServletResponse`, `out` es un objeto `PrintWriter`, e `in` es un objeto `BufferedReader`. Las variables predefinidas se usan en los scriptlets de la misma forma que se usan en los servlets, excepto que no las declaramos.

```
<%
strMult = request.getParameter("MULTIPLIER");
socsec = request.getParameter("SOCSEC");
integerMult = new Integer(strMult);
multiplier = integerMult.intValue();
bonus = 100.00;
%>
```

Expresiones

Las expresiones JSP nos permiten recuperar dinámicamente o calcular valores a insertar directamente en la página JSP. En este ejemplo, una expresión recupera el número de la seguridad social desde el bean de entidad `Bonus` y lo pone en la página JSP.

```
<H1>Bonus Calculation</H1>
Social security number retrieved:
<%= record.getSocSec() %>
<P>
Bonus Amount retrieved: <%= record.getBonus() %>
<P>
```

Etiquetas específicas de JSP

La especificación JavaServer Pages 1.1 define etiquetas específicas de JSP que nos permiten extender la implementación JSP con nuevas características y ocultar mucha complejidad a los diseñadores visuales que necesitan buscar la página JSP y modificarla

```

<%@ page language="java" contentType="text/html" %> JSP element
<html> template text
<body bgcolor="white">
<jsp:useBean JSP element
id="usrInfo"
class="com.ora.jsp.beans.userInfo.UserInfoBean">
<jsp:setProperty name="userInfo" property="*/">
</jsp:useBean>
The following information was saved: template text
<ul>
<li>User Name:
<jsp:getProperty name="userInfo" JSP element
property="userName"/>
<li>Email Address: template text
<jsp:getProperty name="userInfo" JSP element
property="emailAddr"/>
</ul>
</body> template text
</html>
    
```

Fig. Estructura de una página JSP

Procesamiento de la página JSP

Cuando se llame a ésta página (date.jsp), será compilada (por el motor JSP) en un Servlet Java. En este momento el Servlet es manejado por el motor Servlet como cualquier otro Servlet. El motor Servlet carga la clase Servlet (usando un cargador de clases) y lo ejecuta para crear HTML dinámico para enviarlo al navegador, como se ve en la Figura 2. Para este ejemplo, el Servlet crea un objeto Date y lo escribe como un String en el objeto out, que es el stream de salida hacia el navegador.

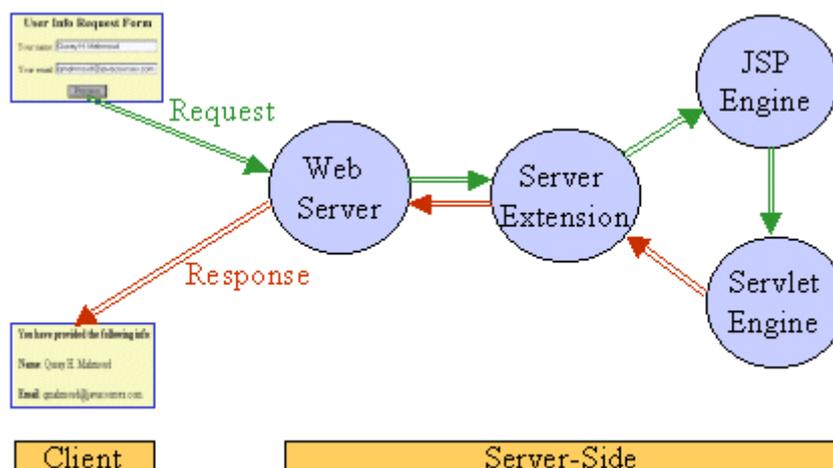


Figura. 2**Objetos implícitos**

El motor JSP nos ofrece instancias de un conjunto de clases. Son objetos ya establecidos, que no tenemos más que usar (no hay que instanciarlos). Deben utilizarse dentro del código Java.

Algunos objetos implícitos:

page (*javax.servlet.jsp.HttpJspPage*): Instancia del servlet de la página. Esto es sólo un sinónimo de `this`, y no es muy útil en Java. Fue creado como situación para el día que el los lenguajes de script puedan incluir otros lenguajes distintos de Java.

config (*javax.servlet.ServletConfig*): Datos de configuración del servlet.

request (*javax.servlet.http.HttpServletRequest*): Datos de la petición, incluyendo los parámetros. Este es el `HttpServletRequest` asociado con la petición, y nos permite mirar los parámetros de la petición (mediante `getParameter`), el tipo de petición (GET, POST, HEAD, etc.), y las cabeceras HTTP entrantes (cookies, Referer, etc.). Estrictamente hablando, se permite que la petición sea una subclase de `ServletRequest` distinta de `HttpServletRequest`, si el protocolo de la petición es distinto del HTTP. Esto casi nunca se lleva a la práctica.

response (*javax.servlet.http.HttpServletResponse*): Datos de la respuesta. Este es el `HttpServletResponse` asociado con la respuesta al cliente. Como el stream de salida tiene un buffer, es legal seleccionar los códigos de estado y cabeceras de respuesta, aunque no está permitido en los servlets normales una vez que la salida ha sido enviada al cliente.

out (*javax.servlet.jsp.JspWriter*): Flujo de salida para el contenido de la página. Este es el `PrintWriter` usado para enviar la salida al cliente. Sin embargo, para poder hacer útil el objeto `response` esta es una versión con buffer de `PrintWriter` llamada `JspWriter`. Podemos ajustar el tamaño del buffer, o incluso desactivar el buffer, usando el atributo `buffer` de la directiva `page`. Se usa casi exclusivamente en scriptlets ya que las expresiones JSP obtienen un lugar en el stream de salida, y por eso raramente se refieren explícitamente a `out`.

session (*javax.servlet.http.HttpSession*): Datos específicos de la sesión de un usuario. Este es el objeto `HttpSession` asociado con la petición. Las sesiones se crean automáticamente, por esto esta variable se une incluso si no hubiera una sesión de referencia entrante. La única excepción es usar el atributo `session` de la directiva `page` para desactivar las sesiones, en cuyo caso los intentos de referenciar la variable `session` causarían un error en el momento de traducir la página JSP a un servlet.

application (*javax.servlet.ServletContext*): Datos compartidos por todas las páginas de una aplicación. El `ServletContext` obtenido mediante `getServletConfig().getContext()`.

pageContext (*javax.servlet.jsp.PageContext*): Datos de contexto para la ejecución de la página. JSP presenta una nueva clase llamada `PageContext` para encapsular características de uso específicas del servidor como `JspWriters` de alto rendimiento. La

idea es que, si tenemos acceso a ellas a través de esta clase en vez directamente, nuestro código seguirá funcionando en motores servlet/JSP "normales".

exception (*java.lang.Throwable*): Errores o excepciones no capturadas.

Ejemplo:

```
<%  
String strParam = request.getParameter("nombre_del_parametro");  
out.println( strParam );  
>%
```

3. EJBs

Un EJB es un componente del lado del servidor que encapsula la lógica del negocio de una aplicación. En cualquier aplicación, los *beans enterprise* implementan los métodos de la lógica del negocio, que pueden ser invocados por clientes remotos para acceder a los servicios importantes proporcionados por la aplicación.

Beneficios

Los EJBs simplifican el desarrollo de grandes aplicaciones empresariales seguras y distribuidas por las siguientes razones:

- **Los desarrolladores pueden concentrarse en solventar la lógica del negocio:** el contenedor EJB proporciona servicios a nivel del sistema como el control de transacciones y las autorizaciones de seguridad. Por lo tanto, los desarrolladores no tienen que preocuparse de estos problemas.
- **Cientes pequeños:** Los desarrolladores no tienen que desarrollar código para las reglas de negocio o accesos a bases de datos; pueden concentrarse en la presentación del cliente. El resultado final es que los clientes son pequeños, y esto es especialmente importante para clientes que se ejecutan en pequeños dispositivos con recursos limitados.
- **Desarrollo rápido:** Los EJBs son componentes portables, y por lo tanto los ensambladores de aplicaciones pueden construir nuevas aplicaciones desde beans existentes. Las aplicaciones resultantes se pueden ejecutar en cualquier servidor compatible J2EE.

Componentes

Hay dos tipos principales de componentes EJB : **session** y **entity**. Un EJB de sesión se usa para realizar una tarea para un cliente, y un EJB de entidad es específico del dominio y se usa para representar un objeto de entidad del negocio que existe en un almacenamiento persistente. Sin embargo, los beans de entidad y de sesión tienen algunas diferencias que podemos ver en la siguiente tabla:

| EJB de Sesión | EJB de Entidad |
|--------------------------------------|---|
| Transitorio | Persistente |
| Puede ser usado por un sólo cliente. | Puede ser usado por muchos clientes. |
| No tiene identidad | Tiene una identidad (como una clave primaria) |

Desarrollar EJBs

Para desarrollar EJBs, todo bean enterprise necesita:

- Un **interface remoto** que exponga los métodos que soporta bean enterprise.
- Un **interface home** que proporciona los métodos del ciclo de vida del bean enterprise.

- Una clase de implementación, incluyendo la lógica de negocio que necesite.

EJBs contra Servlets

A primera vista, los EJBs y los Servlets son tecnologías similares porque ambos son componentes distribuidos del lado del servidor. Sin embargo, hay una diferencia importante entre los dos en el tipo de solución que ofrecen; los EJBs no pueden aceptar peticiones HTTP. En otras palabras, los EJBs no pueden servir peticiones que vienen directamente desde un navegador Web, y los servlets sí pueden. Servlets y JSPs se pueden usar para implementar presentación y control web, pero al contrario que los EJBs, no pueden manejar transacciones distribuidas. Los EJBs pueden ser llamados desde cualquier cliente basado en Java.

¿Cuándo usar EJBs?

Los EJBs son buenos para las aplicaciones que tienen alguno de estos requerimientos:

- **Escalabilidad:** si tenemos un número creciente de usuarios, los EJBs nos permitirán distribuir los componentes de nuestra aplicación entre varias máquinas con su localización transparente para los usuarios.
- **Integridad de Datos:** los EJBs nos facilitan el uso de transacciones distribuidas.
- **Variedad de clientes:** si nuestra aplicación va a ser accedida por una variedad de clientes (como navegadores tradicionales o navegadores WAP), se pueden usar los EJBs para almacenar el modelo del negocio, y se puede usar una variedad de clientes para acceder a la misma información.

4. Algunos ejemplos

Elementos de Script

En el ejemplo `date.jsp` se usa todo el nombre de la clase `Date` incluyendo el nombre del paquete, lo que podría llegar a ser tedioso. Si queremos crear un ejemplar de la clase `Date` usando simplemente: `Date today = new Date();` sin tener que especificar el path completo de la clase, usamos la directiva `page` de esta forma:

Ejemplo #1: fecha.jsp

```
<%@page import="java.util.*" %>
<HTML>
  <HEAD>
    <TITLE>JSP Example</TITLE>
  </HEAD>
  <BODY BGCOLOR="ffffcc">
    <CENTER>
      <H2>Date and Time</H2>
      <%
        java.util.Date today = new java.util.Date();
        out.println("Today's date is: "+today);
      %>
    </CENTER>
  </BODY>
</HTML>
```

Todavía hay otra forma de hacer lo mismo usando la etiqueta `<%=` escribiendo: `Today's date is: <%= new Date() %>`

Como podemos ver, se puede conseguir el mismo resultado usando diferentes etiquetas y técnicas. Hay varios elementos de script JSP. Hay algunas reglas convencionales que nos ayudarán a usar más efectivamente los elementos de Script JSP.

- Usamos `<% ... %>` para manejar declaraciones, expresiones, o cualquier otro tipo de código válido.
- Usamos la directiva `page` como en `<%@page ... %>` para definir el lenguaje de script. También puede usarse para especificar sentencias `import`. Aquí hay un ejemplo:

```
<%@page language="java" import="java.util.*" %>
```

- Usamos `<%! %>` para declarar variables o métodos. Por ejemplo:

```
<%! int x = 10; double y = 2.0; %>
```

- Usamos `<%= ... %>` para definir una expresión y forzar el resultado a un `String`. Por ejemplo: `<%= a+b %> 0 <%= new java.util.Date() %>`.
- Usamos la directiva `include` como en `<%@ include ... %>` para insertar el contenido de otro fichero en el fichero JSP principal. Por ejemplo:

```
<%@include file="copyright.html" %>
```

Manejar Formularios

Una de las partes más comunes en aplicaciones de web es un formulario HTML donde el usuario introduce alguna información como su nombre y dirección. Usando JSP, los datos del formulario (la información que el usuario introduce en él) se almacenan en un objeto `request` que es enviado desde el navegador hasta el contenedor JSP. La petición es procesada y el resultado se envía a través de un objeto `response` de vuelta al navegador. Estos dos objetos están disponibles implícitamente para nosotros.

Para demostrar como manejar formularios HTML usando JSP, aquí tenemos un formulario de ejemplo con dos campos: uno para el nombre y otro para el email. Como podemos ver, el formulario HTML está definido en un fichero fuente JSP. Se utiliza el método `request.getParameter` para recuperar los datos desde el formulario en variables creadas usando etiquetas JSP.

La página `procesar.jsp` imprime un formulario o la información proporcionada por el usuario dependiendo de los valores de los campo del formulario. Si los valores del formulario son `null` se muestra el formulario, si no es así, se mostrará la información proporcionada por el usuario. Observa que el formulario es creado y manejado por el código del mismo fichero JSP.

Ejemplo #2: procesar.jsp

```
<HTML>
  <HEAD>
    <TITLE>Formulario Ejemplo</TITLE>
  </HEAD>

  <BODY BGCOLOR="#ffffff">
    <% if (request.getParameter("name")==null &&
        request.getParameter("email")== null) { %>
      <CENTER>
        <H2>User Info Request Form</H2>
        <FORM METHOD="GET" ACTION="procesar.jsp">
          <P>Nombre: <input type="text" name="name" size=26>
          <P>email: <input type="text" name="email" size=26>
          <P><input type="submit" value="Process">
        </FORM>
      </CENTER>
    <% } else { %>
      <!-- String name, email; -->
      <% name = request.getParameter("name");
        email = request.getParameter("email"); %>
      <P><B>Ha introducido la siguiente información</B>:
      <P><B>Nombre</B>: <%= name %>
      <P><B>Email</B>: <%= email %>
    <% } %>
  </BODY>
</HTML>
```

Comentarios: En el contexto de JSP, los JavaBeans contienen la lógica de negocio que devuelve datos a un script en una página JSP, que a su vez formatea los datos devueltos por el componente JavaBean para su visualización en el navegador. Una página JSP utiliza un componente JavaBean fijando y obteniendo las propiedades que proporciona.

Hay muchos beneficios en la utilización de JavaBeans para mejorar las páginas JSP:

- Componentes Reutilizables: diferentes aplicaciones pueden reutilizar los mismos componentes.

- Separación de la lógica de negocio de la lógica de presentación: podemos modificar la forma de mostrar los datos sin que afecte a la lógica del negocio.
- Protegemos nuestra propiedad intelectual manteniendo secreto nuestro código fuente.

Uso de JavaBeans con JSP

Ahora, veamos como modificar el ejemplo anterior, `procesar.jsp` para usar JavaBeans. En el formulario anterior había dos campos: `name` y `email`. En JavaBeans, son llamados propiedades. Por eso, primero escribimos un componente JavaBean con métodos `setX` `getX`, donde `X` es el nombre de la propiedad. Por ejemplo, si tenemos unos métodos llamados `setName` y `getName` entonces tenemos una propiedad llamada `name`. El ejemplo #3 muestra un componente `FormBean`.

Los buenos componentes deben poder interoperar con otros componentes de diferentes vendedores. Por lo tanto, para conseguir la reutilización del componente, debemos seguir dos reglas importantes (que son impuestas por la arquitectura JavaBeans):

1. Nuestra clase bean debe proporcionar un constructor sin argumentos para que pueda ser creado usando `Beans.instantiate`.
2. Nuestra clase bean debe soportar persistencia implementando el interface `Serializable` o `Externalizable`.

Ejemplo #3: FormBean.java

```
package userinfo;
import java.io.*;

public class FormBean implements Serializable {
    private String name;
    private String email;
    public FormBean() {
        name = null;
        email = null;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getEmail() {
        return email;
    }
}
```

Para poder usar el componente `FormBean` en el fichero JSP, necesitamos ejemplarizar el componente. Esto se hace usando la etiqueta `<jsp:useBean>`. La siguiente línea `<jsp:setProperty>` se ejecuta cuando se ha ejemplarizado el bean, y se usa para inicializar

sus propiedades. En este caso, ambas propiedades (`name` y `email`) se configuran usando una sola sentencia. Otra posible forma de configurar las propiedades es hacerlo una a una, pero primero necesitamos recuperar los datos desde el formulario. Aquí tenemos un ejemplo de como configurar la propiedad `name`:

```
<%! String yourname, youremail; %>
<% yourname = request.getParameter("name"); %>
<jsp:setProperty name="formbean" property="name"
value="<%=yourname%>" />
```

Una vez que se han inicializado las propiedades con los datos recuperados del formulario, se recuperan los valores de las propiedades usando `<jsp:getProperty>` en la parte `else`, como se ve en el Ejemplo #4:

Ejemplo #4: procesar2.jsp

```
<jsp:useBean id="formbean" class="userinfo.FormBean" />
<jsp:setProperty name="formbean" property="*" />
<HTML>
<HEAD>
<TITLE>Form Example</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffffcc">
<% if (request.getParameter("name")==null
  && request.getParameter("email") == null) { %>
<CENTER>
<H2>User Info Request Form </H2>
<form method="GET" action="procesar2.jsp">
<P>
Your name: <input type="text" name="name" size=27>
<p>
Your email: <input type="text" name="email" size=27>
<P>
<input type="submit" value="Process">
</FORM>
</CENTER>
<% } else { %>
<P>
<B>You have provided the following info</B>:
<P>
<B>Name</B>: <jsp:getProperty name="formbean" property="name" />
<P>
<B>Email</B>: <jsp:getProperty name="formbean" property="email" />
<% } %>
</BODY>
</HTML>
```

6. Ejercicios prácticos

Se muestra a modo de ejemplo un pequeño código que consta de 3 páginas:

- `index.jsp`: Muestra y envía una serie parámetros a la página `parámetros.jsp`.
- `parametros.jsp`: Recoge los parámetros de la página anterior y mete alguno de ellos en la session. Enlaza con la página `session.jsp`.
- `session.jsp`: Si viene de la página `parámetros`, mostrará los parámetros que tiene. Si se pone directamente la URL en el navegador no muestra nada.

Se propone como ejercicio sobre el problema del Portal de coleccionistas del vino, realizar lo que sería una página de selección de productos para añadir a una cesta, ver el contenido de la misma (otra página) y procesar la compra mediante un pequeño formulario que pase los parámetros a una página de confirmación (cuatro en total).

5. Próximas sesiones

JSP y XML en la tecnología Web

JSTL y desarrollar Etiquetas JSP Personalizadas

Implementación de MVC mediante Struts

EJB y acceso a datos (JDBC e Hibernate)

Depuración de aplicaciones Web

La capa cliente y AJAX

La capa servidor y uso de Servicios Web

TODO

Anexo I. Lenguajes del lado cliente y servidor

| Lenguajes del lado cliente | |
|---|---|
|  | <p>El lenguaje llamado HTML indica al navegador donde colocar cada texto, cada imagen o cada video y la forma que tendrán estos al ser colocados en la página.</p> <p>El lenguaje consta de etiquetas que tienen esta forma <code></code> o <code><P></code>. Cada etiqueta significa una cosa, por ejemplo <code></code> significa que se escriba en negrita (bold) o <code><P></code> significa un párrafo, <code><A></code> es un enlace, etc. Casi todas las etiquetas tienen su correspondiente etiqueta de cierre, que indica que a partir de ese punto no debe de afectar la etiqueta. Por ejemplo <code></code> se utiliza para indicar que se deje de escribir en negrita. Así que el HTML no es más que una serie de etiquetas que se utilizan para definir la forma o estilo que queremos aplicar a nuestro documento. <code>Esto está en negrita</code>.</p> <p>Esta página es un claro ejemplo de uso del HTML.</p> |
| <h3>JavaScript</h3> | <p>JavaScript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Su uso se basa fundamentalmente en la creación de efectos especiales en las páginas y la definición de interactividades con el usuario.</p> <p>Las sentencias escritas en javascript se encapsulan entre las etiquetas <code><script></code> y <code></script></code>. por ejemplo, si en el código de una página Web incluimos la sentencia</p> <pre data-bbox="288 1406 986 1503"><script> window.alert("Bienvenido a mi sitio web. Gracias...") </script></pre> <p>al abrir la página con el navegador se nos mostrará una ventana de bienvenida </p> |
|  | <p>APPLETS </p> <p>Es otra manera de incluir código a ejecutar en los clientes que visualizan una página web. Se trata de pequeños programas hechos en Java, que se transfieren con las páginas</p> |

web y que el navegador ejecuta en el espacio de la página.

Los applets de Java están programados en Java y precompilados, es por ello que la manera de trabajar de éstos varía un poco con respecto a los lenguajes de script como Javascript. Los applets son más difíciles de programar que los scripts en Javascript y requerirán unos conocimientos básicos o medios del lenguaje Java.

La principal ventaja de utilizar applets consiste en que son mucho menos dependientes del navegador que los scripts en Javascript, incluso independientes del sistema operativo del ordenador donde se ejecutan. Además, Java es más potente que Javascript, por lo que el número de aplicaciones de los applets podrá ser mayor.

Como desventajas en relación con Javascript cabe señalar que los applets son más lentos de procesar y que tienen espacio muy delimitado en la página donde se ejecutan, es decir, no se mezclan con todos los componentes de la página ni tienen acceso a ellos. Es por ello que con los applets de Java no podremos hacer directamente cosas como



abrir ventanas secundarias, controlar Frames, formularios, capas, etc.

VBScript

Es un lenguaje de programación de scripts del lado del cliente, pero sólo compatible con Internet Explorer. Es por ello que su utilización está desaconsejada a favor de Javascript.

Está basado en Visual Basic, un popular lenguaje para crear aplicaciones Windows. Tanto su sintaxis como la manera de trabajar están muy inspirados en él. Sin embargo, no todo lo que se puede hacer en Visual Basic lo podremos hacer en Visual Basic Script, pues este último es una versión reducida del primero.

El modo de funcionamiento de Visual Basic Script para construir efectos especiales en páginas web es muy similar al utilizado en Javascript y los recursos a los que se puede acceder también son los mismos: el navegador.



Flash es una tecnología, y un programa, para crear efectos especiales en páginas web. Con Flash también conseguimos hacer páginas dinámicas del lado del cliente. Flash en realidad no es un lenguaje; Sin embargo, si tuvieramos que catalogarlo en algún sitio quedaría dentro del ámbito de las páginas dinámicas de cliente.

Para visualizar las "películas" Flash, nuestro navegador debe tener instalado un programita (plug-in) que le permita visualizarlas.



CSS, es una tecnología que nos permite crear páginas web de una manera más exacta. Gracias a las CSS somos mucho más dueños de los resultados finales de la página,

pudiendo hacer muchas cosas que no se podía hacer utilizando solamente HTML, como incluir márgenes, tipos de letra, fondos, colores... Incluso podemos definir nuestros propios estilos en un archivo externo a nuestras páginas; así, si en algún momento queremos cambiar alguno de ellos, automáticamente se nos actualizarán todas las páginas vinculadas de nuestro sitio.

CSS son las siglas de Cascading Style Sheets, en español Hojas de estilo en Cascada.

Lenguajes del lado servidor

CGI

Es el sistema más antiguo que existe para la programación de las páginas dinámicas de servidor. Actualmente se encuentra un poco desfasado por diversas razones entre las que destaca la dificultad con la que se desarrollan los programas y la pesada carga que supone para el servidor que los ejecuta.

Los CGI se escriben habitualmente en el lenguaje Perl, sin embargo, otros lenguajes como C, C++ o Visual Basic pueden ser también empleados para construirlos.

Perl

Perl es un lenguaje de programación interpretado, al igual que muchos otros lenguajes de Internet como Javascript o ASP. Esto quiere decir que el código de los scripts en Perl no se compila sino que cada vez que se quiere ejecutar se lee el código y se pone en marcha interpretando lo que hay escrito. Además es extensible a partir de otros lenguajes, ya que desde Perl podremos hacer llamadas a subprogramas escritos en otros lenguajes. También desde otros lenguajes podremos ejecutar código Perl.



ASP (Active Server Pages) es la tecnología desarrollada por Microsoft para la creación de páginas dinámicas del servidor. ASP se escribe en la misma página web, utilizando el lenguaje Visual Basic Script o Jscript (Javascript de Microsoft).



PHP es el acrónimo de Hipertext Preprocesor. Es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.



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

Con JSP podemos crear aplicaciones web que se ejecuten en variados servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Por tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual.

Lenguajes del lado cliente-servidor

dhtml

DHTML no es precisamente un lenguaje de programación. Más bien se trata de una nueva capacidad de la que disponen los navegadores modernos, por la cual se puede tener un mayor control sobre la página que antes.

Cualquier página que responde a las actividades del usuario y realiza efectos y funcionalidades se puede englobar dentro del DHTML, pero en este caso nos referimos más a efectos en el navegador por los cuales se pueden mostrar y ocultar elementos de la página, se puede modificar su posición, dimensiones, color, etc.

DHTML nos da más control sobre la página, gracias a que los navegadores modernos incluyen una nueva estructura para visualizar en páginas web denominada capa. Las capas se pueden ocultar, mostrar, desplazar, etc.

Para realizar las acciones sobre la página, como modificar la apariencia de una capa, seguimos necesitando un lenguaje de programación del lado del cliente como Javascript o VBScript.

En la actualidad, DHTML también puede englobar la programación en el servidor.

Dentro del concepto de DHTML se engloban también las Hojas de Estilo en Cascada o CSS (Cascade Style Sheets), que veremos a continuación.

XML

XML es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes.

Anexo II. Instalación de Tomcat para uso de JSP

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 instaló 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 de 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 compilación 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 que queda así:

```
<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.

Referencias

- <http://plato.acadiau.ca/courses/comp/dsilver/2513/EclipseAndTomcatTutorial/> -
“Tutorial de Integración de Eclipse y Tomcat”
- <http://www.programacion.net/tutorial/j2ee/5>. “Introducción a programación J2EE - SUN (trad. J.A.Palos)”
- www.desarrolloweb.com – Varias referencias
- www.adelat.org . “Portales en Internet”. Aníbal de la Torre - 2006
- www.sicaman-nt.com – Dpto. Desarrollo – SICAMAN Nuevas Tecnologías, SL