ESTADO DEL ARTE EN LA GENERACIÓN DE CONTENIDO DINÁMICO PARA LA WEB

La generación de contenido dinámico para la web, requiere que el servidor realice algún tipo de procesamiento adicional sobre la petición HTTP iniciada por el cliente, con el fin de generar una respuesta personalizada.

Como solución a esta carencia la mayoría de los sistemas de generación de contenido dinámico (**ColdFusion** de Allaire, **ASP** de Microsoft, **Server-Side JavaScript** de Netscape, **PHP,...**) posibilitan el uso de lenguajes de script. La utilización de estos lenguajes no requiere el ciclo clásico editar-compilar-linkar, posibilitando así una rápida codificación y visión de resultados. Debido a que las etiquetas HTML proporcionan un marco donde el contenido dinámico generado se inserta, estas herramientas se llaman comúnmente *template systems*.

A continuación se tratarán las siguientes tecnologías:

- Common Gateway Interface (CGI)
- ColdFusion
- Active Server Pages (ASP)
- WebClass
- Server Side JavaScript (SSJS)
- PreProcessor HiperText (PHP)
- Servlets
- JavaServer Page (JSP)
- Fast Common Gateway Interface (FastCGI)
- Zope
- Poor's Men Zope (PMZ)
- Y otras tecnologías

Breve introducción al protocolo HTTP

HyperText Transfer Protocol es el protocolo del nivel aplicación de la World Wide Web, y está estandarizado por W3C. Como todo protocolo, es un conjunto de reglas de comunicación para intercambiar datos o multimedia (gráficos, audio,...) Su especificación actual es HTTP/1.1 , que considera la existencia de ordenadores intermedios, como proxi y gateways, caché de ficheros, mantenimiento de conexiones abiertas y otros aspectos.

Las conexiones HTTP se realizan con el protocolo de trasporte TCP/IP. El cliente abre una conexión en la IP de un servidor en un puerto determinado, por defecto el 80. Entonces transmite su solicitud, que incluye:

- Tipo de operación que solicita el cliente (obtención del recurso, envío de parámetros, o información del recurso).
- Identificación del documento, fichero o recurso que quiere recuperarse mediante un *Uniform Resource Identifier* (URI oURL).
- La versión del protocolo HTTP que implementa (HTPP/1.0 ó HTTP/1.1).
- Y una serie de modificadores aplicables a la misma petición, como por ejemplo, versión y nombre del programa que realiza la petición (User-Agent), tipo de datos MIME del fichero de respuesta que se espera (Accept)...
- Adicionalmente, pueden incluirse datos en la petición (parámetros de un <FORM>) dentro del cuerpo de ésta.

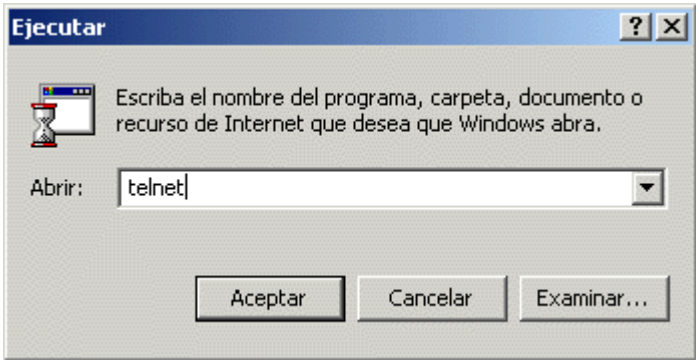
El servidor, por su parte, incluye en su respuesta:

- Una línea de estado con el código y mensaje de éxito o error según corresponda, además de la versión del protocolo que implementa.
- Información propia del servidor (nombre y versión del programa).
- Información sobre el documento o recurso solicitado (fecha de última modificación, tamaño,...)
- Y el propio documento o recurso solicitado.

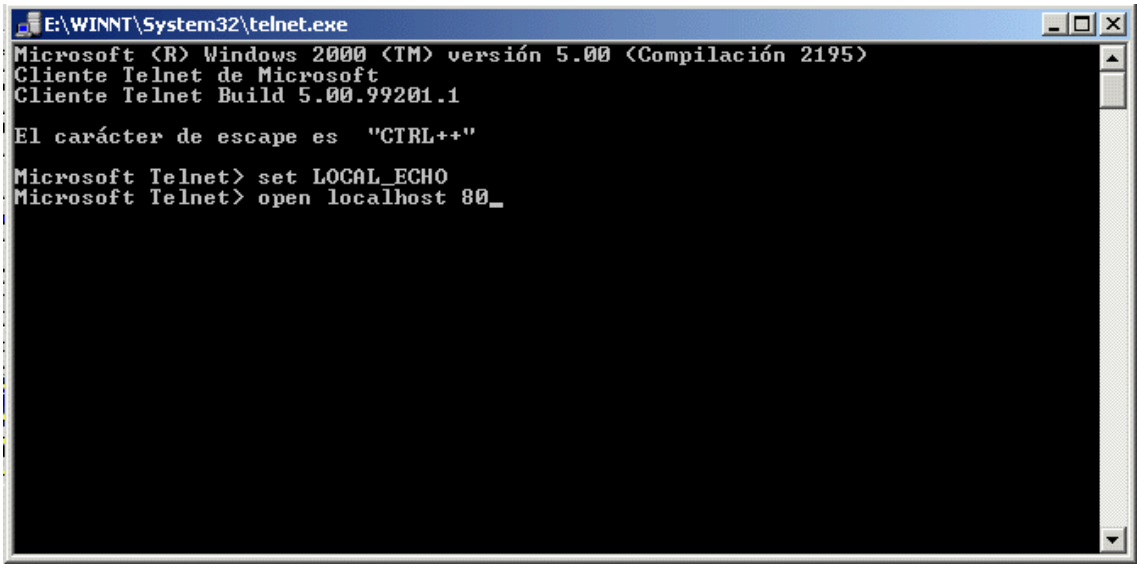
Y finalmente, el servidor cierra la conexión, aunque este último aspecto depende de la versión HTTP que implemente el servidor. La especificación HTTP/1.1 implementa el mantenimiento de conexiones abiertas, por lo que es necesario incluir el modificador *Connection: close* para que el servidor cierre la conexión después de dar la respuesta. Las conexiones abiertas deben usarse cuando un cliente vaya a solicitar un gran número de recursos del servidor en un corto espacio de tiempo.

Los servidores HTTP/1.0 cierran la conexión después de enviar la respuesta.

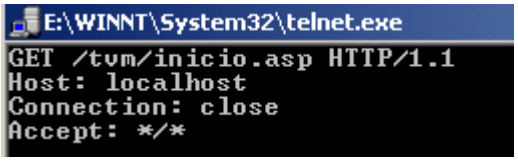
Puede utilizarse el programa TELNET (en Windows, seleccionar *Inicio*, *Ejecutar*, escribir *Telnet* y *Aceptar*) para ver el funcionamiento del protocolo HTTP sobre TCP/IP.



Debe activarse la opción de terminal Eco local para ver el texto que se introduzca en la consola (en Windows'98, menú *Terminal*, opción *Preferencias* y en Windows'2000 desde línea de comandos escribir *set LOCAL_ECHO*).



Para cada recurso hay que realizar una petición al servidor.



El cliente puede comprobar el resultado de cada petición en la línea de estado que se encuentra en la cabecera de la respuesta.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 15 Aug 2001 18:01:35 GMT
Connection: close
Content-Length: 3659
Content-Type: text/html
Set-Cookie: ASPSESSIONIDQQGQQJAG=EGLDILFBJDEIPPIMJKODPJFJ; path=/
Cache-control: private


<HTML>
<HEAD>

<BODY text="Teal">
<body topMargin=0 leftmargin=0 bgcolor=linen>

<!-- Encabezado com·n de las pβginas -->
<TABLE border=0 width=640 >
<TR>
<TD align=left><A href="inicio.asp"><IMG
```

Estos códigos de respuesta enviados por el servidor podemos verlos en la Tabla siguiente:

Códigos de respuesta enviados por el servidor
Información 1xx
100 Continue
101 Switching Protocols
Éxito 2xx
200 OK
201 Created
202 Accepted
203 Non-Authoritative Information
204 No Content
205 Reset Content
206 Partial Content
Redireccionamiento 3xx
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
306 (Unused)
307 Temporary Redirect
Error del cliente 4xx
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict

410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
Error del servidor 5xx
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

Para salir de telnet desde la línea de comandos tecleamos *quit*.

Solicitud HTTP

Una solicitud se compone de una línea de solicitud, uno o más campos de encabezado de solicitud opcionales y un cuerpo de entidad opcional. Las líneas se separan por medio de un retorno de carro para avance de línea (crlf: carriage-return(line-feed). El cuerpo de entidad va precedido por una línea en blanco. He aquí los detalles:

```
<method><resource identifier><HTTP versión><crlf>
[<Header> : <value>]<crlf>
…
[<Header> : <value>]<crlf>
      blank line <crlf>
[Entity body]
```

La línea de solicitud se compone de tres campos de texto, separados por espacios en blanco. El primer campo especifica el método –o comando- por aplicar al recurso de un servidor. El método más común es GET, por medio del cual se le pide al servidor en viar copia del recurso al cliente. En el siguiente recuadro de detalles se explican los demás métodos de HTTP. El segundo campo especifica el nombre del recurso destino; es el URL sin el protocolo ni el nombre de dominio del servidor. El tercer campo identifica la versión del protocolo usada por el cliente; por ejemplo, HTTP/1.0.

Los campos de encabezamiento de solicitud ofrecen información adicional sobre la solicitud, y sobre el cliente mismo, al servidor. Cada campo de encabezado consiste en un nombre, seguido por dos puntos (:) y el valor del campo. El orden en que se transmiten los campos de encabezado no es significativo.

El cuerpo de entidad es empleado en ocasiones por los clientes para transmitirle al servidor información de masa.

Métodos de HTTP			
Método	**HTTP/1.0**	**HTTP1.1**	**Descripción del método**
GET	S	S	Recuperar el URL especificado
GET condicional	N	S	Recupera el recurso si se cumplen las condiciones incluidas en la petición. Si añadimos el parámetro If-Modified-Since, con una fecha en formato HTTP, el servidor comprueba que el recurso ha sido actualizado con posterioridad a la fecha recibida en la petición e incluirá el recurso en su respuesta. En otro caso, indicará que el recurso no ha sido modificado y por lo tanto no será enviado. También puede incluir parámetros en la URL. Por ejemplo, una petición HTTP con GET condicional sería: *GET /default.htm http/1.1* *If-Modified-Since: Sat, 02 Dec 2000 01:15:48 GMT* *Host: www.microsoft.com* *Connection: close* *Accept: */**
HEAD	S	S	Idéntico a GET, salvo que el servidor no envía el documento en respuesta; sólo envía los encabezados. Los clientes lo usan para obtener metadatos de recursos o para probar la validez de vínculos de hipertexto.
POST	S	S	Enviar estos datos al URL especificado
PUT	N	S	Almacenar estos datos en el URL especificado, en reemplazo del contenido anterior.
PATCH	N	S	Similar a PUT, salvo que contiene una lista de diferencias entre la versión original de URL y el contenido deseado tras la aplicación del método.
COPY	N	S	Copiar el recurso identificado por el URL en la(s) ubicacion(es) especificada(s).
MOVE	N	S	Trasladar el recurso indicado por el URL a la(s) ubicación(es) especificada(s). Este método es equivalente a COPY/DELETE (copiar/borrar).
DELETE	N	S	Borrar el recurso identificado por el URL.
LINK	N	S	Establecer una o más relaciones de vinculación entre el recurso identificado por el URL y otros recursos.
UNLINK	N	S	Eliminar una o más relaciones de vinculación en el URL especificado.
TRACE	N	S	Notificar todo lo que se reciba del cliente en el cuerpo de entidad de la respuesta.

OPTIONS	N	S	Solicita información sobre las opciones de comunicación disponibles en la cadena de solicitud/respuesta para el URL especificado. Permite a los clientes determinar las capacidades de un servidor sin recuperar un recurso.
WRAPPED	N	S	Permite que solicitudes se envuelvan en conjunto y quizá también se codifiquen para reforzar la seguridad y/o privacidad de la solicitud. El servidor de destino debe desenvolver el mensaje y cederlo al manipulador apropiado.

En las siguientes tablas listamos los campos de encabezado HTTP. Adviértase que HTTP evoluciona muy rápido. HTTP/1.1 soporta más de 41 campos de encabezado, contra 17 de HTTP/1.0. Para ayudarle toda esta información, hemos dividido los campos de encabezado en cuatro categorías: 1) encabezados generales, que pueden incuirse en mensajes tanto de solicitud como de respuesta; 2) encabezados de solicitudes, que sólo pueden aparecer en mensajes de solicitud; 3) encabezados de respuestas, que sólo pueden aparecer en mensajes de respuesta, y 4) encabezados de entidad, que pueden aparecer en solicitudes o respuestas. Los encabezados de entidad describen el contenido de los datos del mensaje: por ejemplo, un documento devuelto por un servidor o datos de formatos enviados por un cliente.

Encabezados generales de HTTP			
Encabezado	**HTTP/1.0**	**HTTP/1.1**	**Descripción del método**
Cache-Control	N	S	Contiene instrucciones de solicitud/respuesta de memoria caché.
Connection	N	S	Contiene información que puede no transmitirse a gateways.
Date	S	S	Contiene fecha y hora de origen del mensaje.
Forwarded	N	S	Contiene información usada por gateways para rastrear pasos intermedios y evitar lazos de solicitudes.
Keep-Alive	N	S	Contiene información de diagnóstico.
MIME-Version	S	S	Contiene la versión de MIME empleada para codificar el mensaje.
Pragma	S	S	Contiene instrucciones de implementación (por ejemplo, sin memoria caché).
Upgrade	N	S	Lista protocolos de comunicación adicionales que soporta un cliente y que querría usar previo acuerdo del servidor.
Encabezados de solicitudes de HTTP			
Encabezado	**HTTP/1.0**	**HTTP/1.1**	**Descripción del método**
Accept	N	S	Lista contenido aceptable de tipo/subtipo MIME.

Accept-Chars	N	S	Lista contenido aceptable de tipo/subtipo MIME.
Accept-Encoding	N	S	Lista conjuntos de caracteres aceptables.
Accept-Language	N	S	Lista codificaciones aceptables, como compresión y compresión con zip.
Authorization	S	S	Lista ejemplos naturales aceptables.
From	S	S	Transmite esquemas de autenticación y codificación del usuario.
Host	N	S	Contiene dirección de correo electrónico en Internet del usuario.
If-Modified-Since	S	S	Contiene nombre de anfitrión destino.
Proxy-Authorization	N	S	Contiene una fecha/hora usada por GET para la descarga condicional de documentos.
Refer	S	S	Permite a clientes presentar su identidad a un representante.
Unless	N	S	URL del documento de origen de esta solicitud.
User-Agent	S	S	Lista condiciones de encabezado que deben cumplirse para que un método sea aplicado a un recurso.
Encabezados de respuestas de HTTP			
Encabezado	HTTP/1.0	HTTP/1.1	Descripción
Location	S	S	Envía la ubicación exacta del recurso.
Proxy-Authenticate	N	S	Envía el esquema de codificación/autorización usado en esta sesión.
Public	N	S	Lista todos los métodos no estándar soportados por un servidor.
Retry-After	N	S	Indica (en segundos) cuándo volver a intentar un servicio.
Server	S	S	Envía información sobre el software usado en el servidor.
WWW-Authenticate	S	S	Envía esquema de codificación/autorización que desea usar el servidor en todas sus interacciones en el Web.
Encabezados de entidad de HTTP			
Encabezado	**HTTP/1.0**	**HTTP/1.1**	**Descripción**
Allow	S	S	Lista los métodos soportados por el recurso de URL.
Content-Encoding	S	S	Especifica codificación de respuesta, como comprensión y comprensión con zip.
Content-Language	N	S	Especifica lenguaje natural de respuesta (por ejemplo, francés).

Content-Length	S	S	Extensión en bytes del cuerpo de entidad.
Content-Type	S	S	Tipo de contenido MIME de respuesta.
Content-Version	N	S	Contiene un número de versión del recurso.
Derived-From	N	S	Identifica la versión anterior del recurso.
Expires	S	S	Contiene fecha/hora después de la cual el documento caduca.
Last-Modified	S	S	Contiene fecha/hora de la modificación más reciente del recurso.
Link	N	S	Contiene la información de vinculación del documento.
Title	N	S	Contiene el título del documento.
Transfer-Encoding	N	S	Identifica una transformación aplicada al documento (o cuerpo del mensaje).
URL-Header	N	S	Contiene la parte del nombre del recurso del URL.

Respuesta HTTP

La sintaxis de una respuesta HTTP consiste en una línea de encabezado de respuesta, uno o más campos de encabezado de respuesta opcionales y un cuerpo de entidad opcional. Las líneas se separan por medio de un retorno de carro/avance de línea (crlf). El cuerpo de entidad debe ir precedido por un espacio en blanco. He aquí los detalles:

La línea de encabezado de respuesta envía la versión de HTTP, el estado de la respuesta y explicación del estado de devolución.

Los campos de encabezado de respuesta envían información en la que se describen los atributos del servidor y el documento HTML enviado al cliente. Cada campo de encabezado se compone de un nombre, seguido por dos puntos (:) y el valor del campo. El orden en que el servidor remite los campos de encabezado no es importante.

El cuerpo de entidad contiene, por lo general, un documento HTML que un cliente haya solicitado.

```
<HTTP Versión><result code>[<explanation>]<crlf>
[<Header> : <value>]<crlf>
…
[<Header> : <Value>]<crlf>
   blank line  <crlf>
[Entity body]
```

Referencias

Cliente/Servidor Guía de supervivencia

Autores: Robert Orfali, Dan Harkey, Jeri Edwards

Editorial: McGraw-Hill

ISBN: 970-10-1760-9

Capítulo 27. Cliente/servidor en el Web: la era de hipertexto (página 463). Este capítulo explica el protocolo HTTP y contiene un tutorial de HTML.

http://www.w3c.org/Protocols/

Información sobre el protocolo HTTP.
http://www.ietf.org/rfc/rfc2068.txt
RFC protocolo HTTP versión 1.0
http://www.ietf.org/rfc/rfc2616.txt
RFC protocolo HTTP versión 1.1

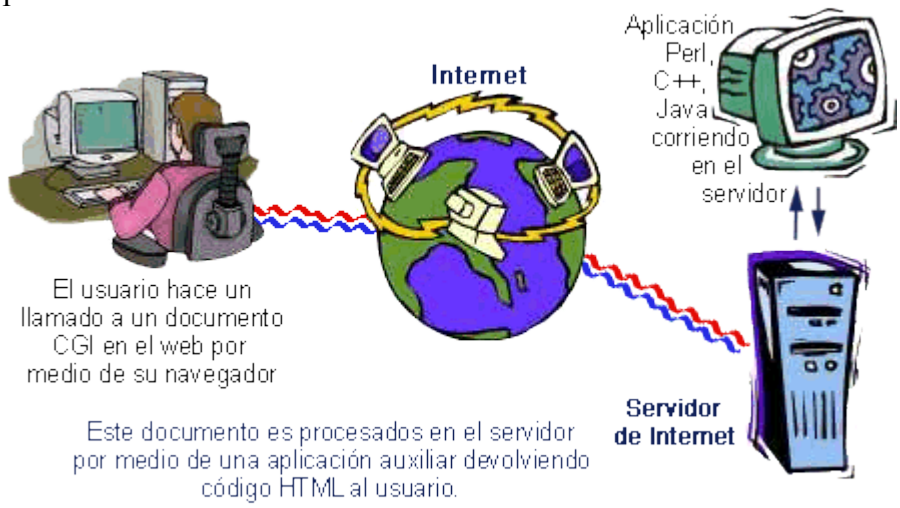
Evolución de tecnologías para la generación de contenido dinámico para la web

Common Gateway Interface. CGI

Introducción

En los primeros años de existencia de la web, tan sólo se podía acceder a información estática. De esta forma, la interactividad con el usuario no existía.

El tiempo fue pasando, y poco a poco las necesidades de un servicio interactivo vía web fueron creciendo. En aquél entonces es cuando se empezó a definir una interfaz, para permitir aumentar la interactividad ofrecida por los servidores web. Esta interfaz se dio en llamar interfaz CGI (Common Gateway Interface, Interfaz Común de Pasarela). La interfaz nos permite comunicar el servidor de web con otros programas que realicen tareas diversas. Estos programas se ejecutan como tareas independientes del servidor de web.



Utilidades de los CGIs

Un uso de los CGIs es el de los contadores de visitas, también llamados contadores de impactos, dando una idea del número de visitas que ha recibido.

Aunque su mayor funcionalidad es la generación de páginas HTML automáticamente teniendo en cuenta: consultas a bases de datos, día que se trate, datos recibidos en un formulario,...

Los populares Altavista, Yahoo, etc utilizan CGIs como motores de búsqueda. ¿Por qué aún hoy no se implementan con Java? La lentitud de Java respecto a un CGI es conocida por todos. Aunque en Java disponemos de JDBC para acceder a base de datos, no sería recomendable para una tarea como la que realiza un buscador de recursos.

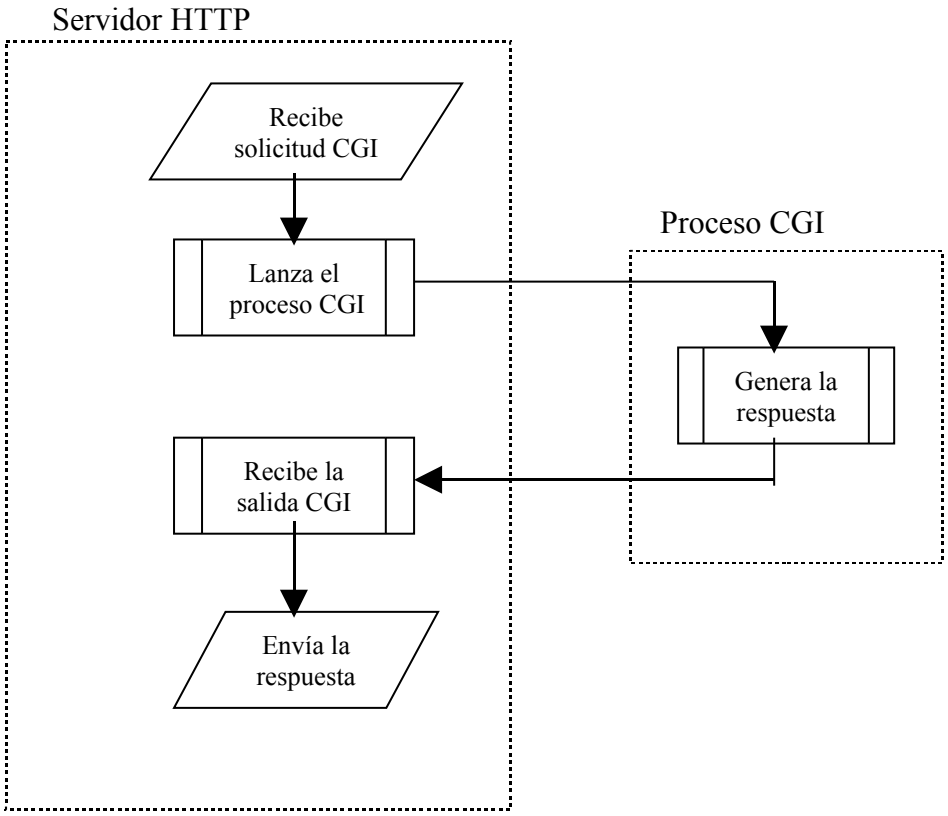
Cómo funciona un CGI

Hay varias formas de invocar a un CGI desde una página web. La más común es mediante un formulario, en el que se introducirán los datos que requiere el CGI y se enviarán al servidor.

Pero también pueden hacerse llamadas a CGIs cada vez que se hace referencia a un documento o una dirección, como un enlace o una imagen.

Así el funcionamiento de un CGI se puede esquematizar de la siguiente forma:

1. El usuario solicita una página web del servidor.
2. El servidor envía la página al usuario.
3. El usuario envía al servidor los datos para el CGI, (mediante un formulario, por ejemplo).
4. El servidor pasa los datos recibidos a la aplicación CGI.
5. La aplicación CGI procesa los datos y genera un resultado que es enviado al servidor.
6. El servidor reenvía los resultados de la aplicación CGI al usuario.



Hemos de distinguir entre tres métodos diferentes de envío de los datos al servidor. Para empezar, tenemos el método *GET*, en el que el programa CGI recibirá los datos a través de la variable de entorno *QUERY_STRING*. El programa deberá interpretar el contenido de esta cadena y usar los datos como sean necesarios. Este método es el indicado cuando no se van a modificar valores en el servidor, o cuando la cantidad de datos enviados no es superior a 1024 bytes, que es el límite de tamaño definido para una URL. Y es que los datos enviados siguiendo este método, se encapsulan en la URL.

Este encapsulado de los datos es muy habitual, y nos lo encontramos muy frecuentemente en las páginas que visitamos que hacen uso de CGIs, utilizando formularios para pasar unos pocos datos, como es el caso de los buscadores.

Un ejemplo de encapsulado de los datos en la URL sería el siguiente:

http://myhost/cgi-bin/search?nombre=valor

Aquí, el nombre del CGI que se va a ejecutar es *search*. Los parámetros se separan del nombre de la aplicación con el carácter de fin de interrogación (?). A partir de ese momento lo que aparecerán serán los parámetros para el programa CGI, generalmente en la forma *nombre = valor*. Hay que destacar tan solo dos puntos más. Aquellos caracteres que no se pueden usar en una URL, es decir, que forman parte de un juego de caracteres extendido, se codificarán sustituyéndose por el carácter % seguido del valor en hexadecimal del carácter. (Como el popular %7E, equivalente al carácter ~).También hay que tener en cuenta que los espacios serán sustituidos por el signo más (+).

Otro método muy usado es el método *POST*. Con él, los datos se pasan a través de la entrada estándar, es decir, *stdin*. Debemos considerar que el servidor no marcará el final de los datos con el carácter *EOF (End Of File)*. Así, para determinar

la cantidad de los datos deberemos acceder a la variable *CONTENT_LENGTH* que almacenará la cantidad en bytes de datos enviados. Este método es el idóneo cuando se envían grandes cantidades de datos, sobre todo cuando el método *GET* no es suficiente.

Finalmente tenemos el método *HEAD*, idéntico al método *GET*, con la salvedad de que el servidor tan solo envía las cabeceras http al navegador.

Para poder adaptarse mejor las aplicaciones CGI, además de los datos proporcionados por el usuario, cuentan con unas variables de entorno. Algunas de estas variables contienen valores que pueden ser importantes, como la variable *REQUEST_METHOD*, que contiene información acerca del método empleado en la solicitud. Otras son meramente informativas, como las que guardan la referencia al cliente utilizado por el usuario.

VARIABLES DE ENTORNO RELATIVAS AL SERVIDOR	
GATEWAY_INTERFACE	La versión de la interfaz CGI soportada por el servidor. Por ejemplo: CGI/1.2
SERVER_NAME	La dirección IP o el nombre del servidor.
SERVER_PORT	El puerto en el que el servidor recibió la solicitud. Generalmente se tratará del puerto 80.
SERVER_PROTOCOL	Nombre y versión del protocolo usado por el servidor para procesar la solicitud. Por ejemplo: HTTP/1.1
SERVER_SOFTWARE	Nombre, y posiblemente la versión, del software del servidor. Por ejemplo: Apache/1.3.9

VARIABLES DE ENTORNO RELATIVAS AL CLIENTE	
ACCEPT	Lista de los tipos de respuesta aceptados por esta solicitud.
ACCEPT_ENCODING	Lista de los tipos de codificación soportados por el cliente.
ACCEPT_LANGUAGE	Identifica el código ISO del lenguaje que el cliente espera recibir.
AUTHORIZATION	Identifica a los usuarios autentificados.
CHARGE_TO	Selección tarficiación automática (reservado para uso futuro).
FROM	Lista la dirección de correo electrónico del cliente.
IF_MODIFIED_SINCE	Acompaña a una solicitud de tipo GET, de forma que sólo se enviarán los datos si el documento es más actual respecto a la fecha indicada.
PRAGMA	Activa directivas del servidor o proxies para uso futuro.
REFERER	Identifica la URL del documento que dio el enlace al documento actual.
USER_AGENTE	Identifica el software usado por el

	cliente, normalmente incluyendo información acerca de la versión.

VARIABLES DE ENTORNO RELATIVAS A LA SOLICITUD	
AUTH_TYPE	Esquema de autentificación usado por el servidor. Ejemplo: basic.
CONTENT_FILE	Nombre del fichero que contiene los datos para la aplicación CGI. Sólo usada en WinCGI/Windows HTTPd.
CONTENT_LENGHT	Número de bytes enviados a la entrada estándar (STDIN), debido a una solicitud tipo POST.
CONTENT_TYPE	El tipo MIME de los datos que están siendo enviados. Por ejemplo:text/html.
OUTPUT_FILE	Nombre del fichero en el que el programa CGI debería escribir los datos. Sólo usada con WinCGI/Windows HTTPd.
PATH_INFO	Información adicional acerca de la ruta para el programa CGI, pasada como parte de la URL después del nombre del programa. Por ejemplo: /docs/septiembre.
PATH_TRANSLATED	La versión traducida de PATH_INFO, que apunta a la dirección absoluta. Por ejemplo: /home/staff/docs/septiembre.
QUERY_STRING	Los datos pasados al programa CGI como parte de la URL, consistente de cualquier cosa que vaya después del cierre de interrogación (?).
REMOTE_ADDR	La dirección IP de la máquina que realizó lasolicitud.
REMOTE_HOST	El nombre de la máquina que realizó la solicitud.
REMOTE_IDENT	El nombre del usuario, según se define en la RFC 931.
REMOTE_USER	Nombre del usuario, si ha sido autentificado.
REQUEST_LINE	La solicitud HTTP que ha sido enviada al servidor, al completo. Por ejemplo: GET /cgi-bin/cgi.pl? dato1=valor1&dato2= valor2 HTTP/1.1
REQUEST_METHOD	El método usado para pasar los datos como parte de la solicitud HTTP. Por ejemplo: GET.
SCRIPT_NAME	Nombre del CGI que está siendo ejecutado. Por ejemplo: cgi.pl

VARIABLES DE ENTORNO RELATIVAS AL PROTOCOLO HTTP	
HTTP_ACCEPT	Contiene los valores correspondientes a la cabecera Accept enviada por el cliente, y se corresponde con los tipos MIME que soporta.
HTTP_REFERER	Contiene la URL del formulario en el que se originó la petición al CGI.
HTTP_USER_AGENT	Contiene el nombre del cliente que realizó la solicitud.

Programando CGIs

La programación de CGIs puede realizarse en cualquier lenguaje. Desde PERL hasta C o C++, pasando por Pascal, un script de la shell de un sistema UNIX, e incluso un fichero bat de MS-DOS. Todo es válido mientras se sigan las especificaciones de la interfaz CGI. Incluso se pueden diseñar CGIs en Java, aunque no es habitual.

El lenguaje por excelencia más usado para la programación de CGIs es PERL (*Practical Extraction and Report Language*). Fundamentalmente debido a que se trata de un lenguaje idóneo para el manipulado de textos. En definitiva, tanto una página HTML como la comunicación seguida mediante el protocolo http no es más que simple texto.

A esto debemos añadirle que el servidor web más popular, el Apache, incluye un módulo para interpretar PERL directamente, sin tener que recurrir a ejecutar el intérprete cada vez que se solicita la ejecución de un CGI. Con esto, Apache gana en rendimiento y disminuye la carga de proceso del sistema.

Si vamos a programar un script que procese datos recibidos y genere un resultado en consecuencia, debemos saber que tenemos a nuestra disposición varias librerías o módulos que nos facilitarán ciertas tareas. En el caso de PERL, el módulo más importante, al que siempre haremos referencia cuando programemos un CGI, es el módulo *cgi.pm*.

Así, para simplificar tareas, simplememtne al principio de nuestro programa indicaremos que se va a usar ese módulo con *use CGI*.

Scripts CGI en Perl

Vamos a ver las principales rutinas para el trabajo con los datos enviados a través de un formulario. Como se comentó hace un momento el uso del módulo CGI facilitaría este trabajo. En el apartado de CGI del anexo A se implementan los scripts haciendo uso tanto de este módulo como de otras rutinas encontradas en la Web (las rutinas no están en este documento debido a la extensión).

El módulo CGI.pm, desarrollado por Lincoln D. Stein, utiliza el estilo de programación orientada a objetos. Necesitaremos la versión 5.0001 o superior para utilizarlo; Perl 5.0003 es la versión recomendada.

Los derechos del copyright del módulo CGI.pm pertenecen a Lincoln D. Stein, pero es posible copiarlo y distribuirlo gratuitamente. Puede obtener la última versión de CGI.pm en la página oficial de CGI.pm en el siguiente URL:

http://www.genomre.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html

En esta página Web,. Encontrará una completa documentación sobre CGI.pm e instrucciones sobre cómo descargar el módulo. Recuerde que deberá copiarlo en el directorio de librerías Perl de su sistema.

Al igual que la librería cgi-lib.pl (que comentaremos más tarde), CGI.pm permite aceptar una consulta enviada a través de un formulario HTMl y extraer fácilmente los parámetros enviados por el usuario. A diferencia de la interfaz procedural de cgi-lib.pl, CGI.pm proporciona su funcionalidad a través de un objeto CGI. Para utilizar el módulo CGI, introduzca la sentencia siguiente en su script Perl:

```
use CGI;
#Crear un objeto CGI
$query = new CGI;
```

La creación del objeto CGI desencadena el procesamiento de la consulta CGI y extrae todas las variables de ambiente importantes para la programación CGI. El valor devuelto, $query, es una referencia al objeto CGI. A partir de esta referencia, es posible acceder a los métodos del objeto CGI.

El procesamiento de la consulta CGI es sólo una parte del módulo CGI. Adicionalmente, el objeto CGI posee otras muchas capacidades, como las siguientes:

- Métodos para crear las cabeceras HTTP que se deben incluir en los documentos HTML que se envían de vuelta al usuario a partir del script.
- Métodos para crear formularios HTML.
- Métodos de gestión de capacidades avanzadas como marcos, cookies y JavaScript.

No necesitará conocer la funcionalidad completa del módulo CGI.pm. De hecho, sólo deberá conocer los métodos que le resulten útiles a la hora de escribir scripts que gestionen la entrada del usuario.

Una vez que haya creado el objeto CGI y obtenido una referencia a dicho objeto, podrá acceder a los parámetros introducidos en el formulario HTML de la siguiente forma:

```
# Con la variable anterior
$query->param('Nombre_Parametro');

# Para trabajar con las cabeceras tendríamos
print $query->header;

# El resto del documento HTML
print $query->start_html("Título de la página");

# Mostrar información en HTML
print $query->end_html;
```

La librería cgi-lib.pl de Steven Brenner, trabaja con la cadena de consulta CGI que el servidor Web envía al programa CGI. Usando la librería cgi-lib.pl se pueden crear rápidamente programas Perl que controlen la entrada de un formulario.

La librería cgi-lib.pl (incluida en el CD) cuenta con las siguientes subrutinas Perl:

Nombre	Descripción
ReadParse	Lee y procesa la consulta CGI para los métodos de envío GET y POST. El resultado del procesamiento se almacena en un array asociativo que se suministra como argumento. Devuelve VERDADERO si existe una consulta; FALSO en caso contrario.
PrintHeader	Muestra la cabecera MIME (Content-type: text/html) con una línea en blanco extra para indicar el final de la cabecera.
HtmlTop	Muestra el encabezado del documento HTML, incluyendo un título que se suministra como argumento.
HtmlBot	Muestra el pie de página del documento HTMl con las etiquetas de finalización </body> y </html>.
MethGet	Devuelve VERDADERO si la consulta CGI se transmitió como una petición GET.
MethPost	Devuelve VERDADERO si la consulta CGI se transmitió como una petición POST.
MyURL	Devuelve el URL HTTP completo del script.
CgiError	Muestra un documento HTMl con un mensaje de error. Si se suministra un array de cadenas como argumento, la primera cadena se utiliza como título y el resto se muestra en el cuerpo del documento HTML. Si no se proporcionan argumentos, se utiliza un mensaje de error genérico como cuerpo del documento HTML.
CgiDie	Se llama a la subrutina CgiError y el script termina su ejecución mediante la función die.
PrintVariables	Muestra las variables en un array asociativo que se suministra como argumento (utilice esta subrutina para mostrar los campos junto a sus valores correspondientes).

Es casi inmediato utilizar las subrutinas de la librería cgi-lib.pl. La secuencia general de llamadas a subrutinas son las siguientes:

- Llamar a la subrutina ReadParse para procesar la consulta CGI.
- Si ReadParse devuelve VERDADERO, procesar la consulta (que en este momento se encuentra disponible en un array asociativo). La salida del documento HTML se consigue llamando a las subrutinas PrintHeader, HtmlTop y HtmlBot.
- Llamar a la subrutina CgiDie o CgiError para informar sobre cualquier error que pudiese producirse en la generación del documento HTML.

Para comprender con mayor profundidad la secuencia anterior, estudie el siguiente script a modo de esquema:

```
   #!/usr/local/bin/perl

   # Se incluye la librería cgi-lib.pl mediante la función
require
   require "cgi-lib.pl";

   # Lectura y procesamiento de la entrada (en este caso el
   # resultado se encuentra en un array asociativo llamado
input)
   if(&ReadParse(*input))
   {

   # La lectura y procesamiento fue satisfactorio.
   # Accedemos a los campos específicos del array input. Por
   # ejemplo, si tenemos un campo llamado keyword,
   # el acceso al valor del campo
   # se realiza mediante la construcción: $input('keyword')

   # La salida HTML se obtiene mediante las rutinas
   # PrintHeader y HtmlTop
         print &PrintHeader;
         print &HtmlTop("Titulo de página");

   # Mostramos el texto que se desea incluir en el documento
HTML
   # Terminamos el documento mediante HtmlBot
         print &HtmlBot;
   }
   else{
   # Se desea generar un mensaje de error si ReadParse falla.
   # Se llama a las subrutinas Call CgiDie o CgiError
   # para conseguirlo
         &CgiDie("Error al leer la entrada","Mensaje error
descripción");
   }
```

Ejemplos de uso

En los siguientes ejemplos se hará uso de los lenguajes PERL y C, por ser los dos lenguajes más utilizados a la hora de programar CGIs.

1.- En el siguiente ejemplo se muestra un pequeño script en PERL que genera automáticamente una página HTML. Sería el típico *Hola mundo* pasado a script CGI.

```
#!/usr/bin/perl -w
print << FIN;
Content-type: text/html

<HTML>
  <HEAD>
  <TITLE>Hola Mundo</TITLE>
 </HEAD>
  <BODY>
  <H1>Texto html</H1>
  </BODY>
</HTML>
Fin
```

Este programa lo único que hace es generar una salida predefinida automáticamente, sin modificar nada, sin usar datos suministrados por el usuario ni nada por el estilo.

2.- Este CGI hecho en C devuelve la hora del servidor al cliente que realice la petición. Este ejemplo es recopilado de la ayuda de Borland C con la única inclusión de la línea en negrita. Con esta línea especificamos el contenido que recibe el navegador.

```
#include <time.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   time_t timer;
   struct tm *tblock;

      /* conseguir time  */
   timer = time(NULL);

      /* convertir date/time a una estructura */
   tblock = localtime(&timer);

      printf("Content-type: text/html%c%c",10,10);
      printf("<br><h1     align=center>Hora     local:
%s</h1>", asctime(tblock));

   return 0;
}
```

3.- Implementación en C del típico contador de visitas. Se podrían utilizar ficheros gif con la numeración del 0 al 9 y en la instrucción switch dependiendo del valor pasar una imagen u otra.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

long expo(long x, long y);

void main(int argc,char *argv, char *argp)
{
      int i=0, longitud=0;
      long numero=0;
      FILE *pf;
      char numero_s[10],digito='0';

      printf("Content-type: text/html%c%c",10,10);
      pf=fopen("numero.txt","r+");
      if(pf==NULL)
      {
            printf("Error:     No     puedo     abrir
numero.txt");
            return;
      }
```

```
        longitud=8;
        while((digito=fgetc(pf))!=EOF)
        {
                numero+=(digito - 48)*(expo(10,(longitud-
i)));
                i++;
        }
        numero++;

        sprintf(numero_s,"%09ld",numero);
        rewind(pf);
        fputs(numero_s,pf);
        fclose(pf);
        printf("<center><font size=3>");
        for(i=0;i<9;i++)
        {
                switch(numero_s[i])
                {
                        case '0':
                                printf("0");
                                break;
                        case '1':
                                printf("1");
                                break;
                        case '2':
                                printf("2");
                                break;

                        ...
                        case '9':
                                printf("9");
                                break;
                }
        }
        printf("</font></center>");
}

long expo(long x, long y)
{
        int a=0;
        long x2=x;

        if(y<0)
        {
                return 0;
        }

        if(y==0)
        {
                return 1;
        }

        for(a=1;a<y;a++)
        {
                x*=x2;
        }
        return x;
}
```

Referencias

Libros/Revistas:

Manual fundamental de Perl 5.

Autor: Naba Barkakati
Editorial: ANAYA S.A.
ISBN: 84-415-0378-8

Perl es un lenguaje script mundialmente utilizado en la programación de CGIs. Este manual es una primera referencia para el que quiera aprender este versátil lenguaje. Posee un manejo de cadenas bastante potente. También utilizado en la administración de sistemas.

Más PC n º 9.

Artículo: Páginas interactivas Programación de CGIs.
Páginas: 175-177
Editorial: Prensa Técnica.

Artículo que introduce en el mundo de los CGIs. Introduce las utilidades de los CGIs que aún se utilizan hoy en día.

Más PC n º 9.

Artículo: Introducción a la tecnología CGI.
Páginas: 188-191
Editorial: Prensa Técnica.

Artículo en el que se explican los CGIs en especial para su trabajo en sistemas LINUX.

La Biblia de la programación CGI.

Autor: Ed Tittel, Mark Gaither, Sebastián Hassinger y Mike Erwin.
Editorial: ANAYA S.A.
ISBN: 84-415-0182-3

Fundamentos de HTML y CGI en primeros capítulos para llegar a elementos y técnicas avanzadas de programación de CGI.

Enlaces:

http://www.ora.com/info/cgi/

Información sobre CGIs, y libros publicados sobre el tema por O'Reilly & Associates Inc.

http://www.worldwidemart.com/scripts/

Scripts CGI realizados por programadores de todo el mundo en todo tipo de lenguajes hasta ofertas de trabajo, pasando por libros, artículos de revistas y abundante documentación sobre la interfaz CGI.

http://www.cgi-world.com/

Scripts de todo tipo escritos fundamentalmente en PERL.

ColdFusion

Introducción

Este producto de Allaire, proporciona un conjunto de etiquetas similares a las de HTML, que encapsulan peticiones a bases de datos desde páginas web. Actualmente estas etiquetas se han extendido para soportar un amplio rango de orígenes de datos. Soporta plataformas UNIX y Microsoft Windows. Nos proporciona las herramientas para crear las páginas web a través de su propio lenguaje ColdFusion Markup Language (CFML), basado en etiquetas, para crear los scripts del servidor que interactúan con las bases de datos y nos facilitan la creación de las páginas dinámicas.

Es una herramienta de desenvolvimiento de aplicaciones web que permite crear páginas web dinámicas para Internet, intranets o extranets a través de integración sofisticada entre elementos como: base de datos, aplicaciones de e-mail, ...

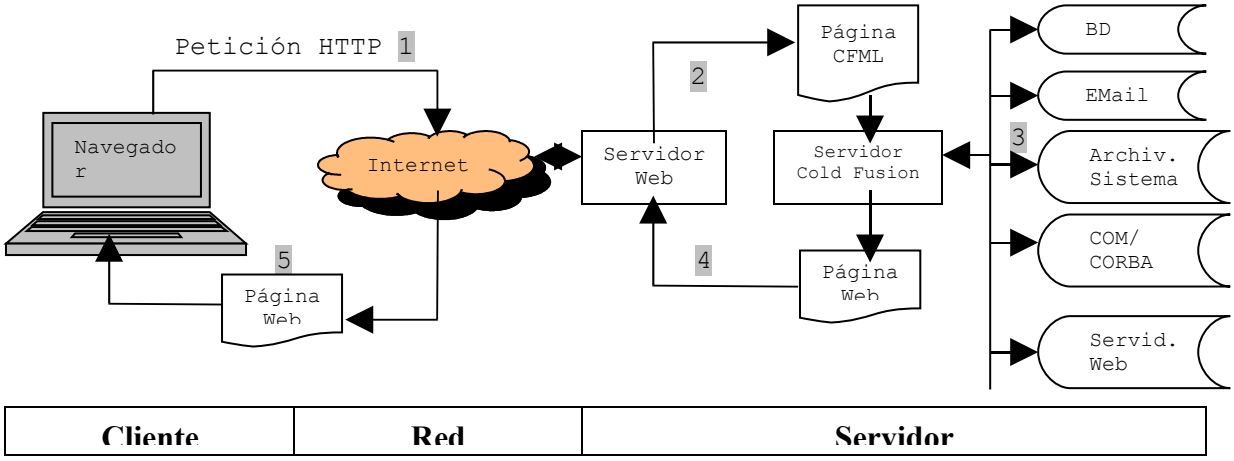
Esto significa que una vez instalado, tendremos acceso a base de datos SQL de forma dinámica, sin la necesidad de generar varias páginas para cada ítem de consulta. Podrá aún generar formularios de envío de e-mails, generar agendas y calendarios, etc.

Cold Fusion utiliza un lenguaje denominada CFML (Cold Fusion Markup Language).

Se basa en tags, como HTML, convirtiéndose en un lenguaje de fácil aprendizaje.

Cómo funciona CFML (Cold Fusion Markup Language)

Cuando una página dentro de la aplicación Cold Fusion es requerida por un browser (usuario), el Cold Fusion procesa el CFML, interactuando con sistemas back-end, y genera dinámicamente una página de retorno para el browser con los datos requeridos. El diagrama siguiente, muestra como Cold Fusion trabaja cuando un browser invoca una página CFML.



1. Cuando un usuario hace click en un botón “Submit” (Enviar) en un formulario o en un link de una página, el navegador envía una petición HTTP para el servidor Web vía Internet o intranet.
2. El servidor Web pasa los datos del cliente y la página apropiada para el Servidor Cold Fusion a través de la API.
3. Cold Fusion lee los datos del cliente y procesa el CFML dentro de la página. Basado en el CFML, el servidor interactúa con servidores de base de datos, archivos de sistema, servidores SMTP, y potencialmente otras aplicaciones y extensiones a través de la API Cold Fusion o a través de COM / DCOM.
4. Cold Fusion genera dinámicamente una página HTML que es retornada por el servidor Web.
5. El servidor Web entonces retorna la página HTML para el navegador del usuario.

Programando Cold Fusion Markup Language (CFML) y ejemplos de uso

En este apartado vamos a juntar las dos secciones que venimos tratando en las otras tecnologías: Programando en ... y Ejemplos de uso. Esto es debido a los variados ejemplos que se van incluyendo al explicar la programación en ColdFusion.

CFML dispone de un comprensivo ambiente de script basado en etiquetas, lo que hace el desarrollo de aplicaciones mucho más rápido y fácil. Por ser un lenguaje basado en etiquetas, el CFML se integra fácilmente al HTML.

Un template Cold Fusion (archivo .cfml), noes más que un archivo de texto, exactamente como un archivo .htm/.html, siendo posible ser editado a través de un simple editor de textos, como por ejemplo, Wordpad de Windows.

A través de sus tags se puede manipular variables, utilizar funciones de fecha, hora, matemáticas, búsqueda e string, hacer declaraciones condicionales, loop, utilizar declaraciones avanzadas SQL.

Aquí está un pequeño ejemplo de CFML:

```
<html>
<head>
<title>
¡Hola Mundo!
</title>
</head>
<body bgcolor=”#FFFFFF” text=”#000000">
<cfset hola=’Hola Mundo’>
<cfoutput>
<font size=”5">#hola#</font>
</cfoutput>
</body>
```

Veremos que las etiquetas de Cold Fusion (CFML) son similares a la sintaxis de las etiquetas HTML, pero, no iguales, estas non ayudarán a generar páginas dinámicas y mostrar los datos de una consulta en su base de datos sin la necesidad de generar varias páginas para eso.

Ya sabemos que un template Cold Fusion puede ser editado a través de un editor de textos, para esta primera aplicación de ejemplo es lo que basta, pero si se prefiere se puede editar con HomeSite o Cold Fusion Studio, ambos encontrados en site de Allaire (http://www.allaire.com).

Vamos asumir que ya configuramos nuestra fuente de datos con el nombre de agenda y que lee de la base de datos en Access llamada agenda.mdb.

En esta base de datos tenemos actualmente apenas una tabla llamada Personal, que tiene como campos Nombre y Teléfono.

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
</CFQUERY>
```

Comience criando en su template el acceso a su base de datos:

Estamos listando todos los ítems contenidos en la tabla Personal de base de datos. Y nombramos nuestra consulta como AgendaPersonal para que podamos exhibirla como sigue:

```
<CFOUTPUT QUERY= "AgendaPersonal" >
#nombre# - #telefono#
<HR>
</CFOUTPUT>
```

El resultado de esta consulta, asumiendo tener apenas tres registros en la base de datos, generará un HTML de la siguiente forma:

```
José da Silva – 11-234-9980
<HR>
Manuel Alves – 19-345-3328
<HR>
Maria Lima – 13-231-9915
<HR>
```

Note que dentro del tag CFOUTPUT fue colocada una etiqueta HTML, <HR>. No existen problemas en intercalar etiquetas CFML con etiquetas HTML, la excepción de comillas dobles (") y sostenido (#). Para estos casos debe repetirlos dos veces (##, """).

Para salvar esta aplicación es necesario que se coloque con extensión de archivo CFM, por tanto podemos llamar esta primera aplicación de agendapersonal.cfm.

Después de instalar el Cold Fusion, el servidor Web reconoce un archivo con la extensión CFM como siendo una aplicación del Cold Fusion Server, por tanto debemos colocar nuestros archivos cfm abajo del directorio principal del servidor Web (wwwroot) u otro mapeado por el servidor (pudiendo ser, por ejemplo, el directorio scripts).

Declaraciones SQL

Como ya vimos anteriormente, SQL es un lenguaje específica para tratamiento con base de datos, para poder tener mejores resultados en próximas aplicaciones es necesario que se tenga un buen conocimiento de esta lenguaje (mirar anexo SQL).

Vamos seleccionar todos los registros de la tabla Personal que contenga en el campo nombre la siguiente información: José da Silva.

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE nombre LIKE 'José da Silva'
</CFQUERY>
```

Note que estamos usando ahora la cláusula WHERE, que es responsable por los filtros de las consultas, y el término LIKE para comparar un campo texto. Si estuviésemos haciendo una comparación de números, utilizaremos el signo "=".

En este ejemplo será mostrado apenas un único registro como resultado de la consulta.

Para seleccionar todos los registros de la tabla que inicien con la sentencia "José" en el campo nombre, usamos el siguiente:

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE nombre LIKE 'José%'
</CFQUERY>
```

Si quisiéramos seleccionar registros que terminen con la sentencia "Silva" usamos:

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE nombre LIKE '%Silva'
</CFQUERY>
```

Ahora queremos registros que contengan la sentencia "ma" utilizamos:

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE nombre LIKE '%ma%'
</CFOUERY>
```

Note que de esta vez tendremos como respuesta dos registros.

CFIF, CFELSE y CFELSEIF

Estas son las etiquetas de condición utilizadas por Cold Fusion, con ellas podremos realizar las tareas condicionales de sus aplicaciones.

En la primera aplicación podemos hacer algunas consistencias para ilustrar esto.

Vamos hacer con que caso el contenido del campo de consulta sea "José da Silva", sea presentado "Sí", caso contrario irá ser presentado "Otros".

```
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel#
<CFIF nombre IS 'José da Silva'>
<br>Sí
<CFELSE>
<br>Otros
</CFIF>
<HR>
</CFOUTPUT>
```

Aquí utilizamos el operador IS para comparar el campo nombre con la sentencia. Podríamos estar utilizando el operador EQ. Y para números podríamos estar utilizando:

LT Menor que
LTE Menor o igual a
GT Mayor que
GTE Mayor o igual a

Ahora ilustraremos la utilización del CFELSEIF:

```
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel#
<CFIF nombre IS 'José da Silva'>
<br>Sí
<CFELSEIF nombre IS 'Manuel Alves'>
<br>Manuel
<CFELSEIF nombre IS 'Maria Lima'>
…
```

Entendiendo más sobre CFOUTPUT

La etiqueta CFOUTPUT es utilizada para mostrar resultados de consultas a base de datos y el resultado de operaciones con variables.

La etiqueta CFOUTPUT puede ir seguida del atributo *QUERY*, que es opcional e indica el nombre de la consulta que se pretende exhibir los datos.

La etiqueta CFOUTPUT tiene otros atributos:

GROUP Opcional. Este atributo no exhibe registros duplicados de acuerdo con el campo ordenado en la query. Este parámetro es case sensitive.

GROUPCASESENSITIVE Opcional. Este campo indica si el campo de opción GROUP será o no case sensitive.

STARTROW Opcional. Especifica una línea que se quiere iniciar a exhibir los datos.

MAXROWS Opcional. Especifica el número máximo de líneas que se quiere exhibir. Teniendo como base la datasource *agenda,* definida anteriormente, veremos ahora como utilizar estos nuevos parámetros de la etiqueta CFOUTPUT.

Uso del parámetro GROUP

Inserte dos registros más en nuestra base de datos:

Nombre	Tel
André Soares	11-234-9980
josé da silva	13-561-7898

Ahora crearemos el acesso a la base de datos, muy importante para la utilización del parámetro *GROUP*:

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
ORDER BY tel
</CFQUERY>
```

Observando arriba, notará que la cláusula *ORDER BY* mostrará sus registros de forma ordenada a través del campo *tel*. Creado el aceso a su base de datos, veremos ahora como usar el parámetro *GROUP*:

```
<CFOUTPUT QUERY="AgendaPersonal" GROUP="tel">
#nombre# - #telefono#
<HR>
</CFOUTPUT>
```

Note que como respuesta tendremos apenas cuatro registros, siendo que en la base de datos existen cinco. El registro no exhibido fue el de José da Silva, que tiene el mismo número de teléfono que André Soares, este que fue exhibido por ser la última ocurrencia encontrada en la base de datos.

Uso del parámetro GROUPCASESENSITIVE

Continuando con la misma base de datos, veremos ahora el parámetro GROUPCASESENSITIVE. Para eso precisaremos cambiar en la cláusula ORDER BY, el nombre del campo para nombre, quedando asi:

ORDER BY nombre

GROUPCASESENSITIVE sólo es empleado en campos string.

Ahora coloque la etiqueta CFOUTPUT de la siguiente forma:

```
<CFOUTPUT QUERY="AgendaPersonal" GROUP="nombre"
GROUPCASESENSITIVE="yes">
#nombre# - #telefono#
<HR>
</CFOUTPUT>
```

Veremos que como respuesta tendremos los cinco registros de la base de datos.

Cambie ahora el parámetro GROUPCASESENSITIVE para NO. Tendremos como respuesta apenas cuatro registros, siendo el registro de José da Silva no siendo mostrado.

Uso del parámetro STARTROW

Este parámetro es usado para indicar en que registro de nuestra query deseamos comenzar a exhibir. Para ilustrar mejor, hagamos de la siguiente forma:

```
<CFOUTPUT QUERY="AgendaPersonal" STARTROW="2">
#nombre# - #telefono#
<HR>
</CFOUTPUT>
```

Veremos que tenemos como respuesta cuatro registros, ya que iniciamos en el registro de número 2 de nuestra query.

Uso del parámetro MAXROWS

Usado para limitar el número máximo de registros de la query a ser exhibidos.

Vamos aumentar el ejemplo del parámetro STARTROW el parámetro MAXROWS, y ver como este nuevo parámetro funciona:

```
<CFOUTPUT QUERY="AgendaPersonal" STARTROW="2" MAXROWS ="3">
#nombre# - #telefono#
<HR>
</CFOUTPUT>
```

En este ejemplo tendremos como respuesta tres registros. Note que el parámetro MAXROWS nos limita al número máximo de 3 registros y no que la exhibición de los registros deba parar en el registro de número 3.

Insertando datos en una base de datos

Antes de iniciar inserte un campo más llamado código (número entero), como llave primaria, en la tabla Personal y coloque códigos en los registros ya existentes.

Siguiendo con nuestra primera aplicación, cree un archivo HTML conteniendo un formulario con la opción method = post, la action = http://su_servidor/scripts/Tutorial_1/inserte.cfm, tres campos de texto, de nombres: codigo, nombre y tel, que representan los nombres de los campos de la tabla Personal en nuestra base de datos. Cree también el botón de envío, *submit*.

La opción Action puede contener el nombre del archivo destino, el archivo html será grabado en el mismo directorio del archivo de aplicación. Ej.: action = inserte.cfm creado el archivo HTML, iremos ahora crear el archivo aplicación para insertar datos en nuestra base de datos.

Inserte.cfm

```
<CFINSERT DATASOURCE="Agenda" TABLENAME="Personal">
```

Entenderemos ahora un poco más sobre los atributos más utilizados de esta etiqueta:

DATASOURCE Requerido. Nombre del datasource que contiene la base de datos.

TABLENAME Requerido. Nombre de la tabla donde se irá insertando los datos.

USERNAME Opcional. Si es especificado, cancela el username especificado en el setup de ODBC.

PASSWORD Opcional. Si es especificado, cancela el password especificado en el setup de ODBC.

FORMFIELDS Opcional. Indica los campos que deben ser insertados. Los campos deben estar separados por comas.

Para que no haya problemas a la hora de insertar los datos, es necesario que el nombre de los campos en el formulario sean idénticos a los usados en la base de datos. Veremos más adelante como quebrar esta regla.

El CFML es muy parecido a HTML y por eso también acepta sus tags. Veamos a continuación:

Selecionando el registro a ser alterado

Primero precisamos crear una consulta en nuestra base de datos y mostrar el resultado de esta consulta para que podamos escoger que registro vamos alterar.

Para mostrar el resultado de esta consulta, será necesario crear un pequeño formulario con un campo escondido (hidden) para poder indicar a la aplicación que este es el registro que se desea alterar.

Altera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel# -
<FORM METHOD="POST" ACTION="formaltera.cfm">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="SUBMIT" VALUE="Alterar">
</FORM>
</CFOUTPUT>
```

Creando un formulario de alteración

El próximo paso es crear un formulario con los datos de registro escogido para ser alterados.

Crearemos entonces una nueva consulta en nuestra base de datos donde el número de campo sea igual al del registro escogido en la aplicación Altera.cfm. Hecho esto colocaremos sus valores en un formulario semejante al que usamos para insertar nuevos datos, visto anteriormente.

FormAltera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE codigo = #form.codigo#
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
<FORM METHOD="POST" ACTION="AlteraFim.cfm">
<INPUT TYPE="TEXT" NAME="nombre" VALUE="#nombre#">
<INPUT TYPE="TEXT" NAME="tel" VALUE="#tel#">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="SUBMIT" VALUE="Alterar">
</FORM>
</CFOUTPUT>
```

Finalizando la alteración

El último paso es crear el archivo que registrará las alteraciones en la base de datos. Semejante a la etiqueta <CFINSERT>, utilizaremos la etiqueta <CFUPDATE> para alterar los datos del registro seleccionado.

AlteraFim.cfm

```
<CFUPDATE DATASOURCE="Agenda" TABLENAME="Personal">
<HTML>
<HEAD>
<TITLE>Alteración en base de datos</TITLE>
</HEAD>
<BODY>
<H2>Datos Alterados con Éxito!</H2>
```

```
</BODY>
</HTML>
```

Algunos atributos utilizados en esta tag:

DATASOURCE Requerido. Nombre del datasource que contiene la base de datos

TABLENAME Requerido. Nombre de la tabla donde se irá insertando los datos.

USERNAME Opcional. Si se ha especificado, cancela el username especificado en el setup de ODBC.

PASSWORD Opcional. Si se ha especificado, cancela el password especificado en el setup de ODBC.

FORMFIELDS Opcional. Indican que campos deben ser insertados. Los campos deben estar separados por comas. Insertando, Actualizando y Eliminando registros con CFQUERY

Vimos anteriormente como insertar y actualizar registros con <CFINSERT> y <CFUPDATE> respectivamente. Mas en el caso que necesitemos de inserciones y actualizaciones más complejas podemos utilizar instrucciones SQL para ejecutarlas. Utilizaremos aqui los formularios de ejemplo utilizados en los tópicos anteriores.

Insertando

Para insertar un registro basta substituir la tag <CFINSERT> como sigue:

Inserte.cfm

```
<CFQUERY DATASOURCE="Agenda" NAME="Inserte">
INSERT INTO Personal (codigo, nome, tel)
VALUES (#form.codigo#, '#form.nome#', '#form.tel#')
</CFQUERY>
<HTML>
<HEAD>
<TITLE>Inserción en Base de Datos</TITLE>
</HEAD>
<BODY>
<H2>Datos registrados con éxito!</H2>
</BODY>
</HTML>
```

Alterando

Para alterar un registro basta substituir la tag <CFUPDATE> como sigue:

AlteraFim.cfm

```
<CFQUERY DATASOURCE="Agenda" NAME="Altera">
UPDATE Personal
SET nombre = '#form.nombre#',
tel = '#form.tel#'
WHERE codigo = #form.codigo#
</CFQUERY>
<HTML>
<HEAD>
<TITLE>Alteración en Base de Datos</TITLE>
</HEAD>
```

```
<BODY>
<H2>Datos Alterados con Éxito!</H2>
</BODY>
</HTML>
```

Eliminando registros

Para excluir registros de otro modo sin utilizar instrucciones SQL a través de la tag <CFQUERY>. Para ilustrarnos como funciona, vamos comenzar seleccionando que registro deseamos excluir de nuestra base de datos.

Exclui.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel# -
<FORM METHOD="POST" ACTION="excluifim.cfm">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="SUBMIT" VALUE="Eliminar">
</FORM>
</CFOUTPUT>
```

Ahora vamos ver como funciona la aplicación que eliminado en definitivo el registro:

ExcluiFim.cfm

```
<CFQUERY DATASOURCE="Agenda" NAME="Exclui">
DELETE FROM Personal
WHERE codigo = #form.codigo#
</CFQUERY>
<HTML>
<HEAD>
<TITLE>Exclusión en Base de Datos</TITLE>
</HEAD>
<BODY>
<H2>Datos Excluídos con Éxito!</H2>
</BODY>
</HTML>
```

Pasanod variables a través de Form y URL

Como vimos para insertar, actualizar y eliminar registros en una base de datos, y dado momento, fue mostrado una instrucción de la siguiente forma:

#form.codigo#

Esta instrucción significa una variable que fue pasada a través de un formulario para una aplicación cfm. Por tanto, la instrucción de encima está ligada a una etiqueta de un formulario:

<input type="hidden" name="codigo" value="codigo">

#form.codigo#

Otro modo de pasar variables/atributos para una aplicación cfm es a través de URL como por ejemplo:

http://dominio/cgi-bin/aplicación.cgi?variable=valor

en este caso la aplicación recibe una variable, que viene identificada después de la interrogación (?) junto con su valor.

En la aplicación esta variable será referenciada de forma semejante al de variables pasadas a través de un formulario. Basta sustituir form por url:

#url.variable#

Para pasar más de una variable para la aplicación, basta separar el conjunto variable y valor por un ampersand (&).

http://dominio/cgi-bin/aplicación.cgi?var1=valor1&var2=valor2

Para entender mejor la utilización de estas dos instrucciones, vamos a modificar la aplicación utilizada anteriormente, y vamos mostrar ésta pasando las variables a través de la URL.

Altera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
</CFQUERY>

<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel#
<FORM METHOD="POST" ACTION="formaltera.cfm">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="submit" VALUE="Alterar">
</FORM>
</CFOUTPUT>
```

FormAltera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE codigo = #form.codigo#
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
<FORM METHOD="POST" ACTION="AlteraFin.cfm">
<INPUT TYPE="TEXT" NAME="nombre" VALUE="#nombre#">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="SUBMIT" VALUE="Alterar">
</FORM>
</CFOUTPUT>
```

Ahora veamos como hacer esto con una URL.

Altera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
```

```
#nombre# - #tel#
<a ref.=formaltera.cfm?codigo=#codigo#>Altera</a>
</CFOUTPUT>
```

FormAltera.cfm

```
<CFQUERY DATASOURCE="agenda" NAME="AgendaPersonal">
SELECT * FROM Personal
WHERE codigo = #url.codigo#
</CFQUERY>
<CFOUTPUT QUERY="AgendaPersonal">
<FORM METHOD="POST" ACTION="alteraFin.cfm">
<INPUT TYPE="TEXT" NAME="nombre" VALUE="#nombre#">
<INPUT TYPE="HIDDEN" NAME="codigo" VALUE="#codigo#">
<INPUT TYPE="SUBMIT" VALUE="Alterar">
</FORM>
</CFOUTPUT>
```

Generando y manipulando variables con CFSET

Vimos como son pasadas las variables a través de aplicaciones, ahora vamos a ver como crear variables en la misma aplicación. La etiqueta CFSET es la responsable de la creación de variables y de la manipulación de sus valores.

Con esta etiqueta podremos usar valores estáticos, parámetros dinámicos o expresiones para crear una variable. A continuación se muestra su sintaxis:

<CFSET Nombre_de_la_variable = Valor Estático, parámetro o expresión>

Veamos ahora ejemplos de la utilización de estos tipos:

Valores estáticos

Vamos a crear una variable llamada verifica y colocar su valor como José da Silva:

```
<CFSET verifica="José da Silva">
```

Por tanto en nuestra aplicación agenda ilustrada en el primer ejemplo de este tutorial podríamos estar utilizando algo así:

```
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel#
<CFSET verifica="Jose da Silva">
<CFIF nombre IS verifica>
<br> Sí
<CFELSE>
<br> Otro
</CFIF>
<HR>
</CFOUTPUT>
```

Parámetros Dinámicos

La etiqueta CFSET también acepta parámetros pasados por otras aplicaciones, como variables pasadas a través de formularios y URL.

Para crear una variable que tenga un valor o contenido de una variable pasada por un formulario, basta con usar:

<CFSET nombre_variable= form.variable>

Otro tipo de uso de CFSET es la combinación de parámetros dinámicos con string:

```
<CFSET nombre_variable='#PARÁMETRO# texto'>
En nuestra aplicación agenda podríamos tener algo así:
```

```
<CFOUTPUT QUERY="AgendaPersonal">
#nombre# - #tel#
<CFSET verifica='Jose da Silva'>
<CFIF nombre Is verifica>
      <CFSET msg ='#nombre# su apellido es Sí'>
<CFELSE>
      <CFSET msg='#nombre# usted no tiene apellido'>
</CFIF>
<HR>
</CFOUTPUT>
```

Expresiones

La etiqueta CFSET también acepta como contenido expresiones matemáticas y de concatenación de strings y valores numéricos.

<CFSET ValorTotal= 2*(10+5)>

<CFSET Divida= 'Jose da Silva' & ValorTotal>

Creando Cookies

Las cookies son mecanismos para almacenar información de cada usuario que visita nuestro site. Cold Fusion puede utilizar estas informaciones, almacenarlas en browsers en nuestras aplicaciones. La etiqueta que Cold Fusion usa para esto es CFCOOKIE.

La etiqueta CFCOOKIE tiene los siguentes atributos:

NAME Requerido. Nombre de la variable cookie a ser creada.

VALUE Opcional. Indica el valor determinado para la variable.

EXPIRES Opcional. Este atributo determina el tiempo de expirado de la cookie. Puede ser especificado con una fecha (como 31/12/2002), número de días (como 1, 10, 100), NOW, o NEVER. Utilizando NOW o cookie es apagado del browser del usuario.

SECURE Opcional. Indica que la variable debe ser transmitida seguramente. Si el browser no soporta SSL, o cookie no es enviado.

PATH Opcional. Especifica los subdirectorios de URLs en que la cookie es aplicada.

DOMAIN Opcional. Especifica el dominio en que la cookie es válido.

Veamos ahora como crear un cookie:

<CFCOOKIE NAME="ID" VALLUE="1234" EXPIRES="1">

Creamos una cookie con un nombre ID de valor 1234 y que expirará en 1 día.

Verificando la existencia de una variable

En muchos casos, algunas aplicaciones necesitan que sean pasadas determinadas variables, y en caso de que estas no sean pasadas es presentado un mensaje de error podrá de Cold Fusion. Para que esto no ocurra, podemos utilizar una función de Cold Fusion que verifica la existencia de variables.

La función utilizada para esto es ISDEFINED. Abajo sigue su sintaxis:

IsDefined("nombre da variable")

Esta función verifica si la variable especificada existe, si retorna TRUE y caso contrario retorna FALSE.

Definiendo un valor patrón para variables

Definir valores patrones de variables en algunas aplicaciones puede ser muy útil, para hacer eso, basta utilizar la etiqueta CFPARAM.

La etiqueta CFPARAM puede ser usada para variables creadas por la etiqueta CFSET, formularios, URL y Cookie. Su sintaxis es la siguiente:

<CFPARAM NAME=”nombre de la variable” DEFAULT=”Valor”>

Donde:

NAME Es el nombre de la variable

DEFAULT El valor patrón de la variable

Uso practico de la función IsDefined y la etiqueta CFPARAM

Ya sabemos como funcionan IsDefined y la etiqueta CFPARAM, vamos ver ahora como utilizar ambos en una única aplicación y entender mejor donde aplicarlas.

Vamos crear una aplicación que tendrá dos links para una misma aplicación. Su única diferencia será que en uno de ellos colocaremos una variable llamada teste con su respectivo valor.

```
Param1.cfm
<a ref.=param2.cfm?teste=”link-1”>Link-1</a><br>
<a ref.=param2.cfm>Link-2</a>
```

El segundo archivo:

```
Param2.cfm
<CFIF IsDefined(“url.teste”)>
        <CFOUTPUT>#url.teste#</CFOUTPUT>
</CFELSE>
        <CFPARAM NAME=”url.teste” DEFAULT=”Link-2”>
        <CFOUTPUT>#url.teste#</CFOUTPUT>
</CFIF>
```

Tendremos como resultado en caso de que se cliqueado el primer link, el mensaje Link 1, y si elegimos el segundo enlace, el mensaje será Link 2.

Mostrar las variables

Hasta ahora vimos y entendimos como crear y verificar la existencia de una variable. ¿Pero como hacer para mostrar su contenido?

En el último ejemplo fue dada una pequeña muestra de cómo hacer esto:

<CFOUTPUT>#url.teste#</CFOUTPUT>

Para mostrar los tipos de las variables de Cold Fusion, basta utilizar algunos de los conceptos ya aprendidos en este tutorial, la etiqueta CFOUTPUT y la señal de #, más un identificador de tipo de variable. Por tanto tendremos:

<CFOUTPUT>#tipo_de_variable#</CFOUPUT>

Abajo tenemos un cuadro que ilustra como proceder para cada tipo de variables:

Variable creada por:	Sintaxis
Form	<CFOUTPUT>#form.variable#</CFOUTPUT>
URL	<CFOUTPUT>#url.variable#</CFOUTPUT>
Cookie	<CFOUTPUT>#cookie.variable#</CFOUTPUT>
CFSET	<CFOUTPUT>#variable#</CFOUTPUT>

Note que las variables creadas por la etiqueta CFSET no precisan de identificador de tipo.

Usando CFSWITCH, CFCASE y CFDEFAULTCASE

Comenzaremos viendo como tratar condiciones de múltiples opciones con las tags <CFSWITCH>, <CFCASE> y <CFDEFAULTCASE>. Estas tres tags son usadas en conjunto siendo la CFSWITCH responsable de la expresión, la CFCASE por las comparaciones y CFDEFAULTCASE para cuando ningún valor atiende las comparaciones hechas por CFCASE.

Veremos ahora la estrutura de utilización de estas tres tags:

<CFSWITCH EXPRESSION=”expresión”>
<CFCASE VALUE=”valor de comparación” DELIMITERS=”delimitadores”>
tags CFML o HTML
</CFCASE>
comparaciones adicionales <CFCASE></CFCASE>
<CFDEFAULTCASE>
tags CFML o HTML
</CFDEFAULTCASE>
</CFSWITCH>

Abajo entenderemos cada parámetro pasado en estas tags:

Parámetro Descripción

EXPRESSION Obligatorio. Aquí debe ser colocado la expresión/variable que irá ter su valor comparado.

VALUE Obligatorio. Este parámetro es responsable por los valores que serán comparados a la expresión.

DELIMITERS Opcional. Define que caracter separa una lista de valores no parámetro VALUE. Su valor por defecto es la coma (,).

Como podemos ver, utilizando estas tags podremos substituir una serie de CFIFs anidados. Para entendernos mejor tendremos abajo tres ejemplos de como utilizarlos.

Utilizando CFSWITCH, CFCASE y CFDEFAULTCASE con una variable

```
<CFSET valor1=1>
<CFSET valor2=2>
<CFSET total=#valor1#+#valor2#>
<CFSWITCH EXPRESSION="#total#">
<CFCASE VALUE="1">
Uno
</CFCASE>
<CFCASE VALUE="2">
Dos
</CFCASE>
<CFCASE VALUE="3">
Tres
</CFCASE>
<CFDEFAULTCASE>
Ninguna de las anteriores
</CFDEFAULTCASE>
</CFSWITCH>
```

En el ejemplo anterior la variable TOTAL es la suma de las otras dos variables (VALOR1, VALOR2), tendremos como respuesta “Tres”. Utilizando CFSWITCH, CFCASE y CFDEFAULTCASE con una variable y el parámetro DELIMITERS.

```
<CFSET valor1=1>
<CFSET valor2=2>
<CFSET total=#valor1#+#valor2#>
```

```
<CFSWITCH EXPRESSION="#total#">
<CFCASE VALUE="1;4" DELIMITERS=";">
Uno
</CFCASE>
<CFCASE VALUE="2">
Dos
</CFCASE>
<CFCASE VALUE="3">
Tres
</CFCASE>
<CFDEFAULTCASE>
Ninguna de las anteriores
</CFDEFAULTCASE>
</CFSWITCH>
```

Note que el ejemplo anterior sólo hubo cambios en el primer CFCASE, que retornará "Uno" en caso de la variable TOTAL sea igual a 1 o 4. Utilizando CFSWITCH, CFCASE y CFDEFAULTCASE con una query.

```
<CFQUERY DATASOURCE="coldfusion" NAME="Consulta">
select * from catastro
</CFQUERY>
<CFOUTPUT QUERY="Consulta">
<CFSWITCH EXPRESSION="#ciudad#">
<CFCASE VALUE="Madrid">
Ciudadanos de Madrid:<BR>
#nombre# - #email#
</CFCASE>
<CFCASE VALUE="Barcelona">
Ciudadanos de Barcelona:<BR>
#nombre# - #email#
</CFCASE>
<CFCASE VALUE="Sevilla">
Ciudadanos de Sevilla:<BR>
#nombre# - #email#
</CFCASE>
<CFDEFAULTCASE>
Ciudadanos de otras localidades:<BR>
#nombre# - #email#
</CFDEFAULTCASE>
</CFSWITCH>
</CFOUTPUT>
```

Vimos en este ejemplo que podemos también trabajar con resultados de una query. En el ejemplo anterior los nombres son clasificados de acuerdo con su localidad.

Funciones

Veremos ahora algunas funciones muy importantes para el desenvolvimento de aplicaciones en ColdFusion. Encontraremos aqui funciones que tratan fecha, hora, formateo de datos, matemáticas y strings.

Funciones de fecha

- Day

Devuelve el dia, de 1 a 31, para una fecha.

Sintáxis: Day(data)

Ejemplo:

<CFSET dia_venta = Day(“01/02/2000”)>

Note que el valor de la variable DIA_VENTA será 02 y no 01, pues la función utiliza el formato mes/dia/ano.

- Month

Devuelve el mes, de 1 a 12, para una fecha.

Sintáxis: Month(data)

Ejemplo:

<CFSET mes_venda = Month(“01/02/2000”)>

El ejemplo de la función Day, lo mismo sucede aquí; el resultado de la variable MES_VENDA es 01 y no 02.

- Year

Devuelve el ano de una fecha.

Sintáxis: Year(data)

Ejemplo:

<CFSET ano_venda = Year(“01/02/2000”)>

Funciones de Hora

- Hour

Devuelve la hora, entre 0 y 23, de un horario.

Sintáxis: Hour(horario)

Ejemplo:

<CFSET hora_venda = Hour ("15:07:33")>

- Minute

Devuelve los minutos de un horario.

Sintáxis: Minute(horario)

Ejemplo:

<CFSET minuto_venda = Minute("15:07:33")>

- Second

Devuelve los segundos de un horario.

Sintáxis: Second(horario)

Ejemplo:

<CFSET segundo_venda = Second("15:07:33")>

Ahora veremos una función que es muy útil tanto para la utilización con funciones de fecha cuando para funciones de hora.

- Now

Esta función devuelve la fecha y hora del sistema, en su servidor Cold Fusion.

Sintáxis: Now()

Ejemplo:

<CFSET data_atual = Now()>

Utilizando las funciones de Fecha y Hora

Para terner una mayor comprensión de la utilización de estas funciones veremos ahora como utilizarlas en la práctica, a través de un ejemplo que incorpora todas las funciones pasadas nasta ahora.

<CFSET fecha_atual = Now()>
<CFSET dia_atual = Day(Now())>
<CFSET mes_atual = Month(Now())>
<CFSET ano_atual = Year(Now())>
<CFSET hora_atual = Hour(Now())>
<CFSET minuto_atual = Minute(Now())>

```
<CFSET segundo_atual = Second(Now())>
<CFOUTPUT>
Fecha generada por el sistema: #fecha_atual#<p>
Dia: #dia_atual# Mes: #mes_atual# Ano: #ano_atual#<br>
Fecha generada por la aplicación: #dia_atual#/#mes_atual#/#ano_atual#<p>
Hora: #hora_atual# Minuto: #minuto_atual# Segundo: #segundo_atual#<br>
Hora generada por la aplicación: #hora_atual#:#minuto_atual#:#segundo_atual#
</CFOUTPUT>
```

Funciones de Formateo

- DateFormat

Devulve una fecha formateada de acuerdo con una máscara especificada. Si ninguna máscara fuera definida, la función devuelve la fecha usando el formato ddmmm-yy.

Sintáxis: DateFormat(fecha, máscara)

Máscara:

d – dia sem zero a esquerda
dd – dia com dois dígitos incluindo o zero
m – mês sem zero a esquerda
mm – mês com dois dígitos incluindo o zero
mmm – abreviação do mês com as três primeiras letras
mmmm – mês por extenso
yy – ano com dois dígitos
yyyy – ano com quatro dígitos

Creando variables con las funciones de fecha y hora.

Ejemplo:

```
<CFSET fecha_venda = (“01/02/2000”,”dd/mm/yyyy”)>
```

Tendremos como resultado: 02/01/2000, recordando que el formato patrón de ColdFusion es mes/dia/ano.

- TimeFormat

Esta función devuelve la hora formateada de acuerdo con una máscara especificada. Si ninguna máscara fuese definida, por patrón será aceptado el formato hh:mm tt.

Sintáxis: TimeFormat(hora, máscara)

Máscara:

h – hora sin cero a la izquierda. Reloj de 12h
hh – hora con dos dígitos incluyendo el zero. Reloj de 12h
H – hora sin cero a la izquierda. Reloj de 24h
HH – hora con dos dígitos incluyendo el cero. Reloj de 24h
m – minutos sin cero a la izquierda
mm – minutos con dos dígitos incluyendo el cero
s – segundos sin cero a la izquierda
ss – segundos con dos dígitos incluyendo el cero
t – identifica el horario como siendo A (antes medio-dia) o P (después medio-dia)
tt – identifica el horario como siendo AM o PM

Ejemplo:

```
<CFSET hora_actual = TimeFormat("13:54:22","hh:mm:sstt")>
```

Tendremos como resultado: 01:54:22PM.

Utilizando las funciones de formateo DateFormat y TimeFormat vimos en el ejemplo anterior como utilizar en la práctica las funciones de fecha y hora, y ahora podremos optimizar este ejemplo utilizando las funciones DateFormat y TimeFormat. Veamos como:

```
<CFSET fecha_actual = DateFormat(Now(),”dd/mm/yyyy”)>
<CFSET hora_actual = TimeFormat(Now(),”HH:mm:ss”)>
<CFOUTPUT>
Fecha generada por el sistema: #Now()#<p>
Fecha generada por la aplicación: #fecha_actual#<p>
Hora generado por la aplicación: #hora_actual#
</CFOUTPUT>
```

- URLEncodedFormat

Esta función convierte una string para formato de URL. O sea, si usted tiene una string con espacios, esta función sustituye estos espacios por los caracteres especiales, para que el string puede ser pasada a través de una URL.

Sintáxis: URLEncodedFormat(string)

Ejemplo:

```
<CFSET nombre = “José Joaquim Silva">
<CFSET nombreURL = URLEncodedFormat("nombre")>
```

Como resultado tendremos: Jos%E9%20Joaquim%20Silva

Funciones Matemáticas

- Max

Devuelve el valor mayor entre dos números.

Sintáxis: Max(número1, número2)

Ejemplo:

```
<CFSET mayor = Max(1,2)>
```

- Min

Devuelve el valor menor entre dos números.

Sintáxis: Min(número1,número2)

Ejemplo:

```
<CFSET menor = Min(1,2)>
```

- RandRange

Devuelve un número aleatório entre dos números. Los números deben ser menores que 100.000.000.

Sintáxis: RandRange(número1, número2)

Ejemplo:

```
<CFSET sorteio = (1,35)>
```

Funciones de String

- Len

Esta función devuelve el total de caracteres de un string.

Sintáxis: Len(string)

Ejemplo:

```
<CFSET tamanho = Len(“coldfusion”)>
```

- LTrim

Borra todos los espacios en blanco a la izquierda de un string.

Sintáxis: LTrim(string)

Ejemplo:
<CFSET palabra = Rtrim(“ coldfusion“)>
- RTrim
Borra todos los espacios en blanco a la derecha de un string.
Sintáxis: RTrim(string)
Ejemplo:
<CFSET palabra = Rtrim(“coldfusion “)>

Referencias
Libros/Revistas:
Certified ColdFusion Developer Study Guide
Mastering – ColdFusion 4.5
ColdFusion 4 for Dummies

Enlaces:
http://www.allaire.com/handlers/index.cfm?ID=18445
Últimos service packs para ColdFusion Server.

http://www.tutorialfind.com/tutorials/webdevelopment/coldfusion/default.asp?ct=21&ob=3
Conjunto extenso de tutoriales. Algunos son tutoriales únicos, merece la pena echar un vistazo.

http://www.zeal.com/Internet___Computers/Programming/Languages/ColdFusion__CFML_/
Etiquetas habituales de Coldfusion, links, webs y más.

http://www.katungroup.com/katuncoldfusiontips.htm
Códigos de ejemplo en ColdFusion y tips.

Active Server Pages. ASP

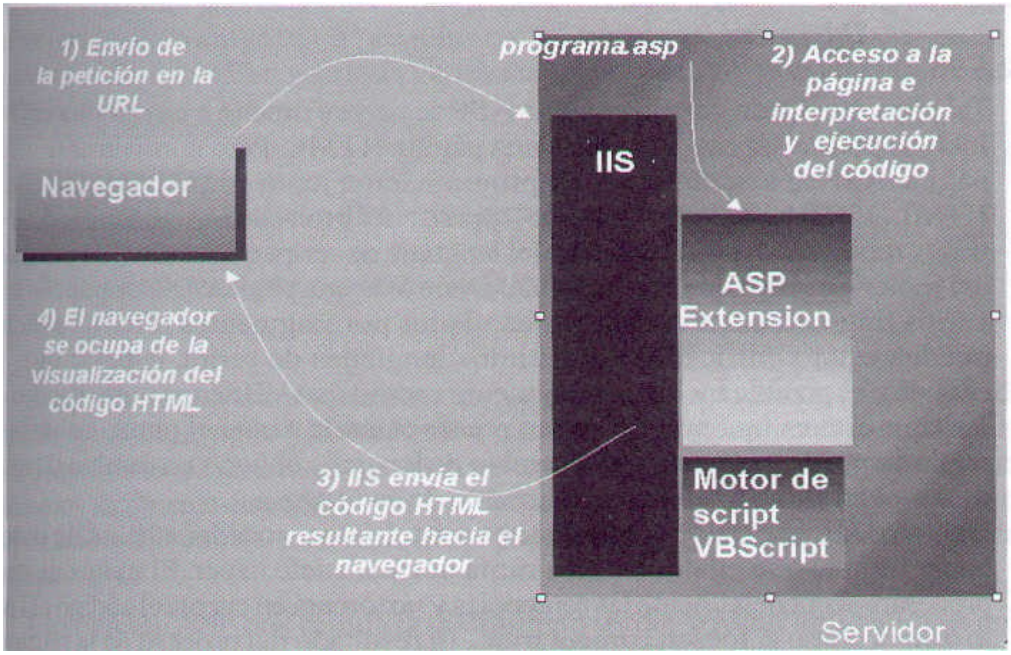
Introducción

ASP o Active Server Pages es la respuesta de Microsoft a la necesidad de crear páginas dinámicas. Podemos generar código HTML dinámicamente en base a la lógica del programa, podemos acceder y actualizar bases de datos, podemos utilizar objetos como si estuviésemos programando en Visual Basic, podemos escribir rutinas con algoritmos complejos, etc.Su funcionamiento se basa, principalmente, sobre servidores Microsoft con Internet Information Server para Windows NT o 2000, y en caso de contar con un sistema operativo Windows 95 o 98 se utiliza un servidor web personal, especialmente el Personal Web Server.

Hay que destacar que las páginas ASP se ejecutan del lado del servidor, devolviendo al cliente los resultados, esto quiere decir que no importa el navegador o sistema operativo del usuario, ya que el mismo vera una simple pagina Html. Para correr aplicaciones ASP bajo sistemas con Windows 95 o 98 es necesario contar con el *Personal Web Server*, el cual está incluido en el CD de Instalación de Windows 98. Si el sistema es Windows NT o 2000 hay que contar con el IIS (Internet Information Server). Hoy en día, es posible correr páginas ASP bajo servidores Unix/Linux, utilizando algún software como puede ser Instant ASP o Chilisoft.

Cómo funciona ASP

En el siguiente gráfico se esquematiza el funcionamiento interno de una aplicación Web basada en una página ASP. Tal como se puede ver todo comienza y finaliza de la misma manera que una petición HTTP, pero en el servidor la actividad es diferente.



1. Todo comienza con una petición o requerimiento HTTP iniciado en un navegador por un cliente que quiere acceder a un recurso de nustro sitio Web por medio de una dirección URL. La dirección URL apunta a la localización física de una página de extensión .asp.

2. En esta página asp por tener esta extensión, cuando llega al servidor IIS se pasa a un módulo denominado ASP extensión que es la DLL ISAPI que es capaz de ejecutar las páginas asp.
3. IIS, es decir, el servidor Web es el que se encarga del envío de la respuesta que le preparó la página ASP hacia el navegador que había realizado la petición.
4. El navegador es el encargado de la visualización del código HTML que recibe a modo de respuesta de la petición.

Aplicaciones

Una de las características mas importantes de las páginas ASP es la posibilidad de conectar con diferentes tipos de bases de datos, para extraer-agregar-eliminar datos de ellas, y generar páginas con esos datos. Estas paginas se generan en forma dinámica, dependen de las sentencias que se establezcan, para si obtener los resultados del proceso realizado.

Pueden conectarse a motores de bases de datos SQL, Access, Oracle, y a cualquier otro con soporte de conexión ODBC.

¿Con que hacer aplicaciones ASP?

No hay un software especifico para realizar páginas ASP, se puede utilizar hasta el Bloc de Notas de Windows, Frontpage, DreamWeaver, entre otros. Pero siempre es conveniente utilizar algún editor de texto como el Edit Plus o Gaspy, los cuales hacen mucho mas fácil la programación.

Principios básicos

Al igual que el lenguaje Html, los códigos ASP tiene un tag de inicio y fin de una página. En Html se utiliza <tag> para abrir, y </tag> para cerrar, mientras que en ASP se utiliza <% y %>:

Como ocurre en otros lenguajes de programación, se suele declarar el lenguaje a utilizar. Aunque esta sentencia es opcional en ASP muchos programadores suelen utilizarla:

```
<% LANGUAJE="VBScript"%>
```

Esta declaración se escribe al principio del archivo a utilizar, antes de cualquier otra expresión.

Al igual que otros lenguajes, es posible comentar el código.

```
<% ' Esto es un comentario
rem Esto es un comentario %>
```

Un comentario dentro del código ASP se señala con una ' (comilla simple) antes del comentario, también se suele cambiar la ' por la palabra rem. De esta forma, toda la línea que contenga un comentario el interprete de asp del servidor no lo leerá ni lo ejecutará.

Los comentarios son muy útiles cuando tenemos cientas o miles de línea de código y luego queremos hacer alguna modificación, poniendo diferentes comentarios pueden ayudarnos mucho mas adelante.

Programando con ASP

El aspecto "interno" de una página ASP es el siguiente:

```
<% ' Referencia del lenguaje
' Declaración de variables
```

```
' etc. %>
<html>
 <head>
      <title>Manual de ASP</title>
</head>
 <body>
      <% ' Conexión bases de datos
      ' Cálculos, etc. %>
</body>
</html>
```

Como pueden observar, se trata de lenguaje Html común y corriente con agregados de scripts de código ASP.

Declaración de variables

En una página ASP no hay que señalar de que tipo de variable se trata, todas son del tipo Variant. La declaración de las variables es opcional, pero es recomendable hacerlo ya que evita posibles errores y facilita la lectura del código. Las variables se declaran con el comando Dim. Para forzar a que se declaren todas las variables de una página se utiliza la función "*Option Explicit*". Ejemplo:

```
<% Option Explicit
 ' Declaramos las variables y las cargamos
Dim nombre, apellido, email
nombre = "Tomas"
apellido = "Vilariño"
email = "tomasvf@terra.es" %>
<html>
 <head></head>
  <body> Los datos son:<br> Nombre: <%=nombre%><br> Apellido: <
%=apellido%> <br> Email: <%=email%><br> </body>
</html>
```

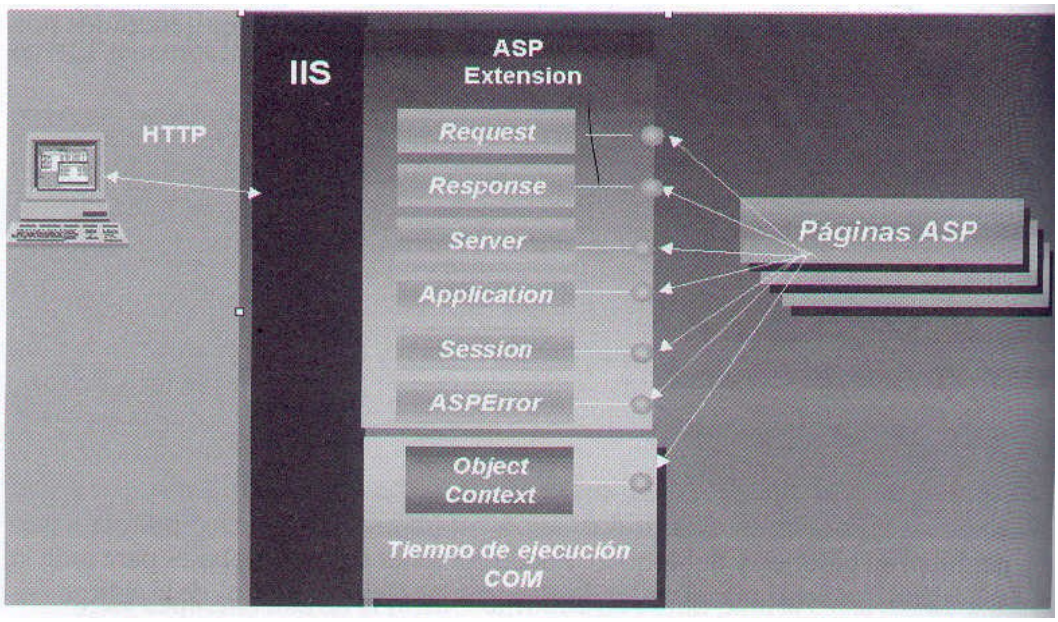
En este ejemplo se declaran tres variables y se cargan con los datos correspondientes, y luego se imprimen en medio del código html normal.

Objetos

La aplicación ASP dispone de una serie de objetos intrínsecos generados por ASP Extensión:

Response: es el objeto que representa la respuesta del servidor; posee los métodos necesarios para gestionar cualquier tipo de respuesta al cliente y permite decidir el tipo de búfer que se utilizará en las respuestas que se envían al servidor.

Este objeto es uno de los mas utilizados en las paginas ASP, ya que dispone de la comunicación entre el cliente y el servidor.



Response.Write

Sirve para escribir en la pagina, ya sea texto común o variables. Siguiendo el ejemplo anterior:

```
<% Option Explicit
' Declaramos las variables y las cargamos
Dim nombre, apellido, email
nombre = "Tomas"
apellido = "Vilariño"
email = "tomasvf@terra.es" %>
<html>
<head></head>
<body>
<% Response.Write "Los datos son:<br>"
Response.Write "Nombre: " & nombre & "<br>"
Response.Write "Apellido: " & apellido & "<br>"
Response.Write "Email: " & email & "<br> %>
</body>
</html>
```

Response.Redirect

Sirve para redireccionar una pagina hacia otra:

```
<% Response.Redirect "pagina2.asp" %>
```

Siempre debe utilizarse antes de los tags Html, ya que de otra forma dará error.

Response.Cookie

Sirve para "plantar" una cookie en la PC del cliente:

```
<% Response.Cookies("nombre")="Tomas"
 Response.Cookies("edad")="21" %>
```

En la cookie *nombre* se guardará la palabra Tomas, y en *edad 21*.

Request: es el objeto que representa la petición HTTP; junto con cada petición se pasa mucha información al servidor Web, la dirección URL de la página, el formato de los datos que se pasan, los datos de usuario. Por ejemplo, se utiliza el

objeto Request para leer los datos introducidos por el usuario en un formulario enviado al aservidor pulsando el botón Submit.

Request.Form

Toma los datos ingresados en un formulario. En el siguiente ejemplo el formulario incluido dentro del archivo formulario.htm, envía a procesar los datos al archivo procesa.asp:

formulario.htm

```
<html>
<head></head>
<body>
<form method="post" action="procesa.asp">
<input type="text" name="camponombre">
<input type="text" name="campoapellido">
<input type="submit" value="Enviar"> </form> </body>
</html>
```

procesa.asp

```
 <%' Como vimos anteriormente, declaramos variables
Option Explicit
Dim nombre, apellido
 ' Recogemos los datos del formulario y los guardamos
 nombre=Request.Form("camponombre")
apellido=Request.Form("campoapellido") %>
<html>
<head></head>
<body> Datos que ingresaste:<br> Nombre:
<%=nombre%><br> Apellido: <%=apellido%><br>
</body>
</html>
```

Y luego se imprimen los resultados.

Request.QueryString

Este método se utiliza para pasar valores de una página hacia otra a través de un link:

tomardatos.asp?**id=121&plataforma=windows**

En lo remarcado en negrita, pueden notar las variables:

id=121

plataforma=windows

Para recoger estos datos, se utiliza (dentro del archivo tomardatos.asp):

```
<% Option Explicit
Dim id, plataforma
' Recogemos los datos del querystring
id=Request.QueryString("id")
plataforma=Request.QueryString("plataforma") %>
```

Request.Cookies

Con este método podemos recoger los datos almacenados en una cookie, a la cual cargamos con algunos datos con el objeto Response.

```
Request.Cookies("nombredelacookie")
```

Así obtendríamos la información del PC del usuario. Un ejemplo para utilizarlo, es crear un formulario que pida un nombre, y luego guardar ese valor en una cookie, para cuando nos vuelta a visitar lo reconozcamos y lo saludemos:

```
<% nombre=Request.Cookies("nombre")
 Response.Write "Hola, " &nombre& "." %>
```

Session: es el objeto que representa una sesión de usuario; hay un ejemplar de este objeto para cada usuario conectado al sitio. Se utiliza para almacenar valores asociados a una sesión determinada.

Permite almacenar información necesaria para una sesión del usuario. Las variables almacenadas con el objeto Session no se pierden al cambiar de página, sino que perduran hasta que se eliminado por el servidor, o bien cuando se llame al método Abandon que cierra la sesión.

```
<% session("nombre")="Root"%>
```

De esta forma, almacenaremos dentro de la variable de sesión *nombre* la palabra Root. También podemos, a partir de una variable de session, guardar la información en otra variable:

```
<% nombre= session("nombre")%>
```

Para destruir una variable session puede ocurrir dos cosas: una es que el usuario esté 20 minutos sin actividad dentro del sitio, y la otra es llamar al método Abandon:

```
<% Session.Abandon%>
```

Application: este objeto representa a la aplicación ASP, por lo tanto sólo existe un único ejemplar en el servidor; al ser un objeto único posee métodos que permiten su bloqueo y desbloqueo. Puede contener valores globales para toda la aplicación que serám visualizados desde todas las sesiones activas de la aplicación.

Este objeto se utiliza para compartir información entre todos los usuarios de una aplicación. Como varios usuarios pueden compartir este objeto, se utilizan los métodos Lock y Unlock para no permitir que dos o más usuarios puedan al mismo tiempo modificar la propiedad.

Lock

Asegura que solo un usuario pueda modificar el objeto Application a la vez.

<% Application.Lock%>

Unlock

Desbloquea al objeto previamente bloqueado, para poder ser modificado por otro usuario después de haberlo bloqueado mediante el método Lock. Si no se desbloquea el objeto Application, el servidor lo hace automáticamente cuando la página .asp termina o transcurre el tiempo de espera.

<% Application.UnLock%>

Ejemplo:

```
<% Application.Lock
Application("visitas") = Application("visitas")+1
Application.Unlock %>
Sos el visitante Nº <%=Application("visitas")%>
```

Server: el objeto Server nos permite crear ejemplares de componentes, y expone métodos que se utilizan para la codificación de URL y del texto HTMl.

ASPError: es un objeto nuevo disponible a partir de Windows 2000; se utiliza por medio del objeto GetLastError del objeto Server y nos brinda información de detalle acerca del último error ASP ocurrido dentro de la aplicación. Es una mejora importante para la depuración del código ASP que hasta la versión 2.0 daba bastante trabajo al programador.

ObjectContext: no es un objeto ASP, es un objeto que nos da el tiempo de ejecución COM que nos permitirá implementar la transaccionalidad de la aplicación y gestionar la seguridad y los manejos de errores en los objetos COM que utilicemos.

Funciones básicas

Fecha y hora

La fecha y hora son muy utilizadas en las páginas ASP.

Fecha

Date devuelve la fecha actual
Day(fecha) devuelve el número de día
Month(fecha) devuelve el número de mes
Year(fecha) devuelve el año

Hora

Now devuelve la fecha y hora actual
Time devuelve la hora actual
Hour devuelve la hora
Minute devuelve los minutos
Second devuelve los segundos

Ejemplos de fecha y hora

Imprimir fecha y hora actual

Código

```
La fecha actual es <%=date()%>
<br> La hora actual es <%=time()%>
<br> Resultado La fecha actual es 25/3/2001 La hora actual es
21:25:5
```

Imprimir Día, Mes y Año por separado

Código

```
<% ' Declaramos variables
Option Explicit
Dim fecha, anio, mes, dia
' Guardamos la fecha actual dentro de fecha
fecha=Date()
' A partir de fecha, sacamos el dia, mes y año
anio=year(fecha)
mes=month(fecha)
dia=day(fecha) %>
Dia: <%=dia%><br>
```

```
Mes: <%=mes%><br>
Año: <%=anio%><br>
```

If...Then...Else

Esta instrucción es conocida en todos los lenguajes de programación. Su tarea es ejecutar una instrucción a partir del resultado de una condición.

If condicion then instruccion
if condicion then
instruccionA
else
instruccionB
end if

Ejemplos:

```
<% Option Explicit
Dim numero, dia numero=1 dia="viernes"
if numero = 1 AND dia="viernes" then
color="verde"
else color="negro"
end if %>

<% Option Explicit
Dim numero, dia numero=1 dia="viernes"
if numero = 1 OR dia="viernes" then
Response.Write "Color Verde"
else Response.Write "Color Negro"
end if %>
```

For...Next

Esta instrucción sirve para repetir un grupo de instrucciones una determinada cantidad de veces.

for contador = principio to fin [Step incremento]
[instrucciones]
exit for
[instrucciones]
next

Ejemplos:

```
<% for cont = 1 TO 10
Response.Write "Esta es la línea número: " & cont
next %>
<% for cont = 1 TO 10
   If cont=2 then
      Response.Write "La línea N° 2"
   end if
next %>
```

Introducción a las bases de datos

Tipo de conexiones

Existen varios tipos de conexión a bases datos. Algunos, como el ODBC requieren se la configuración de un DSN en el servidor, otros en cambio, trabajan directamente si tener que realizar alguna configuración. Algunas de las conexiones que no requieren de DSN son OLEDB o DBQ.

OBDC

```
<%
Set con = Server.CreateObject("ADODB.Connection"
' creamos el objeto de conexión
con.Open "NombreConexion"
' abrimos la conexión por ODBC al DSN NombreConexion
conn.close
' cierra la conexión %>
```

Por OLEDB

```
<%
set con=Server.CreateObject("ADODB.Connection")
' Conexión por OLEDB
con.Open"Provider=Microsoft.Jet.OLEDB.4.0;Data   Source=   "   &
Server.MapPath ("basededatos.mdb")&";"
%>
```

Por DBQ

```
<% set con=Server.CreateObject("ADODB.Connection")
' Conexión por DBQ
con.Open  "Driver={Microsoft  Access  Driver  (*.mdb)};  DBQ="  &
Server.MapPath ("basededatos.mdb") %>
```

Recordset

Se utiliza para realizar operaciones sobre las tablas de la base de datos. Para crear el objeto RecordSet se utiliza la línea set rs = Server.CreateObject ("ADODB.Recordset") seguida de una instrucción SQL para realizar alguna operación.

Instrucciones SQL

A través de estas instrucciones, podemos determinar que tipo de operación vamos a realizar sobre la tabla de la base de datos, puede ser seleccionar (select), borrar (delete), agregar (insert), entre otros. Para ejecutarla hay que crear primero el objeto del RecordSet, y luego de la instrucción SQL escribir:

```
set rs=oConn.Execute(SQL) .
```

Ejemplo sencillo de consulta:

```
<%
' Creamos los objetos de conexión y recordset
set con = Server.CreateObject("ADODB.Connection")
set rs=Server.CreateObject("ADODB.Recordset")
' Conexión por OLEDB
con.Open "Provider = Microsoft.Jet.OLEDB.4.0; Data Source = " &
Server.MapPath ("basededatos.mdb")&";"
 sql="select * from Alumnos order by nombre"
rs=con.Execute(SQL)
' Hacemos un listado simple de los registros
Response.Write                                                "-
"&rs.fields("apellido")&","&rs.fields("nombre")&".   "&   rs.fields
("edad") &" años.<br>"
' Cerramos la conexión y el recordset
rs.Close
con.Close
set rs=nothing
set con=nothing
```

```
%>
```

Ejemplos de uso

A parte de los ejemplos utilizados en la sección anterior se van a mostrar ejemplos en Jscript. Estos ejemplos son para mostrar que ASP puede utilizar varios motores como PerlScript, Jscript, VBScript entre otros.

1.- Ejemplo en Jscript que muestra el uso del objeto String:

```
<HMTL>
<HEAD><TITLE>Ejemplo en JScript</TITLE></HEAD>
<BODY>
<%=@ LANGUAGE=Jsscript %>
<%= "Inicio".anchor("Inicio") %><BR>
<%="Hiperenlace".link(http://www.ran.es) %><BR>
</BODY>
</HTML>
```

2.- Uso del objeto Date en Jscript.

```
<HTML>
<HEAD><TITLE>Uso objeto Date</TITLE></HEAD>
<BODY>
<%@ LANGUAGE=Jscript %>
<%
Fecha = new Date()
%>
<%= Fecha%>
Fecha:<%=Fecha.getDate()%>
Día de la semana: <%=Fecha.getDay() %><BR>
</BODY>
</HTML>
```

3.- Gestión de ficheros con Jscript, para ello hacemos uso de los objetos FileSystemObject y TextStream de VBScript.

```
<HTML>
<HEAD><TITLE>Gestión de ficheros</TITLE></HEAD>
<BODY>
<%@ LANGUAGE=Jscript %>
<%
Conexion = new ActiveXObject("Scripting.FileSystemObject")
Fichero = Conexion.CreateTextFile("c:\prueba.txt")
Fichero.WriteLine("Línea insertada")
Fichero.WriteLine("Otra línea")
Fichero.Close()

Conexion2 = new ActiveXObject("Scripting.FileSystemObject")
Fichero2 = Conexion2.OpenTextFile("c:\prueba.txt")

do{
      Linea=Fichero2.ReadLine() %>
      <%= Linea %><BR><%
}while(!Fichero2.AtEndOfStream)
Fichero2.Close()
%>
</BODY>
</HTML>
```

Alojamiento ASP

En la actualidad están apareciendo muchos servidores que dan servicio gratuito de hosting con soporte de ASP y bases de datos Access, con la única condición de que aparezcan sus banners o ventanas con publicidad.

Websamba (http://www.websamba.com)

Ofrece 30mb, soporta ASP y conexiones a bases de datos Access sin limitaciones, y la posibilidad de subir nuestras páginas por FTP. El único inconveniente que encontramos en este servidor son sus continuas caídas, quedando a veces varias horas sin funcionar. No pone publicidad.

Brinkster (http://www.brinkster.com)

Al igual que Websamba, ofrece 30mb de espacio, soporte asp y bases de datos access. No incluye publicidad en las páginas, pero nos vemos obligados a subir los archivos a través de un File Manager desde su sitio.

Referencias

Libros/Revistas:

Creación de aplicaciones Web en Windows NT Active Server Pages

Autores: Jesús Bobadilla Sancho, Alejandro Alcocer Jarabo.
Editorial: Rama
ISBN: 84-7897-333-8

Explica el uso de IIS 4.0 y su configuración, acceso a bases de datos a través de SQL, Visual Basic Script y Java Script en servidor, numerosos ejemplos, ...

Programación avanzada con Microsoft Visual Basic 6.0

Autor: Francesco Balena
Editorial: McGrawHill
ISBN: 84-481-2681-5

Manual avanzado de Visual Basic 6.0 que tiene capítulos referentes a la programación en internet (HTML dinámico, ASP, WebClass, ActiveX).

Desarrollo de Componentes ASP

Autora: Shelley Powers.
Editorial: ANAYA O'REILLY
ISBN: 84-415-1245-0

Libro que muestra la integración de ASP (WebClass) con otros productos Microsoft de servidor, como son los Servicios de componentes (COM+), y permite que las páginas ASP accedan a objetos COM y COM+.

Programación con Visual Basic 6

Autor: Francisco Charte
Editorial: ANAYA
ISBN: 84-415-0590-X

Desarrollo de aplicaciones IIS que se ejecutan en un servidor Web para ofrecer contenido dinámico.

Enlaces:

http://www.asptutor.com

Site dedicado a ASP, con numerosos ejemplos y utilidades.

http://www.aspfacil.com
Site dedicado a ASP, con numerosos ejemplos y utilidades.

WebClass

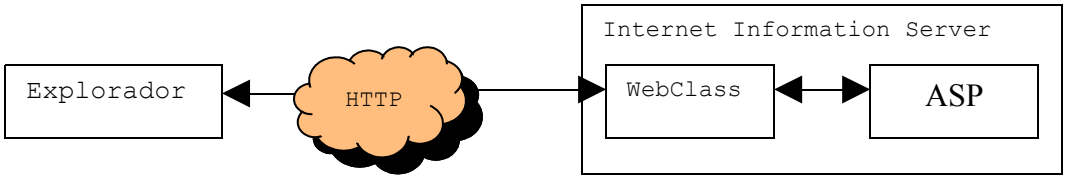
Introducción

Esta tecnología esta fuertemente ligada a ASP. Se trata de un componente que agrega Visual Basic 6 como técnicas de programación para Internet. Las clases Web (WebClasses) no permitirán hacer algo que no pueda hacer también con un script ASP y un componente personalizado escrito en Visual Basic (u otro lenguaje capaz de generar un DLL ActiveX).

Un WebClass es un componente en-proceso que se ejecuta dentro de IIS y que intercepta y, posteriormente, procesa todas las peticiones que los clientes realizan a un documento ASP.

Cómo funcionan las WebClasses

La diferencia entre un script ASP y WebClasses está en la forma en que las tecnologías trabajan a bajo nivel y en cómo las puede utilizar el programador. En una aplicación ASP estándar, el script ASP toma el control cuando el explorador cliente solicita un documento ASP y renuncia a él cuando se vuelve a enviar la página HTML resultante al cliente. En una aplicación WebClass, el WebClass entra en funcionamiento cuando el explorador cliente hace referencia a una página en la aplicación y, desde ese instante, reacciona a las acciones que el usuario realiza sobre la página, tal como pulsar sobre un hipervínculo o sobre el botón Enviar.

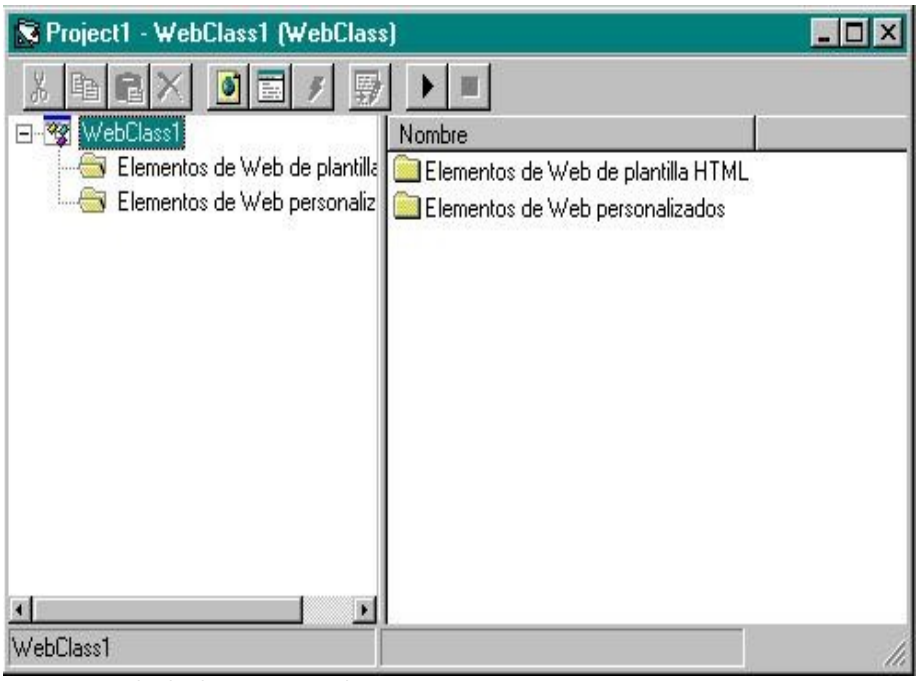


Una ventaja clave de las clases Web sobre la programación ASP tradicional basada en script y en componentes es que podrá desarrollar por completo una aplicación WebClass dentro del entorno de Visual Basic, por lo que podrá utilizar todas las herramientas de depuración habituales. Como en WebClass suele estar formado por varias páginas, se deberá considerar un componente de nivel superior. Por ejemplo, un único WebClass puede implementar un completo sistema de gestión de pedidos en línea y podrá, por tanto, reutilizarse en otra aplicación con mayor facilidad de lo que podría hacer con una colección de páginas ASP débilmente acopladas. Otra evidencia del alto nivel de abstracción de las clases Web es que no necesitará hacer nada especial para mantener el estado de una sesión entre dos peticiones consecutivas de los usuarios (por ejemplo, utilizar cookies o variables Session) porque las clases Web lo hacen todo automáticamente, aunque siempre a expensas de perder escalabilidad (podrá optar por desactivar esta opción y producir componentes WebClass más escalables aunque sin estado). Finalmente, a diferencia de lo que sucede con los scripts ASP, el código del WebClass no guarda ninguna relación con el código HTML que afecta al aspecto de sus páginas; por tanto, resulta más sencillo dividir el desarrollo de la aplicación entre varios diseñadores y programadores HTML.

Programando WebClass

Iniciamos un nuevo proyecto de tipo IIS verá que existe un módulo de clase IwebClass. Si lo abre se encontrará con el diseñador de clases Web, un diseñador que cuenta con dos paneles y una barra de botones. El panel izquierdo muestra una lista de las plantillas y elementos Web existentes en el módulo, mientras que el de la

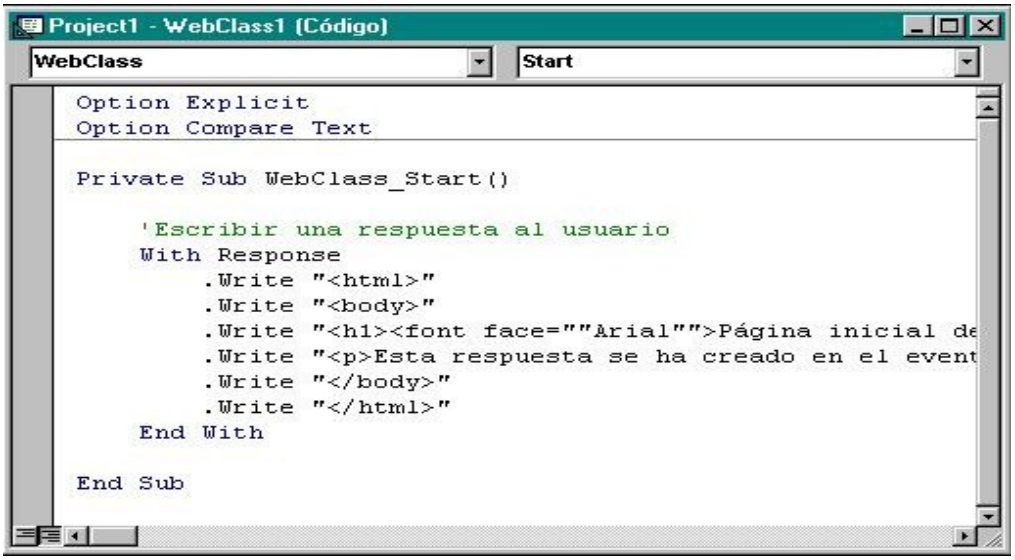
derecha mostrará los elementos existentes en el módulo seleccionado en el panel izquierdo.



Aspecto inicial del diseñador de clases Web.

Los botones que hay en la parte superior facilitan las habituales operaciones de cortar, copiar y pegar, así como otras específicas que insertan nuevos elementos Web, plantillas HTMl y eventos a medida.

Si abre la ventana de código verá que existe un procedimiento, llamado WebClass_Start, que utiliza el método Write de un objeto llamado Response para enviar al cliente una página HTML generada "al vuelo". Este procedimiento se ejecuta al generarse el evento Start del objeto WebClass, es decir, en el momento en que un cliente solicita la recuperación de información.



Pulsando la tecla F5, sin necesidad de introducir elemento o código alguno, aparecerá una primera ventana en la que se pregunta qué componente es el que hay

que iniciar. Pulsamos Intro e inmediatamente después se abre el cliente Web que tengamos instalando en nuestro sistema mostrando una página Web que, si nos fijamos, es la generada en el procedimiento anterior.

Lógicamente esta página no es muy útil, pero nos ofrece un punto de partida para trabajar insertando nuestro propio código y generando nuestras páginas HTML.

Añadir plantillas HTML

Los documentos HTML que se facilitarán desde nuestra clase Web pueden proceder de dos orígenes diferentes: una plantilla previamente diseñada o una generación dinámica del documento. Las plantillas son simplemente documentos HTML que creamos, con nuestro editor preferido, y posteriormente almacenamos en la propia clase Web.

El primer paso, por lo tanto, será crear nuestra plantilla usando un editor HTMl o el propio Bloc de notas de Windows. Guardado el documento en un archivo, desplegaremos el menú emergente del WebClass y elegiremos la opción Añadir plantilla HTML. Antes de dar este paso, no obstante, es preciso haber guardado el proyecto actual. Se selecciona el documento HTML y automáticamente aparece un nuevo elemento en la carpeta de plantillas, invitándose a cambiar el nombre. En el panel derecho podrá ver información diversa acerca de la plantilla.

Al añadir el documento a la lista de plantillas del WebClass lo que obtenemos es una copia, es decir, no se crea un enlace con el documento original, por lo que cualquier cambio en éste último no se reflejará de forma automática.

Suponiendo que hayamos llamado “Plantilla” a la plantilla que acabamos de añadir, podríamos sustituir el código existente en el método WebClass_Start por la siguiente sentencia:

```
Plantilla.WriteTemplate
```

Con esto enviaríamos como respuesta al cliente el contenido de la plantilla “Plantilla”. Si se ejecuta el proyecto podrá ver que se abre su cliente Web y muestra el documento que habíamos creado previamente, sin modificación alguna.

Sustitución de etiquetas

Desde luego una aplicación para devolver una página HTMl tal cual, sin realizar ningún tipo de proceso, no tiene demasiado sentido. Es más fácil facilitar al usuario directamente la página, descargando al servidor de cierto trabajo de proceso.

El uso de plantillas, sin embargo, adquiere sentido cuando no se envían sin más al cliente, realizándose un proceso previo consistente en sustituir ciertas etiquetas por contenidos que cambian dinámicamente. Aquí es donde tiene aplicación el evento ProcessTag con que cuentan las plantillas HTML de un WebClass, facilitando la operación de cambiar el contenido de una etiqueta sin necesidad de tener que recorrer todo el documento manualmente.

Suponga que desea añadir a la página creada con anterioridad un contador, elemento muy habitual, y la fecha y hora local en que se ha descargado el documento del servidor. Estos datos no pueden introducirse directamente en el documento HTMl, puesto que tienen que cambiar en cada solicitud. La solución está en usar unas etiquetas que posteriormente, al recibir una solicitud, sean interceptadas por el WebClass pudiendo sustituir su contenido.

Estas etiquetas especiales tendrán un nombre que empezará por el prefijo indicado en la propiedad TagPrefix, que podemos establecer según nos interese. Suponiendo que hayamos dado el valor WC a dicha propiedad, podríamos incluir las marcas necesarias en el documento HTML anterior para insertar el contador y la

fecha. Tras el prefijo WC hemos dispuesto la arroba (@) y un nombre, CONTADOR en el primer caso y FECHA en el segundo. Como cualquier otra marca, está también dispone de un cierre que es la misma marca precedida de una barra.

Tras cambiar el documento HTML es necesario incluirlo de nuevo en la plantilla de nuestro WebClass. Podemos conseguir esto desplegando el menú emergente de la plantilla que ya teníamos insertada y eligiendo la opción Actualizar plantilla HTML. También podemos elegir editar directamente la plantilla, en lugar del documento original.

Nos resta escribir el código necesario para realizar la sustitución del contenido de las etiquetas. Haga doble clic sobre la plantilla, para abrir el editor de código, y a continuación elija el evento ProcessTag de la lista de eventos. El código, que puede ver a continuación, comprueba el nombre de la etiqueta, que se recibe en el parámetro TagName, modificando de acorde el parámetro TagContents, en el que se devuelve el resultado.

```
Private Sub Plantilla_ProcessTag(ByVal TagName As String, _
    TagContents As String, SendTags As Boolean)

    Select Case TagName
    Case "WC@CONTADOR":
        Dim Contador As Integer
        On Error Resume Next
        Open "Contador.dat" For Input As #1
        If Err Then
            Contador = 0
        Else
            Input #1, Contador
            Close #1
        End If
        On Error GoTo 0
        Contador = Contador + 1
        Open "Contador.Dat" For Output As #1
        Print #1, Contador
        Close #1
        TagContents = Contador

    Case "WC@FECHA":
        TagContents = "-" & Date & "-" & Time
    End Select
End Sub
```

Al ejecutar por primera vez el proyecto, abriendo el explorador, se creará el archivo Contador.dat y se inicializará el contador. Posteriores solicitudes del documento, que puede efectuar pulsando el botón de actualización de su navegador, irán incrementando el contador.

Objetos WebItem

Una sede Web generalmente está compuesta de múltiples documentos, a los que se accede mediante un enlace existente en la página inicial o en un documento intermedio. Añadir varias plantillas a un proyecto Web es sencillo, basta con elegir la opción que ya conocemos y facilitar el camino y nombre del documento. El cliente, sin embargo, accede a una URL en la cual no puede hacer referencia a una plantilla concreta existente en el servidor.

Cada una de las plantillas existentes en el proyecto es un objeto WebItem, objeto al que se asocia un determinado identificador. El cliente puede solicitar dicho objeto facilitando el identificador adecuado. Cuando esto ocurre el WebItem genera un evento Respond, que podemos aprovechar para devolver al cliente el contenido de la plantilla que corresponda usando el método WriteTemplate que ya conocemos.

Direcciones de enlace

Cuando compilamos un proyecto IIS, o bien cuando lo ejecutamos desde el entorno de Visual Basic, se genera automáticamente un página ASP y una entrada, que usa como nombre el del proyecto, mediante la cual es posible acceder a la aplicación desde cualquier cliente Web simplemente facilitando la URL completa. Si estamos haciendo pruebas locales, esta URL podría ser http://localhost/Wplantillas/WebPlantillas.asp, en la cual Wplantillas es el nombre del proyecto y WebPlantillas el del módulo WebClass. En realidad Wplantillas_WebPlantillas.ASP es el nombre de la página ASP que se encuentra en la entrada Wplantillas.

Cuando se accede a la URL anterior se carga el documento por defecto, establecido en el evento Start del WebClass. Si facilitamos el parámetro adecuado, sin embargo, podemos recuperar otro de los documentos existentes. Uno de los parámetros posibles en WCIID, cuyo valor será el identificador del WebItem a ejecutar. La forma más sencilla de introducir los URL correctos de enlace en el documento inicial, o en cualquier otro donde sea necesario, consiste en generarlos mediante el método URLFor. Éste toma como parámetro el nombre del WebItem, devolviendo la URL completa en la que se encuentra.

Tendríamos, por lo tanto, que añadir dos nuevas etiquetas sustituibles a la plantilla HTML que estábamos utilizando, etiquetas que cuyo contenido se generaría al producirse el evento ProcessTag usando un código similar al mostrado en el siguiente fragmento.

```
Case "WC@IIS":
      TagContents = "<a href=" & URLFor(AplicacionesIIS)& ">" & _
            " Aplicaciones IIS</a>"
Case "WC@DHTML":
      TagContents = "<a href=" & URLFor(HTMLDinamico)& ">" & _
            " HTML Dinámico</a>"
```

Observe que el contenido es una etiqueta de enlace, "a href.", que tiene como dirección destino la devuelta por URLFor y como título el facilitado mediante código. Hechos estos cambios, al ejecutar de nuevo el proyecto podrá ver los dos enlaces en la página inicial. Observe en la parte inferior de la ventana la URL de destino, que es la misma de origen con el parámetro WCIID y el identificador del WebItem correspondiente.

Una alternativa a la generación de los URL de enlace durante la ejecución consiste en introducirlos directamente en la plantilla HTML usando el parámetro WCI=nombre en lugar de WCIID=identificador. Con WCI se usa como argumento el nombre del objeto WebItem que se desea recuperar, en lugar de un identificador numérico. En cualquier caso, generar los URL de enlace durante la ejecución desde el interior del propio módulo Web es mucho más seguro, ya que los enlaces no se rompen al introducir modificaciones.

WebItem personalizados

Un objeto WebItem no tiene necesariamente que tener una plantilla HTMl asociada, es posible crear también WebItem que generan código personalizado. Si despliega el menú emergente sobre el panel izquierdo del diseñador WebClass podrá ver que exista una opción que le permite añadir objetos Web personalizados o a medida. En principio estos elementos son idénticos a los vistos en un punto anterior, es decir, objetos WebItem a los que se accede mediante una cierta URL y que generan los eventos Respond y ProcessTag cuando procede.

Es apropiado utilizar un elemento Web a medida cuando el contenido del documento se va a generar, en su mayor parte, de forma dinámica, exigiendo un mayor control que la sustitución de etiquetas usada en los ejemplos anteriores.

Incluir parámetros adicionales

Hasta ahora hemos visto cómo las clases Web usan parámetros en la URL para saber qué documento, qué objeto WebItem, es el que tiene que devolverse. En ocasiones pueden ser interesante incluir parámetros adicionales, parámetros que pueden, por ejemplo indicarnos una selección por parte del usuario en base a la cual generar un nuevo documento.

Una de las formas que tenemos de incluir parámetros adicionales en el documento HTML consiste en asignarlos a la propiedad URLData del objeto WebClass.Cualquier cadena asignada a esta propiedad será incluida, por ejemplo, al generar un enlace mediante el método URLFor. De esta forma, cuando el usuario pulse sobre dicho enlace no sólo estará seleccionando un documento sino que, además, estará devolviéndonos esa misma cadena, que recuperaremos con la misma propiedad URLData.

Eventos propios

Los objetos WebItem cuentan con unos eventos predefinidos, como Respond o ProcessTag, que tienen un cometido ya conocido, como es devolver el documento o procesar una etiqueta sustituible. Nosotros podemos añadir otros eventos a un elemento Web de dos formas: creándolos durante el diseño y creándolos durante la ejecución.

Para añadir un evento personalizado durante el diseño, lo primero que haremos será elegir el elemento WebItem apropiado. Acontinuación abrimos el menú emergente y seleccionamos la opción de añadir un nuevo evento a medida, introduciendo el nombre que deseemos darle. Añadiendo el evento, podemos hacer doble clic sobre él para abrir el método correspondiente. Éste, como podrá ver, tiene un nombre compuesto del nombre del WebItem y del propio evento.

El evento añadido se generará en el momento en que el cliente lo solicite, usando para ello un parámetro adicional incluido en la URL. Podemos generar la URL de un cierto evento utilizando el método URLFor que ya conocemos pero con dos parámetros: el objeto WebItem y el nombre del evento en forma de cadena. En realidad el segundo parámetro de URLFor puede ser cualquier nombre de evento, incluso aunque no se haya definido durante el diseño.

Uando un WebItem recibe un parámetro WCE seguido del nombre de un evento, lo primero que hace es comprobar si existe un evento predefinido con ese nombre. En caso afirmativo lo genera directamente. De no existir se genera el evento UserEvent, facilitando como parámetro el nombre de evento. Esto permite que un programa gestione uno o múltiples eventos definidos durante la ejecución sin necesidad de conocerlos durante la ejecución.

El objeto WebClass

Vamos a adentrarnos en los detalles de la clase WebClass, que representa a un módulo Web completo con sus WebItem, ya sean plantillas o no, y sus eventos. Un mismo proyecto puede contener varios objetos WebClass.

Aplicación y sesiones

Un objeto WebClass es un componente ActiveX, alojado en una librería de enlace dinámico, que IIS, el servidor Web, carga en memoria y pone en marcha cuando el cliente solicita un cierta página ASP. Ésta normalmente contiene una llamada a la función CreateObject que se encarga de crear el citado componente, tal y como se aprecia en el siguiente código, que es el contenido de una página ASP simple.

```
<%
Server.ScriptTimeout = 600
Response.Buffer = True
Response.Expires = 0

Set WebClass = Server.CreateObject("Project1.WebClass1)
WebClass.[_Process] True
%>
```

Cuanod esto ocurre por primera vez, es decir, cuando la aplicación recibe la primera solicitud, se crea automáticamente un objeto Application al que podemos acceder mediante la propiedad del mismo nombre del objeto WebClass. Con cada solicitud, además, se crea un objeto Session, accesible también con una propiedad que tiene el mismo nombre.

Estos dos objetos, Application y Session, pueden ser usados fundamentalmente para mantener información global o a nivel de sesiones. Para ello podemos crear variables, asignarles valores y recuperarlos cuanod nos interese. Ambos objetos cuentan con una propiedad llamada Value, que además es la propiedad por defecto, que es como una matriz cuyo índice es el nombre de una variable, obteniéndose un valor de tipo variant.

El siguiente código sería un ejemplo muy sencillo de uso de los objetos Application y Session. Cuando se recibe la primera solicitud la variable Inicio no existe en el objeto Application, por lo que se indica que la aplicación acaba de cargarse. En todos los accesos posteriores dicha variable ya existe. Cada vez que la solicitud procede de un usuario distinto se crea una nueva sesión, por lo que la variable Inicio del objeto Session nos permite saber si el mismo usuario ha accedido varias veces a la página.

```
Private Sub WebClass_Start()
      Dim sQuote As String
      SQuote = Chr$(34)
      With Response
            .Write "<HTML><BODY>"
            If Application("Inicio") = True Then
                  .Write "<p>El servidor ya estaba cargado</p>"
            Else
                  .Write "<p>El servidor se ha cargado</p>"
            End If
            If Session("Inicio") = True Then
                  .Write "<p>Has vuelto a cargar la página </p>"
```

```
            Else
                  .Write "<p>Has cargado la página - 1ª vez</p>"
                  Session("Inicio") = True
            End If
      End With
End Sub
```

Mediante variables del objeto Application es posible, por ejemplo, almacenar datos para realizar una estadística global de los accesos a la aplicación. El objeto Session, por su parte, podría utilizarse para impedir que un contador se incremente en caso de que un mismo usuario solicite múltiples veces el mismo documento.

Uso de cookies

Las cookies son trozos de información que un servidor Web almacena en el ordenador del cliente, de tal forma que cuando éste vuelve a conectar con el servidor es posible recuperar los datos previamente almacenados. Usando esta técnica los servidores pueden saber las veces que hemos conectado con ellos, en qué fecha y a que hora, qué paginas visitamos la última vez, etc. Basándose en estos datos, una aplicación puede ofrecer al usuario información personalizada.

Ambos objetos, Request y response, cuentan con una propiedad llamada Cookies que nos permite acceder a la colección de este tipo de elementos que nuestra aplicación pueda tener almacenados en el ordenador del cliente. En esta colección el índice es el nombre de la cookie, siendo el valor un número, una cadena o cualquier otro dato que nos interese conservar. Podemos usar la propiedad Cookies del objeto Request sólo para recuperar valores y la propiedad Cookies del objeto Request sólo para asignarlos.

En el siguiente código puede ver cómo utilizando dos cookies reconocemos a cada usuario que accede a nuestro servidor, indicándole cuántas veces nos ha visitado y, si procede, en qué fecha fue la última visita.

```
Private Sub WebClass_Start ()
      Dim contador As Integer

      If Request.Cookies("Contador") = "" Then
            Contador = 1
      Else
            Contador = Request.Cookies ("Contador")
            Contador = Contador + 1
      End If

      Dim sQuote As String
      SQuote = Chr$(34)
      With Response
            .Write "<HTML><Body>"
            If Request.Cookies("Fecha") <> "" Then
                  .Write "<p>La última vez fue el " & _
                  Request.Cookies("Fecha") & "<p>"
            End If
            .Write "</body></html>"
            .Cookies("Contador") = Contador
            .Cookies("Fecha") = Date
      End With
End Sub
```

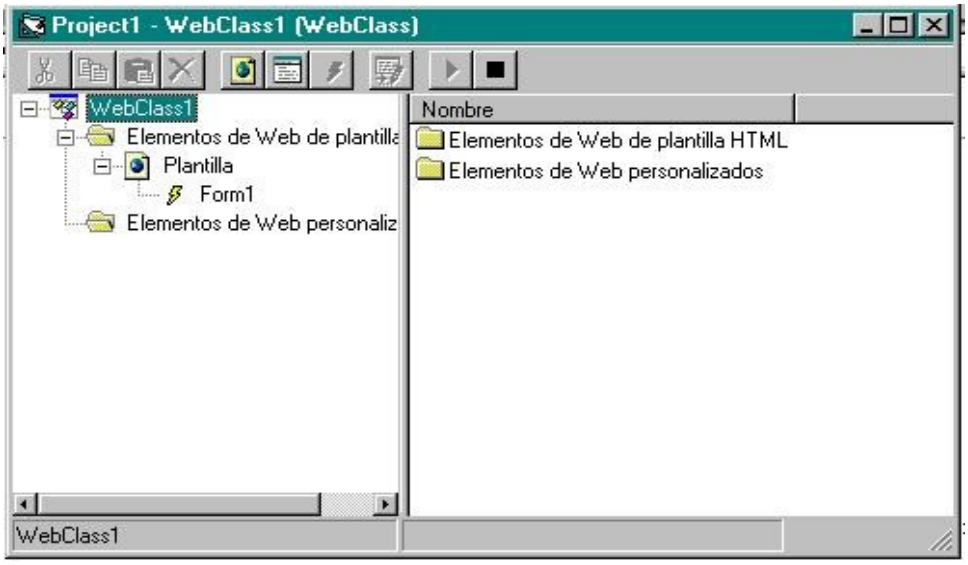
Hay que tener en cuenta que no todos los cliente Web permiten el uso de cookies, ya que las últimas versiones permiten al usuario rechazar estos elementos.

No es aconsejable,por lo tanto, crear una dependencia de las cookies en nuestra aplicación, aunque si están disponibles para ser utilizadas.

Datos de formularios

El objeto Request dispone de diversas propiedades que nos permiten recuperar información del cliente, entre ellas la colección Cookies que ya hemos visto. Otra propiedad de interés es Form, que nos permite acceder a los campos de un formulario HTML recuperando los datos introducidos por el cliente. Al igual que cookies la propiedad Form es una colección en la que se usa como índice el nombre del campo al que quiere accederse. El valor obtenido es el dato introducido en dicho campo. Para que podamos recibir información de un formulario, es necesario conectar el método POST con un WebItem o un evento de nuestro módulo. Esta tarea se realiza de forma muy sencilla.

Diseñe un formulario simple en el cual se solicite el nombre del cliente. Guárdelo en un archivo HTML y recupérelo como una plantilla, llamándola Plantilla. Seleccione ahora la plantilla Plantilla, pulse en el panel derecho sobre el elemento Form1 y abra el menú emergente, seleccionando la opción Conectar a WebItem.



Por último vamos a añadir un nuevo WebItem, al que llamaremos Contenido, que será el encargado de generar el documento en caso de que ya conozcamos al usuario que nos visita.

Como siempre comenzamos por el evento Start, en el que cual se comprueba si disponemos o no del nombre del cliente. De no ser así se facilita la plantilla que contiene el formulario, en caso contrario se indica que es el Webitem Contenido el que generará el contenido.

El objeto WebItem

Los objetos de esta clase, como podrá suponer, son mucho más sencillos que los WebClass. De hecho conocemos los tres eventos que puede generar ProcessTag, respond y UserEvent, el único método de que dispone: WriteTemplate, y una de sus propiedades: TagPrefix.

Tan sólo encontramos tres propiedades más en este objeto: Name, properties y ReScanReplacements. La primera de ellas contiene el nombre del objeto WebItem, como la propiedad Name de cualquier otro objeto. La segunda en una colección de propiedades de usuario. La tercera tiene que ver con el proceso de búsqueda de etiquetas sustituibles. Por defecto tiene el valor False, de tal forma que una vez que se genere el evento ProcessTag se sustituye el contenido original por el facilitado en el código y se termina. Si le damos el valor True conseguiremos que el proceso de búsqueda de etiquetas se reinicialice hecha la sustitución, lo cual sólo tiene sentido si en el código introducimos nuevas etiquetas sustituibles.

Referencias

Libros/Revistas:

Programación avanzada con Microsoft Visual Basic 6.0
Autor: Francesco Balena
Editorial: McGrawHill
ISBN: 84-481-2681-5

Manual avanzado de Visual Basic 6.0 que tiene capítulos referentes a la programación en internet (HTML dinámico, ASP, WebClass, ActiveX).

Desarrollo de Componentes ASP
Autora: Shelley Powers.
Editorial: ANAYA O'REILLY
ISBN: 84-415-1245-0

Libro que muestra la integración de ASP (WebClass) con otros productos Microsoft de servidor, como son los Servicios de componentes (COM+), y permite que las páginas ASP accedan a objetos COM y COM+.

Programación con Visual Basic 6
Autor: Francisco Charte
Editorial: ANAYA
ISBN: 84-415-0590-X

Desarrollo de aplicaciones IIS que se ejecutan en un servidor Web para ofrecer contenido dinámico.

Server-Side JavaScript. SSJS

Introducción

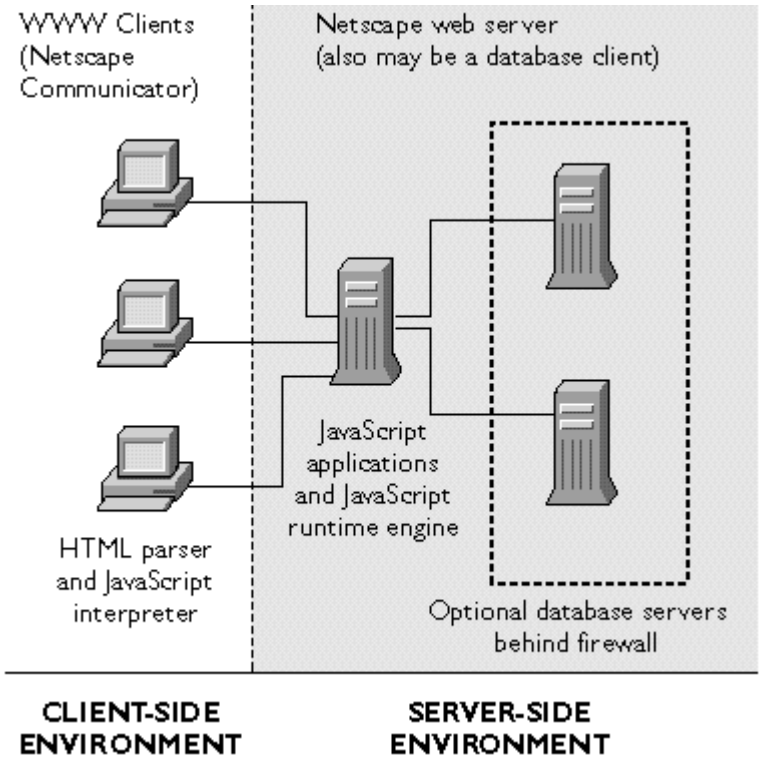
El lenguaje que soporta esta tecnología es una extensión del núcleo de JavaScript. JavaScript es un lenguaje basado en prototipos, no en clases, donde se usan objetos pero no clases ni herencia. SSJS incorpora funcionalidades para soporte de bases de datos y correo electrónico, manejo de sesiones e interrelación con clases del lado servidor mediante la tecnología de Netscape LiveWire.

En contraposición con otros lenguajes de script, SSJS es un lenguaje compilado, donde un conjunto de páginas web que contienen SSJS se compilan en una aplicación web que se ejecuta siempre que se solicita la URL asociada.

El código compilado SSJS no depende de ninguna plataforma (hardware y/o sistema operativo), aunque sí es especifico de los servidores web de Netscape (*Netscape Enterprise Server* y *Netscape Application Server*).

Cómo funciona Server Side JavaScript

La arquitectura de aplicaciones JavaScript pueden correr en el lado cliente y en el servidor. En adición, muchas aplicaciones JavaScript usan el Servicio LiveWire Database para conectar la aplicación a una base de datos relacional. Por esta razón, puedes pensar en las aplicaciones JavaScript que siguen una arquitectura three-tier client-server, como se ilustra en la figura



Arquitectura client-server JavaScript application environment

Las tres capas son:

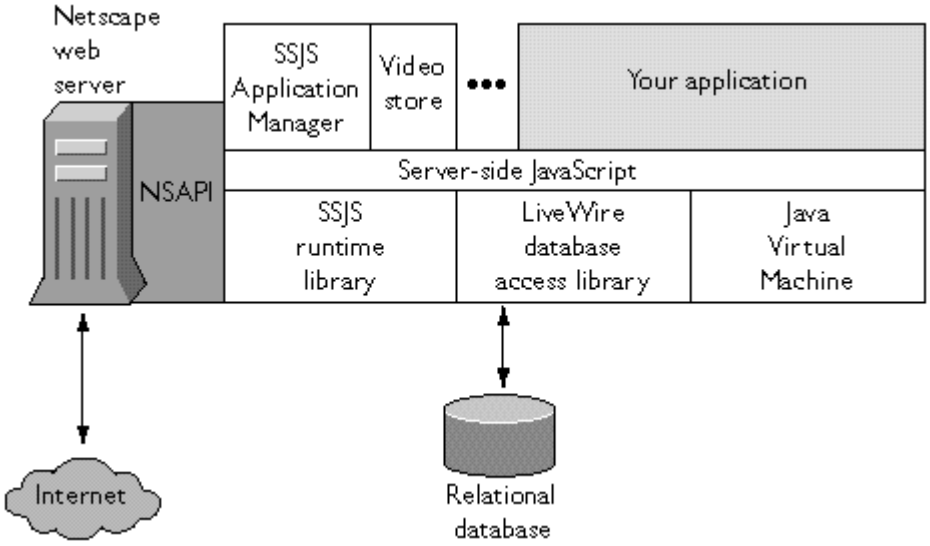
Clientes WWW (tal como clientes Netscape Navigator): Esta capa provee una interfaz/plataforma usuario-final a la aplicacion. Esta capa puede también contar con alguna lógica de negocio, tal como reglas de validación de datos implementadas en JavaScript del lado cliente (client-side JavaScript, CSJS).

Servidor Netscape WWW/cliente base de datos: Esta capa consiste de un servidor Netscape, con server-side JavaScript habilitado. Contiene la lógica de negocio, control de seguridad, y control de accesos a la aplicación por múltiples usuarios, usando server-side JavaScript. El servidor WWW además actúa como cliente del servidor de base de datos.

Servidor base de datos: Esta capa consiste en servidor de base de datos SQL, tipicamente corriendo en estaciones de trabajo que ofrecen alto rendimiento. Contiene todos los datos, metadatos, y normas de integridad referencial requeridas por la aplicación. Esta capa típicamente está dentro de un firewall corporativo y puede proveer una capa de seguridad adicional en adición a la que proveeel servidor WWW. Netscape Enterprise Server soporta el uso de ODBC, DB2, Informix, Oracle, y Sybase como servidores de base de datos. Netscape FastTrack Server soporta sólo ODBC.

El entorno de JavaScript del lado cliente corre como parte del cliente WWW, y el entorno JavaScript lado servidor corre como parte de Netscape web server con acceso a un o más servidores de base de datos. La siguiente figura muestra más detalles de como el entorno de JavaScript del lado servidor, y las aplicaciones se construyen para este entorno, adecuándose dentro de Netscape web server.

En la parte superior de la figura muestra como adecuar JavaScript del lado servidor dentro de Netscape web server. Dentro del web servidor, el runtime SSJS está construido desde tres componentes principales los cuales están listados debajo. El JavaScript Application Manager corre encima de SSJS.



Server-side JavaScript in the Netscape server environment

Estes con los tres principales componentes del entorno runtime JavaScript:

La librería runtime JavaScript: Este componente provee de la funcionalidad básica JavaScript.

La librería de acceso a base de datos LiveWire: Este componente extiende la funcionalidad base de server-side JavaScript con clases y objectos que proveen sin costuras acceso a un servidor externo de base de datos.

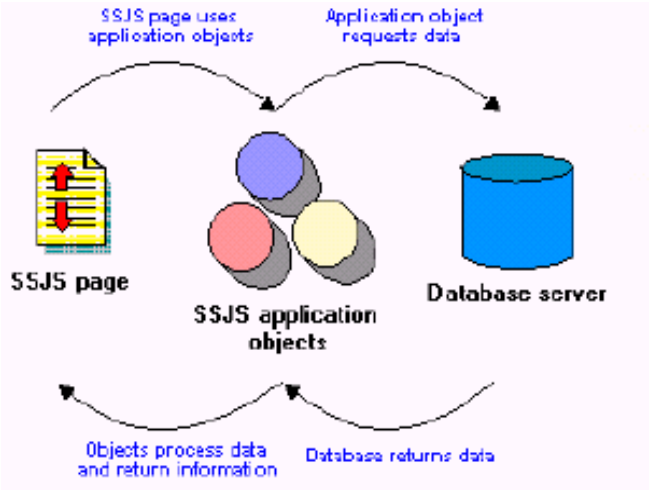
La máquina virtual Java: Diferente a los otros componentes, la Java virtual machine no es para usar sólo con JavaScript; algunas aplicaciones Java corriendo sobre el servidor usan esta máquina virtual. La Java virtual machine ha sido

aumentado para permitir a las aplicaciones JavaScript acceder a clases Java, usando la funcionalidad JavaScript's LiveConnect.

En general, una aplicación JavaScript puede contar con sentencias interpretadas por el cliente (con el intérprete de JavaScript que provee Netscape Navigator o some otro browser) y por el servidor (con el motor runtime JavaScript). Cuando corres una aplicación JavaScript, una variedad de cosas ocurren, algunas en el cliente y otras en el servidor. Aunque el usuario final no necesite conocer los detalles, es importante para el desarrollador de la aplicación, entender que sucede "por debajo".

Las páginas HTML pueden contener sentencias server-side y client-side JavaScript. En el código fuente HTML, client-side JavaScript es delimitado por la etiqueta SCRIPT y server-side JavaScript por la etiqueta SERVER. Podemos escribir ficheros que sólo contengan sentencias JavaScript y ninguna HTML. Un fichero JavaScript puede contener cualquier client-side JavaScript o server-side JavaScript; pero no puede contener ambos objetos client-side y server-side o funciones.

Si el fichero HTML y JavaScript contienen server-side JavaScript, entonces se compila dentro de un fichero ejecutable de aplicación JavaScript. El ejecutable es nombrado a un fichero web y tiene la extensión .web. El compilador devuelve código independiente de la plataforma, parsing y compilando las sentencias server-side JavaScript. Finalmente, desplegar la aplicacion sobre el servidor web y usar el JavaScript Application Manager para instalar e iniciar la aplicación, a fin de que los usuarios puedan acceder a la aplicación.



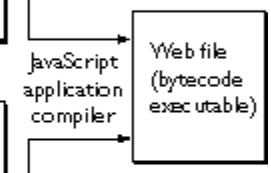
En tiempo de ejecución, cuando un cliente pida una página de la aplicación, el motor runtime localiza la representación de ese fichero en el archivo web de la aplicación. Corre todo el código servidor encontrado y crea una página HTML para enviar al cliente. Esa página puede contener código HTML y sentencias client-side JavaScript. Todo el código sercidor corre en el servidor, antes de enviar la página al cliente y antes de que ningún HTML o client-side JavaScript sea ejecutado. Por consiguiente, el código server-side no puede usar ningún objeto client-side.

```
...
function Substitute( guess, word, answer)  {
   var result = "";
   var len = word.length;
   var pos = 0;
   while( pos < len ) {
      var word_char  = word.substring( pos, pos + 1);
      var answer_char = answer.substring( pos, pos + 1 );
      if ( word_char == guess ) result = result + guess;
      else result = result + answer_char;
      pos = pos + 1;
   }
   return result;
}
```

hangman.js

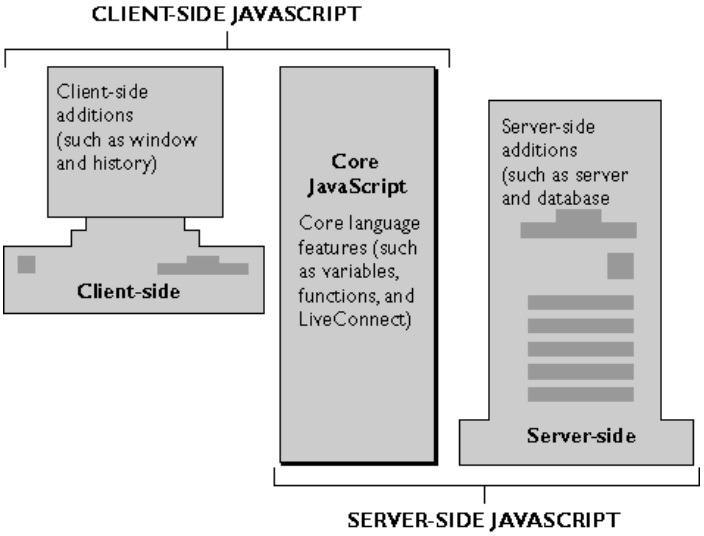
```
<HTML> <HEAD> <TITLE> Hangman </TITLE></HEAD>
<BODY> </H1> Hangman </H1>

<SERVER>
if (client.gameno == null)  {
   client.gameno = 1
   client.newgame = "true"
}
</SERVER>
You have used the following letters so far:
<SERVER>write(client.used)</SERVER>
<FORM METHOD="post" ACTION="hangman.htm">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

hangman.htm



Programando una aplicación SSJS

Client-side y server-side JavaScript ambos implementanel lenguaje JavaScript 1.2. En adición, cada añadido de objectos y funciones especifica para introducir poco a poco en el entorno cliente o servidor. Por ejemplo, client-side JavaScript incluye el objeto form para representar un formulario en una página HTML, mientras que server-side JavaScript incluye el objeto database para conectarse a una base de datos relacional externa.



Server-side JavaScript implementa el lenguaje JavaScript 1.2 completamente, tal como hace client-side JavaScript. Aún así, server-side JavaScript difiere de la especificación JavaScript 1.2 en el tratamiento de los operadores de comparación.

También, server-side JavaScript implementa prototipos tal y como se define en la especificación, pero la implicación es algo diferente en el entorno servidor que en el entorno cliente.

El comportamiento de los operadores de comparación.

El comportamiento de los operadores de comparación cambia entre JavaScript 1.1 y JavaScript 1.2. JavaScript 1.1 provee conversion automática de operandos para comparar. En particular:

Si ambos operandos con objectos, JavaScript 1.1 compara la referencia del objecto.

Si cualquier operando es null, JavaScript 1.1 convierte el otro operando a objecto y compara las referencias.

Si un operando es un string y el otro es un objecto, JavaScript 1.1 convierte el objeto a un string y compara los caracteres del string.

Sin embargo, JavaScript 1.1 convierte ambos operandos a números y compara numéricamente.

JavaScript 1.2 no provee conversion automática. En particular:

JavaScript 1.2 nunca intenta convertir operandos de un tipo a otro.

JavaScript 1.2 siempre compara la entidad de operandos como type. Si los operandos no son del mismo tipo, entonces no son iguales.

Server-side JavaScript puede proveer cualquier comportamiento. Por defecto, server-side JavaScript provee la convesión automática de operandos como se hacía en JavaScript 1.1. Pero se puede tener el mismo comportamiento, si se quiere que en JavaScript 1.2, especificando la opción -a 1.2 a jsac, el compilador de JavaScript.

Prototipos

Como describe la referencia a Server-Side JavaScript, puedes usar las propiedades de los prototipos demuchas clases para conseguir nuevas propiedades e instanciarlas todas. Para las nuevas clases que tengan propiedades prototipo, se trabaja exactamente en server-side JavaScript como en client-side JavaScript. Se pueden añadir nuevas propiedades prototipo a el Blob, Connection, Cursor, DbPool, File, Lock, Resultset, SendMail, y clases Stproc. No puedes usar prototipos con los objetos client, project, request, y server. Asímismo, por lo que respesta a client-side JavaScript, puedes usar las propiedades prototipo para alguna clase que definas para tu aplicacion. Recordar que todas las aplicaciones JavaScript en el servidor corren en el mismo entorno. Esto es porque puedes compartir información entre cliente y aplicaciones. Una consecuencia de esto, sin embargo, es que si usas las propiedades prototipo para añadir una nueva propiedad a alguna clase server-side añadida por JavaScript, la nueva propiedad está disponible a todas las aplicaciones que están corriendo en el servidor. Esto provee de un fácil mecanismo para añadir funcionalidad a todas las aplicaciones JavaScript applications en nuestro servidor. En contraste, si añades una propiedad a una clase tu la defines en tu aplicación, esa propiedad está disponible sólo a la aplication en la cual fue creada.

Uso

Necesitas ser consciente de como el compilador de la aplicación reconoce client-side y server-side JavaScript en un fichero HTML.

Las sentencias client-side JavaScript pueden aparecer en varias situaciones:

Incluyéndolas como sentencias y funciones cin una etiqueta SCRIPT

Especificando un fichero JavaScript con source en la etiqueta SCRIPT

Especificando expresiones JavaScript como el valor de un atributo HTML
Incluyendo sentencias como manejadores de eventos dentro de otras etiquetas HTML

Las sentencias Server-side JavaScript pueden aparecer en estas situaciones:
Incluyéndolas como sentencias y funciones con la etiqueta SERVER
Especificando un fichero como fuente al compilador de la aplicación JavaScript
Especificando una expresión JavaScript como el valor o nombre de un atributo HTML
Fíjate que no puedes especificar sentencias server-side JavaScript como un manejador de eventos.

La ventaja de LiveConnect del núcleo de lenguaje JavaScript trabaja diferentemente en el servidor que en el cliente.
JavaScript provee funcionalidad adicional sin el uso de objetos. Puedes acceder a esta funcionalidad a través de funciones no asociadas a objetos (funciones globales). El núcleo del lenguaje JavaScript provee las funciones globales descritas en la siguiente tabla.

Funciones globales del núcleo JavaScript	
Función	**Descripción**
escape	Devuelve el hexadecimal codificado de un argumento en el conjunto de caracteres ISO Latin-1; usado para crear strings para añadir a una URL.
unescape	Devuelve la cadena ASCII para el valor especificado; usado en el parsing de una cadena añadida a un URL.
isNaN	Evalua un argumento para determinar si no es numérico.
parseFloat	Convierte un argumento string a un número en coma flotante.
parseInt	Convierte un argumento string y devuelve un integer.

Server-side JavaScript añade las funciones globales descritas en la siguiente tabla.

Funciones globales de JavaScript Server-Side	
Función	**Descripción**
write	Añade sentencias a el lado cliente en la página HTML que está siendo generada.
flush	Limpia el buffer de salida.
redirect	Redirecciona a la URL especificada.
getOptionValue	Consigue valores individuales en un elemento SELECT de un formulario HTML.
getOptionValueCount	Consigue el número de opciones en un elemento SELECT de un formulario HTMl.
debug	Visualiza valores de expresiones in la ventana trace o frame.
addCliente	Añade información del cliente a las URLs.
registerCFunction	Registra funciones para usarlas en SSJS.
callC	Llama a funciones nativas.
deleteResponseHeader	Borra información del envio al cliente.

addResponseHeader	Añade información a la cabecera de respuesta enviada al cliente.
ssjs_getClientID	Devuelve el identificador del objeto cliente para utilizar en técnicas de mantenimiento de JavaScript cliente.
ssjs_generateClientID	Devuelve un identificador usado para especificar excepcionalmente el objeto cliente.
ssjs_getCGIVariable	Devuelve el valor especificado de la variable de entorno CGI.

Clases y Objetos

Para soportar las diferentes tareas a desempeñar en cada lado, JavaScript tiene clases y objetos predefinidos que trabajan en el cliente pero no el servidor y otras clases y objetos predefinidos que trabajan en el servidor pero no en el cliente.

Importante: estos nombres de estos objetos son reservados para JavaScript. No crees tus propios objetos usando estos nombres.

El núcleo del lenguaje JavaScript provee las clases descritas en la siguiente table.

Clases del núcleo de JavaScript	
Clase	**Descripción**
Array	Representa un array.
Boolean	Representa un valor booleano.
Date	Representa una fecha.
Function	Especifica un string de código JavaScript a ser compilado como function.
Math	Provee constantes y funciones matemáticas, por ejemplo la propiedad PI contiene el valor de pi (3,141..)
Number	Representa valores numéricos primitivos.
Object	Contiene la funcionalidad base para compartir todos los objetos JavaScript.
Packages	Representa un paquete Java en JavaScript. Usado con LiveConnect.
String	Representa un string JavaScript.

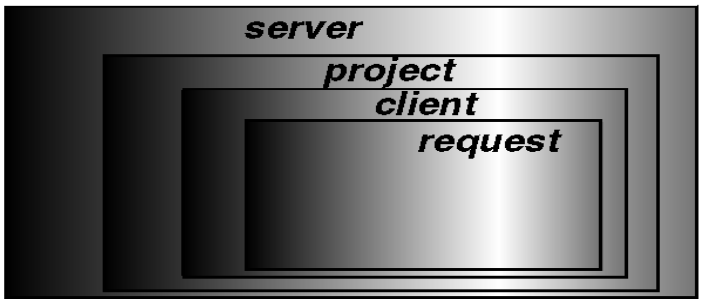
Server-side JavaScript incluye las clases del núcleo, pero no clases añadidas a client-side JavaScript. Server-side JavaScript tiene su propio conjunto de clases adicionales para soportar funcionalidades necesitadas, como describe la tabla siguiente.

Clases Server-Side JavaScript	
Clase	**Descripción**
Connection	Representa una simple conexión a base de datos desde un pool de conexiones.
Cursor	Representa un cursor de la Base de Datos.
DbPool	Representa un pool de conexiones a bases de datos.
Stproc	Representa un procedimiento almacenado de una base de datos.
Resultset	Representa la información devuelta por un procedimiento de base de datos.
File	Provee acceso al sistema de ficheros del servidor.

Lock	Provee la funcionalidad para compartir datos de forma segura en medio de peticiones, clientes y aplicaciones.
SendMail	Provee la funcionalidad para enviar e-mail desde una aplicación JavaScript.

En adición, server-side JavaScript tiene objetos predefinidos que se describen en la siguiente tabla. Estos objetos están disponibles para cada petición HTTP. No podemos crear instancias individuales de ninguno de estos objectos.

Objetos singleton Server-Side JavaScript	
Objeto	**Descripción**
database	Representa una conexión a una base de datos.
client	Encapsula información sobre un par cliente/aplicación.
project	Encapsula información sobre una aplicación que permanece hasta que la aplicación es parada en el servidor.
request	Encapsula información sobre una simple petición HTTP.
server	Encapsula información global sobre el servidor que dura hasta que el servidor el parado.



Hay dos caminos para embeber sentencias server-side JavaScript en una página HTML:

Con la etiqueta SERVER

Usando esta etiqueta para incluir una simple sentencia JavaScript o varios sentencias. Precedemos las sentencias JavaScript con <SERVER> y para terminarlas con </SERVER>.

Cuando tu embebes server-side JavaScript en una página HTML, el motor runtime JavaScript sobre el servidor ejecuta las sentencias encontradas mientras procesa la página. Muchas sentencias ejecutan alguna acción en el servidor, como abrir una base de datos o bloquear un objeto compartido. Sin embargo, cuando tu usas la función write en una etiqueta SERVER o encierras sentencias en backquotes, el motor runtime dinámicamente genera nuevo HTML para modificar la página enviada al cliente.

La etiqueta SERVER es el camino más común para embeber server-side JavaScript en una página HTML. Puedes usar la etiqueta SERVER en alguna situación; típicamente, sin embargo, usa backquotes en lugar de eso si estas generando nombres de atributos o valores para la página HTML.

Las sentencias entre las etiquetas <SERVER> y </SERVER> no aparecen en la página HTML enviada al cliente. En lugar de eso, las sentencias son ejecutadas en el servidor. Sin embargo, la salida de las llamadas a la función write function aparecen en el HTML resultante.

Usa backquotes (`) para encerrar expresiones server-side JavaScript como sustituciones para nombres de atributos HTML o valores de atributos. JavaScript embebido en HTML con backquotes automáticamente genera HTML; no se necesita escribir con write.

En general, las etiquetas HTML tags son del formulario

<TAG ATTRIB="value" [...ATTRIB="value"]>

donde ATTRIB es un atributo y "value" es un valor. Las expresiones entre paréntesis indican que algún número de pares atributo/valor pairs es posible.

Cuando tu encierras una expresión JavaScript entre backquotes para ser usada como un valor de atributo, el motor de runtime JavaScript automaticamente añade comillas para rodear el valor entero.

Compilando una Aplicación

Puedes compilar una aplicación JavaScript usando el compilador jsac. El compilador crea un fichero web de los ficheros fuente HTML y JavaScript.

Por facilidad de acceso al compilador, puedes añadir un directorio en el cual este instalada la variable de entorno PATH adecuada para poder compilar.

El compilador está disponible desde el prompt del sistema. Usa las siguientes sintáxis de línea de comandos para compilar y enlazar aplicaciones en el servidor:

```
jsac [-h] [-c] [-v] [-d] [-l]
   [-o outfile.web]
   [-i inputFile]
   [-p pathName]
   [-f includeFile]
   [-r errorFile]
   [-a 1.2]
   script1.html [...scriptN.html]
   [funct1.js ... functN.js]
```

La sintáxis es mostrada en múltiples líneas por claridad. Debe haber por lo menos un fichero HTML. Por defecto, el fichero HTML y JavaScript son relativos al directorio actual.

Las siguientes opciones en la línea de comandos están disponibles:

-h: Visualizar la sintáxis de ayuda del compilador.

-c: Chequea sólo la sintáxis; no genera un web file.

-v: (Verbose) Visualiza información sobre la ejecución del compilador.

-d: Visualiza contenido generado por JavaScript.

-l: Especifica el caracter firmado para usar cuando se compila (tal como iso-8859-1, x-sjis, o euc-kr)

-o outfile: Crea un bytecode-format web file, nombrado outfile.web.

-i inputFile: Te permite especificar un input file usando su pathname dentro de un pathname relativo. Puedes proveer sólo un nombre de fichero a esta opción.

-p pathName: Especifica un directorio para ser el raíz de todos los pathnames relativos usado durante la compilación.

-f includeFile: Especifica un fichero que es actualmente una lista de input files, permitiendote evadir el caracter final para una línea de comandos. Puedes proveer sólo un filename a esta opción. La lista de ficheros es delimitada por el espacio en blanco. Si el nombre de un fichero contiene un espacio debes encerrar este entre dobles comillas.

-r errorFile: Redirecciona a la salida estándar (incluyendo mensajes de error) para el fichero especificado.

-a 1.2: Cambia para que el compilador maneje comparación de operadores en el servidor.

Por ejemplo, los siguientes comandos compilan y enlazan dos JavaScript-mejorado HTML páginas, main.html y hello.html, y un fichero server-side JavaScript, support.js, creando un binario ejecutable nombrado myapp.web. En adición, durante la compilación, el compilador imprime información del progreso a la línea de comandos.

```
jsac -v -o myapp.web main.html hello.html support.js
```

Como un segundo ejemplo, el siguiente comando compila los ficheros listados en el fichero looksee.txt dentro de un binario ejecutable llamado looksee.web:

```
jsac -f looksee.txt -o looksee.web
```

Aquí, looksee.txt podría contener lo siguiente:

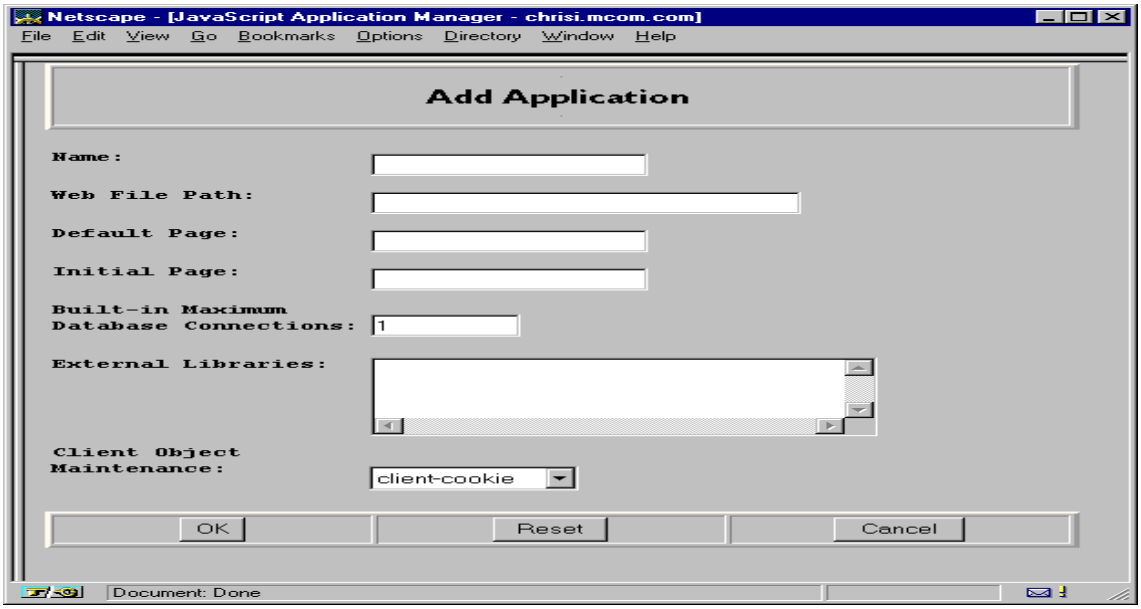
```
looksee1.html
looksee2.html
\myapps\jsplace\common.js
looksee3.html
```

Instalando una nueva Aplicación

Tu no puedes correr una aplicación y los clientes no pueden acceder a ella hasta que se instale. Instalando una aplicación se identifica en el servidor. Puedes instalar alrededor de 120 JavaScript applicaciones en un servidor.

Antes de instalar, debes mover todos los ficheros relacionados con la aplicación al directorio correcto, para publicar los ficheros. De otra manera, se obtendrá un errir cuando tú instales la aplicación. Por razones de seguridad, podrías no publicar el código fuente JavaScript en tu servidor de despliegue.

Para instalar una nueva aplicación con el Application Manager, click en Add Application. Como respuesta, el Application Manager visualizará el siguiente formulario:



Formulario para añadir Aplicaciones

Rellena los campos en el formulario Add Application, como sigue:

Name: el nombre de la aplicación. Este nombre define la URL de la aplicación.

Web File Path: el pathname del fichero application web file. Este campo es requerido.

Default Page: la página por defecto que el motor de JavaScript usará si no se indica la página en la aplicación. Esta página es análoga a index.html para un URL standard.

Initial Page: página que el motor de runtime de JavaScript ejecuta cuando tu iniciar la aplicación en Application Manager. Esta página es ejecutada exactamente una vez por ejecución de la aplicación. Se usa generalmente para inicializar valores, crear bloqueos, y establecer conexiones a bases de datos.

Built-in Maximum Database Connections: Número de conexiones máximas a bases de datos que la aplicación puede tener a la vez usando el objeto predefinido database.

External Libraries: los pathnames de librerías externas a ser usadas con la aplicación. Si especificas múltiples libreréas, delimita los nombres con comas.

Client Object Maintenance: la técnica usada para salvar las propiedades del objeto cliente. Pueden ser cookies, URL clientes, server IP, server cookie, o server URL.

Después de proveer la información requerida, click Enter para instalar la aplicación, Reset para limpiar todos los campos, o Cancel para cancel la operación.

Debes parar y restaurar tu servidor antes de añadir o cambiar librerías externas para una aplicación. Puedes restaurar un servidor desde tu Server Manager.

Ejemplos de uso

1.- Ejemplo del típico Hola Mundo:

```
<P>Ejemplo de
<SERVER>
write("Hola Mundo");
</SERVER>
```

2.- Sencillo ejemplo acceso a base de datos:

```
flush();
conn.beginTransaction();
cursor = conn.cursor ("SELECT * FROM CUSTOMER", true);
while ( cursor.next() ) {
   // ... process the row ...
   flush();
}
conn.commitTransaction();
cursor.close();
```

3.- Uso de SSJS para procesar un elemento SELECT de un formulario que nos llega como petición. Por ejemplo para un SELECT como sigue:

```
<SELECT NAME="what-to-wear" MULTIPLE SIZE=8>
   <OPTION SELECTED>Jeans
   <OPTION>Wool Sweater
   <OPTION SELECTED>Sweatshirt
   <OPTION SELECTED>Socks
```

```
   <OPTION>Cape
</SELECT>
```

Podría ser procesada la entrada de este select como sigue:

```
<SERVER>
var i = 0;
var howmany = getOptionValueCount("what-to-wear");
while ( i < howmany ) {
   var optionValue =
      getOptionValue("what-to-wear", i);
   write ("<br>Item #" + i + ": " + optionValue + "\n");
   i++;
}
</SERVER>
```

4.- Uso del sistema de ficheros con Server-Side Javascript, como se puede ver es bastante similar al ejemplo visto con ASP utiliando un motor Jscript.

```
x = new File("\tmp\names.txt");
fileIsOpen = x.open("r");
if (fileIsOpen) {
   write("file name: " + x + "<BR>");
   while (!x.eof()) {
      line = x.readln();
      if (!x.eof())
         write(line+"<br>");
   }
   if (x.error() != 0)
      write("error reading file" + "<BR>");
   x.close();
}
```

5.- Ejemplo que hace uso de la clase SendMail para el envio de correos electrónicos.

```
<server>
SMName = new SendMail();
SMName.To = "vpq@royalairways.com";
SMName.From = "thisapp@netscape.com";
SMName.Subject = "Here's the information you wanted";
SMName.Body = "sharm, phuket, coral sea, taveuni, maui,
   cocos island, marathon cay, san salvador";
SMName.send();
</server>
```

Hipertext PreProcessor PHP

Introducción

Es un producto de Open Source (gratuito). Proporciona grandes facilidades para acceso a bases de datos y *pattern matching*. También incorpora extensiones para comunicación con otros recursos de red como correo electrónico y servidores de directorio. PHP es compatible con Linux, Windows NT y varios sistemas operativos UNIX, así como con un amplio número de servidores web como Apache, Microsoft IIS y Netscape Enterprise Server.

Breve historia de PHP

En 1994, un programador llamado Rasmus Lerdorf desarrolló un conjunto de herramientas que usaban un parser para interpretar varias macros, tales como libros de visitas, contadores, y algún que otro elemento web más, que estaban en su punto más álgido, cuando la Web estaba naciendo. Rasmus combinó estas herramientas con un paquete intérprete de formulario (Form Interpretation), que programó él, añadiendo soporte para bases de datos, lo que se conoce como PHP/FI. En esta etapa PHP se conocía como Personal Home Page tools.

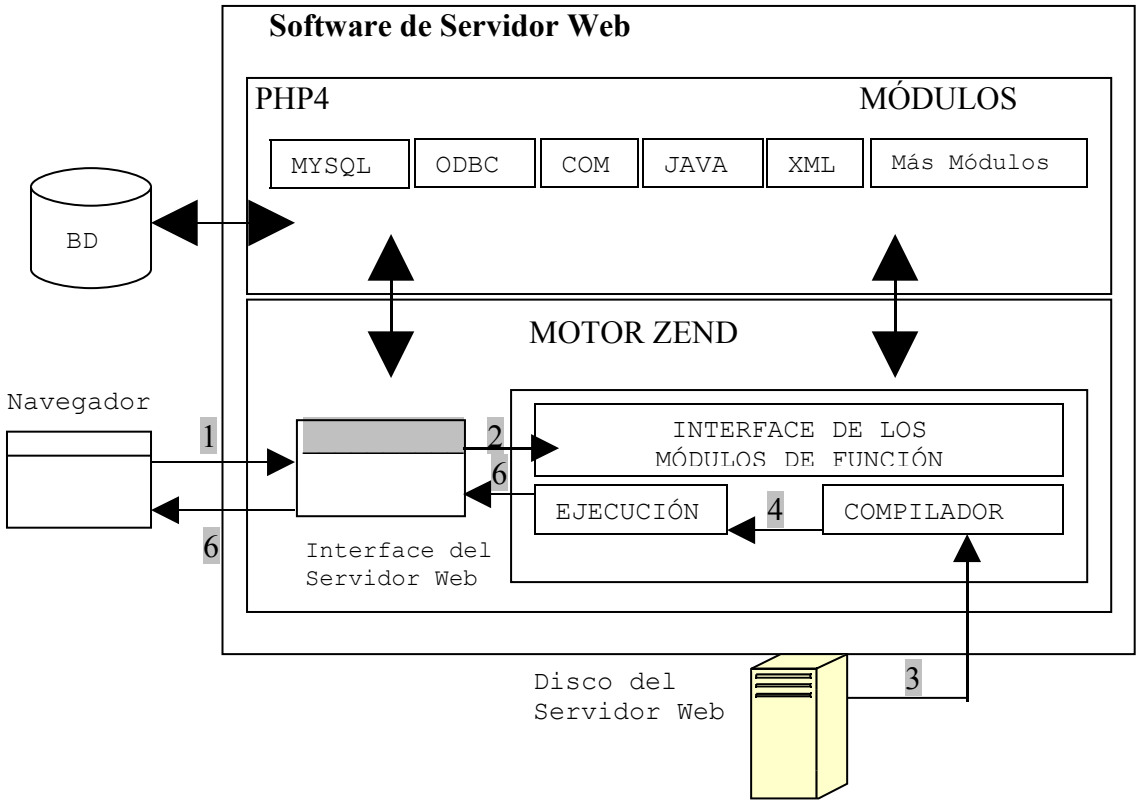
Con la llegada del Código Abierto (Open Source software), programadores de todo el mundo contribuyeron en el desarrollo de PHP/FI. En 1997 más de 50.000 sitios web usaban PHP/FI para conectar bases de datos y visualizar contenido dinámicamente.

En este punto empezó un proceso de desarrollo del lenguaje bastante fuerte, creándose equipos de trabajo. La versión 3.0 se creó con la colaboración de los programadores Zeev Suraski y Andi Gutmans. La versión final de PHP3 apareció en Junio de 1998, cuando se incluyó soporte para numerosos sistemas operativos (Windows, Linux, Unix, etc.), servidores web, bases de datos, SNMP (Simple Network Management Protocol) e IMAP (Internet Message Protocol).

La conclusión es que PHP3 se utiliza en más de un millón de servidores web. PHP3 se incluye como módulo de Apache en la mayoría de distribuciones Linus (Red Hat, Suse, Mandrake, etc.). PHP4 está ya disponible y ha sido creado gracias a una herramienta llamada Zend (un intérprete). Este intérprete ha sido diseñado para ser incluido también en otro tipo de aplicaciones. PHP4.0 es la primera aplicación que utiliza Zend, y también podría ser incluido en otros programas como MySQL. Algunas de las importantes características que incorpora esta nueva versión es el soporte para objetos COM/DCOM y Java.

Cómo funciona un PHP y el intérprete Zend.

En la siguiente figura se puede observar de manera gráfica cómo funciona el intérprete Zend.



El navegador realiza una llamada al servidor web, que pasa la petición a través de la interfaz PHP del servidor web(1). La interfaz del servidor web llama al intérprete Zend (2), que accede al disco donde están las páginas PHP. El servidor web recupera el código de la página PHP y lo envía al compilador (3). El compilador de Zend crea una versión compilada de la página que es pasada al módulo de ejecución de Zend (4). Este módulo genera el código HTML que será visto por el navegador. Si existieran llamadas a otros módulos, como bases de datos, XML o Java, el intérprete de Zend la envía a través del módulo específico para procesar la petición, y devuelve el resultado a la interfaz del servidor web (5) que enviará el código HTML al navegador (6).

Con Zend Compiler se permitirá a los desarrolladores compilar sus propios script de PHP y distribuirlos. De esta forma se pretende proteger el código fuente y así crear una especie de copyright, permitiendo a las empresas crear aplicaciones y soluciones software basado en PHP. Además también se espera la presentación de productos como Zend Caché, un módulo que almacena en una memoria intermedia del servidor web la aplicación PHP ya interpretada. Este proceso consigue mejorar el tiempo de respuesta de cara al usuario siendo ideal para website con gran cantidad de visitas. De esta forma se trata de evitar las continuas compilaciones de las páginas cada vez que son solicitadas.

Configuración de Apache + PHP + mySQL

La configuración del servidor web (Apache) está guardada en el fichero llamado http.conf dentro del directorio \Apache Group\Apache\conf. Este fichero de tipo texto puede ser editado con un editor de texto normal y corriente. Una vez abierto se observa que existen líneas precedidas por el símbolo "#" estas líneas con comentarios que describen el parámetro y el valor que hay a continuación.

Para comenzar a utilizar el servidor Apache es necesario especificar, en primer lugar, dos parámetros: el nombre del servidor (host) y el puerto que se utilizará.

Utilizando el comando Buscar del editor utilizado y como cadena de búsqueda ServerName se edita esa línea por la siguiente:

```
ServerName http://localhost
```

De esta forma, para conectarse al servidor web, bastará con escribir localhost, el nombre que se ha asignado a dicho ordenador en las Propiedades de Red o la dirección IP conocida como LoopBack: 127.0.0.x.

El siguiente cambio, o comprobación, es el puerto utilizado para conectarse al servicio web. Si el usuario no tiene ningún otro servidor HTTP instalado y funcionando, lo lógico es utilizar el puerto por defecto: 80. Para comprobarlo, se utiliza de nuevo el comando Buscar del editor de texto utilizando la cadena de búsqueda Port, si el valor especificado es 80, es correcto, si se desea cambiar bastaría poner otro puerto.

Si se hace uso de un puerto diferente al 80, será necesario especificarlo cuando se realicen las conexiones al servidor web a través de un cliente (navegador) de la siguiente forma:

```
http://localhost:8080 (si el puerto es el 8080)
```

El siguiente parámetro a modificar es establecer el directorio donde debe ir a buscar todos los documentos que se vayan a publicar en el servidor web, tanto si son HTML, PHP, etc. Para modificar este parámetro se busca la cadena DocumentRoot y se edita esta línea para poner el directorio de publicación. Se puede crear el directorio C:\Inetpub\httpd\html, que será el lugar donde se almacenen todos los ficheros de la aplicación web. De esta forma esta línea queda de la siguiente forma:

```
DocumentRoot "c:/Inetpub/httpd/html"
```

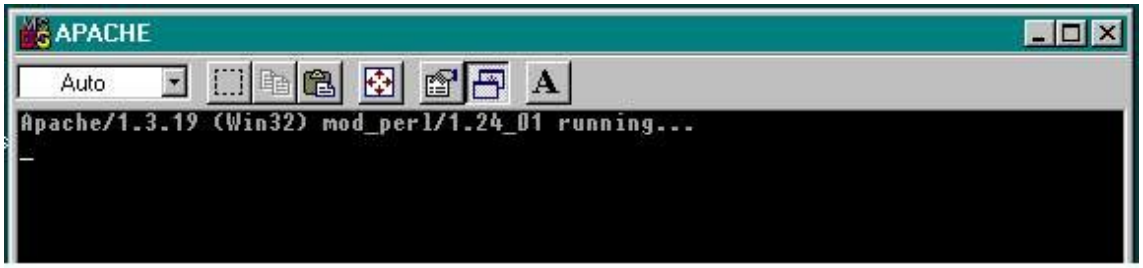
Obsérvese que el tipo de barra utilizada no es la habitual de los directorios de Windows, sino la contraria.

Existen otros parámetros que es interesante especificar aunque no estrictamente necesarios. Este es el caso de ErrorLog, que define el lugar en el que guardará el fichero log con los errores generados por el servidor HTTP.

```
ErrorLog logs/error.log
```

Una vez concluidas todas estas modificaciones se está en disposición de probar el servidor web Apache. Para ello ejecutamos el icono creado en el grupo de programas Apache Web Server llamado Start Apache y, a continuación, aparecerá una ventana de MS-DOS anunciando que el servidor web se encuentra funcionando.

Es importante fijarse en la ventana ya que, si en alguna modificación de las realizadas anteriormente se ha cometido algún fallo, se mostrará un mensaje de error en esta ventana y el servidor web no podrá comenzar a funcionar.



Para finalizara el funcionamiento de servidor Apache se pueden hacer dos cosas diferentes: la primera de ellas es pulsar las teclas Ctrl. + C en la ventana de MS-DOS donde se está ejecutando. La segunda es utilizar el icono Stop Apache situado dentro del grupo de programas de Apache. En ambos casos, el servidor web dejará de responder peticiones HTTP y la ventana de MS-Dos se cerrará.

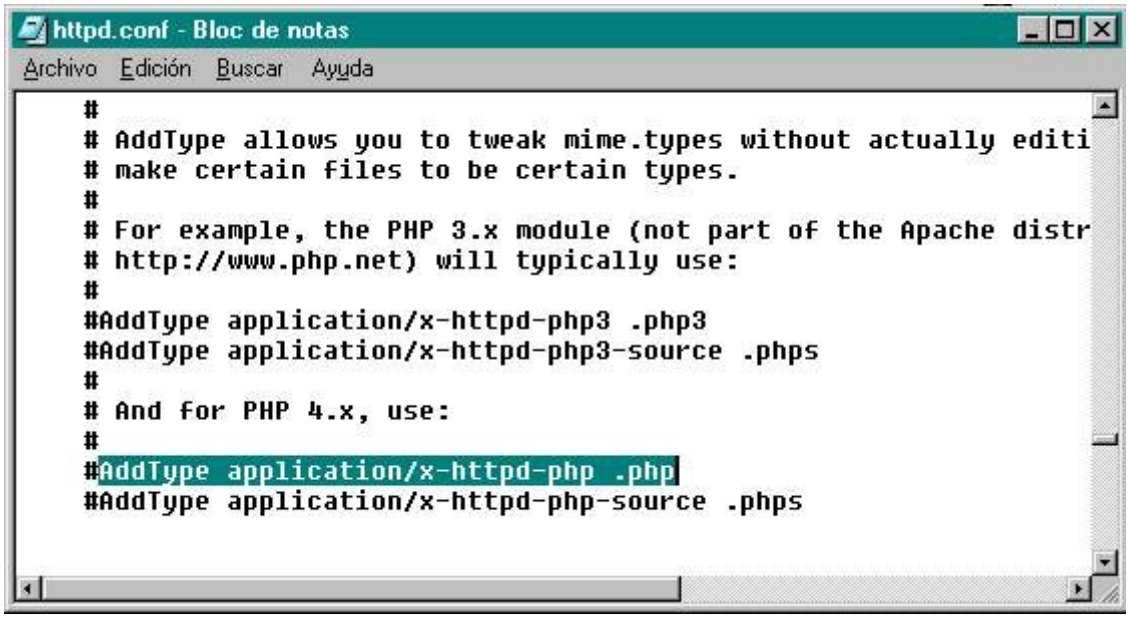
El siguiente paso es la instalación de los ficheros necesarios para interpretar los script PHP que se programen. En primer lugar es necesario crear un directorio en la raíz del disco duro C llamado php4 donde se copiarán todos los ficheros que hay en el fichero zipeado \php\php-4.0.6-Win32.zip del CD-ROM. En este fichero se encuentran las librerías necesarias para interpretar el código PHP así como unos ficheros de inicialización.

A continuación hay que volver a editar el fichero http.conf que se modificó en la configuración del servidor web Apache con el fin de añadir los parámetros necesarios para que el servidor web pueda recibir las peticiones de fichero PHP y pase la llamada al intérprete. En este fichero es necesario añadir tres líneas: en primer lugar hay que buscar la cadena ScriptAlias para, después de la línea que contiene el valor /cgi-bin , añadir la línea:

```
ScriptAlias /php4/ "c:/php4/"
```

Siguiendo con la opción de Buscar y como cadena de búsqueda AddType, se añade después del último AddType la línea:

```
AddType application/x-httpd-php .php
```



De esta forma el servidor web reconocerá la extensión PHP con el fin de ser procesada y no mostrará la habitual ventana de Guardar o Abrir que aparece cuando se solicita un fichero que su formato no es reconocido por un servidor.

A continuación, se vuelve a realizar un proceso de búsqueda para encontrar la cadena Actino y se añade la siguiente línea, después de la línea que contiene /cgi-script/location:

```
Action application/x-httpd-php "/php4/php.exe"
```

Éste es el parámetro que debe conocer el servidor con el fin de enviar las peticiones de páginas PHP a donde se encuentra el intérprete.

Con estas modificaciones el servidor web Apache ya estaría en disposición de reconocer y pasar al intérprete de páginas PHP que hemos grabado en el directorio c:\php4 para devolver el resultado en forma de código HTML al navegador del usuario. Para comprobar que todo funciona correctamente vamos a crear una sencilla página con una función que nos devolverá abundante información sobre la instalación y versión de PHP instalado. La página la llamaremos prueba.php y el código es el siguiente:

```
<?
      phpinfo();
?>
```

La página será grabada en el directorio que hemos elegido para guardar todas las páginas que deban ser vistas con Apache, en el directorio C:\Inetpub\httpd\html. A continuación, arrancamos de nuevo el servidor web y observamos la ventana de MS-DOS por si se produce algún error. Abrimos un navegador escribimos la dirección http://localhost/prueba.php, y al cabo de unos instantes, aparecerá una página ofreciendo información de un gran número de parámetros de PHP.

En estos momentos ya podemos decir que tenemos una plataforma funcionando para crear aplicaciones PHP. Si aún no puede ver esta página, revise de nuevo los pasos realizados en el fichero httpd.conf para comprobar que no ha cometido ningún error.

Hay otras dos opciones más que se pueden configurar para ahorrar esfuerzo al programador. Por defecto, cuando el servidor Apache accede al directorio que le ha sido configurado busca un fichero que es el primero que carga. En la mayoría de los casos este fichero es el que da comienzo a la aplicación web y suele llamarse default, index, home, etc. En nuestro caso es posible añadir el nombre de la página con la que comenzará nuestra aplicación. Para ello se busca la cadena DirectoryIndex y en la línea donde está:

```
DirectoryIndex index.html
```

Se añaden las páginas que queramos:

```
DirectoryIndex index.html index.php home.php default.php
```

Cuando un usuario se conecte al servidor web, éste comenzará a buscar en el directorio de la aplicación la primera página: index.html, luego la segunda index.php, y así sucesivamente, hasta que de con alguna. En caso de no encontrarla mostrará el contenido de todo el directorio.

La siguiente opción es la posibilidad de añadir alias de directorio al servidor web ya que, por defecto, sólo se podría acceder a un único directorio configurado cuando se accede al servidor, pero también es posible tener otros directorios y acceder a ellos añadiendo /<nombre_alias> al final del servidor web. Para poder hacer esto hay que buscar la cadena Aliases y añadir tanto alias como necesitemos después del último, como por ejemplo:

```
Alias /Temp./ "C:/Temp."
Alias /e-commerce/ "C:/Inetpub/httpd/html/e-commerce"
Alias /pruebas/ "C:/Inetpub/httpd/html/pruebas"
```

Por último, en el directorio que hemos creado /php4/ hay un fichero llamado php.ini-dist. Este fichero contiene diversos parámetros de configuración que no son estrictamente necesarios para que PHP funcione correctamente; sí es necesario por ejemplo, para establecer los parámetros necesarios copiar ficheros del cliente al servidor (upload). Si se desea utilizar, primero debe ser renombrado a php.ini y copiarse al directorio c:\windows o donde esté la instalación del sistema operativo. Los parámetros que especifica este fichero son constantes referentes a los diferentes módulos que incluye PHP (mySQL, XML, ODBC, etc.), directorios temporales, etc.

El proceso para instalar la base de datos de mySQL utiliza el programa de instalación que se puede encontrar dentro de la carpeta \mysql del CD-ROM. Una vez ha concluido la instalación es necesario abrir una ventana de MS-DOS ya que es a través de línea de comandos como se utiliza esta base de datos (aunque existen programas de terceras empresas para hacerlo de forma visual, en el CD-ROM se incluye mySQLFront).

A continuación, se ejecuta el programa mysqld situado en el directorio \mysql\bin cada vez que se quiera utilizar esta base de datos. Este programa es el encargado de recoger las solicitudes de conexión de las aplicaciones que accedan a una base de datos creadas en mySQL. Después se puede ejecutar el programa mysql que muestra la interfaz de la base de datos al usuario mediante el prompt mysql>. (Para más detalles consultar el anexo de mySQL situado al final de este trabajo).

Para utilizar mySQL junto a PHP no es necesario realizar ninguna modificación en ningún fichero ya que el propio intérprete de PHP incluye soporte para esta base de datos.

El único requisito es que el programa mySQL esté funcionando cada vez que se esté utilizando una aplicación PHP que acceda a una base de datos mySQL.

Programando un PHP

PHP es un lenguaje de procesos relativamente moderno que aúna cualidades de PERL, C y Java para conseguir un entorno bastante sencillo de aprender y pensado para dar resultados inmediatos. La estructura de los procesos en PHP empieza con la combinación de caracteres "<?php" y terminan con "?>". Estas dos marcas encierran el proceso (script) y deben ponerse siempre sin hacer referencia a la versión del lenguaje.

Las variables en PHP van precedidas por el símbolo $ muy al estilo de PERL. Esta es la forma que tiene el intérprete de PHP de reconocer y tratar como tal a una variable declarada. Las variables en PHP pueden contener cualquier tipo de dato y tienen que ser precedidas como string, integer, etc.

PHP tiene tres tipos básicos de datos: integer, double y string. Hay también algunos tipos no básicos llamados array y objetos. Cada variable tiene un tipo específico, aunque como ya se ha mencionado puede cambiar de tipo on-fly (en el momento de ejecución). Las variables utilizadas en PHP son de tipo case-sensitive, es decir que distinguen entre mayúsculas y minúsculas.

Utilizando la función define() es posible definir constantes para ser utilizadas a lo largo de la página PHP. La forma de crear estas constantes es :

```
<?
      define("CIUDAD","Ourense");
?>
```

En PHP es posible convertir las variables de un tipo a otro, para ello se antepone el tipo de dato que se quiere obtener al de la variable a convertir (como los casting de C o Java):

```
<?
      variable = 11.1;
      $variable = (int) $variable;
?>
```

Funciones de utilidad para el trabajo con variables son:

- gettype(), esta función devuelve el tipo de dato pasado como parámetro.
- settype(), establece el tipo de dato que va a guardar una variable.
- isset(), se utiliza para determinar si una variable ha sido inicializada con un valor.
- unset(), utilizada para destruir una variable y, por lo tanto, liberar los recursos dedicados a la misma.

- empty(), devuelve el valor true si la variable aún no ha sido inicializada, tiene un valor igual a cero o es una cadena vacía.
- is...(), is_integer(), is_double() y is_string() determinan si el tipo de dato pasado como argumento es de tipo integer, double o string respectivamente.
- ...val(), las funciones intval(), doubleval() y strval() realizan conversiones a los tipos integer, double y string. Estas funciones no pueden ser utilizadas para convertir arrays o objetos.

La sintaxis de las sentencias condicionales y instrucciones de bucle es similar a la utilizada en Java. En el apartado de "Ejemplos de uso" se muestran ejemplos de dicha sintaxis.

PHP dispone de dos funciones que permiten leer y ejecutar código que no pertenece a la página que está interpretando, sino que está en un fichero diferente. Este fichero puede ser, a su vez, otra página PHP o fichero no interpretado (HTML, CSS, TXT, etc.).

Esta característica es de gran ayuda al programador ya que permite reutilizar funciones y código en general al tener este código en un fichero separado, y que puede ser utilizado por página PHP en el momento en el que el programador lo necesite.

- require(), reemplaza el contenido del fichero al que haga referencia. Su principal uso es proporcionar constantes y funciones que no van a ser modificadas en la aplicación web.
-

```
<?
      require("variables.php");
?>
```

- include(), también accede a un fichero externo, pero al contrario que la función require() sí es capaz de procesar el código que contiene cada vez que se llama a esta página externa.
-

```
<?
      for ($i=1; $i<4; $i++) {
            include ("fichero" . $i . ".php");
      }
?>
```

Las funciones en PHP son declaradas utilizando la sentencia function. El siguiente ejemplo muestra una función para calcular el cuadrado de un número:

```
<?
      function cuadrado ($num) {
            return $num * $num;
      }
      print ("Cuadrado");
?>
```

Los parámetros pueden ser pasados a una función "por valor" o "por referencia". Por defecto, los parámetros son pasados a la función "por valor". Esto quiere decir que la variable que recibe el parámetro en la función crea una copia de valor. Si el valor del parámetro cambia, el valor que contiene la llamada de la función no será

cambiado. En el ejemplo anterior se muestra como se pasa un parámetro “por valor”.

Cuando se pasa un parámetro “por referencia”, el valor del parámetro original sí se verá modificado. Para que esto ocurra es necesario colocar delante de la variable que va a recibir el parámetro el símbolo ampersand (&), para indicar que el argumento es pasado por referencia.

Con el uso de las funciones dentro de las páginas PHP nos encontramos con un nuevo concepto: el alcance de las variables. Para acceder a una variable global desde una función se utiliza la instrucción global. Esta sentencia notifica al intérprete de PHP que no se va a crear una nueva variable local, sino que se va a utilizar una ya existente fuera de la función. Una forma alternativa de utilizar variables globales dentro de funciones es mediante la construcción del array $GLOBALS. Este array almacena todas las variables globales que hayan sido definidas a lo largo de la página PHP.

Una de las posibilidades que ofrece PHP es la de asignar una función a una variable. Esto puede resultar de gran utilidad cuando sea una condición la que determine qué función debe ser llamada. Cuando una variable hace referencia a una función, la función puede ser llamada colocando los paréntesis que contienen los argumentos (si hay) después del nombre de la variable.

```
<?
      $ciudad = “Madrid”;
      switch  ($estacion) {
            case “1”:
                  $verZona = “primavera”;
                  break;
            case “2”:
                  $verZona = “verano”;
                  break;
            case “3”:
                  $verZona = “otono”;
                  break;
            case “4”:
                  $verZona = “invierno”;
                  break;
      }
      $verZona($ciudad);
?>
```

De esta forma se puede ejecutar una de las cuatro funciones posibles con el parámetro $ciudad:

```
<?
      primavera ($ciudad);
      verano($ciudad);
      otono($ciudad);
      invierno($ciudad);
?>
```

PHP también cuenta con un conjunto de funciones para la manipulación y uso de las variables de tipo string o cadenas de texto. Dado que PHP hereda en parte de PERL cuenta con un juego de funciones potentes que no describiremos dado el número de ellas.

Pensando en el típico portal de Internet que está formado por varios servicios: servicio de email, un buscador, titulares de noticias, información sobre el tiempo, etc. En este caso, la acción más acertada será hacer cada uno de estos servicios en un fichero independiente, de forma que en la página principal se puedan añadir todos o aquellos que sean necesarios. Mediante PHP se crean aplicaciones independientes que se ofrecen a todos los programadores de este lenguaje; de esta forma, es posible acceder a infinidad de herramientas y utilidades que no se encuentran en la distribución oficial de PHP pero que, obtenidas por separado y configuradas correctamente, pueden dar solución a muchas necesidades que tenga el programador en ese momento.

Este es el caso de la clase FastTemplate, que se incluye en las aplicaciones mediante el uso de la función include() y que tiene su origen en un paquete similar para el lenguaje Perl. Esta clase o utilidad permite separar el código PHP del HTML mediante el uso de una o varias plantillas (templates). La idea original está basada en un primer fichero que es la plantilla original y que contiene diferentes llamadas a las distintas partes que se desean incluir. Por otro lado, existe una segunda página PHP que contiene el código necesario para hacer uso de esta clase y sustituir las variables utilizadas por el contenido real, que será luego mostrado al usuario en su navegador.

Para utilizar el sistema de plantillas proporcionado por FastTemplate hay qye realizar cuatro pasos:

1. Definir la plantilla y sus variables.
2. Asignar los diferentes valores a las variables de la plantilla.
3. Interpretar la plantilla.
4. Mostrar el resultado.

Veamos ahora un ejemplo bastante sencillo de cómo se crea una página a través de una plantilla previamente definida.

En primer lugar, hay que definir la plantilla a utilizar. En nuestro caso podrá ser la siguiente:

```
<title>{TITULO_PAGINA}</title>
{CONTENIDO}
Este es un libro sobre... {TITULO_LIBRO}
```

En primer lugar, la página creada no tiene extensión “php” como es habitual, sino que su extensión es “tpl”, que especifica que se trata de una template (plantilla).

En segundo lugar, se han incluido tres variables encerradas entre llaves (TITULO_PAGINA, TITULO_LIBRO Y CONTENIDO). Estas son las variables que serán sustituidas por su valor real cuando la plantilla sea procesada.

Ahora veamos el fichero PHP que se encargará de interpretar la plantilla antes creada:

```
<?
      $texto=“Este ejemplo muestra uso de plantillas.<br>”;
      include “class.FastTemplate.php”;
      $tpl = new FastTemplate(“.”);
      $tpl->define(array(“primera” => “plantilla1.tpl”));
      $tpl->assign(“TITULO_PAGINA”,”Probando plantillas”);
      $tpl->assign(“TITULO_LIBRO”,”PHP”);
      $tpl->assign(“CONTENIDO”,$texto);
      $tpl->parse(“MAIN”,”primera”);
      $tpl->FastPrint();
?>
```

En primer lugar, se ha incluido la clase “class.FastTemplate.php” que es la que ofrece la funcionalidad del uso de las plantillas y que se encuentra en el mismo directorio donde están las páginas.

La segunda línea crea una instancia de dicha clase que se llamará $tpl, pasando como parámetro el path (camino) en el que se encuentra la plantilla (en este caso, está en el mismo directorio).

El siguiente paso es definir la plantilla mediante un nombre y cómo se llama el fichero utilizando la función define().

La siguiente acción es registrar cada una de las variables definidas en la plantilla (plantilla1.tpl) con el valor que realmente contienen. En nuestro ejemplo se han definido tres variables diferentes: la primera de ellas (TITULO_PAGINA) dará el nombre a la página, la segunda (TITULO_LIBRO) contiene una cadena de texto que se encuentra almacenada en la variable $texto y, la tercera, (TITULO_LIBRO) tendrá como valor el texto “PHP”.

El siguiente paso es interpretar la plantilla con los valores reales que deben contener las variables, este proceso es el resultado de juntar la página patrón con los datos que debe mostrar. A continuación, y si no se ha producido ningún error, se muestra el resultado por pantalla mediante el uso de la función FastPrint().

Dentro de una página PHP se puede acceder a una serie de variables que simplifican el código de las expresiones y scripts PHP. Las más importantes son:

§ PHP_SELF

El nombre del fichero que contiene el script que se esta ejecutando, relativo al directorio raíz de los documentos. Si PHP se está ejecutando como intérprete de línea de comandos, esta variable no está disponible.

§ HTTP_GET_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP GET. Sólo está disponible si –variable tracking-- ha sido activado mediante la directiva de configuración track_vars o la directiva *<?php_track_vars?>*.

§ HTTP_POST_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP POST. Sólo está disponible si –variable tracking—ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

Para realizar proyectos serios, de gran duración, se recomienda aislar la lógica de la aplicación empleado clases. La orientación orientada a objeto ha demostrado que un proyecto puede realizarse y mantenerse más fácilmente utilizando clases y objetos.

En PHP utilizamos la palabra reservada class para definir una clase.

Ejemplo,

```
<?php
      class Coche {
            var $modelo;
            var $color;

            function Coche($modelo, $color) {
                  $this->modelo = $modelo;
                  $this->color = $color;
            }

            function obtenerModelo() {
                  return $this->modelo;
```

```
                }

                function obtenerColor() {
                      return $this->color;
                }
        }
?>
```

En el ejemplo anterior definimos la clase Coche, que tiene dos atributos *modelo* y *color*. La función *Coche*, actúa como constructor de la clase, y se ejecuta cada vez que instanciamos un objeto de este tipo. Como se puede ver, no existe destructor, pues PHP implementa un recolector de basura que se encarga de liberar el espacio de memoria.

PHP también permite la herencia simple de clases, mediante la palabra reservada *extends*. Por ejemplo, comparado con otros lenguajes orientados a objeto, PHP deja bastante que desear. No permite polimorfismo ni sobrecarga de operadores. No existen restricciones en cuanto al ámbito de los atributos, por lo que todos los atributos de cualquier clase son visibles desde cualquier otra.

Las clases PHP suelen tener extensión .inc, y pueden ser empleadas desde otro fichero PHP utilizando la cláusula require() o include().

```
<?php
      class Coche extends Vehiculo {
      .
      .
      .
      }
?>
```

Ejemplos de uso

1.- Vamos a ver un caso especial, como descargar un archivo desde un formulario. Para ello utilizaremos una etiqueta INPUT de tipo FILE, soportada a partir de las versiones de los navegadores Nestcape Navigato 2.0 e Internet Explorer 4.0.

El formulario debe usar el método post, y el atributo post, y el atributo enctype debe tener el valor multipart/form-data. Además al formulario debemos añadirle un campo oculto de nombre MAX_FILE_SIZE, al cuál le daremos el valor en byte del tamaño máximo del archivo a descargar.

```
   <FORM  ENCTYPE="multipart/form-data"  ACTION="7-3.php3"  METHOD="
post ">
   <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="100000">
   <INPUT NAME="archivo" TYPE="file">
   <INPUT TYPE="submit" VALUE="Descargar Archivo">
   </FORM>
```

Cuando el formulario es enviado, PHP detectará automáticamente que se está descargando un archivo y lo colocará en un directorio temporal en el servidor. Dicho directorio será que el que esté indicado en el archivo de configuración php3.ini, o en su defecto en el directorio temporal del sistema.

Cuando PHP detecta que se está descargando un archivo crea varias variables con el prefijo del nombre del archivo pero con distintas terminaciones. La variable terminada en _name contiene eligoenlinea">_name contiene el nombre original del

archivo, la terminad en _size el tamaño en bytes de éste, y la variable terminada en _type nos indicará el tipo de archivo si éste es ofrecido por el navegador.

Si el proceso de descarga no ha sido correcto la variable archivo tomará el valor none y _size será 0, y si el proceso ha sido correcto, pero la variable terminada en _size da 0, quiere decir que el archivo a descarga supera el tamaño máximo indicado por MAX_FILE_SIZE.

Una vez descargado el archivo, lo primero que debemos hacer es moverlo a otro lugar, pues sino se hace nada con él, cuando acabe la ejecución de la página se borrará.

Veamos un ejemplo de todo lo dicho.

```
<HTML>
<BODY>
<?PHP
      if ($enviar) {if ($enviar) {
      if ($archivo != "none" AND $archivo_size != 0){
            echo "Nombre: $archivo_name <BR>\n";
            echo "Tamaño: $archivo_size <BR>\n";
            echo "Tipo: $archivo_type <BR>\n";
            /* para Windows
            if (! copy ($archivo, "C:\\TEMP\\".$archivo_name)) {
                  echo   "<h2>No   se   ha   podido   copiar   el
               archivo</h2>\n";
            }*/

      /* para Linux/Unix */
            if (! copy ($archivo, "/tmp/".$archivo_name)) {
                  echo   "<h2>No   se   ha   podido   copiar   el
               archivo</h2>\n";
            }
      } elseif ($archivo != "none" AND $archivo_size == 0) {
      echo "<h2>Tamaño de arcft: 75">echo "<h2>Tamaño de archivo
superado</h2>\n";
      } else {
            echo   "<h2>No   ha   escogido   un   archivo   para
         descargar</h2>\n";
            echo "<HR>\n";
}
?>
<FORM ENCTYPE="multipart/form-data" ACTION="<?php echo $PHP_SELF
?>" METHOD="post">
<INPUT type="hidden" name="MAX_FILE_SIZE" value="100000">
<p><b>Archivo a descargar <b><br>
<INPUT type="file" name="archivo" size="35"></p>
<p><INPUT type="submit" name="enviar" value="Aceptar"></p>
</FORM>
</BODY>
</HTML>
```

2.- Vamos a ver una aplicación, un ejemplo, de todo lo visto hasta ahora. Escribiremos un script que sirva para buscar una determinada cadena (que recibiremos de un formulario, y la almacenamos en la variable $buscar), dentro de nuestra base de datos, concretamente dentro del campo "nombre".

En primer lugar escribiremos el texto HTML de la página web que nos servirá como formulario de entrada, la llamaremos formulario.htm.

```
<html>
```

```
    <body>
         <form      method      =      "POST"      action      =
"http://mysevidor/php/buscador .php3">
     <strong>Palabra clave:</strong>
     <input type="text" name="buscar" size="20"><br><br>
     <input type="submit" value="Buscar">
    </form>
    </body>
   </html>
```

El siguiente script de búsqueda lo llamaremos buscador.php3, y será el encargado de hacer la búsqueda en la BD, y devolver por pantalla los registros encontrados.

```
   <html>
   <body>
   <?php
         if (!isset($buscar)){
               echo"Debe especificar una cadena a buscar";
               echo &quo
               echo"<p>Debe especificar una cadena a buscar</p> \n";
               echo"<p><a href=buscador_bd.htm>Volver</p> \n";
               echo "</html></body> \n";
               exit;
         }
      $link = mysql_connect("localhost", "nobody");
      mysql_select_db("mydb", $link);
      $sql = "SELECT * FROM agenda WHERE nombre LIKE '%$buscar%'
   ORDER BY nombre";
      $result = mysql_query($sql, $link);
      if ($row = mysql_fetch_array($result)){
         echo "<table border = '1'> \n";
         //Mostramos los nombres de las tablas
         echo "<tr> \ndigo" style="margin-left: 50">echo "<tr> \n";
         mysql_field_seek($result,0);
         while ($field = mysql_fetch_field($result)){
               echo "<td><b>$field->name</b></td> \n";
            }
         echo "</tr> \n";
         do {
               echo "<tr> \n";
               echo "<td>".$row["id"]."</td> \n";
               echo "<td>".$row["nombre"]."</td> \n";
               echo "<td>".$row["direccion"]."</td> \n";
               echo "<td>".$row["telefono"]."</td> \n";
               echo "<td><a
               href='mailto:"
               href='mailto:".$row["email"]."'>".
               $row["email"]."</a></td> \n";
               echo "</tr> \n";
         } while ($row = mysql_fetch_array($result));
         echo "<p><a href=buscador_bd.htm>Volver</p> \n";
         echo "</table> \n";
      } else {
      echo "<p>¡No se ha encontrado ningún registro!</p>\n";
      echo "<p><a href=buscador_bd.htm>Volver</p> \n";
      }
   ?>
   </body>
   </html>
```

Lo primero que comprobamos es que el contenido de la variable $buscar que recibimos del la página web formulario.htm no es una cadena vacía, y esto lo hacemos con la función isset() que devuelve 'falso' si la variable que recibe está vacía. A la función le anteponemos el signo admiración (!) que es equivalente a un NOT, para convertirlo en 'verdadero' en caso de que la variable esté vacía, y en es caso terminamos la ejecución del script con exit.

Lo más importante de este script, es sin duda la sentencia SQL que le enviamos al servidor MySQL, y más concretamente la condición que le imponemos, WHERE nombre LIKE '%$buscar%'. Con la sentencia LIKE buscamos cualquier ocurrencia de la cadena contenida en $buscar, mientras que con los signos de porcentaje (%) indicamos el lugar de la coincidencia, por ejemplo, si hubiésemos puesto nombre LIKE '%$buscar', buscaríamos cualquier ocurrencia al final del campo "nombre", mientras que si hubiésemos puesto nombre LIKE '$buscar%', buscaríamos cualquier ocurrencia al principio del campo "nombre".

Las últimas novedades que hemos incorporado, son las funciones mysql_fetch_field(), con el que obtenemos información acerca de las características de cada campo, como su nombre, tipo, longitud, nombre de la tabla que los contiene, etc. Pero para ejecutar la función anterior debemos colocar el puntero en el primer campo, y eso lo logramos con la función mysql_field_seek(), la cuál mueve el puntero interno a la posición indicada.

3.- En este ejemplo utilizaremos sockets para obtener sólo las cabeceras HTTP que se reciben al solicitar una página web y que el usuario nunca ve. Para evitar mostrar la página web, crea una condición para que sólo se muestre la parte de la cabecera despreciando el código de la página.

```
<?
        $fp = fsockopen("www.php.net",80);
        fputs ($fp, "GET / HTTP/1.0\r\n\r\n");
        while (!feof($fp)) {
                $caracter = fgets($fp,1);
                if ($caracter =="<") {
                        exit;
                } else {
                if (ord($caracter)==13) {
                        print ("<br>");
                } else {
                        print ($caracter);
                }
                }
        }
fclose($fp);
?>
```

Referencias

PHP 4. Guía Práctica para usuarios.
Autor: Esteban Trigos García
Editorial: ANAYA Multimedia, S.A. 2000

ISBN: 84-415-1079-2

Guía para usuarios de nivel Básico / Medio para comenzar a trabajar con PHP. Siguiendo la línea de las guías ANAYA, es un libro sencillo con muchos ejemplos y en castellano, cosa que se echa en falta a la hora de buscar libros de este tipo.

Manual de utilidades & Trucos Internet n º 2
Artículo: Desarrollo web con PHP3.
Páginas: 90-98
Autores del artículo: Rafael Morales Dorado y José Manuel Soto Díaz
Editorial: VNU business publications

Revista que por su envergadura se podría calificar como libro. En sus 198 páginas, sin apenas publicidad, se tratan temas de Internet, utilidades, tutoriales, instalaciones paso a paso, cursos, ... Una guía de referencia para con un completo CD de utilidades a tener por todo usuario de Internet.

Programación Actual n º 44
Artículo: Montaje de un servidor web.
Páginas: 72-75
Autores: Luis Miguel Cabezas y Pedro Miguel Casas
Editorial: Prensa Técnica.

Artículo que muestra paso a paso el montaje de un servidor Web utilizando Xitami como servidor, MySQL como gestor de BD y PHP como motor para la generación de contenido dinámico.

http://www.php.net
Sitio oficial de PHP
http://www.zend.com
Sitio oficial del motor de PHP Zend.
http://personales.jet.es/jcantero/linuxPHP/recursos-php.html
Recursos para PHP en Linux.

Java Servlets

Introducción

Fueron introducidos por Sun en 1996 como pequeñas aplicaciones Java para añadir funcionalidad dinámica a los servidores web. Los Servlets, al igual que los scripts CGI, reciben una petición del cliente y generan los contenidos apropiados para su respuesta, aunque el esquema de funcionamiento es diferente.

El API Servlet ha sido una extensión a Java, desde el JDK 1.2 es un paquete estándar. Los Servlets no son específicos para cada servidor Web, el API es flexible y soporta muchos servidores Internet. Los programadores que están interesados en los servlets, lo están en el HttpServlet concretamente.

Cómo funciona un servlet

Un servlet es una clase Java que implementa la interfaz **Servlet**, que define cinco métodos:

service(), es el corazón de los servlets. El servidor que invoca al método service() para ejecutar respuestas. El método service() acepta como parámetros objetos **ServletRequest**, que encapsula la petición del cliente, y **ServletResponse**, que dispone de métodos para devolver información al cliente.

init(), es el lugar donde se inicializa el servlet. Es un método que acepta como parámetro en objeto de tipo **ServletConfig**, que contiene la configuración del servidor, y se garantiza que solamente se llamará una vez durante la vida del servlet.

getServletConfig(), debe devolver el objeto **ServletConfig** que se pasa como parámetro al método *init*().

destroy(), libera el servlet, se llama cada vez que el servlet debe ser descargado; por ejemplo, debido a que hay que cargar una nueva versión del servlet. Todos los recursos del sistema bloqueados por *init*() son liberados al invocar este método y se garantiza que solamente se llamará una vez durante la vida del servlet.

getServletInfo(), devuelve una cadena con la información de Copyright.

Para asegurar un óptimo rendimiento, el servidor solamente carga una instancia de cada servlet. Una vez cargado, el servlet permanece en memoria, estando disponible en cualquier instante para procesar cualquier petición. Todos los servlets asociados con un servidor se ejecutan dentro de un proceso simple. Cada vez que llega una petición, la JVM crea un hilo Java para manejar la petición, reduciendo así la sobrecarga del sistema.

Por lo tanto, varias tareas pueden llamar simultáneamente al método *service*(), así que la sincronización dentro de *service*() debe ser una premisa a no olvidar. Los servlets, al igual que los *applets*, tienen un ciclo de vida predeterminado. Su conocimiento es algo indispensable si queremos sacar provecho a toda la potencia y versatilidad que proporcionan. El ciclo de vida de una servlet consiste en tres fases: **inicialización, proceso de peticiones y la fase de terminación**.

Inicialización

Un servlet puede ser inicializado de tres formas posibles:

1. Automáticamente cuando el servidor arranca por primera vez, si así se ha configurado. Esta opción depende por completo de las características propias del servidor encargado de dar soporte a los servlets.
2. La primera vez que un cliente realiza una petición. Éste suele ser el caso más frecuente.
3. Cuando el servlet se vuelve a cargar. Esto puede suceder, por ejemplo, cuando éste ha sido modificado y compilado de nuevo. Si esto ocurre, cuando llega una nueva petición el servidor, éste último se da cuenta de que el servlet ha sido recompilado y entonces lo carga de nuevo en memoria.

Una vez que el servlet ha sido cargado, el servidor crea una instancia del mismo y llama al método init. Este método será invocado una sola vez con independencia del número de peticiones que lleguen al servlet. Esto significa que todas las tareas que tengan un propósito global dentro de la aplicación Web a la que pertenece ese servlet deben ser llevadas a cabo dentro de este método: configuración de parámetros globales, inicialización de variables estáticas compartidas por los threads del servlet, etc.

Proceso de las peticiones

Por cada petición, el servidor crea un objeto Request (petición) y un objeto Response (respuesta). En ese momento llama al método *service* del servlet, pasándole como parámetros ambos objetos. El método *service* toma los datos de la petición del objeto Request (éstos pueden ser por ejemplo los datos procedentes de un formulario o la información que transporta la propia dirección URL), procesa la petición y utiliza los métodos del objeto Response para enviar la información de vuelta al cliente. Lo más usual es que ésta sea una página HTML aunque también se puede responder con cualquier tipo de datos.

El método *service* responde a una abstracción que ofrece la clase *GenericServlet*. Con frecuencia suele ser sustituido por los métodos *doGet* o *doPost*, que están directamente relacionados con el protocolo HTTP, aunque su funcionamiento es exactamente el mismo. Estos métodos pertenecen a la clase HttpServlet que depende directamente de la clase GenericServlet.

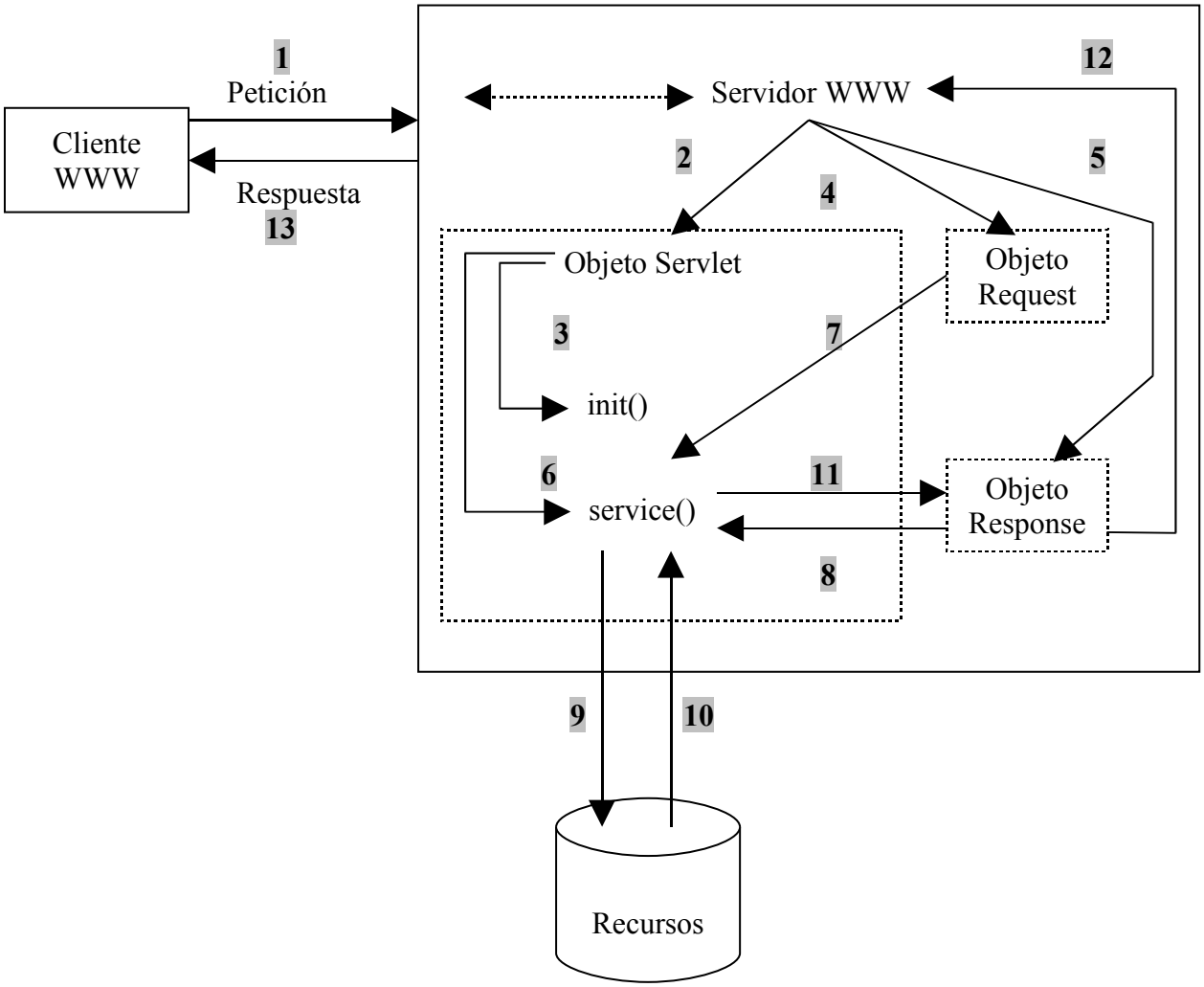
Es importante incidir en que las distintas peticiones se atienden de forma concurrente. Esto permite la coordinación de actividades entre varios clientes. Es posible incluso utilizar variables estáticas para mantener una zona común de información a todos los clientes.

Terminación

Cuando el servidor no necesita el servlet, o el servlet va a ser cargado de nuevo, el servidor llama al método *destroy*. Este método se encarga de liberar todos los recursos que el servlet consume. Como en los casos anteriores, este método puede ser sobrescrito para realizar las operaciones que se consideren necesarias antes de que el servlet "muera".

Ciclo de vida de un servlet

Una representación del ciclo de vida de un servlet:



- El cliente realiza una petición (1). Lo más usual es que la carga del servlet se haga en este momento, como consecuencia de esa primera petición por parte de un cliente, aunque también existe la posibilidad de configurar el servidor de forma que por defecto se carguen algunos servlets cuando éste sea arrancado.
- El servidor crea una instancia del servlet, o lo que es lo mismo, un objeto servlet (2).
- El servidor llama al método init del servlet (3).

Los tres puntos anteriores solamente suceden una vez al iniciarse el servlet. A partir de ese momento todas las peticiones son atendidas por el mismo objeto servlet. El servidor crea para cada una de ellas un thread o hilo de ejecución distinto. Así, cada vez que el servidor reciba una petición ocurre lo siguiente:

- El servidor crea un objeto Request que almacena la información relativa a la petición (4).
- El servidor crea un objeto Response que hace posible la respuesta hacia el cliente por parte del servidor (5).

- El servidor llama al método Service (o doGet, o doPost) del servlet (6), y le pasa como parámetros los objetos Request(7) y Response(8).
- El método Service procesa la petición accediendo a cualquier tipo de recurso (9), si es necesario, y tomando la información necesaria (10).
- El método Service utiliza los métodos del objeto Response (11) para enviar la respuesta al servidor (12) y de ahí al cliente (13).

Servlets y el protocolo HTTP

La clase HttpServlet es una subclase de la clase GenericServlet que se ha especializado fundamentalmente para manejar todos los detalles del protocolo HTTP.

```
class java.lang.Object
        |
        +———class javax.servlet.GenericServlet
                    |
                    +———class javax.servlet.http.HttpServlet
```

En la clase HttpServlet el método *service* no es abstracto, tal y como ocurre en la clase *GenericServlet*. Cuando el servidor llama al método *service* de la clase HttpServlet, determina si el método de envío, lo que se conoce como request meted, ha sido GET o POST e invoca al método que corresponda, doGet o doPost respectivamente.

GET es el método que se utiliza siempre que la información se envía mediante la propia dirección URL, o cuando así se indica en un formulario haciendo que el atributo METHOD de la etiqueta FORM tome el valor "GET". El método POST se utiliza exclusivamente en los formularios y se indica igualmente mediante el atributo METHOD de la etiqueta FORM.

La diferencia que existe entre ambos métodos, más allá de lo explicado, reside en el hecho de que los datos enviados mediante el método GET utilizan variables de entorno mientras que los enviados mediante el método POST se transfieren al servidor a través de los canales de entrada/salida estándar. El espacio del que dispone un servidor para variables de entorno es limitado por lo que en general se recomienda utilizar el método POST cuando el número y tamaño de la información sea considerable.

De la misma forma, los objetos *Request* y *Response* de los que antes hemos hablado se transforman, circunscritos al protocolo HTTP, en los objetos HttpServletRequest y HttpServletResponse. Estos dos objetos constituyen el vehículo a través del cual se produce la comunicación entre el cliente y el servidor.

Métodos de ServletRequest y HttpServletRequest

✍ **Public abstract Enumeration getParameterNames()**

Devuelve los nombres de los parámetros de la petición como un objeto Enumeration de cadenas –objetos de tipo String-, o vacío si no existe ningún parámetro. Los métodos de la interfaz Enumeration nos permitirán recorrer la lista con todos los nombres de los parámetros.

```
for(Enumeration e=req.getParameterNames(); e.hasMoreElements();)
      {
            out.println (e.nextElement());
```

```
}
```

El método hasMoreElements devuelve true en el caso de que todavía queden elementos por recorrer. El método nextElement sirve para obtener el siguiente elemento del conjunto.

✍ **Public abstract String[] getParameterValues (String name)**

Devuelve todos los valores del parámetro que se pasa como argumento, como un array de objetos de tipo String, o null si el nombre del parámetro no existe.

✍ **Public abstract String getQueryString()**

Devuelve el contenido que tendría la variable de entorno QUERY_STRING descrita formalmente por la interfaz CGI.

Como es bien sabido, una dirección URL puede utilizarse como vehículo para enviar información desde el navegador hasta el servidor. El formato en el que se encuentra codificados los datos consiste en un flujo de pares *nombre=valor* separados por el signo "ampersand" (&).

Además, la información se encuentra codificada en formato URL "url-codificada" (URLencoded), lo que significa que los espacios en blanco se sustituyen por el signo más (+) y los caracteres especiales se indican en hexadecimal precedidos por el signo tanto por ciento: **%xx**.

Según dispone la especificación de la interfaz CGI, en la variable de entorno QUERY_STRING queda almacenada cualquier cosa que se escriba a continuación del signo "cierre de interrogación" ? en la dirección URL correspondiente.

Los primeros programas CGI hechos en C o Perl solían tomar esta variable, la procesaban y entonces obtenían los nombres de los parámetros así como sus valores. El entorno que ofrecen los servlets efectúa este proceso de manera transparente al programador por lo que no suele ser necesario conocer el valor de la variable QUERY_STRING. A veces puede resultar muy útil en tareas de depuración, revisión y control del código desarrollado.

En este sentido cabe mencionar el método encodeURL de la clase HttpServletResponse, el cual sirve para codificar una dirección URL. Suele utilizarse con frecuencia en aquellos casos en los que un servlet genera una página HTML como respuesta y en ella existen direcciones URL que contienen parámetros. Nunca se debe olvidar aplicar la codificación ya que de lo contrario el funcionamiento de ese enlace no sería el deseado. Los parámetros no llegarían al servidor o lo haría con errores.

✍ **Public abstract String getParameter(String name)**

Devuelve el valor del parámetro cuyo nombre pasa como argumento, o null si no existe un parámetro con dicho nombre. En el caso de que se trate de un parámetro que puede tomar múltiples valores debemos utilizar getParameterValues.

Acabamos de ver el método getQueryString, el cual proporciona el valor de la variable QUERY_STRING descrita por la interfaz CGI. La API de Java servlets cuenta con algunos métodos más, gracias a los cuales podemos conocer toda aquella información que es puesta por el servidor a disposición de un

programa CGI mediante variables de entorno, tal y como describe la interfaz CGI.

Correspondencia entre la API Java Servlets y algunas de las variables descritas por la interfaz CGI	
Variable CGI	**Método de la interfaz HttpServletRequest**
REQUEST_METHOD	*Public abstract String getMethod()* Método HTTP empleado para enviar información al servidor: normalmente GET oPOST.
SCRIPT_NAME	*Public abstract String getServletPath()* Dirección URL que identifica al Servlet.
PATH_INFO	*Public abstract String getPathInfo()* Es posible pasar cierta información en la dirección URL de invocación del servlet. Esta información se sitúa tras el nombre del servlet en la dirección URL y está precedida por el signo de interrogación (?) que indica el comienzo de la información.
QUERY_STRING	*Public abstract String getQueyString()* La parte que figura a partir del signo de cierre de interrogación (?) en la dirección URL del servlet.
SERVER_PROTOCOL	*Public abstract String getProtocol()* La versión y extensión del protocolo empleado por el servidor de información.
CONTENT_LENGTH	*Public abstract int getContentLength()* Tamaño en bytes de los datos transmitidos.
CONTENT_TYPE	*Public abstract String getContentType()* Tipo MIME de los datos.
SERVER_NAME	*Public abstract String getServerName()* Nombre del servidor tal y como se indicó en la parte host de la dirección URL de invocación del servlet. Este nombre puede ser una dirección IP o un alias de DNS.
REMOTE_ADDR	*Public abstract String getRemoteAddr()* Dirección IP del agente que invocó al servlet.
REMOTE_HOST	*Public abstract String getRemoteHost()* Nombre completo del agente que invocó al servlet.

Métodos de ServletResponse y HttpServletResponse

✍ **Public abstract ServletOutputStream getOutputStream() throws IOException**

Devuelve un canal de salida que se utiliza para escribir la respuesta que el servidor envía al cliente. Antes de utilizar el canal, éste puede ser convertido a otro canal de cualquier tipo aprovechando las facilidades que ofrece Java.

✍ **Public abstract void setContentLength (int len)**

Marca la cabecera CONTENT_LENGTH para esa respuesta, o lo que es lo mismo, indica al navegador del cliente cuál es el tamaño en bytes de la respuesta que va a recibir.

✍ **Public abstract void setContentType(String type)**

Establece la cabecera CONTENT_TYPE para esa respuesta, o lo que es lo mismo, indica al navegador de cliente cuál es el tipo MIME de la información que va a recibir. Evidentemente, este método y el anterior deben ser utilizados antes de enviar cualquier otro dato.

✍ **Public abstract void sendRedirect(String location) throws IOException**

Envía una respuesta al cliente que consiste en una redirección a la dirección URL especificada por el argumento.

Ejemplos de uso

1.- Ejemplo simple de un servlet que devuelve una cadena al cliente. Típico ejemplo Hola mundo.

```
public class SimpleServlet extends HttpServlet
  {
      /**
       * Manejo del método GET construyendo una
       * simple página web.
       */
      public void doGet (HttpServletRequest request,
                         HttpServletResponse response)
      throws ServletException, IOException
      {
    PrintWriter        out;
    String             title = "Salida Simple Servlet";

    // poner content type
    response.setContentType("text/html");

    // escribir los datos de la respuesta
    out = response.getWriter();

    out.println("<HTML><HEAD><TITLE>");
    out.println(title);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + title + "</H1>");
    out.println("<P>La salida de SimpleServlet.");
    out.println("</BODY></HTML>");
    out.close();
      }
  }
```

Referencias

Libros/Revistas:

Desarrollo de Servidores Java: Servlets y JSP

Autores: Genoveva López Gómez, Francisco Javier Soriano Camino, Francisco Fernández Velasco, Juan Pablo Rojas Jiménez.

Referencia obligada para el programador de Servlets y JSP. Es difícil encontrar referencias en castellano tan buenas. Libro de la Facultad de Informática de la Universidad Politécnica de Madrid.

Java Server Programming (Professional)

Autores:Danny Ayers, Hans Bergsten, ...

Editorial: Wrox
Páginas: 1121
ISBN: 1-861003-77-7

Libro completísimo en lo que se refiere a la programación de servidores Java. Destinado a programadores avanzados, es un libro que trata temas como servlets, JSP, XML, EJBs, JDBC, JNDI, JavaMail, RMI, CORBA, Jini/JavaSpaces, Apache,... Mencionar que el número de autores de este libro asciende a 12.

Enlaces:

http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

Tutorial de Servlets con referencias también a JSP.

JavaServer Pages (JSP)

Introducción

Basadas en la tecnología de Servlets, las JSP se han convertido en uno de los principales elementos de la programación Java en el servidor.

Las JSP combinan cualquiera de los lenguajes de marcas como podrían ser HTML o XML, con fragmentos de código Java, para permitir la generación de páginas Web dinámicamente, de forma similar a como funcionan las páginas ASP (Active Server Pages) de Microsoft y su Internet Information Server. Cada página JSP es compilada automáticamente en el momento de su primer acceso, traduciéndose a un Servlet por el motor JSP. Dicho servlet será el que se ejecute en los sucesivos accesos a la misma, siempre y cuando el código JSP asociado a la página no sea modificado.

JSP proporciona multitud de mecanismos para comunicar la página JSP con clases Java, Servlets, Applets y el propio servidor Web desde donde se ejecuta, permitiendo dividir las distintas funcionalidades previstas para nuestras aplicaciones Web en distintos componentes con interfaces bien definidas, enlazados entre sí mediante una o varias páginas JSP.

Este modelo permite la subdivisión de las distintas tareas a realizar, posibilitando que el equipo de desarrolladores construya los componentes a medida, que luego podrán ser "agrupados" por el diseñador de la página con un número limitado de invocaciones a métodos bien definidos, separándose la lógica de la aplicación Web, de la representación de los datos manejados por ésta, que es lo que debería buscarse siempre en el diseño de cualquier aplicación, ya no sólo en lo que se refiere a entornos Web.

JSP (JavaServer Pages) constituyen una especificación que ya está implementada por un gran número de servidores Web, en los que el código asociado a una página JSP podrá ejecutarse sin ningún tipo de cambios, permitiendo así una mayor portabilidad entre las distintas soluciones ofrecidas por los distintos fabricantes (ya sean de libre distribución o de pago).

Cómo funciona JavaServer Pages (JSP)

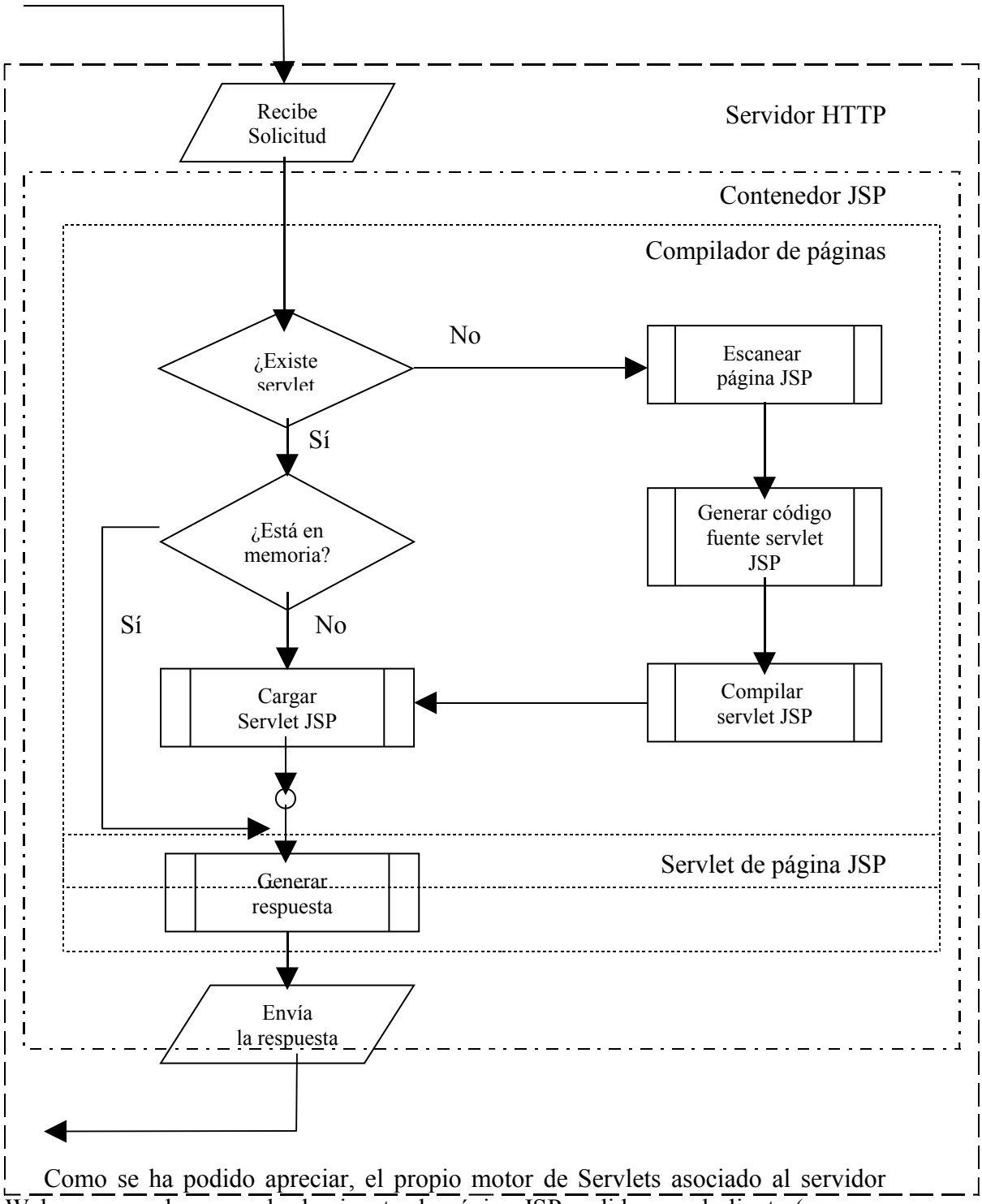
Cuando se produce una solicitud de una página JSP por parte de un navegador, se comienza comprobando si se trata de la primera solicitud de dicha página. En ese caso se compila a un servlet, que es ejecutado y cuya salida es devuelta al usuario que realizó la petición. Las siguientes solicitudes son más eficientes al no necesitar que dicha página sea de nuevo compilada, ya que sólo se invoca al servlet que se generó en la petición inicial. En este sentido esta tecnología es más ventajosa que otras como ASP, que requieren cada vez una nueva compilación.

El proceso de compilación de las páginas consiste en analizar su contenido buscando etiquetas JSP y traduciendo éstas a código Java equivalente. El contenido estático de las páginas (código html) es traducido a cadenas de caracteres en lenguaje Java. Las etiquetas de componentes JavaBeans son traducidas a su correspondiente objeto y llamadas a métodos. Una vez que el código del servlet ha sido construido (se ha codificado su método service()), el compilador de páginas llama al compilador de Java para compilar el código fuente y añade el fichero de bytecodes resultante al directorio apropiado del contenedor JSP.

Una vez que el servlet correspondiente a la página *.jsp* ha sido generado, el compilador de páginas invoca al nuevo servlet para generar la respuesta al cliente. Mientras que el código de la página *.jsp* no se altere, todas las referencias a la página

ejecutarán el mismo servlet. Esto supone una cierta demora en la primera referencia a la página, aunque existen mecanismos en JSP para precompilar la página *.jsp* antes de que se haya producido la primera petición.

El siguiente gráfico muestra el proceso para crear y ejecutar servlets JSP:



Como se ha podido apreciar, el propio motor de Servlets asociado al servidor Web va a ser el encargado de ejecutar la página JSP pedida por el cliente (una vez compilada).

Resulta posible ver el código asociado al servlet generado a raíz de la compilación de una página JSP pedida por un determinado cliente examinando el árbol de directorios asociado al motor de servlets, Esto es especialmente útil en el caso de pretender la depuración de una determinada página JSP, pues de otra forma sería imposible llevar a cabo esta tarea.

Si se echa un vistazo a los ficheros de ayuda asociados a la documentación que acompaña a la implementación de la especificación JSPv1.1, y en particular la asociada a javax.servlet.jsp, aparecen dos interfaces JSPPage y HttpJspPage, que definen los métodos que deben aparecer en una clase generada a raíz de la compilación de una página JSP:

- JSPPage: Es el interfaz que debe implementar cualquier clase generada con un procesador JSP. Dicho interfaz define un protocolo con dos métodos, jspInit() y jspDestroy(), los cuales se van a invocar en el momento de inicialización y de destrucción de la clase asociada a la página JSP, requerida por el cliente y ejecutada a través del motor de servlets del servidor. Estos dos métodos pueden ser definidos por el autor de la página JSP.
- HttpJspPage: Este interfaz, que extiende el interfaz anterior, define un tercer método, _jspService (HttpServletRequest req, HttpServletResponse res) que va a ser el que contenga el código asociado a la página JSP en sí, siendo su creación responsabilidad del motor JSP, por lo que no va a poder ser sobrescrito por el autor de la página JSP.

Como ya se ha comentado a lo largo de este capítulo, la primera vez que se accede a una página JSP desde su última modificación, el motor JSP (JSPengine) va a traducir el contenido de la página en cuestión al código Java correspondiente a su Servlet equivalente. De este modo, dicho servlet heredará los métodos jspInit() y jspDestroy() de su clase "padre", que por omisión estarán vacíos, así como el esqueleto inicial asociado al método _jspService(...) que va a ser el que realmente efectúe todo el "trabajo" de la página.

En lo que se refiere a los métodos jspInit() y jspDestroy(), a pesar de que no suele ser muy común su empleo, simplemente resaltar que el programador de la página JSP podría sobrescribirlos para efectuar distinta tareas de inicialización y/o salvaguarda de estado.

Más interesante es ver el código inicial generado para el método _jspService(), y el cómo las distintas etiquetas JSP lo modifican. Así pues, el código por omisión generado para la página JSP según la especificación de JSP versión 1.1 sería:

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Vector;
import org.apache.jasper.runtime.*;
import java.beans.*;
import org.apache.jasper.JasperException;


public class _0002fHelloWorld_0002ejspHelloWorld_jsp_0 extends
HttpJspBase {


    static {
    }
    public _0002fHelloWorld_0002ejspHelloWorld_jsp_0( ) {
    }

    private static boolean _jspx_inited = false;

    public final void _jspx_init() throws JasperException {
    }

public void _jspService(HttpServletRequest request,
HttpServletResponse response)
```

```
            throws IOException, ServletException {

            JspFactory _jspxFactory = null;
            PageContext pageContext = null;
            HttpSession session = null;
            ServletContext application = null;
            ServletConfig config = null;
            JspWriter out = null;
            Object page = this;
            String  _value = null;
            try {

                if (_jspx_inited == false) {
                    _jspx_init();
                    _jspx_inited = true;
                }
                _jspxFactory = JspFactory.getDefaultFactory();
                response.setContentType("text/html;charset=8859_1");
                   pageContext = _jspxFactory.getPageContext(this, request,
response,
                       "", true, 8192, true);

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

                                                        //    HTML    //    begin
[file="C:\\tomcat\\webapps\\test\\HelloWorld.jsp";from=(0,0);to=(7,7)]
                                out.write("<HTML>\r\n<HEAD>\r\n\t<TITLE>Hola
mundo</TITLE>\r\n</HEAD>\r\n<BODY>\r\n\tHola
Mundo!\r\n</BODY>\r\n</HTML>");
                // end

            } catch (Exception ex) {
                if (out.getBufferSize() != 0)
                    out.clearBuffer();
                pageContext.handlePageException(ex);
            } finally {
                out.flush();
                _jspxFactory.releasePageContext(pageContext);
            }
        }
    }
```

Siendo el código Java anterior el asociado al Servlet resultado del código JSP siguiente:

```
<%@ page info="Ejemplo de JSP" %>
<P> Hola Mundo </P>
```

Programando JavaServer Pages (JSP)

Como se ha visto anteriormente, el empleo de JSP's permite el independizar la "parte" estática de una página Web de la parte dinámica, de modo que en el momento de la codificación, la parte estática de la misma se escribirá exactamente igual a como se haría para una página "normal", empleando todos los mecanismos que nos ofrece el lenguaje de marcas que se esté empleando (ya sea HTML o XML). La parte dinámica de la página a codificar se encerrará empleando etiquetas especiales, que generalmente comenzará mediante '<%' y terminarán con '%>'. Por ejemplo, un fragmento de una página JSP podría ser como el que sigue:

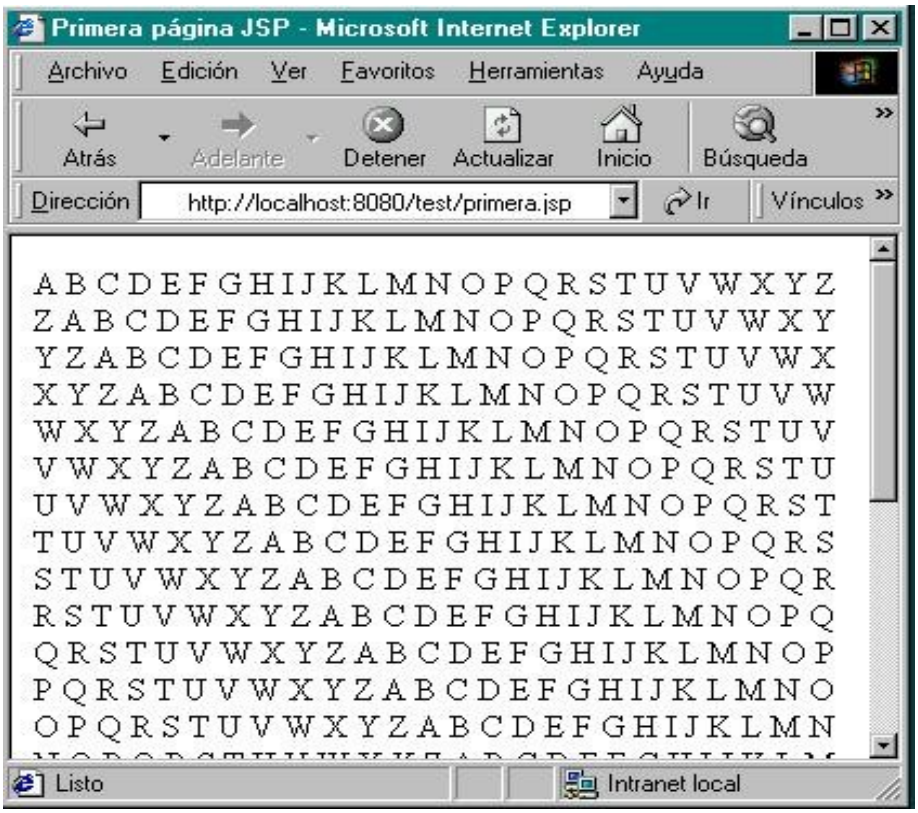
```
<I><%=Request.getParameter("titulo")%></I>
```

Generando el contenido del parámetro título en cursiva. Generalmente, a las páginas JSP se las suele dar la extensión .jsp, soliendo ir emplazadas en cualquier ubicación dentro del árbol de directorios del servidor en la que se ubicaría una página HTML. Además, a pesar de que lo escrito tiene más apariencia de página HTML que de Servlet, por debajo dicha página JSP se va a traducir en un Servlet, en el que el código asociado a la parte estática de la página se va a transformar en una impresión en el OutputStream asociado a la parte estática de la página se va a transformar en una impresión en el OutputStream asociado a la respuesta del método _service del Servlet (representada por el objeto HttpServletResponse). Nótese también que muchos servidores Web permiten la declaración de aliases de modo que una URL que aparentemente referencia a un fichero HTML realmente se refiera a una página JSP o a un Servlet.

A continuación se presenta una página JSP, a partir de la cual se irán identificando los distintos elementos que pueden integrar una página JSP cualquiera:

```
   <HTML>
   <HEAD>
   <TITLE>Primera página JSP</TITLE>
   </HEAD>
   <BODY>
   <!—Establecimiento de la información global de la página -->
         <%@ page language="java" %>
         <!—Declaración de variables -->
         <%! char c = 0; %>
         <!—El código java propiamente dicho - scriplets -->
         <%
         for (int i=0;i<26;i++){
               for(int j=0;j<26;j++){
               //Se mostrarán por pantalla las letras mayúsculas del
alfabeto
               //avanzando cada línea una letra a la derecha
               c = (char)(0x41 + (26 - i + j)%26);
         %>
         <!—Mostraremos el valor de c.toString en la página HTML -->
         <%= c%>
         <%
               }//cerramos el segundo bucle
         %>
         <BR><!—Separamos cada ristra de 26 caracteres en líneas
separadas -->
         <%
         }//cerramos el primer bucle
         %>
   <!—Y por fin terminamos el documento HTML -- >
   </BODY>
   </HTML>
```

Que daría como resultado una página HTML similar a la que se presenta a continuación:



Como puede apreciarse en el código asociado a la página anterior, una página JSP contiene una serie de partes bien diferenciadas dentro de la propia página, aparte de las propias asociadas al lenguaje de marcas empleado para su codificación final:

- Directivas: se emplean para proveer a la página JSP de la información global necesaria para el correcto funcionamiento de la parte dinámica de la misma (el código JSP que va a incluir). Entre ellas se incluyen por ejemplo las sentencias de importación, la página asociada a la gestión de errores, o para indicar si la página en cuestión forma parte de una determinada sesión, como se verá más adelante.
- Declaraciones: empleadas para la declaración de variables o métodos globales a la página en cuestión.
- Scriptlets: el código java propiamente dicho que va a ir embebido en la página.
- Expresiones: formateo de la expresión asociada como un string para su inclusión en la página Web resultado de la ejecución de la página.
- Acciones: aunque éste tipo de estructura JSP se verá al final del capítulo, debido a las similitudes que presentan con el lenguaje XML, simplemente comentar que se encargan de efectuar distintas funciones por debajo de la propia página. Por ejemplo permite buscar o crear una instancia de un determinado Bean, o definir nuevas etiquetas y encapsularlas en librerías de etiquetas (taglibs) para permitir su uso desde varias páginas JSP sin obligar a la inclusión de las mismas en el propio código de la página.

Directivas JSP

Una directiva JSP es una sentencia que permite dar el motor JSP información de la página a la que precede. Estas directivas siguen una sintaxis general de la forma:

```
<%@ directiva {atributo="valor"} %>
```

Donde la aparición entre llaves del par atributo = ”valor” indica la posibilidad de aparición de un número opcional de los mismos.

Las posibles directivas asociadas a partir de la versión 1.0 de la especificación de JSP son las siguientes:

- Page: para la inclusión de información general de la página que se esté codificando.
- Include: para la inclusión en la página de ficheros tal cual (por ejemplo, ficheros conteniendo código JSP, HTML, etc) en el momento en que se traduzca la página a su Servlet asociado.
- TagLib: la URI para una librería de etiquetas que se vayan a emplear dentro de la página JSP.

Directiva Page

La directiva Page como se comentó anteriormente, la estructura declarativa Page permite la aparición de numerosos pares atributo = ”valor”, cuya aparición es opcional, de modo que a aquellos atributos que deban tener un valor de forma obligada, les será asignado el valor “por omisión”.

Atributo y posible valor	Descripción
language=”java”	Indica al servidor el lenguaje se va a emplear en el fichero. De momento java es el único permitido para páginas JSP según la especificación. El soporte para otros lenguajes de script está disponible para JavaScript en http://www.cauco.com, y para polyjsp en http://www.plenix.org/polyjsp .
extends=”package.class”	Define la clase padre del servlet generado a partir de la página JSP. Normalmente es necesaria para emplear cualquier cosa que no se encuentre en las clases base.
import=”package.*, package.class”	Es similar a la primera sección de cualquier programa Java, por lo que debería ponerse siempre al comienzo de cualquier fichero JSP. Lista de paquetes a importar separados por comas.
session=”true|false”	El valor por omisión para esta variable es true, indicando que los datos de la sesión van a estar disponibles desde la página JSP en cuestión.
buffer=”none|8kb|sizekb”	Indica si el OutputStream asociado al servlet resultado de compilar la página en cuestión va a ser Buffered. Por omisión su valor va a ser de 8kb, debiéndose emplear junto con la propiedad autoFlush.
autoFlush=”true|false”	Estando a true, vuelca el contenido del buffer de salida en el momento en que se llene, sin generar una excepción.
isThreadSafe=”true|false”	Por omisión su valor se establece a true, indicando al motor JSP que se deben atender peticiones de la página de forma concurrente (varios clientes se pueden conectar a la página asociada simultáneamente). Nótese que es

	responsabilidad del autor de la página el sincronizar el estado compartido entre las páginas, de modo que ésta sea realmente "segura en threads". Si esta variable se pone a false, sólo se permitirá la existencia de un único thread en ejecución para la página, lo cual se traduce en un único cliente accediendo a la página en un mismo momento. Debido a que nada garantiza que peticiones consecutivas de un mismo cliente se vayan a asociar con una misma instancia de la página JSP pedida, se deben sincronizar aquellos métodos que pudieran generar situaciones conflictivas en lo que se refiere al estado compartido por todas las instancias de la página y, en definitiva, del servlet asociado.
info="text"	Información general asociada a la página accesible mediante el método Servlet.getServletInfo().
ErrorPage= "pathAPaginaDeError"	Path relativo a la página JSP que se va a encargar del manejo de las posibles excepciones no capturadas generadas en una página JSP, y que tendrá isErrorPage=true.
isErrorPage= "true|false"	Marca la página como página de error, como se verá más adelante.
contentType= "text/html; charset=ISO-8859-1"	El tipo MIME asociado a la página resultante de la ejecución de la página JSP que se esté codificando. Debe aparecer antes de que se trate de imprimir cualquier carácter no contenido en el juego de caracteres latin-1 tradicional.

Directiva Include

La directiva Include permite la inclusión de ficheros en la página JSP en el momento en que ésta es traducida a su Servlet correspondiente, siguiendo la sintaxis:

```
<%@ include file="url relativa" %>
```

La URL especificada en el atributo file, el único asociado a esta directiva, normalmente es interpretada como relativa a la página JSP que la referencia, aunque, al igual que ocurre con las URLs relativas en general, se le puede indicar al sistema que dicha URL se interprete como absoluta (relativa al direcotiro raíz del que "cuelga" el servidor Web) simplemente anteponiendo como primer carácter en ella el carácter '/'.

Los contenidos asociados al fichero incluido se filtrarán del mismo modo en que se haría si dicho fichero fuese "cortado" y "pegado" en el código de la página JSP origen, de modo que dicho fichero podría incluir HTMl estático, scriptlets, directivas y acciones; como podría contener cuqlquier página JSP.

Un ejemplo muy usual para el empleo de este tipo de directiva lo constituyen las "barras de navegación" que se pueden encontrar en numerosos sitios Web. Debido a los problemas que pueden llegar a dar los Frames, generalmente dichas barras suelen implementarse mediante una pequeña tabla a lo largo de la parte superior de la página o del lado izquierdo de la misma, apareciendo el código asociado a la misma en todas las páginas del sitio Web.

Así, la forma normal de llevar a cabo esta labor consiste en el empleo de la directiva include, librando al responsable de dichas páginas de mantener actualizadas todas ellas copiando por separado el código HTML asociado en cada fichero.

La solución que se propone es el empleo de la acción JSP jsp:include, la cual inserta el fichero en la página que lo referencia en el momento en que ésta es solicitada por el cliente como se verá más adelante en que se discutirán las denominadas acciones JSP.

Declaraciones JSP

Una declaración JSP , como su propio nombre indica, permite la declaración de variables o métodos que se van a emplear a lo largo del resto de la página JSP, siguiendo la sintaxis que se mencionó anteriormente. Un posible ejemplo de declaraciones de métodos y variables podría ser el siguiente:

```
<%!
      String var = "x";
      int contador = 0;
      private void incrementarContador(){
            contador++;
      }
%>
```

Nótese que después de cada declaración debe aparecer el ';' al igual que se haría durante la codificación de una clase Java cualquiera, incluyéndose el código insertado dentro del cuerpo de la clase asociada al sevlet de la página, fuera de lo que sería el método _jspService (HttpServletRequest req, HttpServletResponse res) comentado anteriormente.

También cabe comentar que éste sería el modo en que podríamos sobrescribir los métodos jspInit(), y jsjpDestroy() que se mencionaron en la primera parte.

Scriptlets JSP

Un Scriptlet se define como cualquier bloque de código Java válido que se encuentre entre los delimitadores <% y %>.

Este código se va a emplazar, según su orden de aparición dentro de la página JSP, en el cuerpo asociado al método _jspService, de modo que el código escrito entre esos delimitadores podrá acceder a cualquier variable, método o bean que hayan sido declaradas dentro de la página. Nótese que todo el código HTMl (o XML) que se incluya en la página JSP, entre los distintos Scriptlets que se codifiquen para la misma, se va a traducir en su correspondiente print dentro de lo que sería el código de _jspService, en el mismo lugar en que aparezcan dentro de la página JSP.

A continuación se listan los posibles objetos accesibles desde el propio entorno asociado al método anterior, esto es, desde el servlet que se genera a partir de la página que se está escribiendo:

Objetos accesibles desde un Scriptlet	Descripción
request	Se refiere a la petición generada por el cliente, siendo generalmente una subclase de HttpServletRequest, la cual contiene la lista de parámetros con que se invocó a la página si existiese.

response	El objeto que identifica el contenedor en que se va a devolver la respuesta a la ejecución de la página. Se podría decir que es el objeto que proporciona el OutputStream que se empleará para la escritura de la página de respuesta a la ejecución de la página JSP.
pageContext	Los atributos y objetos implícitos necesarios para construir el entorno en que se ejecuta una página JSP deberían ser accesibles a través de una API uniforme para permitir al motor JSP compilar dichas páginas. El problema es que cada servidor con soporte JSP emplea una implementación específica de dichos atributos y objetos. La solución empleada para dicho problema es que el motor JSP emplea una fábrica (Factory) que empleará la clase generada para obtener la implementación específica de PageContext en el servidor en que se esté ejecutando. Dicho contexto de la página es inicializado con los objetos asociados a la petición (Request) y a la (response), así como algunos de los atributos declarados mediante la directiva Page, en particular errorPage, session, buffer y autoFlush, además efectuada por un determinado cliente, como se verá más adelante. En resumen, la clase PageContext fue introducida en la especificación JSP para encapsular el uso de características específicas del servidor, como podrían ser JspWriters de alto rendimiento. De modo que accediendo a estas características a través de PageContext, es código generado seguirá funcionando en motores de servlets/JSPs “normales”.
session	El objeto HttpSession asociado con la petición.
application	Es el contexto del servlet devuelto por una llamada a getServletConfig().getServletContext() efectuada en un Servlet “normal”.
out	El objeto asociado al OutputStream que se empleará para la generación de la página Web resultado de la ejecución del JSP. Realmente es uns subclase de jspWriter, una versión con buffer de un PrintWriter.
config	El objeto ServletConfig asociado a la página.
page	El modo en que la página puede referenciarse a sí misma (sería la latenativa a this en una clase Java normal).
exception	La subclase de Throwable no capturada que se pasa a la URL/URI asociada a la página de error (errorPage).

En el fragmento de código siguiente se muestra cómo obtener un posible parámetro de la petición asociada a una invocación a la página JSP, y el modo en que se podría escribir una línea de texto en el stream asociado al resultado de su ejecución.

```
<%
        String var = Request.getParameter(“lname”);
        Out.println(var);
%>
```

Nótese que el código asociado a un Scriptlet es insertado exactamente como se escribió, y cualquier código HTML (estático) antes o después de un Scriptlet es

traducido a sentencias print. Esto significa que un Scriptlet no tiene por qué contener sentencias Java completas, de manera que los posibles bloques sin cerrar podrían afectar, y de hecho lo hacen, a la parte estática de la página (por ejemplo, el hecho de que un comentario HTML se deje inscrito dentro de dos Scriptlets que constituyan un bucle resultará en la aparición de dicho comentario tantas veces como iteraciones tengan lugar en el bucle en la página HTML resultante). Así, el fragmento JSP siguiente, conteniendo una mezcla de código HTMl y Scriptlets:

```
<%
      if(Math.random()<0.5{ %>
      ¡Que tengas un <B>buen</B> día!
      <% } else { %>
      ¡Que tengas un <B>patético</B> día!
      <% }
%>
```

Se transformaría en lo siguiente:

```
if (Math.random()<0.5){
      out.println("¡Que tengas un <B>buen</B> dia!");
} else {
out.println("¡Que tengas un día <B>patético</B> dia!");
}
```

Nótese que si se desea que en la página resultado aparezca una cadena del estilo a '<%', deberá interponerse un blackslash (\), escribiendo así '<\%' en su lugar.

Expresiones JSP

Las expresiones JSP son una herramienta muy útil para embeber valores Java en el código asociado a la página Web generada como resultado de la ejecución de una página JSP. Cualquier cosa encerrada entre <%= y %> será evaluada en el momento de la petición de la página, convertido a String, y después enviado al Stream de salida, siguiendo, por tanto, el siguiente formato:

```
<%= expresión Java %>
```

El hecho de que dicha evaluación se efectúe en tiempo de ejecución permite, por ejemplo, acceder a los datos asociados a la petición (request), a la fecha que se tiene en el servidor, y en general a cualquier información a la que podríamos acceder dentor de un Scriptlet, como se muestra a acontinuación:

```
Fecha y hora actuales: <%= new java.util.Date() %>
Nombre de la computadora cliente
<%= request.getRemoteHost() %>
```

Nótese que las expresiones no deben terminar con un ';', debido esto a que dichas expresiones van a ser embebidas dentro de una llamada a out.println(), por lo que la inclusión de dicho carácter se traduciría en un error en el servidor.

Las expresiones JSP permiten "parametrizar" el código resultante (de igual forma que se haría en una sentencia SQL en la que sólo difieren un número muy limitado de valores). De ese modo, se prodría hacer que un gran número de condiciones y sentencias sean ejecutadas, por ejemplo en un bucle, mostrando una salida en función de los valores obtenidos a raíz de dichas comparaciones y sentencias, simplemente insertando las expresiones a mostrar entre dos o más Scriptlets que constituyen un bucle.

```
<% for (int i=0;i<10;i++){ %>
      <BR> El valor del contador es <%= i %>
<% } %>
```

Etiquetas JSP Avanzadas

Hasta ahora se han visto las estructuras básicas asociadas a la especificación JSP con las que se podrían empezar a desarrollar páginas JSP sencillas. Como podrá imaginarse el lector, lo visto hasta ahora dejaría un poco "en duda" la utilidad del empleo de ésta tecnología pero, obviamente, JSP no acaba aquí. En lo que sigue se expondrá nuevas estructuras JSP que nos van a permitir, desde la comunicación de una página JSP con Servlets, hasta el empleo de JavaBeans independientes de nuestra página, permitiendo, entre otras cosas, el establecimiento de conexiones con Sistemas Gestores de Bases de Datos aprovechando el API JDBC, separando aún más la "lógica de negocio" de la aplicación, de la interfaz de usuario propiamente dicha.

A continuación, se presentan las características JSP, que han sido denominadas "avanzadas", que van a ser ilustradas a lo largo del resto del libro:

- Reutilización de componentes y de librerías de etiquetas. La tecnología JSP enfatiza en el empleo de componentes reutilizables como podrían ser JavaBeans, Enterprise JavaBeans y librerías de etiquetas (taglibs). Siendo estos componentes empleados dentro de herramientas de desarrollo de componentes y de diseñoo de páginas, permitiendo un ahorro importante de tiemo si se aprovechan las características inherentes a Java como son la independencia de la plataforma, su flexibilidad, así como el empleo de otros lenguajes de script.
- Soporte para acciones. Como se verá, el soporte de Acciones, permite la encapsulación de distintas funcionalidades en componentes manipulables por distintas herramientas, lo que, junto con los Scriptlets vistos anteriormente, ofrecen un mecanismo para incluir dicha funcionalidad dentro de una página JSP.
- Capa de acceso vía Web para aplicaciones siguiendo arquitecturas de N-niveles (N-tier). La tecnología JSP forma parte del paquete J2EE (Java 2 Plataform Enterprise Edition), de modo que se pueden desarrollar aplicaciones servidoras intermedias, empleando un sitio Web que haga uso de páginas JSP como interfaz de acceso a componentes EJB (Enterprise JavaBeans) en un entorno conforme a J2EE.

Acciones JSP

Como se ha visto anteriormente, en una página JSP pueden aparecer dos tipos de elementos: directivas y acciones. Las directivas se empleaban para dar información que es válida independientemente del tipo de petición recibida por la página JSP, como por ejemplo el lenguaje a utilizar en los Scriptlets asociados a la página. En lo que se refiere a las acciones JSP, su comportamiento podría depender, y generalmente ocurre así, de los detalles de una petición específica recibida por la página JSP.

En otras palabras, las directivas JSP se podrían ver como proveedores de información que va a influir en el proceso de traducción(compilación de la página en cuestión, mientras que las acciones representan información para el proceso de tratamiento de las subsecuentes peticiones, pudiendo crear objetos y ponerlos a disposición de los posibles Scriptlets de la página mediante determinadas variables específicas para éstos.

Las acciones JSP siguen la sintaxis de los distintos elementos XML, componiéndose de una etiqueta de comienzo, un cuerpo, y una etiqueta de finalización, de la forma siguiente:

```
<miEtiqueta atrib="valor" ... >
cuerpo
</miEtiqueta>
```

O para una etiqueta sin cuerpo...

```
<miEtiqueta atrib1="valor".../>
```

Así, un elemento de este estilo tiene asociado un tipo describiendo el nombre asociado a la etiqueta, un conjunto de atributos válidos, y la semántica asociada a cada uno de estos atributos.

Existen un número de etiquetas predefinidas estándar que debe implementar todo motor JSP, tal y como se indica en su especificación, que son:

- jsp:include: etiqueta que permite la inclusión de un fichero en el momento en que se recibe una petición a la página JSP en que se encuentra.
- jsp:useBean: permite la búsqueda o instanciación del valor para una propiedad de un determinado componente Java (JavaBean).
- jsp:setProperty: para el establecimiento del valor para una propiedad de un determinado componente Java (JavaBean).
- jsp:getProperty: Para mostrar el valor de un determinado componente java en la página resultante.
- jsp:forward: Para enviar al cliente origen de la petición a una nueva página Web.
- jsp:plugin: Para generar el código asociado a cada navegador para el empleo del Java plugin mediante las etiquetas OBJECT o EMBED.

Generalmente, todos los atributos de las acciones, ya sean estándar o a medida, o bien permanecen sin ser interceptados por el contenedor, o bien tienen una semántica bien específica según el tipo de acción para el mismo. A pesar de ello, existen dos excepciones a esta regla general:

- Algunos valores para los atributos son interpretados en el momento de la petición y son procesados por el contenedor.
- Los atributos id y scope tienen una interpretación especial.

Las acciones JSP pueden definir atributos con nombre y sus valores asociados, los cuales, típicamente, serán tratados por la página JSP como fijos e inmutables. Sin embargo, la especificación JSP 1.1 provee un mecanismo para describir un atributo que se ha de tratar en el momento de la petición.

Un valor de atributo de la forma "<%= Scriptlet %>" o '<%= Scriptlet %>' representa un valor de atributo de tiempo de petición, cuyo uso solo está permitido en acciones, y cuyo valor se refiere a aquél asociado al Scriptlet de la expresión. De este modo, si existe más de un atributo de este tipo en la etiqueta asociada, sus expresiones asociadas serán de izquierda a derecha según su orden de aparición en la página JSP.

Nótese que solo pueden asignarse así valores de atributos (por ejemplo, el nombre de un atributo siempre consiste en un nombre explícito, y no el resultado de evaluar una expresión), debiendo aparecer la expresión ella sola, sin permitirse mezclas de expresiones, expresiones y constantes, etc., de modo que en el caso en que esto sea necesario, deberá efectuarse dentro de la propia expresión.

En lo que se refiere al tipo asociado a cada atributo de una acción JSP, éste puede estar definido en función de la propia acción, tratándose dicha semántica en el momento en que se traduce la página JSP. De este modo, si se trata de asignar un valor (resultante de un scriptlet) a un determinado atributo con una semántica específica en tiempo de traducción (cuando se genera el servlet asociado a la página JSP), esto se traducirá en un error de tiempo de traducción.

En definitiva, el que un atributo permita o no que su valor se establezca en el momento de recepción de una petición va a depender del tipo de acción JSP en que vaya a ir inscrito, de modo que antes de tratar de hacer una asignación de este tipo, se deberá comprobar que la acción asociada lo permite.

A continuación, se muestra una lista de las acciones estándar y atributos asociados que permiten el establecimiento de su valor en el momento en que se recibe la petición, como se verá en más detalle:

- Para la acción jsp:setProperty, los atributos value y beanName.
- Para jsp:include, el atributo page.
- Para jsp:forward, el atributo page.
- Para jsp:param, el atributo value.

La tupla atributo/valor id=”nombre” es un elemento con unsignificado especial para el contenedor JSP, tanto en tiempo de traducción de la página, como en el momento de recepción de una petición. En particular:

- El nombre debe ser único durante el momento de traducción, identificando el elemento particular en el que aparece declarado, de modo que si sucede que aparece alguno de éstos duplicado durante el período de traducción de la página, esto se traducirá en un error fatal.
- Además, si el tipo de acción crea una o más instancias de objetos en el momento en que se procesa la petición recibida de un cliente, uno de éstos objetos será asociado por el contenedor JSP con dicho nombre, de modo que este objeto podrá ser accedido mediante dicho nombre en varios contextos a través de pageContext. Empleándose además dicho nombre para referenciar a una variable en el entorno de los Scriptlets de la página JSP asociada.
- Nótese que el ámbito de dicha variable depende de las reglas y capacidades asociadas al lenguaje de script empleado en la página, lo cual implica que dicho nombre deberá cumplir las reglas cintácticas para la nomenclatura de variables de dicho lenguaje.

Más adelante se darán detalles para el caso en que el lenguaje empleado en la página sea “java”. Por ejemplo, la acción <jsp:useBean id=”name” class=”className” .../> ilustrada más adelante, emplea este mecanismo para instanciar, y a continuación poner el nombrado JavaBean disponible para la página en el momento de una petición.

El atributo scope, cuyos posibles valores son: page, request, session y application, modifica el comportamiento del atributo id descrito anteriormente (nótese que este atributo tiene definida su semántica tanto para tiempo de traducción como para el tiempo de tratamiento de la petición). En particular, este atributo describe el nombre del “espacio”, esto es, el ciclo de vida implícito al objeto asociado al nombre indicado mediante el atributo id, y las APIs a utilizar para acceder a dicha asociación, según la siguiente tabla:

Valores ‘scope’	**Descripción**

page	El objeto nombrado será accesible desde javax.servlet.jsp.PageContext para la página actual. Esta referencia debería ser descartada en el momento de la finalización del tratamiento de la petición actual por el cuerpo de la página. Es ilegal el cambiar el objeto asociado, a menos que su tipo en tiempo de ejecución sea un subconjunto del tipo del objeto previamente asociado.
request	El objeto nombrado estará accesible desde la petición (ServletRequest) asociada a la página actual, a través del método getAttribute (name) de dicha petición. Esta referencia al objeto debería descartarse en el momento en que se finalize el tratamiento de la petición actual, siendo ilegal el cambiar el valor de una objeto asociado de este modo, de igual modo que ocurría para el valor anterior.
session	El objeto nombrado estará accesible a través del objeto HttpSession asociado a la página actual, mediante el método getValue(name) de dicho objeto. Dicho referencia debería descartarse en el momento en que la sesión asociada queda invalidada, siendo ilegal el cambio de dicha referencia del mismo modo que ocurría para los dos casos anteriores. Nótese que se produciría un error fatla en tiempo de traducción al tratar de amplear un ámbito (scope) en una página en la que se haya indicado mediante la directiva <%@ page ... %> que dicha página no va a participar en ninguna sesión.
application	El objeto nombrado estará accesible desde la página actual a través de su ServletContext mediante su método getAttribute(name). Esta referencia debería descartarse en el moemto en que lo pida el propio ServletContext. El cambio de la referencia está sujeto a las mismas reglas que la expuestas en el apartado asociado al valor page.

Acciones estándar JSP

La especificación JSP 1.1 define una serie de acciones que siempre estarán disponibles, independientemente de la versión del contenedor JSP o servidor Web que se esté empleando. Por supuesto, todas estas acciones JSP se sumarán a cualquier acción personalizada asociada a la implementación específica del contenedor JSP.

<jsp:useBean>

Una acción del tipo jsp:useBean asocia una instancia de un objeto Java definido con un ámbito dado a un identificador mediante una variable disponible vía Scriptlets con el mismo nombre.

La semántica básica de esta acción trata de encontrar un objeto existente a partir del identificador y ámbito dados y, en el caso de no encontrarlo, tratará de crear el objeto a partir del resto de atributos que acepta, como se verá más adelante, lo que hace que este tipo de acción goze de una gran flexibilidad. También puede emplearse esta acción para darle un nombre localmente a un objeto definido en cualquier otro lugar, ya sea en otra página JSP o en un Servlet (mediante el atributo type, y sin incluir los atributos class y beanName).

En lo que se refiere al empleo de este tipo de acción, ésta deberá aparecer con al menos el atributo type o el atributo class presente en su declaración, no estando permitida la aparición de class y beanName simultáneamente.

El atributo beanName se refiere al nombre de un Bean, tal y como se indica en la especificación asociada a JavaBeans para el argumento asociado al método instantiate() en java.beans.Beans. Por ejemplo, este nombre podría ser de la forma a.b.c para una clase, o a/b/c.ser para la carga de un objeto serializado, pudiendo ser asignado en tiempo de petición. Nótese que si el parámetro no se corresponde con ninguna de las anteriores posibilidades, se generará una excepción en tiempo de petición, tal y como indica la semántica del método comentado anteriormente.

Las etapas asociadas a la inlusión de una etiqueta de este tipo en una página JSP son las siguientes:

1. Se trata de localizar un objeto en base a los atributos asociados al identificador y al ámbito (id y scope), efectuándose dicha inspección de forma sincronizada en el espacio de nombres asociado a cada ámbito, para evitar comportamientos no deterministas.
2. Se define una variable del lenguaje de script asociado a la página JSP (java) con el identificador dado mediante el atributo id, en el ámbito léxico actual de dicho lenguaje de script, con el tipo (type) o clase(class) especificado/s.
3. Si se ha encontrado el objeto de la búsqueda, el valor de la variable es inicializado mediante una referencia a dicho objeto, empleándose un cast al tipo especificado. Nótese, que en el caso en que dicho casting falle, se lanzaría una excepción del tipo java.lang.ClassCastException, completándose el procesamiento de la etiqueta jsp:useBean.
4. En el caso de que el objeto no se encuentre durante el proceso asociado al paso 1, son haberse dado un nombre de clase (atributo class) o de Bean (atributo beanName), se lanzará una excepción del tipo java.lang.InstantiationException, completándose el procesamiento de la etiqueta.
5. Si el objeto no se encontró en el paso 1 pero sí fue incluido el atributo class refiriéndose a una clase no abstracta con un constructor sin argumentos, entonces, se instanciará dicha clase, de modo que la referencia al objeto se asociará a la variable de script determinada por id en el ámbito apropiado, pasándose entonces al paso 7. En el caso de que no se cumpla la condición anterior, se lanzará una excepción del tipo java.lang.InstantiationException, completándose con ello el tratamiento de la cción useBean.
6. Finalmente, en el caso de que no se encuentre el objeto del paso uno y se haya incluido el atributo beanName en la declaración, se invocará el método instantiate() de java.beans.Beans con el classLoader asociado al objeto Servlet y el valor de beanName como argumento. Si el método no falla, la referencia al nuevo objeto será asociada a la variable con el nombre especificado en la etiqueta dentro del ámbito especificado, pasándose después a la fase 7.
7. Si aparece el cuerpo para la etiqueta jsp:useBean, se pasará a procesar el cuerpo de ésta. En primer lugar, la inicialización de la variable se efectuará solo en el ámbito asociado al cuerpo de la etiqueta, tratándose el texto contenido en dicho cuerpo como en cualquier otro sitio. Un uso común del empleo de un cuerpo no vacío sería para completar dicha

inicialización, conteniendo en ese caso acciones del tipo jsp:setProperty, y Scriptlets. Completándose finalmente la ejecución de la acción useBean.

La acción jsp:useBean podría no tener "cuerpo", en cuyo caso la sintaxis asociada será de la forma:

```
<jsp:useBean id="nombre" scope="page|request|session|application" typeSpec />
    typeSpec::=   class="nombreDeClase" |
                  class="nombreDeClase" type="NombreDeTipo" |
                  type="NombreDeTipo" class="nombreDeClase" |
                  beanName="NombreDelBean" type="NombreDeTipo" |
                  type="NombreDeTipo" beanName="NombreDelBean" |
                  type="NombreDeTipo"
```

En el caso de que la acción sí que conste de un cuerpo, éste aparecerá del siguiente modo:

```
<jsp:useBean id="nombre" scope="page|request|session|application" typeSpec />
    body
</jsp:useBean>
```

En este caso, el cuerpo se ejecutará en el caso de que el Bean asociado a la acción sea creado. Normalmente, el cuerpo asociado a una acción jsp:useBean contendrá, o bien Scriptlets, o bien etiquetas del tipo jsp:setProperty, que se emplearán para modificar el objeto que se acaba de crear, aunque el contenido de dicho cuerpo no tienen restricciones especiales.

A continuación, se listan los posibles atributos que son factibles de aparecer en una etiqueta del tipo jsp:useBean:

Atributo	Descripción
id	Indica el nombre a emplear para identificar la instancia del objeto en el espacio de nombres asociado a su ámbito, así como el nombre de la variable declarada e inicializada con la referencia a dicho objeto.
scope	Representa el ámbito en el que dicha referencia va a ser accesible. Su valor por defecto es page. Para más información véase el apartado asociado a scope.
class	El "nombre completo" de la clase que define la implementación del objeto. Si los atributos class y beanName no fuesen especificados, el objeto asociado a id debe estar presente en el ámbito dado.
beanName	El nombre de un Bean, como se esperaría en una llamada a instantiate() en la clase java.beans.Beans. El valor de éste atributo puede ser establecido mediante una expresión a evaluar en el tiempo de petición.
type	Si se especifica, define el tipo de la variable que va a recoger la referencia al objeto asociado. Dicho tipo podría no ser el mismo que el especificado por la clase asociada a la implementación del objeto, pero debe estar relacionado con ésta, ya siendo una superclase de aquella, o un interfaz que implemente. En el caso en que no se cumplan las restricciones anteriores, una excepción del tipo ClassCastException en tiempo de petición, al tratarse de asignar la

	referencia al objeto a la variable en cuestión. Si no se especifica, se asumirá que su valor es el mismo que el del atributo class.

Es importante considerar que la clase especificada para el atributo class debe estar ubicada en el directorio normal de clases del servidor, no en la parte reservada a las clases que se recargan automáticamente si éstas cambiasen (por ejemplo, las clases asociadas a Servlets o páginas JSP). Así, para el JavaWebServer, éstas deberían estar en el directorio classes o en un fchero jar dentro del directorio lib, y no en el directorio asociado a los Servlets.

Un ejemplo del uso de la acción jsp:useBean es el segundo reflejado en la sección de Ejemplos de uso.

<jsp:setProperty>

La acción jsp:setProperty establece el valor de las posibles propiedades asociadas a un Bean. El atributo name indicará un objeto que deberá estar definido previamente a la aparición de esta propiedad.

Existen dos variantes de este tipo de acción, estableciendo ambas el valor de una o más propiedades del Bean basándose en los tipos asociados a dichas propiedades. Efectuándose la introspección usual para todo Bean para descubrir qué propiedades se encuentran presentes, y, para cada una de ellas, su nombre, si son o no indexadas, su tipo, y sus métodos de obtención y establecimiento.

Las propiedades de un Bean pueden establecerse a partir de uno o más parámetros del objeto Request, de una constante de tipo String, o de una expresión evaluada en tiempo de petición. Las propiedades simples e indexadas pueden establecerse mediante setProperty, permitiéndose su establecimiento sólo para los tipos que aparecen en la tabla siguiente, aplicándose la conversión que aparece en la misma. Las expresiones en tiempo de petición pueden asignarse a cualquier tipo de propiedad, sin realizarse conversiones de ningún tipo.

Nótese que la asignación de valores a propiedades indexadas requieren un array como valor a asignar, aplicándose las reglas expuestas en el párrafo anterior para cada uno de los elementos del mismo. Además, si ocurriese algún tipo de error en las posibles conversiones, éste se produciría en el momento de la traducción de la página a su Servlet correspondiente.

Tipo para la propiedad	Conversión a su valor de tipo String
boolean o Boolean	Según java.lang.Boolean.valueOf(String)
byte o Byte	Según java.lang.Byte.valueOf(String)
char o Character	Según java.lang.Character.valueOf(String)
double o Double	Según java.lang.Double.valueOf(String)
int o Integer	Según java.lang.Integer.valueOf(String)
float o Float	Según java.lang.Float.valueOf(String)
long o Long	Según java.lang.Long.valueOf(String)

```
<jsp:setProperty name="NombreDelBean" prop_expr />
prop_expr ::= property="*" |
        property="NombreDePropiedad"|
        property="NombreDePropiedad" param="parameterName"|
        property="NombreDePropiedad" value="ValorDePropiedad"
ValorDePropiedad::= string
```

Nótese que el valor que se ha de asociar a la propiedad puede ser también un atributo de tiempo de petición, como se vió anteriormente.

NombreDePropiedad ::= expr_scriptlet

La acción jsp:setProperty admite los siguientes atributos:

Atributo	Descripción
name	Se refiere al nombre de una instancia de un Bean definido, por ejemplo, mediante una acción jsp:useBean. Dicho Bean debe contener la propiedad que se desea establecer, obligándose también a que aparezca el elemento jsp:useBean antes de jsp:setProperty en el mismo fichero.
property	El nombre de la propiedad del Bean que se desee establecer. En el caso de que se asigne el valor *, esta etiqueta irá recorriendo los distintos parámetros contenidos en ServletRequest, de modo que aquellos cuyo nombre y tipo se asemejen a alguna propiedad del Bean indicado mediante el atributo name serán asociados a la propiedad correspondiente.
param	El nombre del parámetro de la petición (Request) que se quiere asociar a la propiedad. Generalmente éste parámetro suele venir de un formulario. Si se omite el atributo param, se buscará dentro de la petición un atributo con el mismo nombre que la propiedad del Bean a asignar. Si éste no existe en la petición, o si éste tiene el valor de “”, la cción jsp:setProperty no hará nada.
value	El valor que se ha de asignar a la propiedad, pudiando asignarse mediante una expresión a evaluar en el momento de la petición.

En la sección Ejemplo de uso el ejemplo tercero muestra el uso de jsp:setProperty.

<jsp:getProperty>

Una acción del tipo jsp:getProperty sitúa el valor de una propiedad de una instancia de Bean, convertido a String, en el objeto asociado a la salida de la página JSP (out). Dicha instancia de Bean deberá haber sido definida tal y como se indica en el atributo name en un punto anterior de la página (generalmente mediante una cción del tipo jsp:useBean).

Como es lógico, la conversión a String se efectúa del mismo modo que para los métodos println(), empleándose el método toString() del onjeto para instancias de objetos, y convirtiéndose directamente los tipos básicos (int, flota, etc.). Si el objeto no fuese encontrado, se lanzaría una excepción en tiempo de petición.

La sintaxis asociada a jsp:getProperty, debido a su simplicidad, se resume en lo que sigue:

<jsp:getProperty name=”name” property=”propertyName” />

Los atributos asociados simplemente nos servirán para indicar el nombre de la instancia de objeto a emplear, y el nombre de la propiedad cuyo valor se quiere obtener.

Atributo	Descripción
name	Se refiere al nombre de una instancia de un Bean definido, por

	ejemplo, mediante una acción jsp:useBean.
property	El nombre de la propiedad del Bean cuyo valor se desea obtener.

No hay mucho más que decir:
<jsp:getProperty name="user" property="name" />

<jsp:include>

Un elemento del tipo <jsp:include …> permite la inclusión de recxursos estáticos y dinámicos en el mismo contexto que la página actual, especificándose dicho recurso mediante URLs al igual que ocurría para la directiva análoga a ésta que se vió con anterioridad.

Una página "incluida" sólo tiene acceso al objeto JspWriter, no pudiendo establecer cabeceras, lo cual imposibilita, por ejemplo, el invocar a métodos como setCookie() que, en el caso de ser invocados desde una página de este tipo, generarán una excepción en tiempo de petición. Nótese que esta restricción es equivalente a aquella impuesta por el método include() de la clase RequestDispatcher.

Una acción de este tipo podría tener subelementos del tipo jsp:param para proveer de valor a determinados atributos de la petición para ser empleados en el momento de la inclusión, continuándose el procesamiento de la petición una vez se termina la inclusión de la página "auxiliar". Además, si la salida de la página emplea buffering, dicho búfer será "volcado" a la salida previamente a la inclusión de la página.

Para el caso en que no se emplee acciones del tipo param, la sintaxis de una acción del tipo jsp:include sería la siguiente.

<jsp:include page="urlSpec" flush="true"/>

Nótese que el atributo flush es un atributo de aparición abligada, cuyo valor siempre ha de ser true para la especificación de JSP versión 1.1, un valor false no es válido.

Si van a aparecer elementos del tipo param. La sintaxis va a ser como sigue:

<jsp:include page="urlSpec" flush="true">
{ <jsp:param …/> }*
</jsp:include>

Por ejemplo:

<jsp:include page="/template/copyright.html" />

Atributo	Descripción
page	La URL asociada a la página a incluir. Importante el destacar que la asignación de este valor permite el empleo de expresiones a calcular en el momento de la petición.
flush	Atributo lógico obligado. Si se establce a true el buffer se volcará a la salida. Un valor false no es válido.

<jsp:forward>

Un elemento del tipo <jsp:forward page= "urlSpec" /> permite la redirección de la petición actual a un recurso estático, a otra página JSP o a un Servlet Java en el mismo contexto que la página actual, terminando con ello la ejecución de ésta. Nótese que el valor de la petición (Request) se ajustará en base a la nueva URL. Este tipo de acción permite la inclusión de subelementos del tipo param con objeto

de asignar valores a determinados atributos de la petición, para ser empleados después en la "redirección".

Si la salida de la página emplea buffering, entonces éste será "limpiado" previamente a la redirección. En caso contrario, si se ha efectuado alguna escritura, un intento de "redirección" desembocará en la generación de una excepción del tipo IllegalStateException.

Para el caso básico se tendría:

```
<jsp:forward page="urlSpec"/>
```

Y empleando los elementos param:

```
<jsp:forward page="urlSpec">
{ <jsp:param …/> }*
</jsp:forward>
```

En este caso, la etiqueta permite al autor de la página el que la petición en proceso se vea afectada por los atributos especificados.

Atributo	Descripción
page	Una URL similar a la vista para la directiva include. Al igual que para la acción anterior, también se permite el asignar su valor en tiempo de petición, mediante una expresión cuyo resultado sea un String con una URL.

<jsp:param>

<jsp:param> se emplea para dar información en forma de clave-valor útil para otras acciones como son jsp:include, jsp:forward y jsp:plugin.

Para el caso en que esta acción se emplee bajo una del tipo jsp:include o jsp:forward, la página incluída o el destino de la redirección verán el objeto asociado a la petición original, con los parámetros originales, y aquéllos añadidos mediante ésta acción, de modo que si se añade algún atributo existente en la petición, su valor anterior será concatenado al nuevo. Nótese que el ámbito asociado a estos nuevos atributos será el de la página que se incluye o la que se redirige la petición, no menteniéndose una vez ejecutada la acción que los incluya.

Este comportamiento es el mismo al que se puede apreciar para los métodos include() y forward() en el objeto de tipo ServletRequest. Por ejemplo, si la petición tiene un parámetro "nombre=Juan" y un "forwarding" se establece el parámetro "nombre " con valor "Pedro", la petición redirigida tendrá el valor "nombre=Pedro,Juan". El valor nuevo tiene precedencia al anterior.

La sintaxis será:

```
<jps:param name="nombre" value="valor" />
```

Esta acción tiene dos atributps cuya aparición es obligada, name y value. Name indica el nombre del parámetro a establecer, y value, el valor que se ha de asociar a dicho atributo, el cuál podría ser una expresión a evaluar en tiempo de petición.

<jsp:plugin>

La acción plugin permite al autor de la página JSP el generar código HTML que contenga los constructores (OBJECT o EMBED, dependiendo del navegador) que provocará la descarga del software asociado al Java Plugin (si se requiere), y la posterior ejecución del Applet o JavaBean especificado con ésta.

Esta etiqueta es reemplazada por una etiqueta <object> o <embed>, dependiendo del navegador empleado por el cliente origen de la petición, y emitida

a través del stream de salida asociado a la respuesta (response). Los atributos asociables a este tipo de acción establecerán los valores de configuración para la presentación del elemento en cuestión, tal y como se indicará a continuación.

Mediante elementos del tipo <jsp:param> se indicará los parámetros para el Applet o componente JavaBean.

Mediante elementos del tipo <jsp:fallback> se indicará el contenido a emplear por el navegador en el caso en que el plugin no pueda iniciarse. En el caso en que pueda iniciarse dicho plugin, y el Applet o JavaBean no pueda encontrarse o iniciarse, se mostrará al usuario un mensaje específico del propio plugin, como podría ser una ventana emergente mostrando la información asociada a una excepción del tipo ClassNotFoundException.

```
<jsp:plugin type="bean|applet"
        code="objectCode"
        codebase="objectCodeBase"
        { align="alignment" }
        { archive="archiveList" }
        { height="height" }
        { hspace="hspace" }
        { jreversion=" jreversion " }
        { name="componentName" }
        { vspace=" vspace" }
        { width=" width" }
        { nspluginurl="url" }
        { ie pluginurl="url" }
        { <jsp:params>
                { <jsp:param name="paramName" value="paramValue" /> }+
        </jsp:params> }
        { <jsp:fallback> arbitrary_text </jsp:fallback> }
        </jsp:plugin>
```

Atributo	Descripción
type	Identifica el tipo de componente, ya sea un Bean o un Applet.
code	Como se define en la especificación de HTML.
codebase	Como se define en la especificación de HTML.
align	Como se define en la especificación de HTML.
archive	Como se define en la especificación de HTML.
height	Como se define en la especificación de HTML.
hspace	Como se define en la especificación de HTML.
jreversion	Identifica la versión del JRE que necesita el compoenente para funcionar, el valor por omisión es: "1.1"
name	Como se define en la especificación de HTML.
vspace	Como se define en la especificación de HTML.
title	Como se define en la especificación de HTML.
width	Como se define en la especificación de HTML.
nspluginurl	La URL donde puede accederse al JRE Plugin para el Netscape Navigator. Su valor por defecto depende de la implementación.
iepluginurl	La URL donde puede accederse al JRE Plugin para Microsoft Internet Explorer. Su valor por defecto depende de la implementación.

Un ejemplo de uso:

```
<jsp:plugin type=applet code=”Molecule.class” codebase=”/html” >
<jsp:params>
        <jsp:param name=”molecule” value=”molecules/benzene.mol”/>
</jsp:params>
<jsp:fallback><p> Imposible inicializar el plugin </p></jsp:fallback>
</jsp:plugin>
```

Empleo de JavaBeans en páginas JSP

En el momento de establecer la arquitecture de desarrollo de una aplicación que incluya JSP, suele ser recomendable el tratar de encapsular toda la lógica de negocio de la aplicación en componentes reutilizables, pudiéndose insertar éstos en cualquier página JSP que requiera de la funcionalidad contenida en ellos.

Los JavaBeans son la implementación de la idea de componentes particularizada al lenguaje de programación Java, correspondiéndose entonces un JavaBena con una clase Java que se ajusta a los siguientes criterios o restricciones.

1. Es pública.
2. Posee un constructor público sin argumentos.
3. Métodos get y set públicos para simular propiedades. Los métodos get no tendrán argumentos, salvo que se refieran a propiedades indexadas.

La arquitectura de los JavaBeans emplea reflexión para inferir los métodos públicos asociados a la clase en cuestión, aunque se puede emplear una clase adicional de tipo BeanInfo, que deberá tener el mismo nombre que la clase asociada al bean seguido de BeanInfo, para dar explícitamente la información del bean en cuestión.

Un Bean también debe poder ser serializado, guardándose para su uso posterior. Esto se logra haciendo que dicho Bean implemente el interfaz java.io.serializable, de este modo, en el momento en que el componente se serialice, guardará su estado actual, el cual viene definido por el valor de sus propiedades públicas.

Como se ha comentado, el valor de una propiedad de un bean siempre es establecido o obtenido empleando una nomenclatura bien definida. Para cada propiedad, deben existir dos métodos, de establecimiento y obtención, de la forma ..getXXX() y setXXX(...), donde XXX se corresponderá con el nombre de propiedad (habiéndose pasado su primera letra a mayúscula).

A pesar de que un Bean es como una clse Java “normal”, generalmente, estos beans son importados en entornos de desarrollo integrado (IDEs), de modo que la propia herramienta será capaz de entender las propiedades y métodos contenidos en dicho bean, y emplearlos para el beneficio del desarrollador (el cual podrá aumentar la “paleta” de beans gráficos, en el caso que éstos lo sea, con todos aquellos que pueda conseguir o implementar).

Nótese que la mayor parte de esa lógica de negocio que se está comentando aquí debería ir emplazada, en condiciones normales, en los que se denominan Enterprise JavaBeans (EJBs), en los que lo concerniente a transacciones y escalado son problemas a resolver explícitamente por el contenedor y no por el Bean.

Ejemplos de uso

A continuación se listan ejemplos de uso de páginas JSP en la que se muestra el uso de todos los elementos JSP.

1.- En este primer ejemplo recopilatorio se muestran distintas funcionalidades como recuperar el nombre del Host remoto, conseguir la cadena QueryString, uso de accesos a la página y mostrar la fecha del servidor.

```
<HTML>
<HEAD><TITLE>Primer ejemplo recopilatorio</TITLE></HEAD>
<BODY>
<B>Expresión.</B>
Nombre de la máquina cliente: <%=request.getRemoteHost() %>
<B>Scriptlet.</B>
<% out.println("Datos de la petición (GET):" + request.
getQueryString ()); %>
<B>Declaración y expresión.</B>
<%! Private int accessCount = 0; %>
Accesos a la página desde que se arrancó el servidor: <%= +
+accessCount %>
<B>Directiva import y expresión.</B>
<%@ page import = "java.util.*" %>
Fecha actual:
<%= new Date() %>
</BODY>
</HTML>
```

2.- En este segundo ejemplo se muestra el uso de la accion jsp:useBean.

```
<HTML>
<HEAD>
<TITLE>Reutilizando componentes Java en una página JSP</TITLE>
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
      <TR><TH CLASS="TITLE">
      Reutilizando JavaBeans en JSP</TABLE>
<P>
<jsp:useBean id="mensaje" class="MensajeBean"/>
<% mensaje.setMensaje("Hola mundo desde un Bean"); %>
<H1>Mensaje:</H1>
<%=mensaje.getMensaje()%>
</CENTER>
</BODY>
</HTML>
```

Donde el código asociado al bean MensajeBean es el siguiente:

```
public class MensajeBean {
      private String mensaje = "Valor no establecido";
      public String getMensaje() {
                return(mensaje);
      }
      public void setMensaje(String mensaje) {
            this.mensaje = mensaje;
      }
}
```

3.- Los siguientes ejemplos establecen los valores del Bean asociado a partir de los datos contenidos en la petición (Request).

```
<jsp:setProperty name="request" property="*" />
<jsp:setProperty name="user" property="user" param="username" />
```

A continuación se muestra un ejemplo en que se asocia un valor a la propiedad de un determinado Bean mediante una expresión a calcular en el momento de la petición.

```
<jsp:setProperty name="results" property="row" value="<%=i+1%>"  />
```

4.- A continuación se pretende diseñar una aplicación en Web que permita navegar por un árbol de directorios, mostrnado las propiedades de los distintos ficheros contenidos en el mismo.

Para ello, el primer paso será el diseñar de alguna forma la arquitecura de la "aplicación" en cuestión, definiendo los posibles servlets, beans y páginas JSP que pudieran estar involucradas en su funcionamiento. Para el ejemplo en cuestión, esta aplicación constará de un Bean, denominado Explorador, que nos permitirá la apertura de un directorio determinado, y el acceso a los ficheros contenidos en él, en particular su nombre, atributos y tamaño, para mostrarlo después en la parte de la aplicación asociada a su interfaz (vista).

En segundo lugar, como ya se ha dejado entrever, aparecerá una página JSP que será la encargada de mostrar el contenido del directorio en que se encuentre el bean en cuestión en cada momento, generando enlaces para aquellos ficheros que sean a su vez directorios, para poder acceder al contenido de los mismos. Nótese que, con objeto de no complicar en exceso la aplicación, no se permitirá la descarga de dichos ficheros al cliente, puesto que ello conllevaría el comprobar las posibles restricciones de acceso del servidor (contenedor) en que se esté ejecutando la aplicación, comprobando con ello que el fichero se encuentre dentro del subárbol de directorios "controlado" por dicho servidor, etc.

A continuación se muestra el código asociado al bean Explorador, con las funcionalidades básicas para permitir únicamente mostrar el contenido de los directorios.

```
public class Explorador{
      private java.io.File directorioActual = null;
      private java.util.Vector contenido = null;
      private String separator = java.io.File.separator;
      private java.util.Enumeration enumeracionContenido = null;
      private java.io.File archivoActual = null;
public Explorador() {
      super();
      contenido = new java.util.Vector();
}
public bolean esDirectorio() {
      if(archivoActual != null)
            return archivoActual.isDirectory();
      else
            return false;
}
public String getAtributos() {
      if(archivoActual != null) {
            StringBuffer atributos = new StringBuffer();
            if(archivoActual.isDirectory())
               atributos.append("d");
            else if (archivoActual.isFile())
               atributos.append("f");
            else
               atributos.append("-");
```

```
                if(archivoActual.canRead())
                   atributos.append("r");
                else
                   atributos.append("-");
                if(archivoActual.canWrite())
                   atributos.append("w");
                else
                   atributos.append("-");
                return atributos.toString();
          } else {
                return "Archivo no existe";
          }
   }
   public String getDirectorioActual() {
          if(directorioActual != null)
                return directorioActual.getPath();
          else
                return null;
   }
   public String getNombre() {
          if(archivoActual != null){
                return new Long(archivoActual.getName()).toString();
          }
          else
                return "Sin nombre";
   }

   public String getTamanyo() {
          if(archivoActual != null){
                return new Long(archivoActual.length()).toString();
          }
          else
                return "-1";
   }
   public boolean masArchivos() {
          return enumerationContenido.hasMoreElements();
   }
   public void setDirectorioActual(String newDirectorioActual){
          directorioActual = new java.io.File(newDirectorioActual);
          String contents[];
          if((contents = directorioActual.list()) != null){
                contenido.removeAllElements();
                for (int i = 0;i < contents.length; i++)
                      contenido.addElement(new
java.io.File(directorioActual.getPath()      +      separator      +
contents[i]));
          }
          enumeracionContenido = contenido.elements();
   }
   public void siguienteArchivo() {

archivoActual=(java.io.File)enumeracionContenido.nextElement();
   }
   }//fin de la clase
```

Como se ha comentado, el bean anterior será el que emplee la página JSP asociada al interfaz de la aplicación para acceder a los datos relevantes de los ficheros del directorio en que se encuentre la aplicación en cada momento.

El código de dicha página JSP se muestra a continuación:

```
<html>
<head>
<title>Explorador de ficheros</title>
</head>
<body bgcolor=gray>
<%@ page language="java" session="true" %>
<jsp:useBean id="explorador" class="Explorador">
<!--explorador estará accesible desde la session (HttpSession)
con getValue() -->
<jsp:setProperty name="explorador" property="directorioActual"
param="dir"/>
</jsp:useBean>

<center>
<table border=8 bgcolor=orange >
<tr><td><h1><font color=lightblue>
Contenido de <jsp:getProperty name="explordaor"
property="directorioActual"/>
</font></h1>
</td></tr>
</table>
<br>
<hr width="80%">
<p>
<table border=2 bgcolor=black width="80%" cols=10 >
<%
      while (explorador.masArchivos()){
            explorador.siguienteArchivo();
%>
<tr><td>
      <font color=#aaaaaa>
      <%= explorador.getAtributos() %>
      </font>
</td>
<td colspan=8>
      <font color=#aaaaaa>
      <%
      if (explorador.esDirectorio()) {
      %>
<a href="<%= request.getRequestURI()
+"?"+Request.getQueryString()
+java.io.Fileseparator+explorador.getNombre() %>"<
%=explorador.getNombre() %> </a>
<%
}
else{
%>
<%= explorador.getNombre() %>
<%
}
%>
</font>
</td>
<td align=right>
      <font color=#aaaaaa>
      <%=explorador.getTamanyo() %>
      </font>
</td>
</tr>
<%
} // fin del while
```

```
%>
</table></center></body>
</html>
```

Como puede apreciarse en el fragmento anterior, la página en cuestión se limita a construir una tabla para cada uno de los ficheros contenidos en el directorioActual asociado al explorador en un momento dado, a partir de la "pseudoEnumeración" que pone éste a disposición de la página.

Referencias

Libros/Revistas:

Desarrollo de Servidores Java: Servlets y JSP

Autores: Genoveva López Gómez, Francisco Javier Soriano Camino, Francisco Fernández Velasco, Juan Pablo Rojas Jiménez.

Referencia obligada para el programador de Servlets y JSP. Es difícil encontrar referencias en castellano tan buenas. Libro de la Facultad de Informática de la Universidad Politécnica de Madrid.

Java Server Programming (Professional)

Autores:Danny Ayers, Hans Bergsten, ...
Editorial: Wrox
Páginas: 1121
ISBN: 1-861003-77-7

Libro completísimo en lo que se refiere a la programación de servidores Java. Destinado a programadores avanzados, es un libro que trata temas como servlets, JSP, XML, EJBs, JDBC, JNDI, JavaMAil, RMI, CORBA, Jini/JavaSpaces, Apache,... Mencionar que el número de autores de este libro asciende a 12.

Enlaces:

http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

Tutorial de Servlets con referencias también a JSP.

FastCGI (Fast Common Gateway Interface)

Introducción

FastCGI es una evolución natural de CGI en la que el CGI se conserva en memoria de forma persistente, teniendo otras mejoras como la posibilidad de que el cgi-bin resida en máquinas diferentes a la del servidor de web, algo que conlleva ventajas a la hora de repartir la carga y fijar políticas de seguridad.Fast Common Gateway Interface (FastCGI) es una mejora a la existente CGI (Common Gateway Interface), la cual es un estándar para interactuar con aplicaciones externas con servidores Web.

FastCGI es una propuesta abierta y esperamos libre en servidores Web comerciales que lo soporten. FastCGI es incluido en Open Market WebServer y Secure WebServer, versiones 2.0 y superiores.

En el CD se incluye mod_fastcgi.dll que provee de soporte para poder utilizar FastCGI con Apache. En el mismo directorio se encontrará un archivo html con las instrucciones para instalar dicho módulo.

Ventajas de FastCGI

FastCGI expande y realza el modelo CGI en varios caminos:

FastCGI facilita aplicaciones persistentes entre peticiones de clientes, eliminando gastos de arranque de la aplicacion y permitiendo a la aplicacion mantener estado entre llamadas de clientes.

FastCGI permite a las aplicaciones residir en sistemas remotos (más que tener que residir en el mismo sistema como servidor Web).

FastCGI permite una flexibilidad adicional en las aplicaciones, con soporte explícito para aplicaciones que autentifiquen clientes y filtro de entradas.

Las aplicaciones CGI son efímeras y “short-lived”: cada vez que un cliente realiza una petición a una aplicación CGI, el servidor pregunta al sistema operativo para crear un nuevo proceso CGI. Después de que el proceso CGI procesa la petición, el servidor mata el proceso. El servidor crea y posteriormetne mata un nuevo proceso por cada petición cliente.

Las aplicaciones FastCGI son “long-lived”, y pueden persistir entre llamadas de clientes. El servidor crea el proceso FastCGI una vez y continua corriendo y satisfaciendo peticiones cliente hasta que es explicitamente terminado. También es posible invitar al servidor Web para iniciar multiples copias de una aplicación FastCGI, si preveemos que procesando concurrentemente mejorará el rendimiento de la aplicacion.

Las aplicaciones “long-lived” tienen dos importantes ventajas sobre las aplicaciones “short-lived”:

Una aplicación “short-lived” cuesta la sobrecarga del inicio en todas las peticiones; una aplicacion “long-lived” dispersa la sobrecarga sobre muchas peticiones. Para una aplicacion que tiene un coste de arranque excesivo, tal y como abrir una base de datos, haciendo una inicialización sobre todas las llamadas puede ser ineficiente. El proceso de reinicialización para todos los clientes es ineficiente

para programas Perl, donde el interprete lee a través del programa entero antes de ejecutarlo.

Una aplicación “long-lived” puede cachear informacion en memoria entre peticiones, permitiendo responder más rápidamente a peticiones posteriores.

FastCGI no es el único camino para conseguir aplicaciones “long-lived” en la Web. Por ejemplo, hay muchos motores existente que implementan aplicaciones “long-lived”. En muchos casos, estas aplicaciones se acoplan a los servidores Web. En otras palabras, desde muchos servidores Web no soportan aplicaciones “long-lived”, un programador debe codificar este soporte dentro del servidor Web. Este método requiere una tremenda cantidad de trabajo y también ata la aplicación a un servidor particular.

Otro camino para conseguir aplicaciones “long-lived” es escribir código que llama a rutinas de la API del servidor Web. Esta alternativa invuelve mucho código extra, ata la aplicación a un servidor Web particular, y introduce problemas de mantenimiento, escalabilidad, y seguridad.

FastCGI es la estrategia más general y flexible para construir aplicaciones “long-lived” Web. Separar la aplicacion y aplicaciones CGI servidor deben correr sobre el mismo nodo que el servidor Web; las aplicaciones FastCGI pueden correr sobre algún nodo que puede ser alcanzados desde tu servidor Web usando protocolos TCP/IP. Por ejemplo, podemos querer correr la aplicación FastCGI sobre un servidor “high-speed computer” o motor de base de datos, y el servidor web en un nodo diferente.

Roles de Fast CGI

Las aplicaciones CGI y FastCGI son caminos efectivos para permitir a una aplicacion actuar como una extension del servidor Web. CGI no suministra explicitamante soporte para diferentes clases de aplicaciones: bajo CGI, todas las aplicaciones reciben una petición HTTP, haciendo algo con ella, y generando una respuesta HTTP. FastCGI suministra soporte para varios "roles" comunes que las aplicaciones pueden desempeñar.

Los tres papeles soportados por el WebServer 2.0 son:

1. Responder (Contestador)
2. Filter (Filtro)
3. Authorizer (Certificador)

Aplicaciones Responder

Una aplicación responder es la clase más básica de aplicación FastCGI: recibe la información asociada con la petición HTTP y genera una respuesta HTTP. El Responder es el role más similar a la programación tradicional de CGI, y la mayoría de las aplicaciones FastCGI son responders.

Aplicaciones Filter

Un aplicación filtro FastCGI recibe la información asociada con la petición HTTP, más una flujo extra de datos de un fichero almacenado en el servidor Web, y genera una version "filtered" filtrada del flujo de datos como respuesta HTTP. Con las aplicaciones filtro, el administrador del sistema mapea unos tipos MIME para filtrar una aplicación FastCGI particular. Cuando un cliente pide una URL con ese tipo MIME, el servidor Web invoca la aplicación filtro, la cual procesa el fichero

especificado en la URL y envía la respuesta (usualmente texto HTML) de regreso al cliente.

Por ejemplo, escribamos una aplicación filter FastCGI que convierta texto SGML a HTML, y mapee la extensión .sgml (MIME-type SGML) a la aplicación FastCGI filter. Ahora, supongamos una petición de usuario de la siguiente manera:

www.aerjug.com/docs/chap1.sgml

Dada esta URL, el servidor Web pasa chap1.sgml como entrada a la aplicación FastCGI filter, la cual procesa chap1.sgml y devuelve una version HTML de la petición cliente.

Aplicaciones Authorizer

Una aplicación FastCGI authorizer recibe la información en una cabecera HTTP y genera una decisión así como se autorice la petición. Para marcar una aplicación FastCGI con el papel de authorizer, el administrador del sistema nombra la aplicacion dentro del archivo de configuración del servidor, usando una directiva llamada AuthorizeRegion.

Cuando una petición de cliente a una URL que encuentre el criterio AuthorizeRegion, el servidor Web llama a la aplicación FastCGI authorizer. Si la aplicacion concede la autorización (por retorno de una respuesta con código 200 OK), el servidor Web reactiva la ejecución de comandos en la sección AuthorizeRegion. Si la aplicacion deniega la autorización (por respuesta con otro código), el servidor Web para el proceso de comandos siguientes en la sección AuthorizeRegion, y devuelve la respuesta de la aplicación FastCGI al cliente. Las aplicaciones Authorizer pueden devolver cabeceras que contengan variables de netorno. Otros programas CGIs o FastCGI acceden a estas respuestas (incluyendo otras autorizaciones) pueden acceder a estas variables de entorno. Las cabeceras debe tener el siguiente formato:

Variable-name: valor

Por ejemplo, la siguiente cabecera

Variable-AUTH_METHOD: la base de datos mira la variable de entorno AUTH_METHOD para poner a "database lookup" esta petición. Otras aplicaciones CGI o FastCGI corriendo en esta petición pueden acceder al valor de AUTH_METHOD.

Las aplicaciones Authorizer no pueden leer exitosamente desde la entrada standard. El intento de leer desde la entrada standard resulta en un inmediato EOF.

Cómo funciona FastCGI

El trabajo que implica escribir una aplicación FastCGI depende en gran parte del uso de librerías de I/O que se usen. Vamos a describir como escribir aplicaciones FastCGI en términos de librerías de Open Market, las cuales están disponibles para C, Perl, y Tcl. FastCGI es un standard abierto y posibilita la construcción de nuestras propias librerías en otros lenguajes.

En general, el objetivo de las librerías es facilitar la tarea de escribir una aplicación FastCGI tanto como una CGI. Por ejemplo, usando las mismas técnicas para decodificar “query string”, salidas HTML a stdout, usar variables de entorno, y demás. Cuando usamos nuestras librerías, migrando aplicaciones CGI a FastCGI es en mayor parte una tarea de reestructuración del código para tomar ventaja de los rasgos de FastCGI y sus librerías.

Estructura del código

La principal tarea de conversión de un programa CGI a uno FastCGI esta en separar el código de inicialización del código que necesita correr para cada petición. La estructura debería parecerse algo a esto:

Código de Inicialización
    Comienzo del bucle de Respuesta
        Cuerpo del bucle de Respuesta
    Fin del bucle de Respuesta

El código de inicialización es lanzado una vez exactamente, cuando la aplicación es inicializada. La inicialización del código usualmente desempeña operaciones de consumición de tiempo tal y como abrir bases de datos o calcular valores para tablas o bitmaps.

El bucle de respuesta corre continuamente, esperando a que la petición de un cliente llegue. El bucle empieza con una llamada a FCGI_Accept, una rutina en la librería FastCGI. La rutina FCGI_Accept bloquea la ejecución del programa hasta que llegue una petición de cliente a la aplicacion FastCGI. Cuando la petición de un cliente llega, FCGI_Accept se desbloquea, corre una iteracion del cuerpo del bucle de repuesta, y luego se bloquea de nuevo esperando por otra petición de cliente. El bucle termina solo cuando el administrador del sistema o del servidor Web mata la aplicación FastCGI.

Inicialización de Variables de Entorno

Cuando un proceso FastCGI comienza, todavía no ha aceptado ninguna petición, y por consiguiente ninguna de las variables de entorno CGI están determinadas.

Se determina el entorno inicial de un proceso FastCGI empezando por la directiva AppClass usando la opción -initial-env. El proceso podría usar este entorno (environment) para configurar sus opciones y localizar ficheros y bases de datos.

En procesos FastCGI que comienzen por la directiva AppClass con la opción -affinity, la variable FCGI_PROCESS_ID es determinada en el entorno inicial (no en el entorno de la petición).

FCGI_PROCESS_ID es un número decimal number en el rango de 0 a N - 1 donde N es el número de procesos (argumento para la opción -processes de AppClass). El proceso prodría usar FCGI_PROCESS_ID en conjunción con otras variables para localizar ficheros de logs (session-related).

Variables de Entorno Por-Request

En general, , FastCGI usa las mismas variables de entorno por-request que CGI, y podemos acceder a los valores de ellas en las aplicaciones FastCGI tal y como hacíamos con las aplicaciones CGI. La únicas diferencias son las siguientes:

En aplicaciones FastCGI Authorizer, el servidor Web no determina las variables PATH_INFO, PATH_TRANSLATED, y CONTENT_LENGTH.

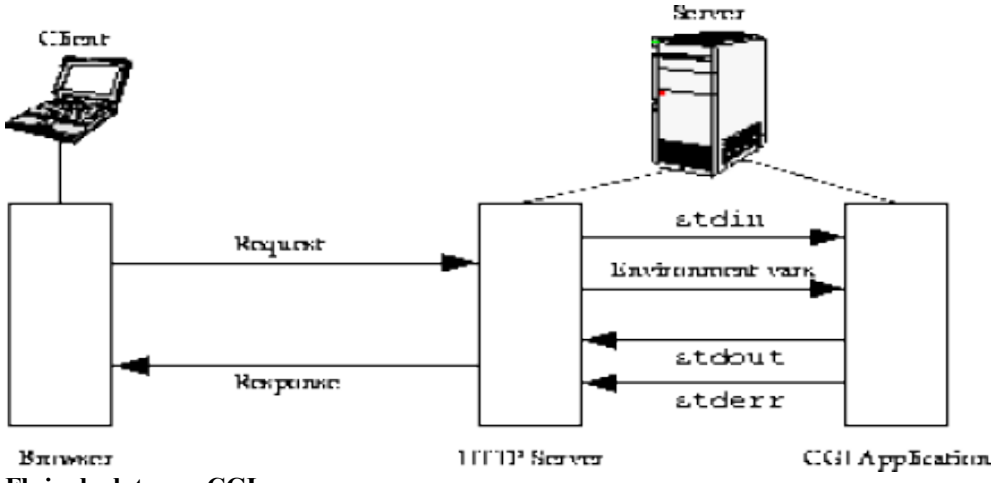
En aplicaciones FastCGI Authorizer, el servidor Web determina dos variables de entorno adicionales:

- FILE_LAST_MOD: El servidor Web determina FILE_LAST_MOD a la fecha y hora que filtro el fichero de entrada por última vez. El formato es el número de segundos desde medianoche (UTC), 1 Enero, 1970.
- FCGI_DATA_LENGTH: La aplicación lee al menos FCGI_DATA_LENGTH bytes del flujo de datos antes de recibir la indicación de fin de flujo.

FastCGI determina FCGI_ROLE por cada petición a RESPONDER, AUTHORIZER, o FILTER.

Detalles de Implementación

Las librerías de aplicación FastCGI son diseñadas para encapsular detalles del diselo de FastCGI. Como se muestra en la imagen siguiente, las aplicaciones CGI usan los tres flujos standard POSIX (stdin, stdout, and stderr), más variables de entorno, para comunicarse con un servidor HTTP.



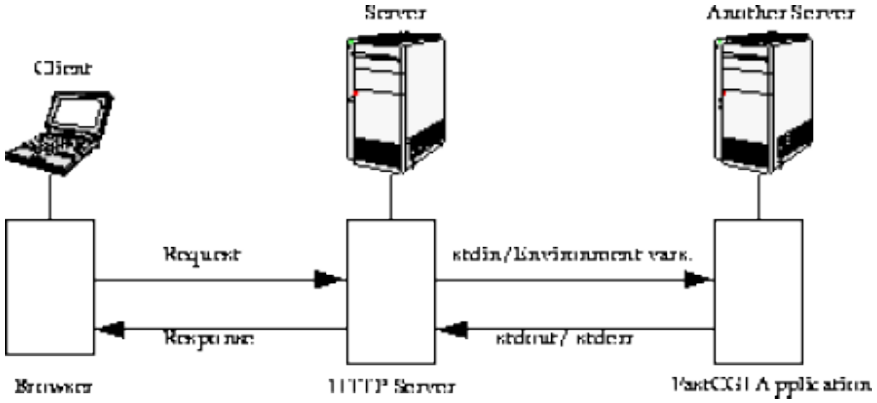
Flujo de datos en CGI

La diferencia fundamental entre FastCGI y CGI es que las aplicaciones FastCGI son “long-lived”, lo cual significa que el servidor necesita reunirse con una aplicación funcionando, más bien que la aplicación iniciada, con propósito de comunicarse con ella.

La implemantación FastCGI básicamente crea una conexión bidireccional entre dos procesos que no tienen relación. FastCGI usa una simple conexión para todos los datos asociados con una aplicación -- stdin, stdout, stderr, y variables de entorno. Los datos en la conexión se encapsulan usando un protocolo FastCGI que permite stdin y variables de entorno compartir la misma media conexión (en el camino in) y stdout y stderr compartir la media conexión (en el camino way).

En el lado de la entrada (input), la aplicación FastCGI recibe datos en la conexión, los desembala para separar stdin de las variables de entorno y luego invoca la aplicación. En el lado de la salida (output), FastCGI envuelve stdout y stderr con cabeceras del protocolo apropiadas, y envia los datos encapsulados fuera al servidor.

Desde una aplicación FastCGI no siempre corre en el mismo nodo que el servidor HTTP, soporta dos implementaciones de conexión: un flujo canalizado, para comunicaciones en la misma máquina, y flujo TCP, para comunicaciones con el cliente y el servidor cuando están en máquinas diferentes.



Flujo de datos en FastCGI cuanod servidor y aplicación están en diferentes máquinas.

La librería fcgi_stdio: Compatibilidad I/O

La implementación para la compatibilidad I/O es que la libreria fcgi_stdio.h contenga macros para traducir los tipos y procedimientos definidos en stdio.h dentro de las llamadas apropriadas a FastCGI. Por ejemplo, considere un programa FastCGI escrito en C conteniendo las siguientes líneas de código:

```
fprintf(stdout, "<H2>Aerobic Juggling</H2>/n");
fcgi_stdio.h archivo de cabecera que contiene la macro

#define fprintf FCGI_fprintf
Por lo tanto el preprocesador traduce la llamada a fprintf
Dentro de la siguiente llamada:

FCGI_fprintf(stdout, "<H2>Aerobic Juggling</H2>/n");
FCGI_fprintf toma los mismos argumentos que fprintf.
```

La implementación de FCGI_fprintf testea el fichero para ver si es un flujo normal C o un flujo FastCGI, y llama a la apropiada implementación.

El fichero de cabecera fcgi_stdio.h contiene macros para traducir llamadas a todas las rutinas ISO stdio.h (y todas las adicionales convenciones Posix, como fileno, fdopen, popen, y pclose) dentro de sus equivalentes FastCGI.

La librería fcgi_stdio: Compatibilidad Binary

La librería fcgi_stdio suministra de toda la compatibilidad binaria entre aplicaciones FastCGI y CGI: podemos correr el mismo binario C de CGI o FastCGI.

La implementación está en FCGI_Accept: la función FCGI_Accept testes el entorno para determinar si la aplicación fue invocada como un CGI o como FastCGI. Si fue invocada como un programa CGI, la petición bucle satisface a una simple petición de cliente y luego sale, produciendo un comportamiento de CGI.

Desarrollando aplicaciones FastCGI en C

Vamos a ver como codificar aplicaciones FastCGI en C y como convertirlas en ejecutables. Si estas convirtiendo una aplicación CGI en una aplicación FastCGI, en la mayoría de los casos solo necesitamos añadir unas líneas de código. En aplicaciones más complejas, se necesitará reorganizar el código.

Las librerías I/O

El FastCGI Software Development Kit que acompaña a Open Market WebServer 2.0 incluye librerías I/O para simplifar el trabajo de conversión de aplicaciones CGI a FastCGI o escribir nuevas aplicaciones FastCGI. Hay dos librerías in el kit: fcgi_stdio y fcgiapp. Debes incluir una de estas cabeceras en tu programa:

- fcgi_stdio.h
- fcgiapp.h

La librería fcgi_stdio es una capa por encima de la librería fcgiapp, y recomendamos que se use, ambas convierten las aplicaciones CGI existentes para escribir nuevas aplicaciones FastCGI. La librería fcgi_stdio ofreces varias ventajas:

Simplicidad: hay solo 3 nuevas llamadas a la API para aprender.

Familiaridad: Si se está convirtiendo una aplicación CGI a FastCGI, debes encontrar algunos cambios entre CGI y FastCGI. Uso de las mismas variables de entorno, mismas técnicas para analizar las QueryStrings, las mismas rutinas I/O, y demás.

Conveniencia: la librería provee de todos los binarios para compatibilidad entre CGI y FastCGI. Esto es, tu puedes correr los mismos binarios tanto sean CGI como FastCGI.

La librería fcgiapp es más específica para FastCGI, sin intentar proveer la tapa de familiaridad con CGI.

Estructura del código

Estructurar el código para FastCGI, hay que separar tu código en dos secciones como vimos anteriormente:

- La sección de inicialización, la cual es ejecutada sólo una vez.
- Sección bucle de respuesta, la cual se ejecuta tantas veces como sea llamada.

El bucle de respuesta típicamente tiene el siguiente formato:

```
while (FCGI_Accept() >= 0) {
# body of response loop
}
```

El FCGI_Accept bloquea hasta que llega una petición del cliente, y devuelve entonces 0. Si hay un fallo del sistema, o el administrador del sistema termina el proceso, Accept devolverá -1.

Si la aplicación fue invocada como un programa CGI, la primera llamada a Accept devuelve 0 y la segunda siempre devuelve -1, produciendo un comportamiento CGI.

Note también que el mundo CGI recomienda pequeños scripts, mientras que FastCGI recomienda combinación de scripts. Podremos volver a pensar la estructura global de tus aplicaciones para tomar una mejor ventajato de las ganancias de rendimiento con FastCGI.

Construcción

Esta sección explica como construir y depurar las aplicaciones FastCGI escritas en C.

El preprocesador C necesita conocer la localización del fichero de cabecera fcgi_stdio.h, el cual está en el siguiente pathname:

$toolkit/include/fcgi_stdio.h

donde *$toolkit* simboliza el directorio en el cual tienes instalado el Software Development Kit para FastCGI.

El enlazador necesita conocer la localización del fichero de librería libfcgi.a, el cual está en el siguiente pathname:

$toolkit/libfcgi/libfcgi.a

Si tu enlazador (linker) no encuentra la librería de Berkeley socket, entonces necesitas forzar las directivas del linker a su búsqueda.

Un ejemplo de Makefile está en el siguiente pathname:

$toolkit/examples/Makefile

Este Makefile contiene las normas necesarias y pathnames para construir la aplicación FastCGI en C. Para construir la aplicación de ejemplo:

$./configure
$ make

Desarrollando aplicaciones FastCGI en Perl

Este capítulo explica como codificar aplicaciones FastCGI en Perl. Antes de construir aplicaciones FastCGI en Perl, debes tener una versión FastCGI-savvy del intérprete de Perl. Open Market desarrolla estos binarios Perl para las plataformas más populares y hacerlas disponibles con un kit de desarrollo.

Los binarios FastCGI-savvy son extensiones de Perl standard, e intentan reemplazar tu instalación existente de Perl.No hay necesidad de mantener dos versiones de Perl: la version que suplimos seguirá funcionando cuando invoquemos desde un shell o programa CGI. Hay además direcciones en el kit de desarrollo para hacer tu porpio Perl FastCGI-savvy, si tu necesitas una versión para la misma plataforma que no suplimos.

FastCGI es ideal para escribir aplicaciones en Perl, porque provee de un gran rendimiento. Cuando tu corres un script en Perl, el intérprete de Perl analiza el script entero antes de executarse. Con FastCGI, podemos factorizar este coste de inicialización y pagarlo sólo una vez, haciendo la ejecución del actual script más rápida en respuestas a el cliente.

La primera línea de un script en Perl especifica típicamente el pathname del intérprete de Perl. En nuestro caso debemos especificar el pathname del Perl FastCGI-savy.

Después, debes contarle a Perl para cargar la extension FastCGI. Hecho así, en la segunda línea cerca del principio del script FastCGI:

use FCGI;

Después, hay que dividir los scripts FastCGI dentro de las dos siguientes secciones:

La sección de Inicializacion, la cual es ejecutada sólo una vez.

La sección del bucle Response, el cual consigue ejecutarse tantas veces como el script FastCGI sea invocado.

Un bucle response típicamente tiene el siguiente formato:

```
while (FCGI::accept >= 0) {
# cuerpo del bucle de respuesta
}
```

La llamada accept devuelve 0 cuantas veces un cliente pida el script FastCGI. De otra manera, la llamada a accept devuelve -1.

Desarrollando aplicaciones FastCGI en Java

Como sabemos Java es adecuado para escribir aplicaciones del lado servidor (Servlets), y este rango de aplicabilidad crece como los compiladores Java y otras herramientasand avanzadas para Java. Los módulos Java, el recolector de basura, y los hilos son especialmente valiosos para escribir aplicaciones 'long-lived' servidoras.

La clase FCGIInterface (no es una interface aunque el nombre de lugar a confusión) provee facilidades para aplicaciones Java analogas a las que proveía fcgi_stdio en aplicaciones C. Incluyendo esta librería tus aplicaciones Java pueden correr como CGI o FastCGI.

Instalación sobre Apache de Scripts FastCGI

Para tener más detalles se debe consultar la documentación del 'FastCGI developer's kit'. Sobre el servidor Apache, la siguiente línea debe ser añadida a srm.conf:

```
AddType application/x-httpd-fcgi .fcgi
```

Los scripts FastCGI deben finalizar en la extensión .fcgi. Por cada script que tu instales debes añadir lo siguiente a srm.conf:

```
AppClass /usr/etc/httpd/fcgi-bin/file_upload.fcgi
-processes 2
```

Estas instrucciones Apache lanzarán dos copias de file_upload.fcgi al inicio.

Ejemplos de uso

1.- Un pequeño ejemplo de una respuesta de una aplicación FastCGI escrita en C: TinyFastCGI

```
#include "fcgi_stdio.h"
```

```
/* fcgi librería, ponerla de primera*/
#include <stdlib.h>
int count;
void initialize(void)
{
  count=0;
}
void main(void)
{
/* Inicialización. */
  initialize();
/* Bucle de respuesta. */
  while (FCGI_Accept() >= 0)   {
    printf("Content-type: text/html\r\n"
           "\r\n"
                              "<title>Hola  FastCGI!  (C,  fcgi_stdio
library)</title>"
           "<h1>Hola FastCGI! (C, fcgi_stdio library)</h1>"
                "Petición número %d running sobre el host <i>
%s</i>\n",
            ++count, getenv("SERVER_HOSTNAME"));
  }
}
```

2.- Un pequeño ejemplo de una respuesta de una aplicación FastCGI escrita en Perl: TinyFastCGI

```
#!fcgi-savvy-perl
use FCGI; # Importar la libreria; requiere la linea
# Código de Inicializacion
$cnt = 0;
# Bucle Response
while (FCGI::accept >= 0) {
  print "Content-type: text/html\r\n\r\n";
    print  "<head>\n<title>FastCGI  Demo  Página  (perl)</title>
\n</head>\n";
  print  "<h1>FastCGI Demo Página (perl)</h1>\n";
  print "Esto viene desde el servidor FastCGI.\n<BR>\n";
      print   "Corriendo   desde   <EM>$ENV{SERVER_NAME}</EM>   a
<EM>$ENV{REMOTE_HOST}</EM>\n<BR>\n";
  $cnt++;
  print "Esta es la conexión número $cnt\n";
}
```

3.- Un pequeño ejemplo de una respuesta de una aplicación FastCGI escrita en Java: TinyFastCGI

```
import FCGIInterface;
class TinyFCGI {
    public static void main (String args[]) {
        int count = 0;
        while(new FCGIInterface().FCGIaccept()>= 0) {
            count ++;
                                System.out.println("Content-type:
text/html\r\n\r\n");
            System.out.println(
                    "<title>Hola FastCGI! (Java)</title>");
            System.out.println("<h1>Hola FastCGI! (Java)</h1>");
            System.out.println(
```

```
                                "request number " + count + " running en
host <i>" +
                            System.getProperty("SERVER_NAME") + "</i>");
            }
        }
    }
```

4.- Generador de números primos. Considerar una aplicación responder que genera el n número primo. Una aplicación CGI podría no ser eficiente. Por ejemplo, si el usuario pregunta por el número primo 50,000, una aplicación CGI tendría que calcular el primer primo, el segundo, y así sucesivamente, hasta el 50,000. La aplicación podrá entonces terminar, realizando con ello todos los duros cálculos. Si un cliente después pide el número primo 49,000, el servidor tendrá que frezar una nueva aplicación CGI la cual tendrá que comenzar a calcular primos desde cero.

Las aplicaciones FastCGI pueden ser más eficientes y ordenar este problema, debido a que pueden mantener el estado. Una aplicación FastCGI puede calcular una tabla extensa de números primos en su fase de inicialización y después mantener la tabla indefinidamente. Cuando un cliente pida un número primo en particular, el bucle respuesta sólo necesita mirar en la tabla.

Aquí el código para el exemplo del cálculo del número primo:

```
#include "fcgi_stdio.h"
#include <stdlib.h>
#include <string.h>

#define POTENTIALLY_PRIME 0
#define COMPOSITE 1
#define VALS_IN_SIEVE_TABLE 1000000
#define MAX_NUMBER_OF_PRIME_NUMBERS 78600

/* All initialized to POTENTIALLY_PRIME */
long int  sieve_table[VALS_IN_SIEVE_TABLE];
long int  prime_table[MAX_NUMBER_OF_PRIME_NUMBERS];
/* Use Sieve of Erastothenes method of building
   a prime number table. */
void
initialize_prime_table(void)
{
 long int prime_counter=1;
 long int current_prime=2, c, d;

  prime_table[prime_counter]=current_prime;

  while (current_prime < VALS_IN_SIEVE_TABLE)   {
   /* Mark off composite numbers. */
     for (c = current_prime; c <= VALS_IN_SIEVE_TABLE;
          c += current_prime)  {
        sieve_table[c] = COMPOSITE;
     }

   /* Find the next prime number. */
     for (d=current_prime+1; sieve_table[d] == COMPOSITE; d++);
   /* Put the new prime number into the table. */
     prime_table[++prime_counter]=d;
     current_prime=d;
  }
}
```

```
void main(void)
{
    char *query_string;
    long int n;

    initialize_prime_table();

    while(FCGI_Accept() >= 0) {
        /*
         * Produce the necessary HTTP header.
         */
        printf("Content-type: text/html\r\n"
               "\r\n");
        /*
         * Produce the constant part of the HTML document.
         */
        printf("<title>Prime FastCGI</title>\n"
               "<h1>Prime FastCGI</h1>\n");
        /*
         * Read the query string and produce the variable part
         * of the HTML document.
         */
        query_string = getenv("QUERY_STRING");
        if(query_string == NULL) {
                printf("Usage: Specify a positive number in the
query string.\n");
        } else {
            query_string = strchr(query_string, `=') + 1;
            n = strtol(query_string);
            if(n < 1) {
                 printf("The query string `%s' is not a positive
number.\n",
                        query_string);
            } else if(n > MAX_NUMBER_OF_PRIME_NUMBERS) {
                   printf("The number %d is too large for this
program.\n", n);
            } else
                    printf("The %ldth prime number is %ld.\n",
prime_table[n]);
            }
        }
    } /* while FCGI_Accept */
}
```

Esta aplicación tiene un notable coste de inicialización mientras rellena la tabla, pero los subsiguientes accesos son rápidos.

Referencias

http://www.fastcgi.com/

Página principal donde se puede encontrar abundante información sobre FastCGI.

http://www.fastcgi.com/devkit/doc/FCGI_Accept.3

`Documentación de FCGI_Accept.3`, describe la función en su formato tradicional.

http://www.fastcgi.com/applibs/

Página Web para descarga de los binarios de Perl y Tcl para diversas plataforams.

http://www.fastcgi.com/servers
Principales servidores que soportan FastCGI.
http://www.openmarket.com/store/eval/swsg.htm
Zona download del 'Open Market Secure WebServer', una versión de libre de este servidor que soporta FastCGI.

Servidor Zope (Z Object Publishing Enviroment)

Introducción

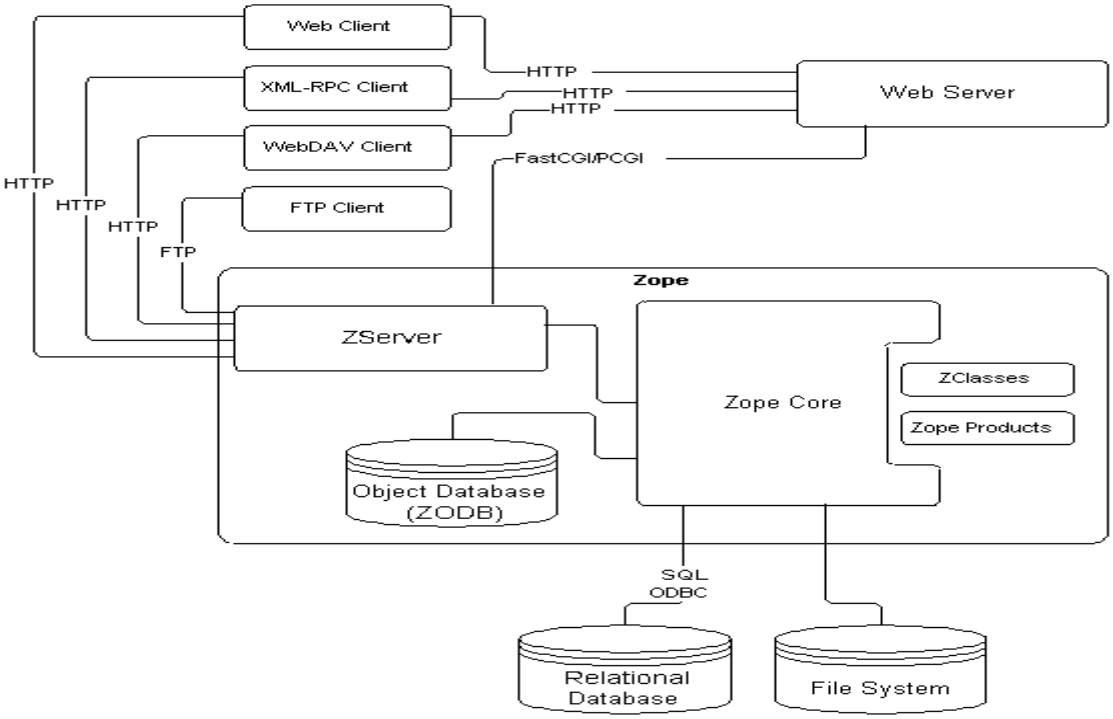
El servidor de web Zope (http://www.zope.org/) es un paso adelante a la hora de programar webs dinámicos. Todas las invocaciones para obtener cualquier página por el cliente provocan la llamada a métodos de objetos residentes en el servidor de aplicaciones web. Aquí el servidor de web deja de ser un conjunto de páginas estáticas y cgi-bins, para pasar a ser un conjunto de objetos configurados por el programador de contenidos. El servidor web Zope (también cuenta con un servicio FTP), facilita la creación de portales para empresas reutilizando cabeceras y pies de página y ofreciendo infinidad de facilidades más.

Cómo funciona Zope

Los lenguajes del lado del servidor, como PHP o ASP, permiten al desarrollador abstraerse de las incomodidades del protocolo CGI y el intercambio de datos, logrando que las aplicaciones basadas en el web sean más sencillas de desarrollar.

Un paso más en este camino hacia el desarrollo de aplicaciones web lo constituyen los servidores de aplicaciones, grupo al que pertenece Zope. Junto con el mecanismo básico de intercambio de datos con los clientes web, dotan al desarrollador de la parte servidor de la aplicación de unas infraestructuras que le facilitan el desarrollo de aplicaciones y reducen el tiempo de su construcción.

La arquitectura de Zope se basa en una serie de mecanismos de intercambio de datos con el servidor Zope y un conjunto de herramientas de apoyo a ese servidor Zope. Lo mejor es ver el esquema que constituye esta arquitectura que se presenta en el portal de Zope (http://www.zope.org/ZopeArchitecture).



Vamos a destacar los distintos clientes de intercambio de información con Zope (cliente Web, XML-RPC, WebDAV y FTP), la posibilidad que incluir un servidor de web como Apache comunicándolo a través de el protocolo CGI persistente, el núcleo de Zope, conocido también como el ORB Zope donde se gestionan las peticiones sobre objetos Zope y donde se interacciona con las clases de objetos Zope y los productos, la base de datos orientada a objetos de Zope y la posibilidad de interaccionar con otros tipos de bases de datos relaciones o sistema de ficheros.

Una de las características fundamentales de Zope es que en su arquitectura todo lo que existe son objetos. Fue una decisión de diseño al intuir en el paradigma de objetos una potente forma de ver las partes de las aplicaciones web. Este tipo de decisiones de diseño son cada vez más comunes en los sistemas de programación actuales y es extraño ver nuevas arquitecturas de propósito general que hayan abrazado el modelo de objetos para sus sistemas.

Como consecuencia de esta decisión hubo que elegir un lenguaje con soporte de objetos. Además de esta característica el lenguaje a elegir debería estar presente en distintas arquitecturas (Windows, GNU/Linux etc.) y debería de facilitar el desarrollo de aplicaciones Internet, lo que básicamente requiere unas librerías de comunicaciones y soporte de HTTP/CGI potentes. De entre las alternativas existentes (Perl, Java, C, C++ ...) se eligió a Python, un lenguaje interpretado, presente en muchas plataformas, con mucha fuerza dentro de la comunidad de desarrolladores de aplicaciones Internet, que ya había mostrado su potencia en el mundo de los CGI y totalmente orientado a objetos (http://www.python.org).

Zope ORB

Ya veremos en los ejemplos que los clientes cuando piden una URL al servidor Zope, lo que están haciendo en realidad es pedir al servidor Zope un objeto. En realidad se pide la ejecución de un objeto con una serie de parámetros. Si el lector es conocedor de CORBA este concepto le recordará a como se ejecutan métodos sobre objetos remotos a través de CORBA.

Cuando el servidor Zope recibe la petición para acceder a un objeto Zope, esta petición va a parar al ORB de Zope que se encarga de localizar al objeto y procesar la petición, mostrando los resultados de la misma.

Este ORB es una parte crítica de la arquitectura ya que de su buen funcionamiento depende el rendimiento y la robustez de la plataforma.

La creación de un web con Zope

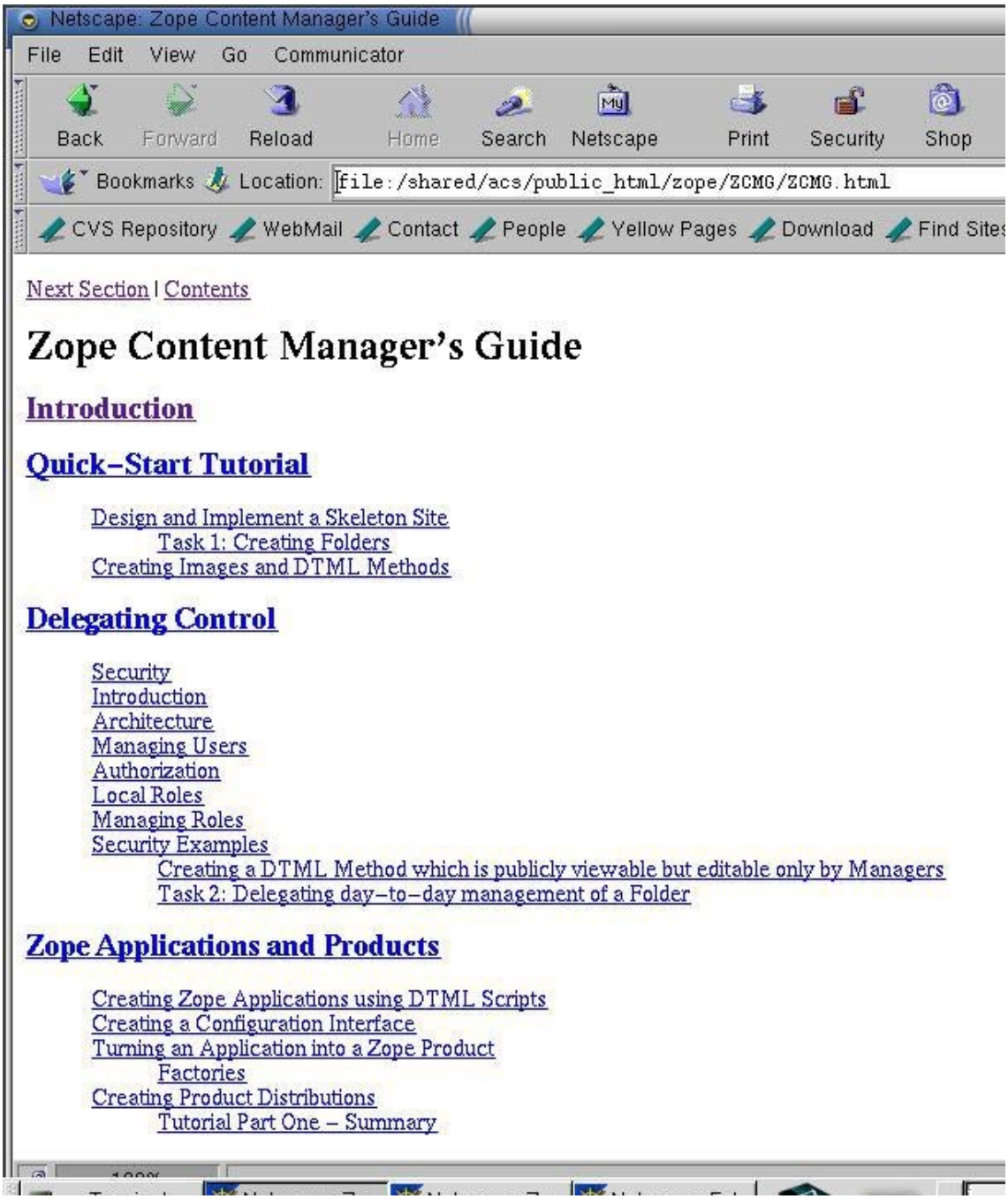
Vamos a analizar con un ejemplo el proceso de publicar objetos Zope. Con Zope el objetivo ha sido el facilitar al máximo la publicación de contenidos a través de páginas HTML. Con el ejemplo que vamos a mostrar podremos analizar con más cuidado si se ha logrado este objetivo o no ha sido así. Para ello será fundamental observar la sencillez del entorno, su potencia a la hora de gestionar los contenidos y los conocimientos técnicos necesarios para publicar contenidos con Zope.

El portal de una empresa

En nuestro caso nos vamos a plantear como ejemplo la realización de un portal para una empresa. Es un ejemplo bastante completo, que encierra muchas de las problemáticas de como llegar a mostrar una imagen dentro de la red por parte de una organización. Y es un ejemplo con el que muchos de los lectores tendrán que enfrentarse en su futuro profesional.

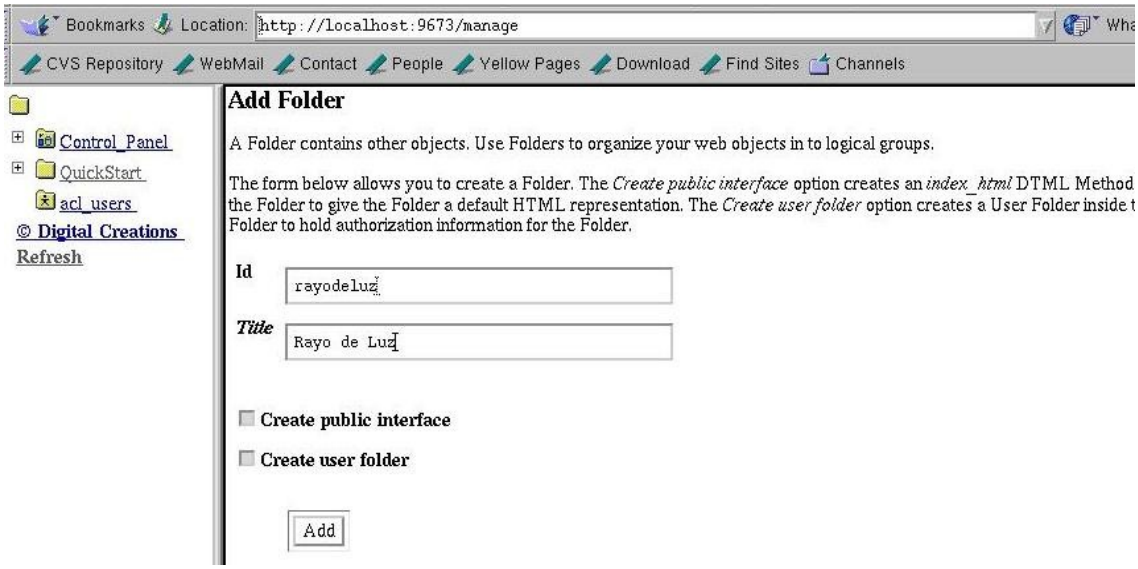
Lo primero que hay que hacer antes incluso de plantearse con que herramienta se va a desarrollar el web es definir que se quiere lograr. Es decir, que contenidos se quieren poner en la red, para quién y como se van a organizar. En nuestro ejemplo nos vamos a centrar en el diseño del portal interno de una compañía, cuyo objetivo es proporcionar a los distintos departamentos de la misma libertad para informar a los demás de sus actuaciones, con el objetivo de mejorar la coordinación y que aumentar el flujo de información, piedras angulares para el buen funcionamiento de una compañía.

Dentro de la amplia documentación de Zope, y para este artículo en concreto, nos va a ser útil la "Guía para el Gestor de Contenidos Zope". Esta guía se puede obtener del web de Zope, en la zona de documentación. En las referencias puedes encontrar los detalles exactos de como descargarla para poderla leer tranquilamente.



Primeros pasos en la creación del web

Nuestra compañía se llama Rayo de Luz por lo que parece lógico que el primer paso que debemos de dar es crear una nueva carpeta (folder) para guardar en él el contenido del web que vamos a crear. Para ello accedemos a la pantalla de gestión de Zope (http://localhost:9673 en Debian) y pasamos a dar este primer paso utilizando la interfaz web de Zope para la publicación de sitios web.



Ya tenemos la carpeta para el web creada y hemos dejado seleccionadas las opciones de crear una interface pública, para que todo el mundo pueda acceder a dicho directorio, y de que se cree una carpeta para los usuarios de esta nueva zona. Gracias a este carpeta de usuarios vamos a poder definir usuarios y grupos dentro de la zona del web de la compañía, usuarios que no serán visibles para otras publicaciones que hagamos con este servidor Zope. Iremos poco a poco viendo la potencia de control de acceso y operación de que dispone Zope, una de las características que nos pueden hacer terminar de decantarnos por este gran producto.

La zona de los departamentos

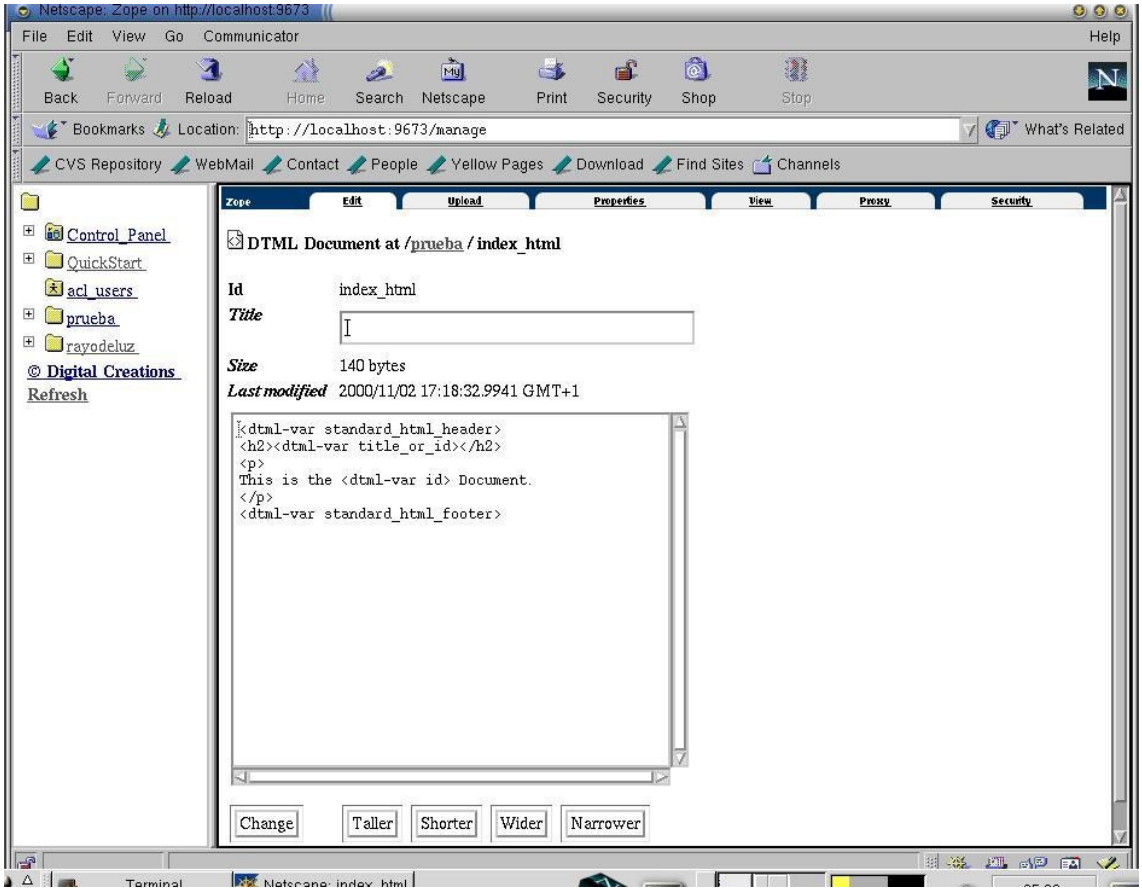
Desde la página principal de nuestra compañía lo mejor que podemos hacer es poner enlaces a los distintos departamentos que forman la compañía un logo de la misma para ir creando la imagen corporativa y mostrarla a nuestros empleados, los visitantes del sitio que estamos construyendo. Para cumplir este objetivo nos vamos a la consola desde la que gestionamos zope (http://localhost:9673/manage) y seleccionamos la carpeta "rayodeluz".

Quizá es un buen momento para recordar al lector que todo lo que está viendo, a pesar de estar organizado como si fueran ficheros de directorios, son objetos, no son ficheros. La carpeta "rayodeluz" es un objeto contenedor con una serie de propiedades. Dentro de este objeto, que es muy similar a una carpeta, nos encontramos con muchos otros objetos, que también pueden ser contenedores, permitiendo la aparición de estructuras en árbol, una abstracción a la que estamos todos acostumbrados gracias a los sistemas de ficheros.

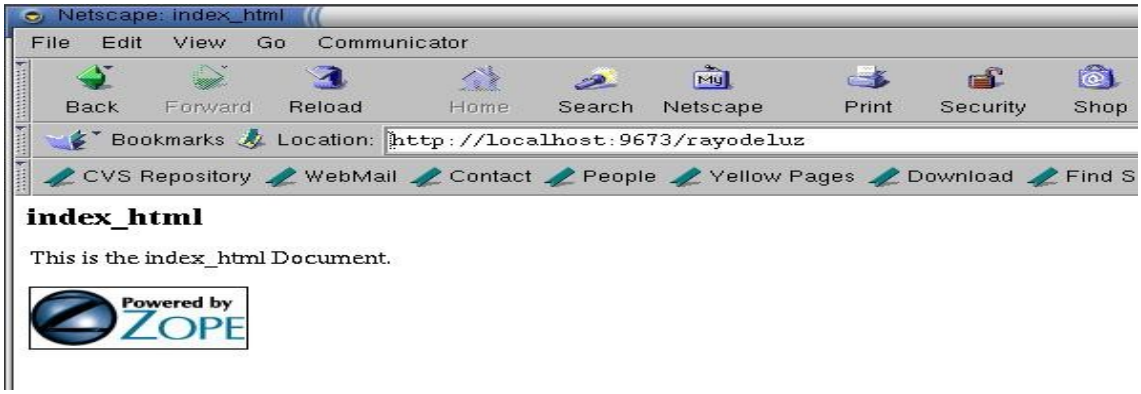
Estamos dentro de la carpeta (objeto) "rayodeluz", que acabamos de crear en el anterior apartado, y cuyo único objeto es un DTML Document llamado "index_html", que se crea de forma automática cada vez que creamos una carpeta. Cuando a través de una URL accedemos a la carpeta, el ORB de Zope de forma

automática nos devuelve el objeto "index_html". De nuevo, es una abstracción muy similar a la que hacen los servidor de web como Apache, donde el fichero índice de los directorios se suele llamar "index.html". El objeto "DTML Document" llamado "index_html" podemos interpretarlo como un fichero que está dentro de la carpeta "rayodeluz", aunque el lector tiene que tener en cuenta que en realidad es un objeto que permite hacer con él bastantes más cosas que con un simple fichero. Esta insistencia en la idea de que todo son objetos es fundamental para lograr comprender la arquitectura con toda su potencia, y no quedarnos en una interpretación demasiado simplista de lo que es Zope.

Vamos a editar el objeto "index_html" para ver como podemos modificar su contenido e incluir el índice a los departamentos de la compañía y el logo.

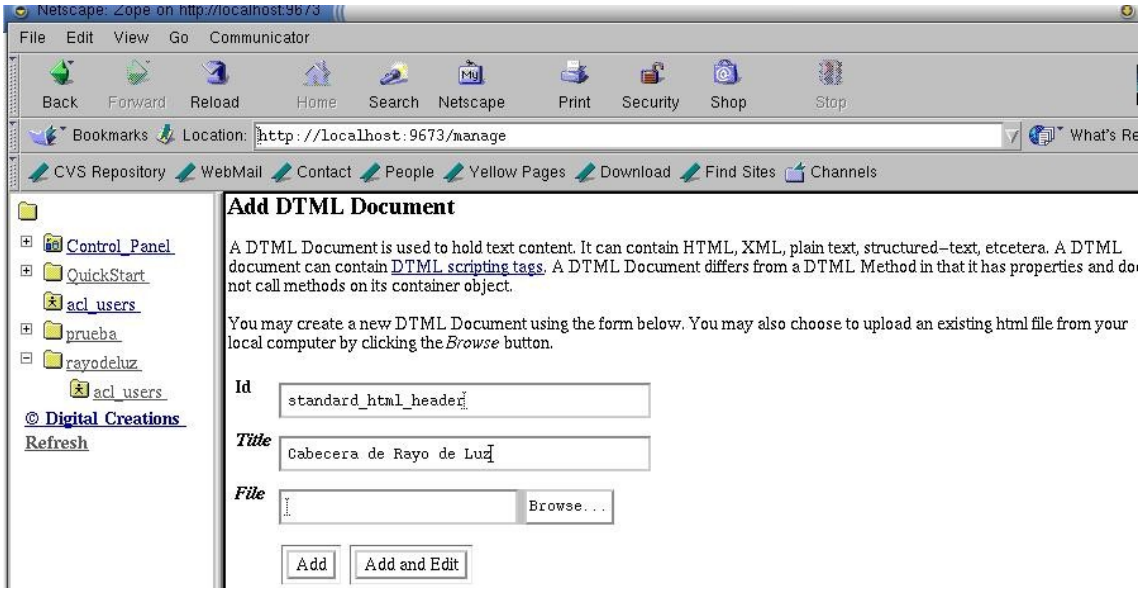


Este sencillo contenido es el que genera la página HTML siguiente cuando accedemos a la URL: http://localhost:9673/rayodeluz.

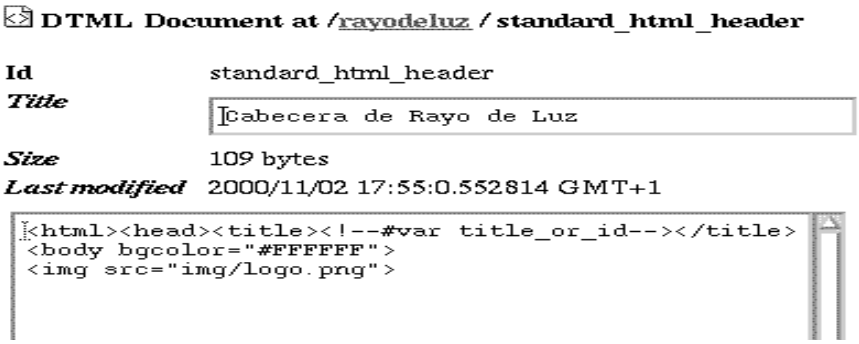


El contenido por defecto de "index_html" hace referencia a la una serie de variables DTML que definen la cabecera y pie de la página, y el título y el nombre del documento actual. Todas estas variables se buscan dentro del contenedor en el que esté el objeto "index_html", es este caso dentro de "rayodeluz". En nuestro caso todas estas variables no están definidas en este objeto por lo que ¿de dónde las saca Zope?.

Aquí ha salido a relucir uno de los mecanismos más potentes de Zope, mecanismo conocido como **adquisición**. Por este proceso, si una variable no se encuentra en el contenedor actual se busca en los contenedores superiores, hasta dar con su valor. De esta forma se pueden centralizar valores que son luego adquiridos por toda una jerarquía de objetos. Veremos que no sólo afecta este mecanismo a la variables, si no que se aplica en otras ocasiones, convirtiendo a la adquisición en uno de los mecanismos estrella de Zope. En nuestro caso la cabecera y pie por defecto, los que se incluyen en la carpeta raíz de Zope, no nos valen, por lo que nos vamos a crear dos objetos Documento DTML cuyos identificadores sean "standard_html_header" y "standard_html_footer" que serán utilizados por todas las páginas HTML del web de nuestra compañía. Vamos con ello.

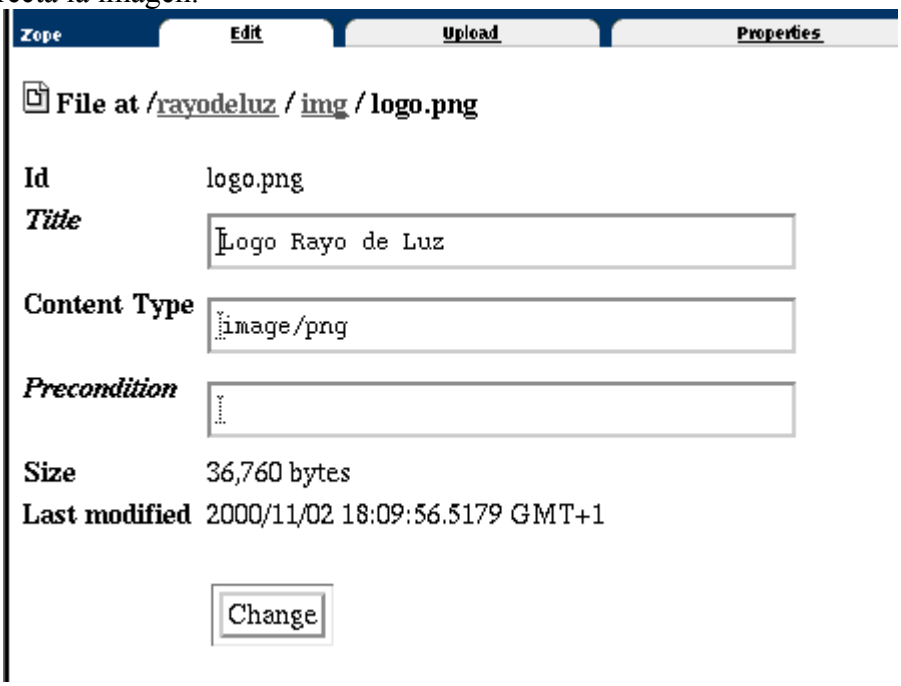


Ya tenemos creado el nuevo objeto que servirá de cabecera para todas las páginas que creemos como parte del web de la compañía. En general, todas estas páginas serán documentos DTML aunque cuando sea necesaria más interactividad, por ejemplo a través del intercambio de información entre el cliente web y Zope, utilizaremos otras alternativas. El contenido de esta nueva cabecera es el siguiente:

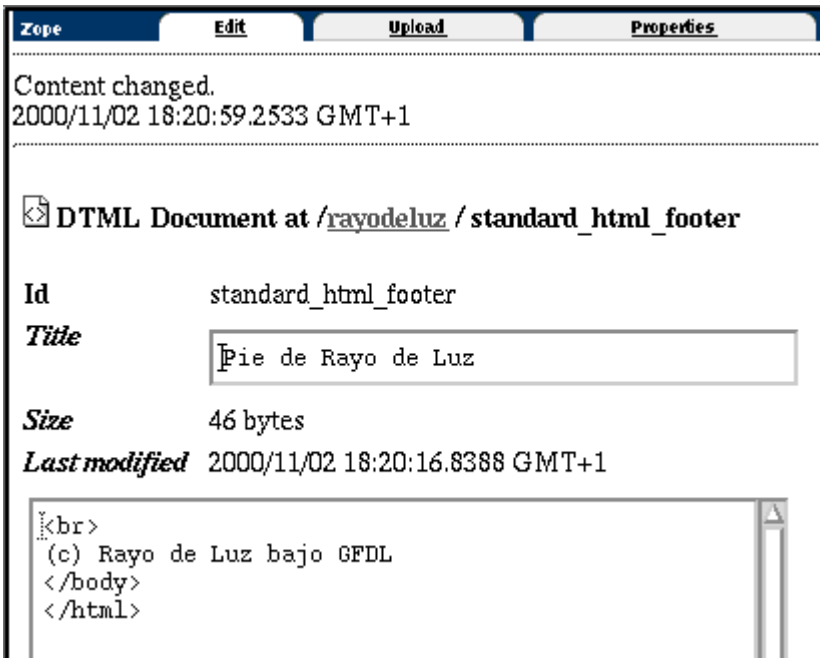


Como vemos, utilizamos la variable "var title_or_id" para que la todas las páginas tengan como título HTML el título que se le de al objeto DTML en el que residen. Tras ello definimos un BODY HTML con un fondo blanco e incluimos una imagen como logo. Pero aquí tenemos un problema. Este imagen es un fichero real, no un objeto Zope y sin embargo, sólo podemos hacer referencia a objetos que residan dentro de Zope para construir la página.

Para resolver este inconveniente tenemos la posibilidad de importar imágenes en Zope, que pasarán a ser objetos disponibles para todos los demás objetos de Zope. Para añadir esta imagen lo primero que hacemos es crearnos una carpeta para almacenar todas las imágenes y tras ello, nos metemos en esa carpeta, a la que hemos llamado "img", y añadimos un nuevo objeto de tipo "File". Como identificador del objeto tenemos que dar el nombre que utilizamos para referirnos a esta imagen dentro de las páginas de Zope, y el título podemos elegir el que queramos. Pulsando el botón de "Browse" podemos navegar por nuestro sistema de ficheros local a la búsqueda de la imagen. Una vez localizada pulsamos sobre el botón "Add" y ya tenemos dentro de Zope la imagen disponible. Podemos ver que una vez dentro de Zope, este automáticamente le asigna un tipo MIME y podremos visualizar la imagen accediendo a la opción de "View" presente en todos los objetos. Si ahora probamos a visualizar la cabecera veremos que ya se muestra de forma correcta la imagen.

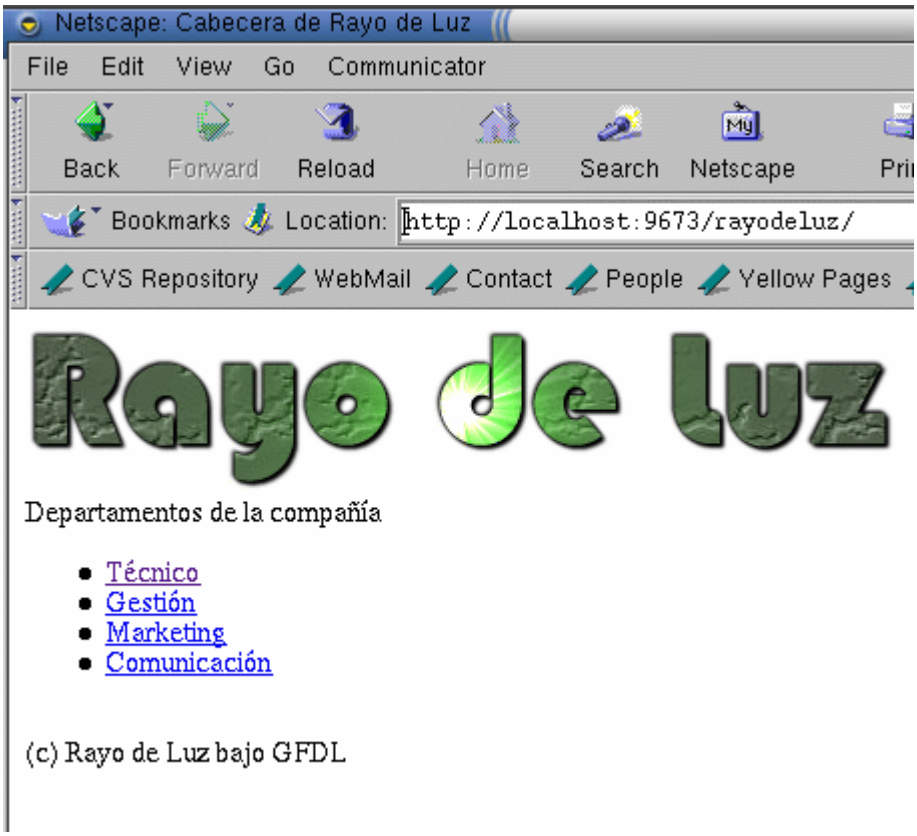


A lo largo de todo el apartado anterior hemos sentado las bases de la construcción de nuestro web con Zope. Si el lector ha ido leyendo con atención, a estas alturas ya estará en disposición de avanzar muy rápido con la creación del web con Zope. El siguiente paso es modificar el pie que va a aparecer en todas las páginas que cuelguen de la carpeta "rayodeluz", algo posible gracias al mecanismo de adquisición. Esta modificación es totalmente paralela a la de la cabecera. Hay que crear un nuevo documento DTML dentro de "rayodeluz" con el nombre "standard_html_footer". Este HTML será incluido de forma automática en todos los documentos DTML nuevos, y estos documentos son los que constituyen las páginas web del sitio.



Bueno, pues ya hemos logrado construir las cabeceras y pies de página para todas las páginas presentes y futuras del web. Sólo nos queda meter dentro de "index_html" los enlaces a las distintas secciones de la compañía. Para ello nada mejor que una simple lista HTML con enlaces a los distintos directorios de cada departamento. Mostremos de nuevo como se inserta este HTML. A partir de este momento ya daremos por supuesto que el lector se maneja con la interfaz de Zope y sabe como editar los objetos y su contenido.

Este contenido nos genera al fin la página principal de la empresa. Es un poco sencilla pero no olvidemos que nosotros somos simples técnicos. Ya se encargará nuestro departamento de diseño de llevarse las manos a la cabeza cuando vea la página y hacer una maqueta que deja boquiabiertos a los navegantes de las páginas web de la compañía. Y ese esquema lo podremos introducir dentro de la cabecera estándar y pasar a ser utilizado de forma automática por todas las páginas que cuelguen de "rayoluz". Un trabajo de días que nosotros podremos hacer en unas pocas horas y podernos dedicar a descansar en el demás tiempo.



Control de acceso

Bueno, ya llegado el momento de comenzar a crear las páginas del departamento técnico. En ellas existirá información pública para todos los empleados de la empresa como los pasos para configurar el acceso a Internet de los computadores, las instrucciones básicas de mantenimiento de los equipos, las normas básicas de navegación por Internet y uso del correo electrónico y los correos y teléfonos de contacto de las personas del departamento. Pero no toda la información que va a estar dentro de nuestro departamento debe ser pública. Los datos sobre las claves de acceso a las máquinas como administrador, la gestión de IP de los equipos y toda la biblioteca de documentos técnicos, así como un foro de discusión interno, serán partes privadas.

Creamos una nueva carpeta para incluir las páginas del departamento técnico, y como identificador utilizamos "tecnico" que es el nombre que dimos como enlace en la página principal. Dentro de esta carpeta modificamos el objeto "index_html" para reflejar la información del departamento y ponemos enlaces a distintas páginas donde se amplía la información. Una vez hecho esto, vamos a crear una carpeta llamada "interno" donde vamos a incluir toda la información que debe ser sólo accedida por las personas de nuestro departamento.

El acceso a esta carpeta llamada "interno" queremos que sea restringido. Para ello nos vamos a dicha carpeta y seleccionamos la solapa "Security" donde se definen las políticas de seguridad de acceso a los distintos objetos de esta carpeta. En general estos permisos se "adquieren" de la carpeta superior. De nuevo sale a la palestra el mecanismo de adquisición: si algo no existe en el contenedor actual, se viaja a los contenedores superiores en la jerarquía buscándolo. De esta forma podemos tener centralizada la seguridad de todo el sitio en la carpeta raíz. Veamos los permisos que vienen por defecto en la carpeta raíz, y que son los que heredan todos los nuevos contenedores que vayamos creando.

Import/Export objects	☐	☐	☐
Join/leave Versions	☐	☐	☐
Manage ZCatalog Entries	☐	☐	☐
Manage properties	☐	☐	☐
Manage users	☐	☐	☐
Open/Close Database Connections	☐	☐	☐
Save/discard Version changes	☐	☐	☐
Search ZCatalog	☐	☐	☐
Undo changes	☐	☐	☐
Use Database Methods	☐	☐	☐
Use mailhost services	☐	☐	☐
View	☐	☐	☐
View management screens	☐	☐	☐

Change

En estas imagen aparecen sólo parte de los permisos, pero son los que queremos destacar en este instante. La primera columna se refiere al usuario anónimo. Si aparece seleccionada esta opción, un usuario sin autenticarse será capaz de realizar esa operación. Por ejemplo, tenemos seleccionado "View" por lo que cualquier usuario podrá ver todas las páginas DTML por defecto. También podrá utilizar el objeto "Z Search Catalog" o utilizar métodos para acceder a las bases de datos. Los permisos que aquí aparecen se pueden ampliar y se pueden crear nuevos roles, a parte del de usuarios anónimo, el de administrador y el de dueño del recurso.

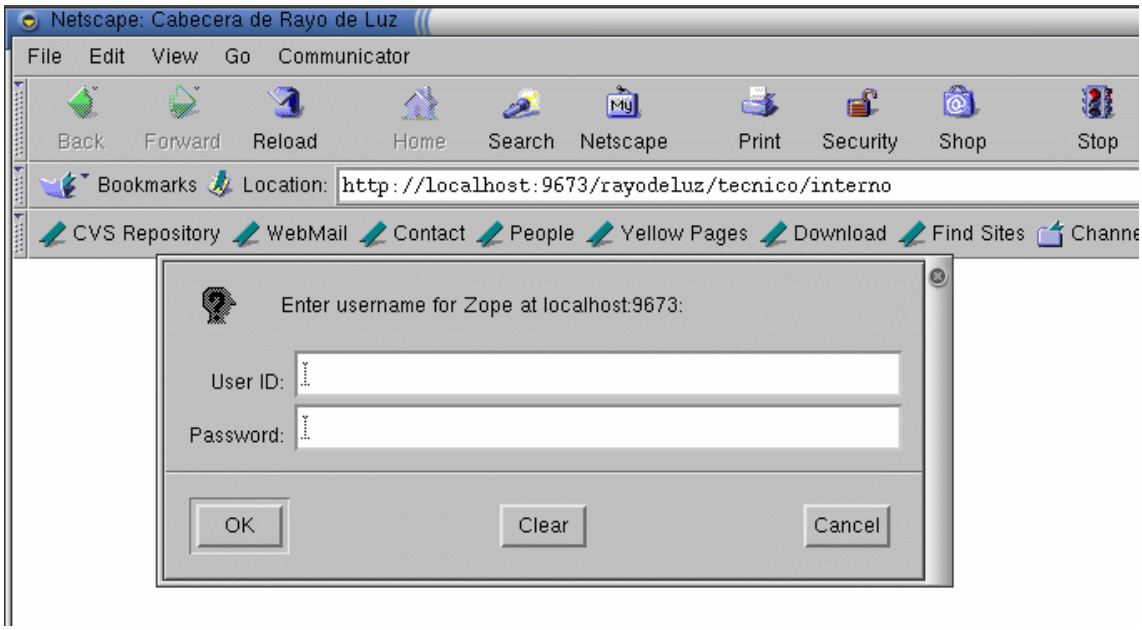
Vemos que por ejemplo el usuario anónimo no tiene acceso a las pantallas de gestión de Zope, algo normal ya que desde allí se puede modificar fácilmente todo el web.

Nuestro objetivo es que para entrar en el directorio "interno" del la parte de departamento técnico, sea necesario autenticarse en el sistema. Para ello nos vamos a dicha carpeta, accedemos a la solapa de seguridad y modificamos la operación de "View" para que no se adquiera. De esta forma sólo podrán ejecutar esa operación los usuarios que tengan el rol de administradores.

Import/Export objects
Join/leave Versions
Manage ZCatalog Entries
Manage properties
Manage users
Open/Close Database Connections
Save/discard Version changes
Search ZCatalog
Undo changes
Use Database Methods
Use mailhost services
View
View management screens

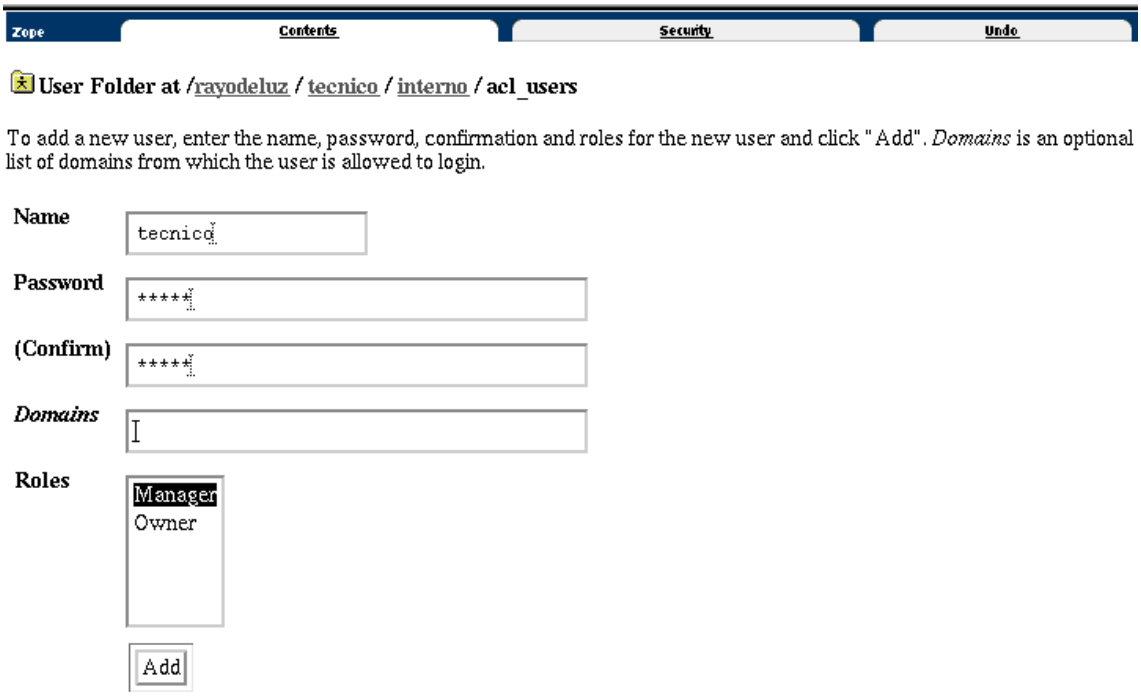
Change

Para probar que ahora se nos pide que nos autentiquemos, podemos apagar el navegador por completo e intentar acceder a dicho directorio. Si el lector ha metido una clave de administrador en algún momento, y esperamos que lo haya hecho para poder seguir el ejemplo, debe apagar el navegador para que deje de tener efecto esta clave para Zope.



¿De dónde consulta Zope este usuario que se nos pide? Recordemos que a la hora de crear una carpeta teníamos al opción de crear una carpeta de usuarios asociada a ella. El primer sitio en el que Zope va a buscar el usuario que se

introduzca en la caja de autenticación va a ser dentro de esta carpeta de usuarios "user_folder". Cuando se crea un usuario, se le asigna un rol. Es este rol el que define que puede y que no puede hacer un usuario. En este caso, para acceder a esta carpeta protegida, es necesario que el usuario tenga el rol de "manager". Vamos a crear un usuarios con estos permisos. Queremos que este usuario sólo tenga el rol de "manager" para esta carpeta en concreto. Por ello lo creamos en el "user_folder" de esta carpeta. Este usuario será "manager" en la carpeta actual y todas las que cuelguen de él, pero no lo será en carpetas superiores. Con este sistema tenemos una gran potencia a la hora de definir permisos a regiones concretas de tal forma que haya distintos administradores para cada zona del sitio.



Este nuevo usuario tendrá el rol de "manager" por lo que podrá acceder a la zona protegida y, más aún, podrá acceder a la interfaz de gestión y modificar el contenido de su zona desde la interafaz Zope.

Iremos viendo a lo largo del curso la enormes posibilidades que ofrecen estos mecanismos de gestión de usuarios que harán triviales las labores más complejas de control de acceso.

Ejemplos de uso

En los siguientes ejemplos se mostrará el uso del lenguaje que emplea Zope para la creación de páginas Web, el Document Template Markup Language (DTML). Para una mayor explotación de las posibilidades del servidor Zope habrá que hacer uso de Python.

1.- El contenido del documento es una mezcla de HTML y etiquetas especiales de DTML. La etiqueta <dtml-var> inserta 'cosas' dentro de páginas web. En esta página insertamos una cabecera, un pie (footer), una imagen, un comentario y un título con DTML.

```
<dtml-var standard_html_header>
<dtml-var header.gif> <p>
```

```
Hola Mundo <i><dtml-var title></i>,
Ejemplo de Hola Mundo en Zope
</p>
<dtml-comment>
Esto es un comentario y no se ejecutará
</dtml-comment>
<dtml-var standard_html_footer>
```

2.- Ejemplo del uso de sentencias condicionales con DTML.

```
<dtml-if expr="num > 5">
      El número es más grande que 5
<dtml-elif expr="num < 5">
      El número es más pequeño que 5
<dtml-else>
      El número es 5
</dtml-if>
```

3.- Envío de correo electrónico con DTML.

```
<dtml-sendmail>
To: <dtml-recipient>
Subject: Curriculum
<dtml-mime type="text/plain" encode="7bit">
Hola, por favor miren mi Curriculum
<dtml-boundary                    type="application/octet-stream"
disposition="attachment"
encode="base64" filename_expr="resume_file.getId()">
<dtml-var expr="resume_file.read()">
</dtml-mime>
</dtml-sendmail>
```

4.- Uso de sentencias SQL y DTMl en conjunción:

```
select * from employees
        <dtml-sqlgroup where>
        <dtml-sqltest salary op=gt type=float optional>
        <dtml-and>
        <dtml-sqltest first op=eq type=string multiple optional>
        <dtml-and>
        <dtml-sqltest last  op=eq type=string multiple optional>
        </dtml-sqlgroup>
```

5.- Uso de Cookies con DTML:

Para escribir cookies:

```
<dtml-call expr="RESPONSE.setCookie('lastVisited', ZopeTime())">
```

Para leerlas:

```
<dtml-if expr="bobobase_modification_time() >
ZopeTime(lastVisited)">
```

PMZ (Poor's Man Zope)

Introducción

Con Python se pueden crear páginas activas muy al estilo de PHP o ASP, de modo muy simple. Vamos a ver PMZ (Poor's Man Zope), una alternativa a otros sistemas más complejos como Zope.

Utilidades de PMZ

PMZ es un sistema simple (tan sólo 5 kb. De código Python) que nos permite empotrar código Python en medio de una página web. Esto nos permite hacer páginas activas de un modo realmente simple: las partes invariables de la página se ponen tal cual, en código HTML, pero si deseamos meter algo de código Python para realizar alguna tarea, como acceso a una base de datos, generación dinámica de algún texto o código HTML, simplemente se engloba el código en una etiqueta XML.

PMZ es, en principio, un pequeño juguete que nos permite lo que se puede hacer con Python, aunque en realidad nos puede servir para crear todo tipo de páginas activas sin demasiados problemas de configuraciones complicadas, ni crear una gran cantidad de código para un pequeño resultado.

Para incluir código Python debemos ponerlo entre las etiquetas <pmz> y </pmz>. Además para asegurar la compatibilidad con el estándar XML, se permite que el código esté dentro de la etiqueta <[¡CDATA[(código python]]>, de modo que un parser normal compatible con el estándar XML no encuentre problemas por culpa del código Python.

PMZ vs Zope

Zope es un sistema que también usa Python para crear páginas activas, pero de un modo mucho más complejo, aunque más potente. La diferencia radica en que, por un lado, PMZ se basa en Apache para servir las páginas, mientras que Zope crea su propio servidor web (que puede convivir perfectamente con Apache). Además, PMZ incluye directamente el código que va a ejecutar; Zope se basa en un uso intensivo de etiquetas que referencia o bien a otros trozos de código DTML (que puede tener referencias Zope) o a pequeños programas hechos en Python.

PMZ está más indicado para pequeñas aplicaciones que no requieren gran complejidad, mientras que Zope se usa más para grandes proyectos, ya que proporciona herramientas que sirven desde unificar de un modo simple la apariencia de todo un gran portal, hasta reutilizar el código Python.

Zope está licenciado con la ZPL (Zope Public License), que intenta mantener el espíritu Open Source; aún así no es compatible con la GPL, aunque la propia Free Software Foundation no desanconseja el uso de Zope por su licencia, pero si la creación de nuevo software bajo ella.

Cómo funciona PMZ

PMZ se basa en la idea de que gracias al módulo *action* del Apache, se pueden asociar un tipo de controladores (handlers) de Apache a un cierto script cgi, en este caso *pmz.py*. De modo que, con crear el handler asociado a la extensión *.pmz* y el handler al *pmz.py*, al script pmz.py se le pasa el nombre del otro archivo, con su camino absoluto en nuestro árbol de directorios. En este momento, *pmz.py* se encarga ya de todo el control, se encarga de crear una serie de restricciones de ejecución de los scripts, basados en la localización y especificados en el propio

script, en el diccionario _settings. Este diccionario contiene por un lado los directorios, con expresiones regulares (por ejemplo .* al final para que sea cualquier cosa desde ese punto) y el tipo de restricción que tiene:

['_rexec=0'] para ninguna restricción y

['_rexec=1'] para ejecución restringida.

Cuando se está en modo no restringido se puede hacer cualquier cosa desde nuestro Python, como manejar sockets. Esta será nuestra opción más común si vamos a crear páginas realmente dinámicas, pero debemos tener mucho cuidado de no creer en un script PMZ de un modo absoluto si no queremos comprometer la seguridad de nuestro servidor. En modo restringido no se pueden cargar ciertos módulos que no se consideran seguros (el módulo *rexec* determina la ejecución segura). Además, en modo restringido no se heredan ciertas variables que nos podrían interesar para crear páginas realmente seguras.

PMZ utiliza varios módulos, entre los cuales quizás se debería destacar el cgi. Este módulo nos facilita la creación de scripts CGI de todo tipo, proporcionándonos funciones que unifican todos los métodos de paso de variables a los scripts, ya sea por shell, o como sea. PMZ lo usa para proporcionarnos la variable CGI_VARS con todos los parámetros que puede recibir nuestro script *.pmz*. Para ello se suele usar la clase *FieldStorage*, que se encarga de leer los parámetros de las variables de entorno o bien de la entrada estándar (por ello no se debe inicializar más de una vez). Una vez tengamos una instancia de esta clase, podemos acceder a sus elementos como si fuera un diccionario. El módulo *re* se utiliza para control de expresiones regulares, como comprobar si una cadena se ajusta a una expresión regular. Este módulo lo usa PMZ para comprobar la ejecución restringida basándose en políticas de ubicación de expresiones regulares. Las expresiones regulares se escriben al más puro estilo UNIX; en la siguiente tabla se explican:

Expresiones regulares para el módulo *re*	
.	Cualquier carácter.
^	Comienzo de línea.
$	Final de línea.
*	0 ó más repeticiones del elemento anterior.
+	1 ó más repeticiones del elemento anterior.
¿	0 ó 1 repetición.
\	Deja escapar un carácter especial.
[]	Engloba un conjunto de caracteres.
|	Deja elección entre la parte anterior y posterior al símbolo.
NOTA: hay más expresiones, pero estas son las más comunes.	

Este módulo puede hacer dos cosas bien distintas: buscar una cadena o comprobar si coincide. Si se busca coincidencia ésta es desde el comienzo de la cadena, mientras que si se busca puede aparecer en cualquier posición. Además, se puede compilar la expresión regular para que búsquedas sucesivas sobre la misma expresión regular sean más rápidas. Para buscar expresiones se utiliza *re.search(regex, cadena [,flags])* y para coincidencias se usa *re.match(regex, cadena [,flags])*.

Gracias al módulo *rexec*, se pueden ejecutar scripts Python dentro de un entorno restringido. En este entorno hay cosas que no se pueden hacer, como cargar ciertos módulos. En el caso de PMZ se encarga de las restricciones básicas: impide la lectura y escritura en disco y el uso de sockets. Hay que tener cuidado de no cargar

módulos que de un modo indirecto pretendan usar sockets (como hacen los módulos CORBA, el módulo cgi y cualquiera relacionado con las redes) o leer/escribir en disco, ya que si aparecen tendremos un precioso traceback mostrándonos dónde está el error. En la ejecución restringida además, podemos elegir qué módulos pueden ser cargados y cuáles no, dónde puede buscar los módulos que se deseen cargar e incluso, qué objetos de módulos como *os* y *sys* estarán disponibles.

Con el módulo traceback podemos tener más control sobre los errores de ejecución que se den. PMZ lo usa para mostrar en la página que está siendo visualizada y, en rojo, el error que se da en el código PMZ, sin parar la ejecución del todo. En particular, usa la función *print_exc([limit[,file]])* para desviar el flujo de información del traceback a la salida estándar.

String proporciona ciertos servicios para el control de cadenas como división en una lista señalando un separador (split(string[,sep[,maxsplit]])), unir una lista de cadenas en una sola cadena(join(lista[,sep])) o tareas más terrenales, como poner una cadena entera en minúsculas o mayúsculas o intercambiar las letras mayúsculas por minúsculas o viceversa,... este módulo es muy útil en todo tipo de tareas.

PMZ usa también, de pasada, el módulo *exceptions*, donde se definen todas las excepciones.

Creación de scripts CGI

En realidad, nuestro archivo no es un CGI propiamente dicho, ya que reside en cualquier sitio de nuestro árbol compartido por web y no en el directorio CGI. Además, su ejecución se basa en la de un auténtico script CGI, *pmz.py*.

Si estamos en modo no restringido, hemos heredado una variable CGI_VARS, que es un diccionario con pares del tipo 'variable':'valor'. Si estamos en modo restringido, lo más que podemos hacer es cargar el módulo *os*; dentro del diccionario *environ* hay una clave, QUERY_STRING, que contiene los datos que se le piden al script, pero no formateados, con lo cual cada uno debe encargarse de ello.

Conclusión

Aunque PMZ no está indicado para sitios con una carga elevada, según indica el propio autor, Andreas Jung, se puede utilizar en sitios pequeños, donde la complejidad de Zope no sea necesaria. En cualquier caso, si se desea usar de un modo abusivo, se pueden utilizar trucos como compilar a bytecodes el propio *pmz.py* o para casos más extremos, pasar el código a lenguajes como C.

Si nuestro sitio en cambio, no necesita la complejidad de Zope o solamente necesitamos scripts en un par de sitios y el consumo de CPU no se espera muy elevado, entonces PMZ es nuestra mejor opción, ya que su uso es extremadamente fácil.

En cuanto al futuro de PMZ, se prevé que pueda usar otros interfaces CGI's como Persistent CGI y/o Fast CGI. Además, poco a poco se van arreglando pequeños bugs que aparecen.

Instalación de PMZ

Instalar PMZ con Apache es muy simple, sólo hay que copiar el script *pmz.py* en el directorio cgi-bin y añadir dos líneas en el archivo de configuración del Apache:

AddHandler python-parsed-html .pmz
Action python-parsed-html /cgi-bin/pmz.py

Además, debemos asegurarnos que Apache carga el módulo dinámico *action_module*; en los sistemas UN*X, y con Apache 1.3; esto se hace poniendo la siguiente línea en el archivo de configuración /etc/Apache/httpd.conf:

LoadModule action_module /usr/lib/Apache/1.3/mod_actions.so

Si nuestro servidor web no es Apache, debemos buscar unas opciones de configuración tales que nos permitan asociar la extensión pmz con el script *pmz.py*, y que pmz.py reciba como parámetro el nombre del archivo .pmz. En este momento, debemos configurar mínimamente el *pmz.py,* en concreto la variable _settings, que indica el tipo de ejecución que tienen determinados directorios (expresados como expresiones regulares). Un ejemplo del contenido de esta variable podría ser este:

_settings = {
'/var/www/pmz/.':['_rexec=1'], # ejecución restringida*
'/var/www/pmz/trusted/.':['_rexec=0'], # ejecución no restringida*
}

Cualquier directorio que no aparezca en este diccionario, no tendrá permitida ningún tipo de ejecución.

Ejemplos de uso

1.- Página PMZ que muestra en una página web una lista con 100 números.

```
<!— Archivo n100.pmz  genera 100 números -->
<HTML>
<HEAD>
      <TITLE>100 números</TITLE>
</HEAD>
<BODY><pmz>
      for i in range (100):
            print i
</pmz></BODY>
</HTML>
```

En este caso no hemos optado por usar la etiqueta de compatibilidad <¡[CDATA[]]>, ya que es opcional. El código Python se debe hacer como normalmente; se debe respetar la identación ya que es la que marca los bloques.

2.- Página HTML que contiene un formulario que llamará a nuestro script *respuesta.pmz* y realizará una serie de tareas.

```
<!—Archivo Formulario.html -->
<HTML>
<HEAD>
      <TITLE>Formulario simple</TITLE>
</HEAD>
<BODY>
<FORM action="respuesta.pmz" method="POST">
<INPUT TYPE="text" SIZE="12" NAME="texto" VALUE=" "><BR>
<INPUT TYPE=2submit" NAME="ok" VALUE="ok">
</FORM>
</BODY>
```

```
</HTML>
```

El formulario utiliza el método POST para enviar los datos; si quisiéramos usar el método GET, sería exactamente igual la creación de nuestro script, usando el contenido de la variable CGI_VARS o mirando en las variables de entorno si estamos en un entorno restringido.

```
<!—Archivo respuesta.pmz -->
<HTML>
<HEAD>
      <TITLE>Respuesta</TITLE>
</HEAD>
<BODY>
El usuario a respondido '
<pmz>
<¡[CDATA[
print CGI_VARS['texto']
]]>
</pmz>
' que tiene
<pmz>
<¡[CDATA[
print len(CGI_VARS['texto'])
]]>
</pmz>
caracteres.
</BODY>
</HTML>
```

El archivo respuesta.pmz se limita a visualizar el valor del campo de texto introducido en el formulario y el número de caracteres que contiene.

Referencias

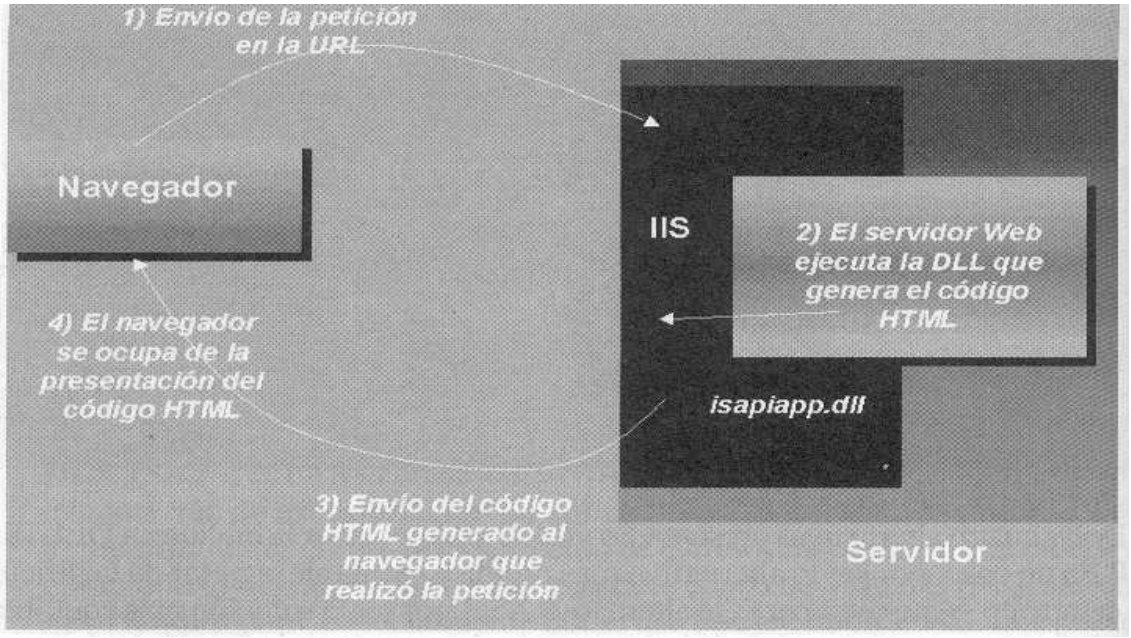
http://pmz.sourceforge.net
Abundante información del proyecto PMZ.
http://www.python.org
Python.
http://www.pythonlabs.com
Pythonlabs.
http://python.sourceforge.net
Pythondev.

Otras tecnologías

DLL ISAPI

Es un mecanismo nativo de IIS; es la biblioteca de Internet Server o la interfaz de programación de aplicaciones de Internet Server (API de IS). Tiene similitudes con el procesamiento de CGI y también se parece en el nivel de complejidad del desarrollo ya que requiere el uso de Visual C++. Tiene un muy buen rendimiento, especialmente por el uso de ejecutables DLL que pueden ser reentrantes. Varios usuarios pueden compartir el mismo ejemplar (instancia) de la biblioteca DLL, lo qie permite una mayor escalabilidad, si comparamos esta solución con la alternativa CGI.

En el siguiente gráfico se esquematiza el funcionamiento interno de una aplicación Web basada en una biblioteca DLL ISAPI:



1. Todo comienza con una petición o requerimiento HTTP iniciado en un navegador por un cliente que quiere acceder a un recurso de nuestro sitio Web por medio de una dirección URL. La dirección URL apunta a la localización física de una biblioteca de enlace dinámico (DLL).
2. En esta DLL está empaquetada la lógica de negocio que se haya programado con un lenguaje como Visual C++ para dar respuesta a la petición del usuario. Por ejemplo, dentro de la DLL se pueden incluir las funciones para validar al usuario, para accederA una base de datos por medio de ADO y para preparar la respuesta en formato HTML que se enviará al usuario.
3. IIS, es decir, el servidor Web es el que se encarga del anvío de la respuesta que le preparó la DLL al navegador que había realizado la petición.
4. El navegador es el encargado de la visualización del código HTML que recibe a modo de respuesta de la petición.

Max Server Pages

Max DOS32/W está inspirado en productos Xbase tales como Clipper, FoxPro y dBASE. Te permite compilar y ejecutar código Xbase en plataforma Windows. Aunque también se puede recompilar para Linux usando Max for Linux. Incluye soporte ODBC y acceso a a archivos de daros de FoxPro, dBASE y Clipper. Max también soporta páginas web dinámicas usando un modulo aparte, el Max Server Pages en el servidor web de Linux o NT.

Con la opción de compilación /msp se genera código para usar Max Server Pages (poca más información se puede obtener de la documentación que se ofrece con el compilador).

PHPNuke 5.0

No es una tecnología sino una herramienta para generar portales. Es de los mejores software del mundo GPL (patrocinado por Mandrake). Los requerimientos para instalar el PHPNuke 5.0 son los siguientes:

- PHP 4.X.X.
- MySQL 3.2X.XX
- Apache 1.3.X con el módulo PHP 4 con soporte MySQL.

Naturalmente, PHPNuke estan multiplataforma como los requerimientos anteriores. Podemos encontrar PHPNuke en plataformas Solaris, Windows NT y Linux.

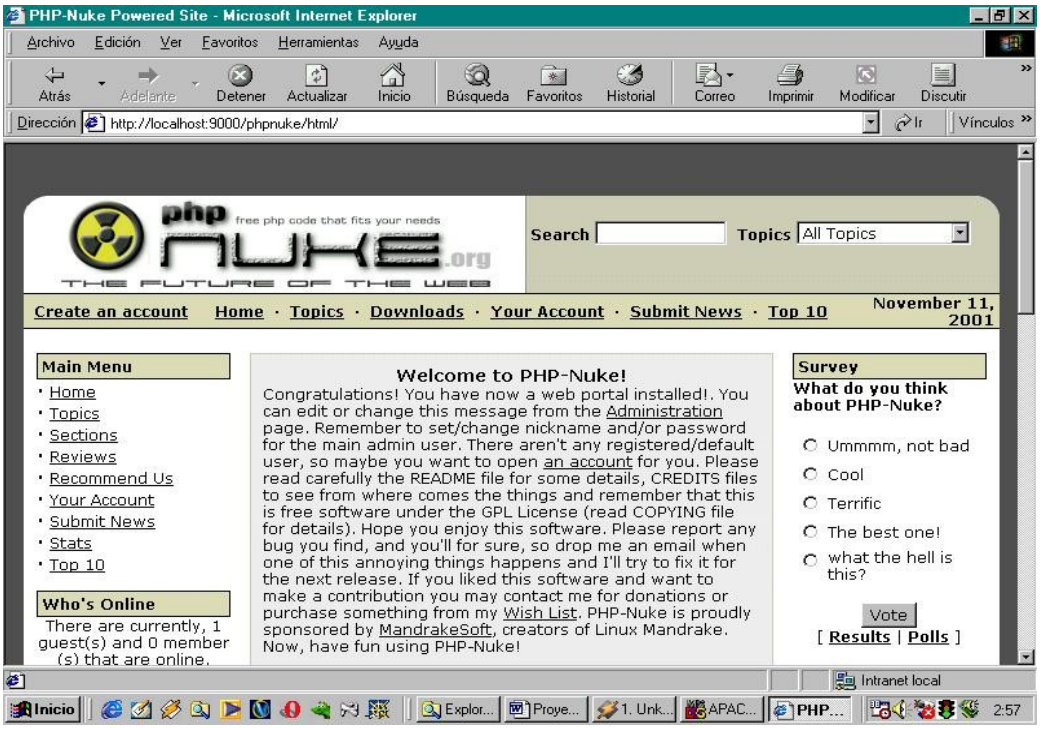
PHPNuke no ofrece ningún servicio que no hallamos visto en la mayoría de portales: noticias, buscador, encuestas, etc. La ventaja que tiene es que ofrece todo esto de forma integrada. Instalando un sólo software tiene todos los servicios que puede necesitar cualquier web master en su portal. Con PHPNuke no hace falta instalar un paquete para cada cosa. No sólo incorpora todo tipo de servicios, sino que además todo es administrable por web.

- Los servicios que incluye son:
- Sección de artículos (Stories).
- Sección de encuestas (Polls).
- Sección de preguntas y respuestas (FAQs).
- Directorio tipo Yahoo (Web links).
- Buscador.
- Sección de Downloads.
- Secciones especiales (artículos fuera de la Home).
- Secciones de temas (Topics). Sirven para clasificar los artículos. Puedes acceder a los artículos por tema.
- Gestión de usuarios registrados (Accounts). Los usuarios pueden personalizar la página.

Todo es administrable desde la Web, incluso lleva un módulo para actualizar los ficheros de la web a través del navegador. Como dato curioso PHPNuke se puede configurar para que el site muestre todas las secciones en gallego.

Es considerada una herramienta para webmasters ‘vagos’ ya que no hay que programar nada.

Si observamos un site con la siguiente apariencia podemos sospechar que se utilizó esta herramienta:



Server Side Includes

Server-side scripting es un camino para conseguir contenido dinámico. Un buen ejemplo de esto es un contador de visitas, o la hora y fecha actual.

Server-side scripting no es JavaScript, la cual es una clase de 'client-side scripting'. SSI (Server Side Includes) son directivas que son colocadas en páginas HTML, y evaluadas en el servidor mientras las páginas están siendo servidas. Ellas permiten generar contenido dinámico, sin que se tenga que servir la página via programa CGI, o otra tecnología dinámica.

A través de un server-side scripting, podemos:

- Insertar el contenido de la última modificación de un fichero.
- Insertar el contenido de otro fichero HTML.
- Usar un documento HTML para producir dos o más páginas web
- Crear formularios de retroalimentación que permite conseguir e-mail directamente desde el web.
- Acceder y modificar bases de datos

Todas las directivas SSI son formateadas como comentarios SGML dentro de un documento HTML y así queda como sigue:

```
<!--#command tag1="value1" tag2="value2" -->
```

Algunos ejemplos de SSI:

Visualiza la fecha local sin ningún formateo:

```
<!--#echo var="DATE_LOCAL" -->
```

Fecha formateada:

```
<!--#config timefmt="%A %B %d, %Y" -->
Hoy es <!--#echo var="DATE_LOCAL" -->
```

Incluir el resultado de un CGI

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

Ejecutando comandos:

```
<pre>
      <!--#exec cmd="ls" -->
</pre>
<pre>
        <!--#exec cmd="dir" -->
</pre>
```

Conclusiones

Ante la variada oferta de tecnologías para generar contenido dinámico cabe plantearse la siguiente pregunta: "¿Cuál escoger?"

Hay varios condicionantes a tener en cuenta:

El primero es ¿qué se pretende hacer?. En error sería pretender generar gráficos a consultas de usuarios mediante CGIs en Perl, teniendo a ColdFusion que los realiza de forma automática y con una amplia variedad. O por el contrario, generar pequeños Scripts con ColdFusion pudiéndolos hacer en Perl y ofreciendo el mismo rendimiento y menor coste monetario.

¿Qué condicionamientos tengo a nivel de SO?. Estaríamos muy condicionados si trabajasemos con ASP en Linux. Se puede trabajar pero no dispondríamos del poder de los ActiveX que es la gran fuerza de ASP (Windows), entre algunos ActiveX cabe descatar ADO (para el acceso a bases de datos). Si nuestro SO es Windows 2000 con ASP podrías tener una integración perfecta.

¿Qué lenguaje de programación utilizar?. Esta respuesta está condicionada a los conocimientos del programador. Trabajar con CGIs en Perl o C, necesita de un conocimiento del lenguaje (Perl o C) bastante a fondo si se pretende hacer una buena aplicación. Y a parte de los conocimientos del lenguaje necesitaríamos conocer el protocolo HTTP, y demás dependiendo de lo que se pretenda realizar.

Java (Servlets y JSP) es la solución más rentable en este caso, puesto que cuenta con infinidad de clases para realizar los trabajos más costosos. Cuenta con otras tecnologías Java de apoyo como son RMI, JDBC, JavaMail, EJBs, CORBA,...

¿A cuantas peticiones tengo que responder?¿Cuál es la carga que va a sufrir mi site?. Esta respuesta viene muy ligada al Hardware del que se dispone. Si tenemos una granja de servidores para repartir la carga tenemos el problema típico de las variables session y la posible interconexión con los demás servidores. Tendríamos que echar un vistazo a ASP.NET que resuelve este problema.

¿Cuánto tiempo tenemos para construir nuestro "portal"?¿Cuánto presupuesto?. En el caso más desfavorable ante estas preguntas me atrevo a recomendar PHPNuke. Es una herramienta basada en PHP, MySQL y Apache todos de libre distribución y multiplataforma. En caso de tener presupuesto y tiempo la respuesta a esta pregunta estaría ligada a la primera.

¿Hay que interconectar con software existente?. Si el software es Microsoft la solución pasa por ASP, puesto que tiene muy fácil integración (es todo lo mismo). No se excluye a otras tecnologías ante esta afirmación. Si el software del que estamos hablando no es Microsoft las principales tecnologías disponen de mecanismos para poder interconectarse.

¿La tecnología que utilizo es multiplataforma?. Aquí es donde cojea ASP y salen a flote soluciones como Servlets, JSP, SSJS o PHP. Los bytecodes Java como SSJS dotan de la capacidad de ser multiplataforma a las aplicaciones que se desarrollan con ellas.

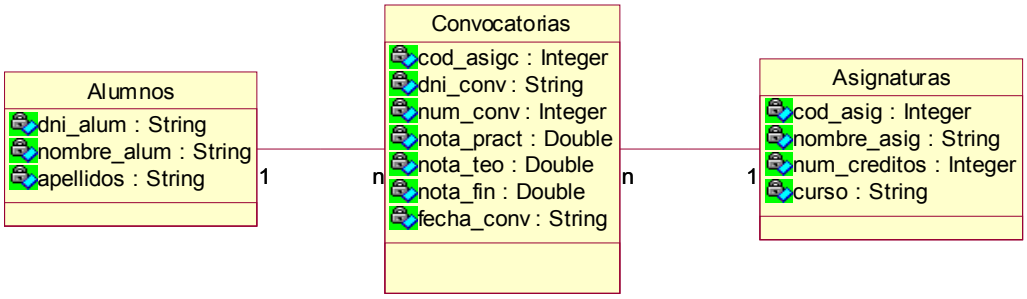
La respuesta a todas las demás preguntas que puedan surgir es: DEPENDE.

Personalmente si se tiene un SO Windows 2000, una buena solución sería ASP.NET (véase el apéndice de ASP.NET) y la construcción de componentes ASP con C++, Delphi, Java, Visual Basic,... que nos permitirán trabajar por ejemplo con la suite Office sin ningún problema. En un SO Linux, PHP o Java (Servlets/JSP) es una solución muy socorrida y no por ello menos potente. Y no por esto olvidar que podemos integrar las soluciones dependiendo de lo que queramos hacer. Podemos tener en un IIS (con ColdFusion) páginas ASP y CFM y funcionar todo en el mismo puerto y mismo dominio. La elección no es tanto de la tecnología a utilizar sino del SO, Hardware, ...

ANEXO A. Ejemplo con las diversas tecnologías

Explicación del ejemplo:

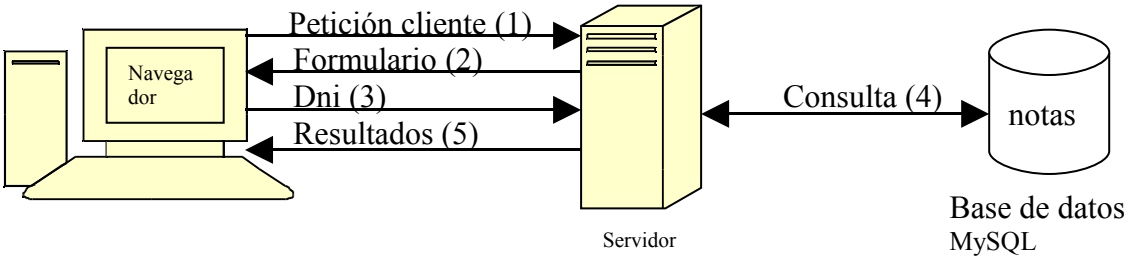
En el ejemplo implementado con las diversas tecnologías se utilizará una base de datos "notas" en MySQL con la siguiente estructura:



La estructura de los ejemplos es la siguiente:

El cliente solicita una página html (1) en la que tenemos un formulario con un campo de texto y un botón de submit(2). En el campo de texto introduciremos el dni del interesado en saber sus notas(3). Tras el envío del dni, el servidor (PWS, Apache, Tomcat, JWS, Xitami,... según el caso) realiza la consulta a la base de datos (4) y devuelve la respuesta con los datos referentes al alumno (5).

Con la siguiente figura mostramos el proceso:

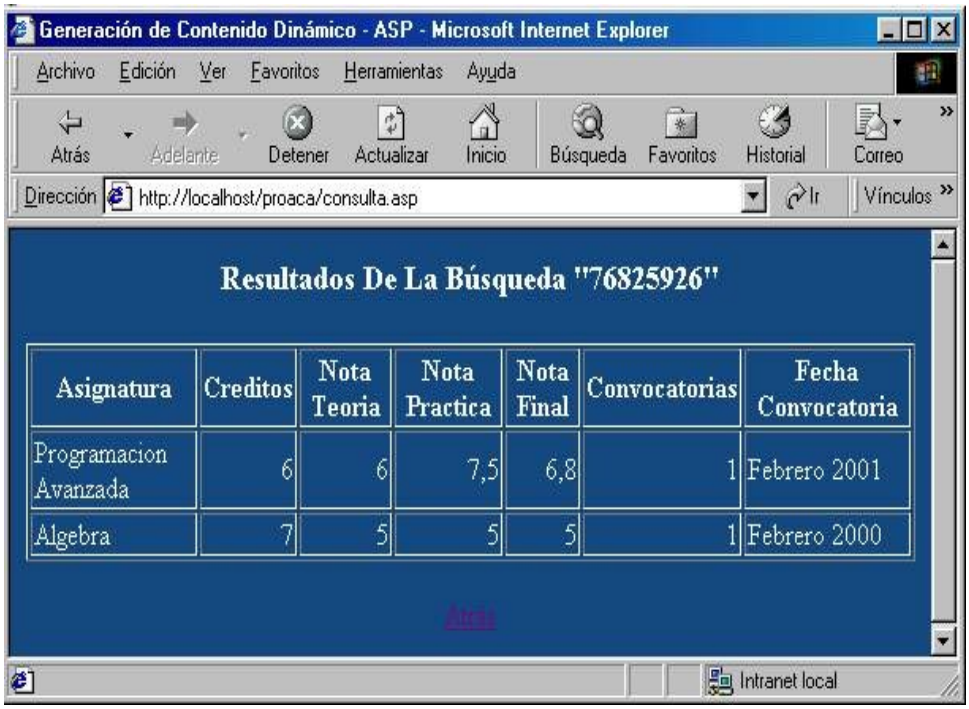


El paso (2) devuelve un formulario al cliente que para todos los ejemplos es el mismo salvo en una línea. En esta línea es donde se hace referencia al script/servlet/... que responderá con la calificación del alumno. El archivo HTML base sería el siguiente (veáse la línea a cambiar en cada caso):

```
<HTML>
<HEAD>
      <TITLE>Generación Dinámica Contenido - Tecnología </TITLE>
</HEAD>
<BODY bgcolor="#1A4B85">
<center>
<FORM  ACTION="SCRIPT"  METHOD=POST>
Introduce el dni:
<INPUT TYPE="text" Name="Dni" Size="10"><br>
<INPUT  TYPE="Submit" Value="Enviar">
</FORM>
</center>
</BODY>
</HTML>
```

Este archivo . html se corresponde con la siguiente imagen:

Y la respuesta que obtenemos dependiendo de cada ejemplo será similar a la siguiente imagen:



Para trabajar con MySQL se utilizó MySQL-Front (incluido en el CD), programa que ofrece una interface gráfica que facilita el uso de MySQL. Como editores y servidores se utilizaron:

Editor	Servidor	Tecnología
Perl editor	Apache Web Server	CGI (Perl)
ColdFusion Studio	ColdFusion Server	ColdFusion
Visual Interdev	Personal Web Server, Internet Information Services 5.0	ASP
Visual Basic	Personal Web Server, Internet Information Services 5.0	WebClass
PHP Coder	Apache Web Server, Xitami	PHP
Visual Café	Java Web Server, iPlanet Web Server	Servlet
DreamWeaver	Tomcat	JSP

CGI

En el ejemplo para la tecnología CGI se utiliza como lenguaje, Perl. Se utiliza también la librería cgi-lib.pl de Steven Brenner para trabajar con la QueryString. Se emplea para la consulta a la base de datos el API que ofrece MySQL para Perl como es DBI.

```
#!perl

# Se incluye la librería mediante función require
require "cgi-lib.pl";
use DBI();

$| = 1;

print PrintHeader();
print HtmlTop("Resultados de la busqueda");

print "<br>";
ReadParse();
$dni = Variables();

#Conexión a la Base de Datos
my $con=DBI->connect("DBI:mysql:database=notas;host=localhost",
      "root","",{'RaiseError'=>1});
#Recogemos datos de la consulta
$consulta = "SELECT * FROM asignaturas,convocatorias,alumnos ";
$consulta .= "WHERE dni_conv='$dni' AND dni_alum=dni_conv AND
cod_asig=cod_asigc";

my $sta = $con->prepare($consulta);
$sta->execute();

print "<table border=1 align=center>";
print "<th>Asignatura</th><th>Num.Creditos</th><th>Nota
Final</th><th>Num. Convocatoria</th><th>Convocatoria</th>";

while(my $rs = $sta->fetchrow_hashref()) {
 print "<tr><td>";
 print $rs->{'nombre_asig'};
 print "</td><td align=right>";
 print $rs->{'num_creditos'};
 print "</td><td align=right>";
 print $rs->{'nota_fin'};
 print "</td><td align=right>";
 print $rs->{'num_conv'};
 print "</td><td>";
 print $rs->{'fecha_conv'};
 print "</td></tr>";
```

```
    }

    print "</table>";

    $sta->finish();
    #Desconectarse de la base de datos
    $con->disconnect();
    print "<br><HR>\n";
    print "<br><center><a href=javascript:history.back()>Atrás<a></center>";


    print HtmlBot();
    exit;
```

En este segundo ejemplo para la tecnología CGI se utiliza el módulo CGI. Como se puede ver este módulo facilita el trabajo con los scripts CGI enormente.

```
    #!perl
    use DBI();
    use CGI;

    $query = new CGI;

    $dni = $query->param('dni');

    #Conexión a la Base de Datos
    my $con=DBI->connect("DBI:mysql:database=notas;host=localhost",
          "root","",{'RaiseError'=>1});
    #Recogemos datos de la consulta
    $consulta = "SELECT * FROM asignaturas,convocatorias,alumnos ";
    $consulta   .=   "WHERE   dni_conv='$dni'   AND   dni_alum=dni_conv   AND
cod_asig=cod_asigc";

    my $sta = $con->prepare($consulta);
    $sta->execute();

    print $query->header;
    print $query->start_html("Buscando...");

    print "<table border=1 align=center>";
    print                  "<th>Asignatura</th><th>Num.Creditos</th><th>Nota
Final</th><th>Num. Convocatoria</th><th>Convocatoria</th>";

    while(my $rs = $sta->fetchrow_hashref()) {
     print "<tr><td>";
     print $rs->{'nombre_asig'};
     print "</td><td align=right>";
     print $rs->{'num_creditos'};
     print "</td><td align=right>";
     print $rs->{'nota_fin'};
     print "</td><td align=right>";
     print $rs->{'num_conv'};
     print "</td><td>";
     print $rs->{'fecha_conv'};
     print "</td></tr>";
    }

    print "</table>";

    $sta->finish();
    #Desconectarse de la base de datos
    $con->disconnect();

    print "<br><HR>\n";
    print       "<br><center><a       href=javascript:history.back()>Atrás<a>
</center>";
```

```
print $query->end_html;
```

Cold Fusion

El ejemplo fue realizado teniendo como servidor Personal Web Server (PWS) y ColdFusion Server 5. La página .cfm fue colocada en c:\inetpub\wwwroot. El ColdFusion Server se adapta perfectamente con PWS ya que una simple instalación (estilo windows) y configuró todo para funcionar. La conexión a la base de datos se realiza mediante ODBC (datasource="NuSphere MySQL") gracias a MyODBC.

```
<html>
<head>
      <title>Generación de Contenido Dinámico - CFusion</title>
</head>
<body bgcolor="#1A4B85" Text="#FFFFFF">
<cfquery name="Alumno"
datasource="NuSphere MySQL">
SELECT *
FROM asignaturas,convocatorias,alumnos
WHERE dni_conv='#Form.Dni#' AND
 dni_alum=dni_conv AND cod_asig=cod_asigc
</cfquery>
<table border=1 align=CENTER>
<Caption><h1 align=center>Resultados de la
busqueda</h1></caption>
<th>Asignatura</th><th>Creditos</th>
<th>Nota Teoria</th><th>Nota Practica</th>
<th>Nota Final</th><th>Convocatorias</th>
<th>Fecha Convocatoria</th>
<cfoutput query="alumno">
<tr><td>#nombre_asig#</td>
<td align=right>#num_creditos#</td>
<td align=right>#nota_teo#</td>
<td align=right>#nota_pract#</td>
<td align=right>#nota_fin#</td>
<td align=right>#num_conv#</td>
<td>#fecha_conv#</td></tr>
</cfoutput>
</table>
<br><center><a href=javascript:history.back()>Atrás<a></center>
</body>
</html>
```

ASP

En este caso no hace falta llamar al formulario para la entrada del dni. La página ASP genera la primera vez que se llama el formulario y al pulsar submit se llama a sí misma generando el resultado conveniente.

Para el acceso a base de datos se utilizó el DSN del sistema, tal y como se comentó en el ejemplo anterior, y los objetos ADO.

```
<html>
<head>
      <title>Generación de Contenido Dinámico - ASP</title>
</head>

<body bgcolor="#1A4B85" Text="#FFFFFF">
<%IF(Request.Form("dni")<>"") then

  ' Código para conectar a la base de datos
  Dim Conn
  'Se crea un objeto conexion
  Set Conn = Server.CreateObject("ADODB.Connection")
```

```
      'Se abre la Base de Datos utilizando MyODBC y
      'teniendo un DSN de sistema
      'que se corresponda con la BD Notas
      Conn.Open "NuSphere MySQL"

      'Se recoge del formulario el dni
      dni = Request.Form("dni")

      'Sentencia SQL
      sql = "SELECT * FROM asignaturas,convocatorias,alumnos "
      sql = sql & "WHERE dni_conv=""" & dni & """
          AND dni_alum=dni_conv AND cod_asig=cod_asigc;"

      'Ejecución de la sentencia SQL
      Set Rs=Conn.Execute(sql)

      'Si el RecordSet contiene valores
      'devolvemos la cabecera de la tabla
      If not Rs.eof then
      %><table border=1 align=CENTER>
        <caption><H3>Resultados De La
         Búsqueda     "<%=dni%>"</H3>     </caption>     <th>Asignatura</th>
<th>Creditos</th>      <th>Nota Teoria</th> <th>Nota Practica</th> <th>Nota
Final</th> <th>Convocatorias</th> <th>Fecha Convocatoria</th>
      <%  while not RS.eof
                        Response.Write "<tr><td>" & RS("nombre_asig") &
    "</td><td align=right>" & RS("num_creditos") & "</td>"
                        Response.Write "<td align=right>" &
   RS("nota_teo") & "</td><td align=right>" & RS("nota_pract") &
   "</td>"
                        Response.Write "<td align=right>" &
   RS("nota_fin") & "</td><td align=right>" & RS("num_conv") &
   "</td><td>" & RS("fecha_conv") & "</td></tr>"
                        Rs.Movenext
            wend
           Response.Write "</table>"

      'Si el RecordSet no contiene valores mostramos
      'siguiente mensaje
      Else
          Response.Write  "<h3  align=center>No  hubo  resultados  para  "  &
busqueda & "</h3>"
      End If

     'Cerramos el RecordSet y la conexión
     RS.Close
     Conn.Close
     Set RS=Nothing ' Liberamos recursos
     Set Conn=Nothing

     Response.write"<br><center><a href=javascript:history.back()>
   Atrás<a></center>"
   Else%>

   <br><h2   align=Center>Consulta   de   Calificaciones   por   DNI   de
alumno</h2><br>
   <table align=CENTER>
    <form method="post" action="consulta.asp" id=form1 name=form1>
     <tr>
         <td align="right">
          <b>Introduce DNI</b>(sin letra):<br>
                <INPUT        NAME="dni"        TYPE="text"        SIZE="10"
MAXLENGTH="10"> 
                <INPUT NAME="accion" TYPE="submit" VALUE="Buscar">
         </td>
      </tr>
     </form>
```

```
</TABLE>
<%
End If
%>
</body>
</html>
```

WebClass

Para este ejemplo se ha de incluir la página HTML con el formulario para la entrada del dni como una plantilla web dentro del proyecto Visual Basic. Para la ejecución se abre el proyecto y se pulsa F5 y ya se abre un explorador con el formulario de entrada.

```
Option Explicit
Option Compare Text



Private Sub Plantilla_Form1()
  Dim Conn
  Set Conn = Server.CreateObject("ADODB.Connection")
  Conn.Open "NuSphere MySQL"

  Dim dni
  dni = Request.Form("dni")

  Dim sql
  sql = "SELECT * FROM asignaturas,convocatorias,alumnos "
  sql = sql & "WHERE dni_conv=""" & dni & """ AND dni_alum=dni_conv AND
cod_asig=cod_asigc;"

  Dim RS
  Set RS = Conn.Execute(sql)
  Response.Write (RS("nombre_alum")) & "<BR>"
  Response.Write (RS("nota_fin")) & "<BR>"

  RS.Close
  Conn.Close
  Set RS = Nothing ' Liberamos recursos
  Set Conn = Nothing
End Sub

Private Sub Plantilla_ProcessTag(ByVal TagName As String, _
    TagContents As String, SendTags As Boolean)

    Select Case TagName
    Case "WC@CONTADOR":
        Dim Contador As Integer
        On Error Resume Next
        Open "Contador.dat" For Input As #1
        If Err Then
            Contador = 0
        Else
            Input #1, Contador
            Close #1
        End If
        On Error GoTo 0
        Contador = Contador + 1
        Open "Contador.Dat" For Output As #1
        Print #1, Contador
        Close #1
        TagContents = Contador

    Case "WC@FECHA":
```

```
            TagContents = "-" & Date & "-" & Time
        End Select
    End Sub


    Private Sub WebClass_Start()

        Plantilla.WriteTemplate

    End Sub
```

PHP

En PHP tenemos abundantes funciones para acceder a bases de datos diversas. PHP carga por defecto las de MySQL y podremos obtener abundante información (en castellano) en la página oficial. Este comentario viene a raíz de que se probó el ejemplo con una base de datos en InterBase; y, las funciones que se ofrecían en los manuales el intérprete las mostraba como desconocidas. Indagando se decubrió que PHP sólo cargaba el módulo para MySQL por defecto, y si se querían otros módulos hay que modificar el fichero php.ini en la sección de módulos y quitar el comentario en la línea oportuna.

```
<html>
<head>
       <title>Generación Dinámica de Contenido - PHP</title>
</head>
<body bgcolor=Linen>
       <?php
      /*Conexión con la base de datos*/
      $host="localhost";
      $usuario="root";
      $clave="";
      MYSQL_CONNECT($host,$usuario,$clave)OR DIE("No ha sido
      posible conectar con la base de datos");

      /*Selección de la base de datos*/
      $bd="notas";
      MYSQL_SELECT_DB($bd)OR DIE("No ha sido posible abrir
      la base de datos");

    $consulta="SELECT  *  FROM  asignaturas,convocatorias,alumnos  WHERE
dni_conv='$Dni' AND dni_alum=dni_conv AND cod_asig=cod_asigc";

      /*Resultado de la consulta*/
      $resultado=mysql_query($consulta);
      $numero=MYSQL_NUMROWS($resultado);

      $contador=0;
      if($numero==0) :
            print"<center><p>No hay respuesta</center>";
      elseif ($numero>0):
             $nombre=mysql_result($resultado,$contador,"nombre_alum");
             $dni=mysql_result($resultado,$contador,"dni_alum");
       print        "<center><hr><br>Nombre:        $nombre        Dni:
       $dni</center><hr><br>";

        print"<table  border=1  align=CENTER><th>Asignatura</th><th>Nota
Final</th><th>Convocatorias</th><th>Fecha Convocatoria</th>";
             while ($contador<$numero):
                                          $asig=mysql_result($resultado,
$contador,"nombre_asig");
                    $nota=mysql_result($resultado,$contador,"nota_fin");
                    $conv=mysql_result($resultado,$contador,"num_conv");
```

```
                                                $fecha=mysql_result($resultado,
$contador,"fecha_conv");
                        $contador++;
                        print"<tr><td>$asig</td><td align=right>$nota</td>";
                                    print"<td    align=right>$conv</td><td
          align=right>$fecha</td></tr>";
                 endwhile;
                 print"</table>";
          endif;
            ?>
            <br><center><a href=javascript:history.back()>Atrás<a></center>
    </body>
    </html>
```

Servlets

El archivo servlet.html en la etiqueta action hace referencia a este servlet (notasServlet). Éste genera una tabla con los resultados obtenidos en la base de datos notas. Se utilizó un driver JDBC propio para MySQL que se utiliza en el método `DriverManager.getConnection("jdbc:mysql://localhost/notas");`. Con ello cargamos el driver de MySQL. El driver podemos ubicarlo físicamente en el mismo directorio al servlet. Para la realización del ejemplo se utilizó Java Web Server y las clases se situaron el el directorio Servlets. Para un mejor funcionamiento habría que dar de alta los servlets con el administrador que ofrece JWS.

```
/*
        Servlet que consulta en la base de datos "notas" y
   devuelve los resultados dependiendo del dni introducido
   a través de la página servlet.html.
        El driver JDBC:MySQL debe estar incluido en el mismo
   directorio que el servlet
   Tomás Vilariño 20 Julio 2001
*/

import java.io.*;
import java.net.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class notasServlet extends HttpServlet {
  Connection con;
  DatabaseMetaData metaData;

  // Este método es el que se encarga de establecer e inicializar
  // la conexión con la base de datos
  public void init( ServletConfig conf ) throws ServletException {
    SQLWarning w;

    super.init( conf );
    try {
      Class.forName( "org.gjt.mm.mysql.Driver" );
      try{
                                                          con        =
DriverManager.getConnection("jdbc:mysql://localhost/notas");
      if( (w = con.getWarnings()) != null ) {
        while( w != null ) {
          log( "SQLWarning: "+w.getSQLState()+'\t'+
               w.getMessage()+'\t'+w.getErrorCode()+'\t' );
          w = w.getNextWarning();
          }
        }
```

```
          }
          catch(SQLException e){}
        } catch( ClassNotFoundException e ) {
          throw new ServletException( "init" );}
      }


      public void service( HttpServletRequest req,HttpServletResponse res )
      throws ServletException,IOException {
        String dni = req.getParameter( "dni" );
        try{
            realizarConsulta( req,res,dni );}
        catch(SQLException e){
            }
      }


      // Se cierra la conexión con la base de datos
      public void destroy() {
        try {
          con.close();
        } catch( SQLException e ) {
          while( e != null ) {
            log( "SQLException: "+e.getSQLState()+'\t'+
                 e.getMessage()+'\t'+e.getErrorCode()+'\t' );
            e = e.getNextException();
            }
        } catch( Exception e ) {
          e.printStackTrace();
        }
      }


      // Este método ejecuta la consulta a la base de datos y devuelve el
      // resultado en una tabla con los distintos campos
               public     void     realizarConsulta(     HttpServletRequest
req,HttpServletResponse res, String dni )
      throws IOException,SQLException {
        Statement stmt = con.createStatement();
        String consulta;
        ResultSet rs;

        res.setStatus( res.SC_OK );
        res.setContentType( "text/html" );

        consulta = "SELECT * ";
        consulta += "FROM asignaturas,convocatorias,alumnos ";
         consulta += "WHERE dni_conv=\"" + dni + "\" AND dni_alum=dni_conv
AND cod_asig=cod_asigc;";

        rs = stmt.executeQuery( consulta );
        PrintWriter out = new PrintWriter( res.getOutputStream() );
        out.println( "<HTML>" );
           out.println(  "<HEAD><TITLE>Generación  Dinámica  de  Contenido  -
Servlet</TITLE></HEAD>" );
                          out.println(      "<BODY      BGCOLOR=\"#1a4b85\"
TEXT=\"#FFFFFF\"><CENTER><H2>Resultado de la busqueda</H2></CENTER>" );
        if(rs.next()){
               out.println( "<br><center>Nombre: <b>" + rs.getString(13));
//Nombre (12)
                        out.println(  rs.getString(14)+  "</b></center>");
//Apellidos (14)
                    out.println(  "<center>DNI:  <b>"  +  rs.getString(6)+
"</b></center><br>");//Dni (6)

            out.println( "<br><TABLE BORDER=1 ALIGN=center>" );
            out.println( "<th>Asignatura</th><th>Creditos</th>" );
```

```
            out.println( "<th>Nota Teoria</th><th>Nota Practica</th><th>Nota
Final</th><th>Convocatorias</th><th>Fecha Convocatoria</th>" );

            do{
                        out.println( "<tr><td>" + rs.getString(2)+ "</td>");
//Asignatura (2)
                out.println( "<td align=right>" + rs.getString(3)+ "</td>");
//Creditos (3)
                out.println( "<td align=right>" + rs.getString(9)+ "</td>");
//Nota Teoria (9)
                out.println( "<td align=right>" + rs.getString(8)+ "</td>");
//Nota Practica (8)
                        out.println( "<td align=right>" + rs.getString(10)+
"</td>");       //Nota Final (10)
                out.println( "<td align=right>" + rs.getString(7)+ "</td>");
//Num_Convocatoria (7)
                        out.println( "<td>" + rs.getString(11)+ "</td>");
//Convocatoria (11)
            }while( rs.next() );
            out.println( "</TABLE></BODY></HTML>" );
        }
        else
              out.println( "<H3><center>No hubo resultados para " + dni +
"</center></H3>");

                                                out.println(          "<center><A
HREF=javascript:history.back()>Atrás</A></center>" );

        out.flush();
        rs.close();
        stmt.close();
    }


     // Devuelve la información del Servlet
     public String getServletInfo() {
       return "Servlet Notas (Proyecto Acad. Dirigido), 2001";
     }
   }
```

JavaServer Pages

Después de que se introduzca el dni en el formulario, el atributo action nos llevará a la siguiente página JSP. Usamos un bean (clase DB) que nos ofrece tres métodos: boolean conectar(), ResultSet ejecutar(String) y void terminar(). El primero establecerá la conexión a la base de datos devolviéndonos True si hubo éxito, False en caso contrario. El segundo método ejecutar al que se le pasa un String en el que irá la consulta y nos devuelve el ResultSet. Y por último void terminar() que se limitará a cerrar la conexión a la base de datos. El único trabajo que realiza la página JSP es la presentación de los datos. Un perfecto modelo en tres capas.

La página JSP debe ubicarse en un directorio que cuente con la carpeta Web-Inf/ Classes dentro del Tomcat. Y las clases que se usen en la página JSP se ubicarán en Web-Inf/Classes.

```
<HTML>
<HEAD>
<TITLE>
Generación de Contenido Dinámico - JSP
</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">
```

```
<%@ page language="java" import="java.sql.*" %>
<jsp:useBean id="db" scope="request" class="DB" />
<jsp:setProperty name="db" property="*" />

<%
int numeroColumnas=0;
ResultSet rs=null;
ResultSetMetaData rsmd=null;
try{
      db.conectar();
}catch(ClassNotFoundException e){
      throw new ServletException("Drivers de la base de datos no
disponibles",e);
}catch(SQLException e){
      throw new ServletException("URL asociada a la base de datos no
existe o incorrecta",e);
}//try

String consulta = "SELECT * FROM asignaturas,convocatorias,alumnos WHERE
dni_conv=\"76825926\" AND dni_alum=dni_conv AND cod_asig=cod_asigc;";
try {
      rs = db.ejecutar(consulta);
}catch(SQLException e){
      throw new ServletException("La sentencia no funciona
correctamente ",e);
}//try

try{
%>
<TABLE BORDER=1 ALIGN=center>
<%
      while( rs.next() ) {
%>
       <tr>
       <td><%=rs.getString(13)%></td>
       <td><%=rs.getString(14)%></td>
       <td><%=rs.getString(6)%></td>
       <td><%=rs.getString(2)%></td>
       <td><%=rs.getString(3)%></td>
       <td><%=rs.getString(9)%></td>
       <td><%=rs.getString(8)%></td>
       <td><%=rs.getString(10)%></td>
       <td><%=rs.getString(11)%></td>
      </tr>
<%
}
    rs.close();
    db.terminar();
}catch(SQLException e){
      throw new ServletException("La sentencia");}

%>
</TABLE>

</BODY>
</HTML>
```

El código del bean que utilizamos en la página JSP es el siguiente:

```
public class DB {
      private java.sql.Connection conexion=null;

      public DB(){
            super();
      }
```

```
          public boolean conectar() throws
ClassNotFoundException,java.sql.SQLException {
                Class.forName("org.gjt.mm.mysql.Driver");
                conexion =
java.sql.DriverManager.getConnection("jdbc:mysql://localhost/notas");
                return true;
                }
          public java.sql.ResultSet ejecutar(String query)throws
java.sql.SQLException {
                java.sql.Statement sentencia= conexion.createStatement();
                java.sql.ResultSet resultados =
sentencia.executeQuery(query);

                return resultados;
                }
          public void terminar() throws java.sql.SQLException {
                conexion.close();
                }
    }
```

ANEXO B. Conceptos Básicos de Redes

En los siguientes párrafos se recogen los conceptos más comúnmente utilizados a la hora de tratar las comunicaciones en redes.

Protocolo de Comunicaciones

Para que dos o más ordenadores puedan conectarse a través de una red y ser capaces de intercambiar datos de una forma ordenada, deben seguir un protocolo de comunicaciones que sea aceptado por todos ellos. El protocolo define las reglas que se deben seguir en la comunicación, por ejemplo, enseñar a los niños a decir por favor y gracias es una forma de indicarles un protocolo de educación, y si alguna vez se olvidan de dar las gracias por algo, seguro que reciben una reprimenda de sus mayores.

Hay muchos protocolos disponibles para ser utilizados; por ejemplo, el protocolo HTTP define como se van a comunicar los servidores y navegadores Web y el protocolo SMTP define la forma de transferencia del correo electrónico. Estos protocolos, son protocolos de aplicación que actúan al nivel de superficie, pero también hay otros protocolos de bajo nivel que actúan por debajo del nivel de aplicación y que son más complicados, aunque, afortunadamente, como programadores Java, no será necesario tener excesivo conocimiento de los protocolos de bajo nivel; nada que vaya más allá del conocimiento de su existencia.

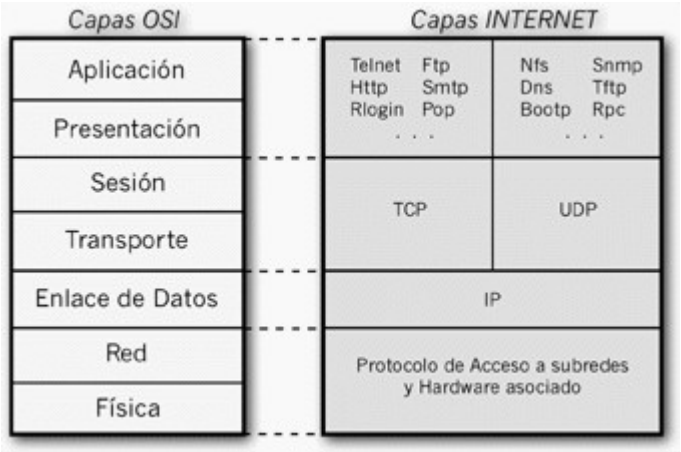
Capas de Red

Las redes están separadas lógicamente en capas, o niveles, o layers; desde el nivel de aplicación en la parte más alta hasta el nivel físico en la parte más baja. Los detalles técnicos de la división en capas o niveles de la red se escapa de la misión del Tutorial y, además, no es un conocimiento imprescindible para que el lector desarrolle programas Java que se comuniquen a través de la red, ya que la gente de JavaSoft se ha encargado de ocultar toda la parafernalia que involucra el manejo de los protocolos de redes de bajo nivel y las capas de más bajo nivel del modelo de comunicaciones.

La única capa interesante para el usuario y el programador es el Nivel de Aplicación, que es el que se encarga de tomar los datos en una máquina desde esta capa y soltarlos en la otra máquina en esta misma capa, los pasos intermedios y los saltos de capas que se hayan producido por el camino están ocultos en el lenguaje Java.

Por si el lector es más curioso y desea conocer un poco el funcionamiento de las capas, la figura siguiente muestra la correlación existente entre el modelo teórico de capas o niveles de red propuestos por la Organización de Estándares Internacional (ISO, International Standards Organisation) y el modelo empleado por las redes TCP/IP. Cuando se presenta un problema de tamaño considerable, la solución más óptima comienza por dividirlo en pequeñas secciones, para posteriormente proceder a solventar cada una de ellas independientemente. Pues el mismo principio de divide y vencerás es el que se sigue a la hora de diseñar redes, es decir, separar en un buen número de niveles el hecho de la transmisión de un sistema a otro. Como referencia, la ISO, creó un modelo de interconexión de sistemas abiertos, conocido como OSI. Ese modelo divide en siete capas el proceso de transmisión de información entre equipos informáticos, desde el hardware físico, hasta las aplicaciones de red que maneja el usuario. Estas capas son las que se pueden ver en la figura siguiente:

física, de enlace de datos, de red, de transporte, de sesión, de presentación y, por último, de aplicación. Cada nuevo protocolo de red que se define se suele asociar a uno (o a varios) niveles del estándar OSI. Internet dispone de un modelo más sencillo; no define nada en cuanto al aspecto físico de los enlaces, o a la topología o clase de red de sus subredes y, por lo tanto, dentro del modelo OSI, sólo existe una correlación con los niveles superiores.



Las aplicaciones que trabajan a un cierto nivel o capa, sólo se comunican con sus iguales en los sistemas remotos, es decir, a nivel de aplicación; un navegador sólo se entiende con un servidor Web, sin importarle para nada cómo le llega la información. Este mismo principio es el que se emplea para el resto de las capas. Para ilustrar este concepto de capas o niveles, puede resultar explicativo ver qué sucede cuando se solicita una página Web. En este caso, el navegador realiza una petición HTTP, petición que se incluye en un paquete TCP, que a su vez es encapsulado y fragmentado en uno o varios datagramas IP, que es la unidad de datos a nivel de red. Dichos datagramas son de nuevo encapsulados en unidades de datos PPP, o frames, que se envían al proveedor de Internet a través del módem, que transforma esas unidades digitales de datos en señales acústicas de acuerdo a una determinada norma, V.34bis o V.90, por ejemplo. El proveedor de Internet ensamblará los paquetes PPP para convertirlos de nuevo en datagramas IP, que son llevados a su destino, donde serán decodificados en sentido inverso al realizado en el equipo originen de la petición, hasta que alcancen el nivel de aplicación, que supone el servidor web.

Resumiendo en primer lugar, que TCP/IP opera sólo en los niveles superiores de red, resultándole indiferente el conjunto de protocolos que se entienden con los adaptadores de red Token Ring, Ethernet, ATM, etc., que se encuentren por debajo. En segundo lugar, que IP es un protocolo de datagramas que proporciona un interfaz estándar a protocolos superiores. Y, en tercer lugar, que dentro de estos protocolos superiores se incluyen TCP y UDP, los cuales ofrecen prestaciones adicionales que ciertas aplicaciones de red necesitan.

Cliente y Servidor

Servidor es aquel que escucha y está siempre a la espera de que el Cliente se conecte y comenzar la conversación entre ambos.

IP, Internet Protocol

Es el protocolo que se utiliza por debajo del Nivel de Aplicación para traspasar datos entre cliente y servidor. El lector no necesita más que saber de su existencia; no obstante, indicar que es un protocolo de red encargado de mover datos en forma de paquetes entre un origen y un destino y que, como bien indica su nombre, es el protocolo que normalmente se utiliza en Internet.

Al igual que antes, si bien ese conocimiento es suficiente para el programador Java, si el lector tiene interés en las redes, debe saber que IP es un protocolo simple, fácilmente implementable, de pequeñas unidades de datos o datagramas, que proporciona un interfaz estándar a partir del cual el resto de los protocolos y servicios pueden ser construidos, sin tener que preocuparse de las diferencias que existan entre las distintas subredes por la cuales circulen los datos.

Todo dispositivo conectado a Internet o a cualquier red basada en TCP/IP, posee al menos una dirección IP, un identificador que define unívocamente al dispositivo que lo tiene asignado en la red.

Un datagrama IP se encuentra dividido en dos partes: cabecera y datos. Dentro de la cabecera se encuentran, entre otros campos, la dirección IP del equipo origen y la del destino, el tamaño y un número de orden. El estudio más en detalle de la estructura interna del datagrama, sí que se escapa del alcance de este breve comentario.

IP opera entre un sistema local conectado a Internet y su router o encaminador más próximo, así como entre los distintos encaminadores que forman la red. Cuando un datagrama llega a un encaminador, éste determina, a partir de su dirección IP de destino, hacia cuál de sus conexiones de salida ha de dirigir el datagrama que acaba de recibir. Por desgracia, en cuanto al transporte, IP provee un servicio que intenta entregar los datos al equipo destino, pero no puede garantizar la integridad, e incluso la recepción de esos datos. Por ello, la mayoría de las aplicaciones hacen uso de un protocolo de más alto nivel que ofrezca el grado de fiabilidad necesario.

Cada datagrama IP es independiente del resto, por lo que cada uno de ellos es llevado a su destino por separado. La longitud del datagrama es variable, pudiendo almacenar hasta 65 Kbytes de datos; si el paquete de datos (TCP o UDP) sobrepasa ese límite, o el tamaño de la unidad de datos de la red que se encuentra por debajo es más pequeño que el datagrama IP, el mismo protocolo IP lo fragmenta, asignándole un número de orden, y distribuye empleando el número de datagramas que sea necesario.

TCP, Transmission Control Protocol

Hay veces en que resulta de vital importancia tener la seguridad de que todos los paquetes que constituyen un mensaje llegan a su destino y en el orden correcto para la recomposición del mensaje original por parte del destinatario. El protocolo TCP se incorporó al protocolo IP para proporcionar a éste la posibilidad de dar reconocimiento de la recepción de paquetes y poder pedir la retransmisión de los paquetes que hubiesen llegado mal o se hubiesen perdido. Además, TCP hace posible que todos los paquetes lleguen al destinatario, juntos y en el mismo orden en que fueron enviados.

Por lo tanto, es habitual la utilización de los dos acrónimos juntos, TCP/IP, ya que los dos protocolos constituyen un método más fiable de encapsular un mensaje

en paquetes, de enviar los paquetes a un destinatario, y de reconstruir el mensaje original a partir de los paquetes recibidos.

TCP, en resumen, ofrece un servicio de transporte de datos fiable, que garantiza la integridad y entrega de los datos entre dos procesos o aplicaciones de máquinas remotas. Es un protocolo orientado a la conexión, es decir, funciona más o menos como una llamada de teléfono. En primer lugar, el equipo local solicita al remoto el establecimiento de un canal de comunicación; y solamente cuando ese canal ha sido creado, y ambas máquinas están preparadas para la transmisión, empieza la transferencia de datos real.

UDP, User Datagram Protocol

Hay veces en que no resulta tan importante el que lleguen todos los mensajes a un destinatario, o que lleguen en el orden en que se han enviado; no se quiere incurrir en una sobrecarga del sistema o en la introducción de retrasos por causa de cumplir esas garantías. Por ejemplo, si un ordenador está enviando la fecha y la hora a otro ordenador cada 100 milisegundos para que la presente en un reloj digital, es preferible que cada paquete llegue lo más rápidamente posible, incluso aunque ello signifique la pérdida de algunos de los paquetes. El protocolo UDP está diseñado para soportar este tipo de operaciones. UDP es, por tanto, un protocolo menos fiable que el TCP, ya que no garantiza que una serie de paquetes lleguen en el orden correcto, e incluso no garantiza que todos esos paquetes lleguen a su destino. Los procesos que hagan uso de UDP han de implementar, si es necesario, sus propias rutinas de verificación de envío y sincronización. Como programador Java, el lector tiene en sus manos la elección del protocolo que va a utilizar en sus comunicaciones, en función de las características de velocidad y seguridad que requiera la comunicación que desea establecer.

Dirección IP

La verdad es que no se necesita saber mucho sobre el protocolo IP para poder utilizarlo, pero sí que es necesario conocer el esquema de direccionamiento que utiliza este protocolo. Cada ordenador conectado a una red TCP/IP dispone de una dirección IP única de 4 bytes (32 bits), en donde, según la clase de red que se tenga y la máscara, parte de los 4 bytes representan a la red, parte a la subred (donde proceda) y parte al dispositivo final o nodo específico de la red. La figura siguiente muestra la representación de los distintos números de una dirección IP de un nodo perteneciente a una subred de clase B (máscara 255.255.0.0). Con 32 bits se puede definir una gran cantidad de direcciones únicas, pero la forma en que se asignaban estas direcciones estaba un poco descontrolada, por lo que hay muchas de esas direcciones que a pesar de estar asignadas no se están utilizando.

Por razones administrativas, en los primeros tiempos del desarrollo del protocolo IP, se establecieron cinco rangos de direcciones, dentro del rango total de 32 bits de direcciones IP disponibles, denominando a esos subrangos, clases. Cuando una determinada organización requiere conectarse a Internet, solicita una clase, de acuerdo al número de nodos que precise tener conectados a la Red. La

administración referente a la cesión de rangos la efectúa InterNIC (Internet Network Information Center), aunque existen autoridades que, según las zonas, gestionan dominios locales; por ejemplo, el dominio correspondiente a España lo gestiona Red Iris.

Los subrangos se definen en orden ascendente de direcciones IP, por lo cual, a partir de una dirección IP es fácil averiguar el tipo de clase de Internet con la que se ha conectado. El tipo de clase bajo la que se encuentra una dirección IP concreta viene determinado por el valor del primer byte de los cuatro que la componen o, lo que es igual, el primer número que aparece en la dirección IP. Las clases toman nombre de la A a la E, aunque las más conocidas son las A, B y C. En Internet, las redes de clase A son las comienzan con un número entre el 1 y el 126, que permiten otorgar el mayor número de direcciones IP (16,7 millones), por lo que se asignan a grandes instituciones educativas o gubernamentales. Las clases B (65536 direcciones por clase), suelen concederse a grandes empresas o corporaciones y, en general, a cualquier organización que precise un importante número de nodos. Las redes de clase C (256 direcciones) son las más comunes y habitualmente se asignan sin demasiados problemas a cualquier empresa u organización que lo solicite. La clase D se reserva a la transmisión de mensajes de difusión múltiple (multicast), mientras que la clase E es la destinada a investigación y desarrollo. La tabla siguiente resume estos datos.

Clase	Nodos por Clase	Máscara Asociada	Dirección de comienzo	Dirección Final
A	224 = 16.777.216	255.0.0.0	0.0.0.0	127.255.255.255
B	216 = 65.536	255.255.0.0	128.0.0.0	191.255.255.255
C	28 = 256	255.255.255.0	192.0.0.0	223.255.255.255
D	.	.	224.0.0.0	239.255.255.255
E	.	.	240.0.0.0	255.255.255.255

Todo lo dicho antes solamente implica a la asignación de direcciones dentro de Internet. Si se diseña una red TCP/IP que no vaya a estar conectada a la Red, se puede hacer uso de cualquier conjunto de direcciones IP. Solamente existen cuatro limitaciones, intrínsecas al protocolo, a la hora de escoger direcciones IP, pero que reducen en cierta medida el número de nodos disponibles por clase que se indicaban en la tabla anterior. La primera es que no se pueden asignar direcciones que comiencen por 0; dichas direcciones hacen referencia a nodos dentro de la red actual. La segunda es que la red 127 se reserva para los procesos de resolución de problemas y diagnosis de la red; de especial interés resulta la dirección 127.0.0.1, bucle interno (loopback) de la estación de trabajo local. La tercera consiste en que las direcciones IP de nodos no pueden terminar en 0, o en cualquier otro valor base del rango de una subred; porque es así como concluyen las redes. Y, por último, cuando se asignan direcciones a nodos, no se pueden emplear el valor 255, o cualquier otro valor final del rango de una subred. Este valor se utiliza para enviar mensajes a todos los elementos de una red (broadcast); por ejemplo, si se envía un mensaje a la dirección 192.168.37.255, se estaría enviando en realidad a todos los nodos de la red de clase C 192.168.37.xx.

Ahora bien, si se quiere que una red local tenga acceso exterior, hay una serie de restricciones adicionales, por lo que hay una serie de direcciones reservadas que, a fin de que pudiesen ser usadas en la confección de redes locales, fueron excluidas de Internet. Estas direcciones se muestran en la siguiente tabla.

Clase	Máscara Asociada	Dirección de comienzo	Dirección Final
A	255.0.0.0	10.0.0.0	10.255.255.255
B	255.255.0.0	172.16.0.0	172.31.255.255
C	255.255.255.0	192.168.0.0	192.168.255.255

Infovía, por ejemplo, al ser una especie de gran Intranet española, utiliza el rango de direcciones 10.xx.xx.xx. Si el lector está considerando la creación de una intranet, debería escoger direcciones IP para su red dentro de alguno de los rangos reservados de la tabla anterior, y emplear un servidor proxy, o cualquier otro mecanismo que enmascare las direcciones IP de esa intranet, de forma que todos los puestos de la red local utilicen una única dirección IP a la hora de salir a la Red.

Actualmente, se intenta expandir el número de direcciones únicas a un número mucho mayor, utilizando 128 bits. E.R. Harold, en su libro Java Network Programming, dice que el número de direcciones únicas que se podría alcanzar representando las direcciones con 128 bits es 1.6043703E32. La verdad es que las direcciones indicadas de esta forma son difíciles de recordar, así que lo que se hace es convertir el valor de los cuatro bytes en un número decimal y separarlos por puntos, de forma que sea mucho más sencillo el recordarlos; así, por ejemplo, la dirección única asignada a java.sun.com es 204.160.241.98.

Dominios

Y ahora surge la pregunta de qué es lo que significa java.sun.com. Como a pesar de que la dirección única asignada a un ordenador se indique con cuatro cifras pequeñas, resulta muy difícil recordar las direcciones de varias máquinas a la vez; muchas de estas direcciones se han hecho corresponder con un nombre, o dominio, constituido por una cadena de caracteres, que es mucho más fácil de recordar para los humanos. Así, el dominio para la dirección IP 204.160.241.98 es java.sun.com.

El Sistema de Nombres de Dominio (DNS, Domain Name System) fue desarrollado para realizar la conversión entre los dominios y las direcciones IP. De este modo, cuando el lector entra en Internet a través de su navegador e intenta conectarse con un dominio determinado, el navegador se comunica en primer lugar con un servidor DNS para conocer la dirección IP numérica correspondiente a ese dominio. Esta dirección numérica IP, y no el nombre del dominio, es la que va encapsulada en los paquetes y es la que utiliza el protocolo Internet para enrutar paquetes desde el ordenador del lector hasta su destino.

Java proporciona clases para la manipulación y conocimiento de direcciones y dominios, concretamente la clase InetAddress permite encontrar un nombre de dominio a partir de su dirección IP; y viceversa, encontrar la dirección IP que corresponde a un dominio determinado.

Si el lector quiere saber cuál es su dirección IP, pues lo cierto es que probablemente no lo sepa. Si, al igual que el autor del Tutorial, utiliza un proveedor de acceso a Internet, realmente no tendrá una dirección IP fija o un dominio específico. Cada proveedor de acceso a Internet dispone de un bloque de direcciones reservadas; cuando se conecta con él y se accede a Internet, el proveedor asigna una dirección de ese bloque, que durará solamente el tiempo que dure la conexión. Si el lector desconecta y vuelve a conectar, casi seguro que la dirección IP para esta nueva sesión será diferente de la utilizada en la anterior conexión.

Puertos y Servicios

Un *servicio* es una facilidad que proporciona el sistema, y cada uno de estos servicios está asociado a un *puerto*. Un puerto es una dirección numérica a través de la cual se procesa el servicio, es decir, no son puertos físicos semejantes al puerto paralelo para conectar la impresora en la parte trasera del ordenador, sino que son direcciones lógicas proporcionadas por el sistema operativo para poder responder.

Sobre un sistema Unix, por ejemplo, los servicios que proporciona ese sistema y los puertos asociados por los cuales responde a cada uno de esos servicios, se indican en el fichero /etc/services, y algunos de ellos son:

```
daytime      13/udp
ftp        21/tcp
telnet      23/tcp      telnet
smtp        25/tcp      mail
http       80/tcp
```

La primera columna indica el nombre del servicio. La segunda columna indica el puerto y el protocolo que está asociado al servicio. La tercera columna es un alias del servicio; por ejemplo, el servicio *smtp*, también conocido como *mail*, es la implementación del servicio de correo electrónico.

Las comunicaciones de información relacionada con Web tienen lugar a través del puerto 80 mediante protocolo TCP. Para emular esto en Java, se utiliza la clase Socket. La fecha (daytime), sin embargo, el servicio que obtiene la fecha y la hora del sistema, está ligado al puerto 13 utilizando el protocolo UDP. Un servidor que lo emule en Java usaría un objeto DatagramSocket.

Teóricamente hay 65535 puertos disponibles, aunque los puertos del 1 al 1023 están reservados al uso de servicios estándar proporcionados por el sistema, quedando el resto libre para utilización por las aplicaciones de usuario. De no existir los puertos, solamente se podría ofrecer un servicio por máquina. Nótese que el protocolo IP no sabe nada al respecto de los números de puerto, al igual que TCP y UDP no se preocupan en absoluto por las direcciones IP. Se puede decir que IP pone en contacto las máquinas, TCP y UDP establecen un canal de comunicación entre determinados procesos que se ejecutan en tales equipos y, los números de puerto se pueden entender como números de oficinas dentro de un gran edificio. El edificio (equipo), tendrá una única dirección IP, pero dentro de él, cada tipo de negocio, en este caso HTTP, FTP, etc., dispone de una oficina individual.

Cortafuegos

Seguramente que el lector habrá leído ya algo sobre los cortafuegos, o *firewalls*. Un cortafuegos es el nombre que se da a un equipo y su software asociado que permite aislar la red interna de una empresa del resto de Internet. Normalmente se utiliza para restringir el grado de acceso a los ordenadores de la red interna de una empresa desde Internet, por razones de seguridad o cualquier otra.

Servidores Proxy

También es posible que el lector haya leído cosas sobre los *servidores proxy*. Un servidor proxy actúa como interfaz entre los ordenadores de la red interna de una empresa e Internet. Frecuentemente, el servidor proxy tiene posibilidad de ir almacenando un cierto número de páginas web temporalmente en caché, para un acceso más rápido. Por ejemplo, si diez personas dentro de la empresa intentan

conectarse a un mismo servidor Internet y descargar la misma página en un período corto de tiempo, esa página puede ser almacenada por el servidor proxy la primera vez que se accede a ella y proporcionarla él, sin necesidad de acceder a Internet, a las otras nueve personas que la han solicitado. Esto reduce en gran medida el tiempo de espera por la descarga de la página y el tráfico, tanto dentro como fuera de la empresa, aunque a veces puede también hacer que la información de la página se quede sin actualizar, al no descargarse de su sitio original.

URL, Uniform Resource Locator

Una ***URL*** , o dirección, es en realidad un puntero a un determinado recurso de un determinado sitio de Internet. Al especificar una URL, se está indicando:

- El protocolo utilizado para acceder al servidor (http, por ejemplo)
- El nombre del servidor
- El puerto de conexión (opcional)
- El camino (directorio)
- El nombre de un fichero determinado en el servidor (opcional a veces)
- Un punto de referencia dentro del fichero (opcional)

A veces el nombre del fichero se puede omitir, ya que el navegador incorporará automáticamente el nombre de fichero ***index.html*** cuando no se indique ninguno, e intentará descargar ese fichero.

Además de indicar el fichero o página a la que se desea acceder, también es posible indicar una referencia, o *anchor*, que se haya establecido dentro de esa página.

La sintaxis general, resumiendo pues, para una dirección URL, sería:

protocolo://nombre_servidor[:puerto]/directorio/fichero#referencia

El *puerto* es opcional y normalmente no es necesario especificarlo si se está accediendo a un servidor que proporcione sus servicios a través de los puertos estándar; tanto el navegador como cualquier otra herramienta que se utilice en la conexión conocen perfectamente los puertos por los cuales se proporciona cada uno de los servicios e intentan conectarse directamente a ellos por defecto.

ANEXO C.Instrucciones SQL

Select

Recupera registros de una tabla.

SELECT [ALL | DISTINCT] lista_selecccion [INTO [nueva_tabla]] [FROM {tabla | consulat} [[, {tabla2 | consulta2} [..., {tabla16 | consulta16}]] [WHERE criterio] [GROUP BY criterio] [HAVING criterio] [ORDER BY criterio] [COMPUTE criterior] [FOR BROWSE]

Delete

Elimina un registro de una tabla.

DELETE [FROM] {tabla| consulta} [WHERE criterio]

Insert

Agrega registros a una tabla

INSERT [INTO] {tabla| consulta [(columnas)] {DEFAULT VALUES | valores | instruccion_select}

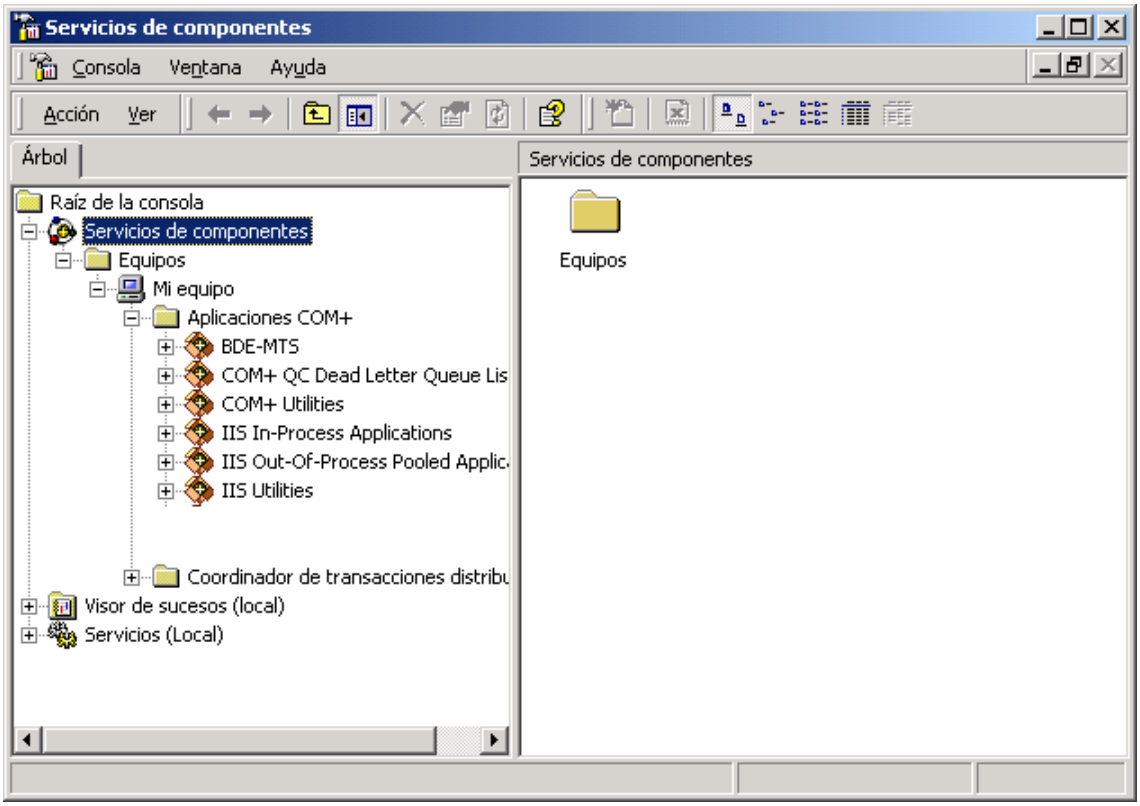
Update

Actualiza los registros de una tabla

UPDATE {tabla | consulta} SET [{tabla | consulta}] { columnas | variables | variables_y_columnas} [, {columnas2 | variables2 | variables_y_columnas2} ... [, {columnasN | variablesN | variables_y_columnasN}]] [WHERE criterio]

ANEXO D.Diferencias con *COM+*

A la hora de programar un componente *COM* o *COM+* no hay demasiadas diferencias. Una diferencia importante es que en *Windows* 2000 no existe la *Microsoft Transaction Library*. Por lo tanto, es inútil buscar la referencia a la misma. Ahora, el *API* de *MTS* se encuentra en la *COM+ Services Type Library*, que será la que tengamos que añadir a nuestros proyectos.

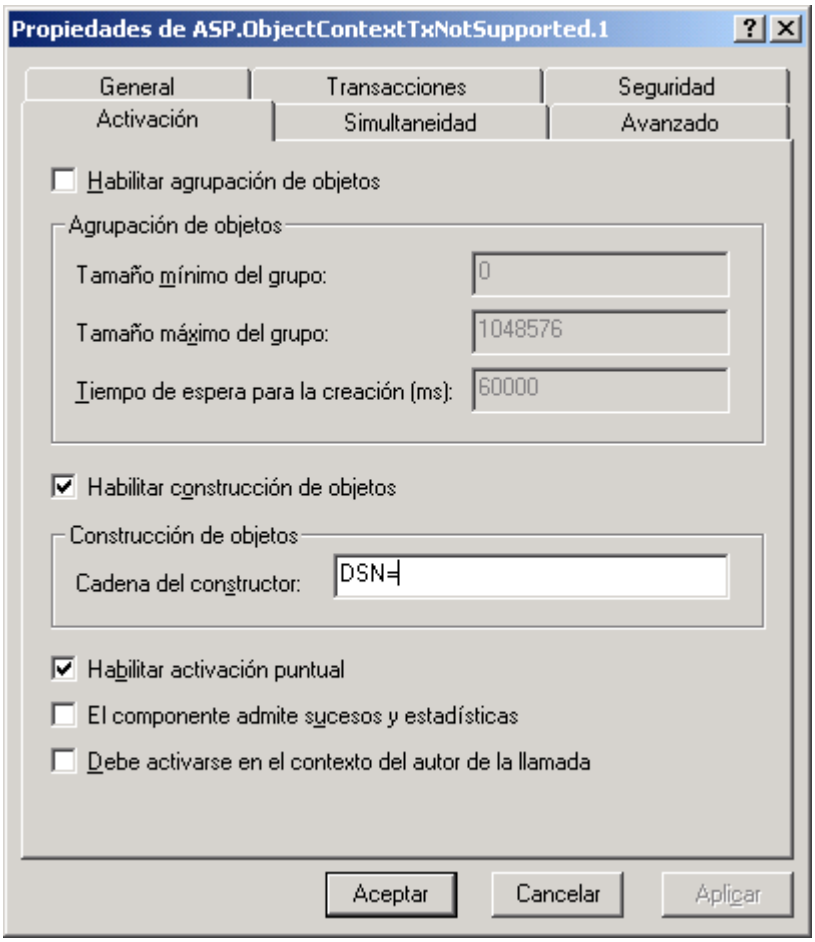


Con respecto al *pooling* (tanto de objetos como de conexiones), se sigue comportando igual que en *MTS*, aunque ahora si se soporta el *pooling* de objetos. No tendremos que hacer ningún cambio a la hora de desarrollar (exceptuando el hecho de cambiar a *Visual C++, Delphi,* o esperar a *Visual Basic* 7 para aprovechar todas las ventajas del *pooling*).

Con respecto a las transacciones, la diferencia reside en como crear los objetos. Antes, para que un objeto hijo actúe en la misma transacción que el padre (o creador), este debe crearse usando *CreateInstance* (en vez de *CreateObject* o *New*). Ahora, no es necesario usar *CreateInstance*: *CreateObject* nos sirve para crear objetos en cualquier situación.

Con respecto a las novedades, hay varias. Algunas están relacionadas con la gestión de eventos, y otras características avanzadas que escapan del ámbito de este articulo. En el *CD* se incluye un archivo con el nombre de una serie de artículos que están disponibles en la *MSDN*, y que deberían ser de obligada lectura. No incluimos las *URL's* de los mismos, porque *Microsoft* las cambia de vez en cuando. Lo mejor será que uséis la herramienta de búsqueda de la MSDN.

Pero hay una novedad que si resulta interesante: los constructores. En efecto, ahora es posible implementar el método *Construct* de la interfaz *ObjectConstruct* que se ejecutará durante el proceso de creación del objeto *COM+* (antes incluso de que salten los eventos *Class_Initialize* y *ObjectControl_Activate*)



Dicho método recibe una cadena como parámetro, que puede contener valores iniciales, una cadena de conexión, o lo que quiera que se nos ocurra....siempre que sea una cadena. Lo más importante: dicha cadena puede ser fijada en tiempo de ejecución. Basta con pinchar con el botón derecho sobre el componente, elegir la pestaña *Propiedades->Activation tab*, e introducir la cadena en el apartado *Constructor String*. Así podemos cambiar fácilmente los parámetros de inicio, sin tener que recompilar, ni recurrir al registro de la maquina para tales menesteres.

ANEXO E. ASP.NET

Debido a la amplia aceptación que ha tenido la tecnología ASP en el mundo de la programación de servidores web, son muchos los desarrolladores que la han incluido en su repertorio técnico, al tiempo que han proliferado cursos, libros, seminarios y eventos sobre la materia. Vamos a explicar las mejoras de ASP 3.0 (e IIS 5) respecto a la versión 2.0 (IIS4).

Server.Transfer

El uso del método *Redirect* para forzar la redirección a otra página, tiene el inconveniente de devolver al navegador del cliente la petición de redirección, lo que impide la utilización de CGIs remotos a los que el cliente no tiene acceso directo, sino a través de otro CGI que actúe como pasarela.

Otro problema, surge al ejecutar una redirección cuando previamente, en la misma página ASP se ha enviado contenido html al cliente, ya que se obtiene un error indicando que no es posible enviar encabezados HTTP al cliente, si posteriormente se realiza una redirección. La solución a esto consiste en activar el buffer con R*esponse.Buffer=True*, de manera que, en caso de no realizar la redirección, bastará con hacer un *Response.Flush* para enviar todo el contenido html almacenado en el buffer de salida. Si, finalmente se realiza la redirección, bastará con hacer un *Response.Clear*.

Además, con la redirección se pierde el contenido de la colección *Request*, y es necesario aislar las transacciones entre páginas. Es decir, una transacción debe ser iniciada, ejecutada y aceptada (o cancelada) en la misma página web.

Para salvar estas limitaciones, se ha introducido el método *Server.Transfer*, que no provoca un retorno de encabezados rápido. La única limitación de este nuevo método, es que sólo es posible realizar una transferencia a otra página que se encuentre en el mismo servidor web que la página que originó la transferencia.

Además, puesto que no es necesaria la intervención del cliente, ahora sí es posible hacer que una transacción abarque más de una página web, aunque no es algo muy recomendable, ya que las transacciones deben estar activas el menor tiempo posible.

Ejemplo de uso de Server.Transfer

ServerTransfer1.asp

```
<%
Response.Write ("Este texto lo generó página 1<BR>")
Server.Transfer("/path/ServerTRansfer2.asp")
%>
```

ServerTransfer2.asp

```
<%
Response.Write("Este texto lo generó página 2<BR>")
%>
```

Salida:

```
Este texto lo generó página 1
Este texto lo generó página 2
```

Server.Execute

Hasta ahora, la única forma de encapsular código ASP en un fichero para su ejecución como parte de otro, era con el uso de la directiva *#include*. El inconveniente de los *includes* es que no se pueden llamar dinámicamente, ya que el intérprete de ASP procesa, en primer lugar, las directivas #include, y luego el código ASP.

Por ejemplo, con el siguiente código, pretendemos que se incluya en la página actual el fichero 3.asp, y para ello, hemos generado una llamada a un include de forma dinámica. Sin embargo, la respuesta del compilador será un "Archivo Include no encontrado", puesto que intentará buscar un fichero llamado codigo.asp, que por supuesto, no existe.

```
<%
      codigo = 3
%>
<!-- #include file = <%=codigo%>.asp -->
```

Sin embargo, con ASP 3, sí es posible tener sentencias include generadas de forma dinámica. En concreto, el equivalente al ejemplo anterior sería:

```
<%
      codigo = 3
      Server.Execute(codigo & ".asp")
%>
```

Además, este método reduce el código de las páginas ASP que el servidor debe procesar, porque sólo se incluye en la página el código necesario, y no el de todo los includes.

Remove y RemoveAll

Tanto las variables de aplicación como, las de sesión, son accesibles mediante la colección *Contents*. Dicha colección devuelve todas las variables que están almacenadas en los objetos *Application* y *Session* respectivamente, junto con sus valores.

Sin embargo, hasta ahora no era posible liberar el espacio de memoria ocupado por dichas variables cuando no se deseaba usarlas más. Para permitir esto, se han añadido dos nuevos métodos a la colección *Contents*, *Contents.Remove* que permite eliminar una variable por su nombre, y *Contents.RemoveAll*, que elimina todas las variables de una vez.

Objeto ASPError

Para permitir un manejo de errores centralizado, se ha creado el objeto *AspError* y el método *Server.GetLastError*, que genera una concreción de este objeto. Sólo tiene sentido utilizar este nuevo objeto si se crean páginas de errores personalizadas que se ejecutan cuando se produce algún tipo de error previsto por el intérprete de ASP.

Esto ya era posible conIIS 4.0 mediante la modificación delas páginas de errores por defecto. Pero con IIS 5.0, el objeto AspError permite personalizar el texto del error que aparece en la página, usando las propiedades de dicho objeto. También es posible utilizar esta información para generar ficheros de logs más comprensibles.

Capacidades del navegador

Con las versiones 3.0 y 4.0 de IIS se incluye un componente que permite determinar las capacidades admitidas por el navegador a partir del análisis del encabezado HTTP correspondiente a la cadena user agent.

Pero esta información está limitada a una lista estática de propiedades, basadas en un fichero del servidor denominado browscap.ini. Es decir, IIS 4 detecta el nombre y la versión del navegador, y consulta dicho fichero para saber qué es lo que admite ese navegador en concreto. Así no hay forma de saber que admite un navegador posterior a la versión de IIS (a menos que actualicemos el fichero .ini).

Con IIS 5.0 e Iexplorer 5 se puede obtener la lista de propiedades que el cliente ha establecido en la configuración del navegador. Así, no sólo podemos saber si un navegador particular admite cookies. Ahora podemos saber también si el usuario desactivó o no el uso de cookies en su sistema, por ejemplo.

Includes con <script>

Hasta ahora, mediante la etiqueta <SCRIPT> era posible ejecutar código en el servidor especificando la propiedad RUNAT = "SERVER". Pero ahora, en vez de tener que incluir el código entre las etiquetas SCRIPT, se puede especificar el nombre del fichero que contiene este código, de manera que pueda ser referenciado desde otras páginas sin necesidad de incluirlo.

```
<SCRIPT RUNAT="SERVER">
Response.Write "Esto se interpreta como código ASP en el servidor."
</SCRIPT>
```

Y el código equivalente:

```
<SCRIPT RUNAT="SERVER" SRC="/include/browser.asp">
```

VBScript y Jscript 5.0

Se han añadido nuevas características a los lenguajes de script que Microsoft proporciona para la creación de páginas ASP. Cabe mencionar la inclusión de un parser de Expresiones Regulares para validaciones complejas y búsqueda de patrones, el uso de la sentencia With con objetos en VBScript, y se ha añadido la posibilidad crear de clases con la sentencia Class.

En Jscript se han incluido los bloques try y catch para el manejo de errores. Esto es muy importante, ya que posibilita controlar los errores de forma mucho más eficiente, y por supuesto, es mucho más potente que el control de errores de VBScript.

Tanto en VBScript como en en Jscript, se ha incorporado el soporte para DCOM que permite proporcionar el nombre del servidor como parámetro extra en la función CreateObject.

MDAC 2.5

La nueva versión de MDAC (Microsoft Data Access Components) incluida por defecto con IIS 5 (actualmente, la última versión es la 2.6) incorpora características para el tratamiento de datos en formato XML.

Ahora es posible almacenar un recordset en un fichero, gracias a los métodos Save y Open. A partir de un recordset se puede obtener su representación en XML, que puede ser almacenado en un fichero, en un objeto Stream, o bien en un objeto DOM y a la inversa. Para realizar esto se utiliza el método Save del objeto Recordset, que devuelve datos en formato XML a partir de un conjunto de registros.

BUFFER activo por defecto

Este cambio se ha hecho para impedir el envío de contenido al cliente mientras se procesa el código de una página ASP, a menos que se llame al método Response.Flush que fuerza el vaciado del buffer.

La utilización del buffer mejora el rendimiento, debido a que se reduce tanto el tráfico de red como los tiempos de respuesta entre cliente y servidor. Aunque esta opción está activada por defecto tras la instalación de IIS 5.0, es configurable para cada sitio web, al igual que lo era en la versión anterior.

IsClientConnected

El método Response.IsClientConnected permite detectar si el cliente se mantiene a la espera de la respuesta de una petición en curso, con el fin de poder abortar el procesamiento de la página en caso de que el cliente haya cancelado la petición, lo que permite liberar recursos que pueden ser utilizados por otras peticiones en espera. Este método ya existía en ASP 2.0, pero en ASP 3.0 no es necesario enviar previamente contenidoo al cliente antes de llamar al método IsClientConnected. Ahora se puede utilizar cuando mejor nos convenga, se haya enviado información al cliente, o no.

Los objetos COM se liberan antes

En IIS 4.0, ante una sentencia del tipo set objeto = nothing se libera la referencia a este objeto, pero no el objeto en sí, que permanece en memoria hasta que la ejecución de la página se ha completado.

En cambio, en IIS 5.0 es posible liberar el objeto antes de que termine la ejecución de la página, mediante la asignación del valor nothing, excepto si se produce alguna de las siguientes condiciones:

El objeto usa el evento OnEndPage.

El contador de referencias para el objeto no es cero.

HTML con extensión .ASP

La ventaja de asignar la extensión .ASP a todos los ficheros que componen una aplicación web radica en que no se necesitan modificar las referencias a ficheros con extensión .HTM o .HTML si se les añade código que deba ejecutarse en el servidor.

Sin embargo, el rendimiento se ve mermado debido a que el contenido de la página es previamente analizado en busca de código script que se ejecute en el servidor, y en caso de no existir, se devuelve la página tal cuál al cliente, sin ser evaluada línea a línea por el intérprete de ASP (ASP.DLL).

Ahora, con ASP 3.0, el intérprete detecta las páginas que sólo contienen código html, y no pierde tiempo en procesarlas. Por supuesto, no es tan eficiente como en el caso de una página con extensión .HTM o .HTML, pero el rendimiento es notablemente mejor que en la versión anterior, y elimina la tarea de renombrar ficheros.

Presente y futuro de ASP: ASP.NET

Este entorno consiste sintéticamente, en una arquitectura basada en servicios independientes de su implementación, a los que se accede a través de la utilización de estándares abiertos, como XML, siendo el ámbito de su explotación Internet. En el llamado .NET Framework se incluyen lenguajes como C#, VisualBasic.NET, y el entorno de desarrollo de aplicaciones web ASP.NET (formalmente ASP*) del que nos ocuparemos en este número.

Aunque ASP.NET hereda muchos de los elementos introducidos en las versiones anteriores de ASP, no se trata de una versión que añade mejoras sobre la actual versión 3, sino de una plataforma construida desde cero, orientada al

desarrollo de aplicaciones web empresariales basadas en un nuevo modelo de programación más robusto y flexible.

Para empezar a trabajar con ASP.NET sólo es necesario instalar ASP.NET Premium Edition sobre Windows 2000 o Windows NT 4.0, con el Service Pack 6ª que se puede descargar de la web oficial de ASP.NET de Microsoft en http://www.asp.net

Incremento del rendimiento

En una página ASP 3, es habitual encontrar código HTML entremezclado con código escrito en alguno de los lenguajes de script soportados por ASP, a saber: VBScript o Jscript. Esto tiene dos consecuencias no deseables.

La primera es que el código resultante es código "espagueti", difícil de entender. La segunda es que al ser el código interpretado, se realiza una llamada al intérprete de ASP por cada bloque de código encontrado. Estas repetidas llamadas al intérprete de ASP tienen un efecto negativo en el rendimiento del servidor web. En ASP.NET este efecto se minimiza debido a que las páginas ASP se compilan la primera vez que se solicitan, y en las sucesivas llamadas se ejecutan directamente.

Hasta ahora, para que una página ASP utilizara funciones de la lógica de negocio de la aplicación web, era necesario, o bien crear módulos en ASP que después serán incluidos en otros ficheros, o bien emplear objetos COM, lo que degrada el rendimiento de la aplicación cuando el número de objetos es elevevado. Esto ya no será necesario, pues es posible crear módulos de clase compartidos por otras aplicaciones web.

Además, la nueva plataforma de desarrollo completamente compatible con la sintaxis y el procesamiento del código de las versiones anteriores de ASP siendo tan sencilla la migración como cambiar la extensión de los ficheros .asp a .aspx para que el código se ejecute en la nueva plataforma.

Web Forms

El intercambio de datos entre un usuario que accede mediante un navegador y la aplicación web del servidor se realiza principalmente a través de los formularios de HTML definidos por las marcas <FORM></FORM>.

Con ASP.NET se introducen los llamados Web Forms, o formularios Web que tienen como principal características la de incorporar un modelo de programación similar al empleado por lenguajes de programación como Visual Basic, en lo referente a la implementación de formularios. Los formularios Web aportan las siguientes ventajas:

- Proporcionan un modelo de programación basado en eventos.
- Permite la separación entre el código HTML y el código de la lógica de la aplicación siendo éste compilado y escrito en cualquier lenguaje soportado por la plataforma .NET.
- Desarrollo Rápido de Aplicaciones (RAD) mediante la herramienta de desarrollo Visual Studio .NET que incorpora controles para formularios en tiempo de diseño.
- Soporta un amplio conjunto de controles y componentes .NET que ofrecen un modelo de objetos con consistencia de tipos.

En sustitución de las marcas HTML empleadas en los formularios tradicionales, en ASP.NET existen los llamados Server Controls (Controles de Servidor) que se utilizan para la creación de los elementos activos de un formulario. Se distinguen dos tipos de controles: controles HTML y controles Web.

Controles HTML

Este conjunto de controles se definen en el espacio de nombres System.Web.UI.HtmlControls de ASP.NET y derivan directa o indirectamente de la clase base HTMLControl.

Los controles HTML tienen su marca equivalentes en HTML, es decir, que todos los controles web se traducen a su correspondiente marca al ser enviados por el servidor web al navegador. Los controles HTML se definen mediante la propiedad runat="server" (restringida en ASP 3.0 a la etiqueta <object> y <script>), lo que permite transformar cualquier formulario HTML en un formulario definido mediante controles HTML, simplemente añadiendo esta propiedad. En la tabla vemos la relación de las marcas HTML con el control HTML equivalente.

Equivalentes HTML – HTML controls	
<a>	HTMLAnchor
<button>	HTMLButton
<select>	HTMLSelect
<textarea>	HTMLTextArea
<input type="check">	HTMLInputCheckBox
<input type="radio">	HTMLInputRadioBox
<input type="text">, <input type="password">	HTMLInputText
<input type="hidden">	HTMLInputHidden
<input type="image">	HTMLInputImage
<input type="file">	HTMLInputFile
<form>	HTMLForm
<img>	HTMLImage
<table>	HTMLTable
<tr>	HTMLTableRow
<td>	HTMLTableCell
Cualquier otra marca como , , etc.	HTMLGenericControl

¿Qué ventaja aportan los controles HTML al empleo de marcas HTML tradicionales? La respuesta es la manipulación de los controles como objetos. Hasta ahora, no existía la manera de acceder a los atributos de los elementos de un formulario desde ASP, sino que había que generar dichos atributos dinámicamente mediante programación. Con los controles HTML podemos hacerlo referenciando los controles como objetos, tal y como se puede ver en el ejemplo:

Antes

```
        <SELECT NAME="provincias">
        <%
              While Not rs.EOF
              Response.Write  "<option  value='"&  rs("provincia").
Value &  "'>"
              Wend
        %>
        </SELECT>
```

Ahora

```
        <%
              While Not rs.EOF
              Provincias.Add(rs("provincias").Value)
              Wend
        %>
```

```
<SELECT ID="Provincias" runat=server>
</SELECT>
```

El objetivo, pues, de los controles HTML es el de mejorar el modelo de objetos de ASP extendiéndolo a las marcas HTML.

Controles Web

El segundo tipo de controles de servidor, se define dentro del espacio de nombres System.Web.UI.WebControls de ASP.NET, y deriva directa o indirectamente de la clase base WebControl. Se trata de controles de más alto nivel de abstracción y proporcionan las siguientes ventajas:

- Un modelo de objetos más rico y consistente. La clase base WebControl implementa un conjunto de propiedades común a todos los controles además de un modelo de objetos con una definición robusta de tipos de datos.
- Detección automática del navegador. Los controles web detectan el tipo de navegador del cliente y generan el código HTML que mejor se adapta a la versión de HTML que el navegador es capaz de reconocer.
- Enlace de datos. Los controles disponen de propiedades que permiten asociarlos a fuentes de datos sin realizar ninguna programación específica.

El formato de declaración de los controles web usa los espacios de nombres con un prefijo para hacer referencia al espacio de nombres del componente, y el resto para definir la clase del control. Al igual que los controles HTML, es necesario el atributo runat="server". Por ejemplo:

```
<asp:TextBox id="nombre" runat="server" Text="Introduzca su
"nombre"> </asp:TextBox>
```

En este caso, asp es el prefijo para el espacio de nombres System.Web.UI.WebControls, como vemos en la siguiente tabla.

Controles web incluidos con la beta 1 de ASP.NET	
Label	Etiqueta.
TextBox	Caja de texto. Una sola línea, múltiples líneas y password.
CheckBox	Casilla de verificación.
RadioButton	Botones de selección excluyente.
DropDownList	Lista desplegable.
ListBox	Lista desplazable.
CheckBoxList	Grupo de casillas de verificación.
RadioButtonList	Grupo de botones de selección excluyente.
Button	Botón para enviar un formulario al servidor.
LinkButton	Botón con apariencia de enlace.
ImageButton	Botón con apariencia de imagen.
HyperLink	Enlace para navegar entre URLs.
Image	Muestra una imagen.
Panel	Agrupa otros controles.
Table	Define una tabla junto con los controles TableCell y TableRow.
Calendar	Calendario.
Repeater	Lista de datos sin apariencia visual.

DataList	Lista de datos con apariencia visual.
DataGrid	Rejilla de datos.
RequiredFieldValidator	Valida que el usuario rellena el control indicado.
RangeValidator	Valida que los datos de entrada están dentro de un rango.
CompareValidator	Compara la entrada en un control sobre otro.
RegularExpressionValidator	Valida la entrada del usuario mediante expresiones regulares.
CustomValidator	Permite establecer la lógica de validación tanto en el cliente como en el servidor.
AdRotator	Muestra banners configurados mediante ficheros XML.

En definitiva, los controles web facilitan al programador de aplicaciones web la creación de interfaces de usuario teniendo en cuenta que pueden construirse controles más complejos a partir de los más básicos, e incluso hacer uso controles desarrollados por terceras partes, al igual que ocurre en otros entornos de programación.

Servicios Web

Los servicios Web forman parte de la plataforma .NET de Microsoft, y como ASP.NET es parte integrante de ella, se ha diseñado para que pueda usar los servicios web.

Un servicio Web no es más que la lógica programable de una aplicación accesible a través de protocolos web estándar. Uno de estos protocolos es SOAP (Simple Object Access Protocol) propuesto al W3C (World Wide Web Consortium), como un estándar por varios fabricantes de software, como IBM, Microsoft, Lotus, etc. Se basa en dos tecnologías estándar, como son HTTP para el transporte, y XML para representar los datos, por lo que los detalles de implementación de los servicios son trasparentes a los consumidores de los mismos, sólo hace falta "entender" HTTP y XML.

El modelo de uso de un servicio web en ASP.NET es el siguiente: un servicio web expone lógica de la aplicación y, o bien otro servicio, o bien una página web, consume dicho servicio accediendo a dicha lógica. El siguiente ejemplo muestra la definición de un servicio web muy simple:

```
<%a WebService Language="VB" Class="Sumador"%>
Imports System.Web.Services
Public Class Sumador
Public Function <WebMethod()>Suma(a As Integer,b As
Integer) As Integer
      Return a + b
End Function
End Class
```

Lo primero que aparece, es una directiva WebService que define tanto el lenguaje utilizado, (en este caso Visual Basic) como el nombre de la clase (en este caso Sumador), e indica ASP.NET que el código de la clase contiene la lógica de la aplicación. Otro aspecto relevante es el uso del atributo <WebMethod()> que se utiliza para indicar que el método va a ser llamado desde web de manera que ASP.NET realizará el trabajo necesario para que el método soporte SOAP. Para que este código sea reconocido como un servicio web, debe almacenarse en un fichero con extensión .asmx

Veamos el código de una página ASP que consuma el servicio web que se ha definido anteriormente:

```
<HTML>
<HEAD>
<Script runat="server">
      Public Sub Suma
      Dim MiSumador As New Sumador
      Dim Suma As Integer

      Suma = MiSumador.Suma(Numero1.value, Numero2.value)
      Resultado.value = suma
      End Sub
</Script>
</HEAD>
<BODY>
<FORM runat="server" actino="sumador.aspx" meted="POST">
<INPUT runat="server" type="text" id="Numero1">
+
<INPUT runat="server" type="text" id="Numero2">
=
<INPUT runat="server" type="text" id="Resultado">
<INPUT   runat="server"   type="submit"   value="Calcular"
OnServerClick="Suma">
</FORM>
</BODY>
</HTML>
```

Para probar este ejemplo, necesitaríamos realizar lo siguiente:

Llamamos desde el navegador al fichero que contiene la definición del servicio web, por ejemplo:

```
http://localhost/webservicedemo/sumador.asmx?SDL
```

y obtendríamos un documento XML con la definición del SDL (Service Definiction Language).

Convertimos este documento en un proxy, es decir, se le añade la lógica necesaria para que pueda ser llamado mediante el protocolo SOAP. Utilizamos las herramientas de línea de comandos del .NET SDK.

```
WebServiceUtil.exe /command:proxy
Sumador.sdl /language:VB
```

Y finalmente compilamos el fichero resultante:

```
vbc /t:library /r:system.web.services.dll
/r:system.xml.serialization.dll
```

Este fichero debe estar en un directorio /bin del servidor web donde se encuentra la página que consume el servicio.

Implantación

Pasar de una versión de desarrollo a producción, a veces puede suponer un verdadero calvario. Pero ASP.NET nos facilita la vida en este aspecto. Ahora basta con copiar los ficheros (si, un simple copy) en la máquina de producción, respetando una determinada estructura de directorios (por ejemplo, las DLL's deben de estar en el directorio /bin). No hace falta registrar ningún objeto, ni parar ningún servicio. Simplemente se copian las páginas web y las dll's al servidor.

Puesto que cada sitio usa sus propias DLL's, (las que están en su directorio /bin), es posible tener diferentes sitios web usando diferentes versiones de un mismo componente, sin ningún tipo de problemas.

Otra novedad es que, por fin, se termina el "infierno de las DLL's". Instalar una nueva DLL es tan fácil como copiar y sobrescribir la antigua. No hay bloqueos que impidan la copia, y no hay necesidad de reiniciar la máquina.

Por último, el método para desinstalar una aplicación web es simple pero eficaz: deltree /s.

Configuración

En ASP 3.0, la mayoría de los parámetros de un sitio web estaban almacenados en la metabase. Ahora, muchos de estos parámetros es encuentran en un fichero de configuración, llamado config.web, que contiene datos en formato XML.

Además de ser fácilmente comprensible, esto permite llevar a límites insospechados el nuevo modelo de instalación basado en copy & run. Y esto no es todo: estos ficheros son heredables. Podemos definir una configuración genérica para todos los sitios, e ir afinando cuando sea necesario. Además, futuros cambios en el fichero de configuración son detectados automáticamente por el servidor, sin que sea necesario reiniciar la máquina o el servicio web.

Estado de la sesión

Uno de los mayores problemas de ASP viene de una de sus mayores facilidades. En efecto, el uso de variables de sesión facilita mucho la programación de sitios web, pero tiene graves consecuencias en la escalabilidad, sobre todo si se intentan montar granjas de servidores.

Sin embargo, ASP.NET de nuevo nos facilita las cosas, ya que es capaz de mantener el estado incluso entre máquinas, con lo que montar granjas de servidores deja de ser un problema: cada petición de datos dentro de una misma sesión de usuario puede ser atendida por una máquina diferente sin problemas.

Gestión de la caché

Todos sabemos que el uso de la caché puede acelerar enormemente nuestras aplicaciones. ASP.NET proporciona 3 tipos de caché. En primer lugar, la caché de toda la vida: es capaz de almacenar las páginas más comunes en memoria.

En segundo lugar, permite almacenar en caché parte de las páginas, siendo el resto generada de forma dinámica.

En tercer lugar, proporciona un API que permite a los desarrolladores incluir cualquier tipo de objetos en la caché. Además esta caché es capaz de notificar cambios en las fuentes de datos que impliquen un refresco de los contenidos. También es posible hacer que los contenidos de la caché expiren en un momento determinado. Incluso proporciona mecanismos de ahorro de memoria, limitando lo que se guarda en caché según la carga.

Seguridad

Además de admitir varios tipos de autenticación, se introduce el concepto de roles. Los roles agrupan a los usuarios en grupos lógicos. A cada uno de dichos grupos se le pueden asignar diferentes permisos. En tiempo de ejecución se puede comprobar fácilmente si un usuario pertenece o no a un determinado rol:

```
User.IsInRole("Ventas")
```

Además, es posible incluir en el fichero config.web información acerca de la seguridad. Podemos especificar qué usuarios o roles pueden acceder a unos directorios o páginas, u otros. Si incluimos la seguridad en dicho fichero, y prescindimos de la seguridad de NT (las famosas ACL), estaremos favoreciendo aún

más el modelo copy & run, ya que con copiar la aplicación a otra máquina, tendremos configurado todo lo relativo a seguridad.

ESTADO DEL ARTE EN LA Generación De Contenido Dinámico Para La Web..1

Breve introducción al protocolo HTTP....2
Solicitud HTTP....5
Respuesta HTTP....9

Evolución de tecnologías para la generación de contenido dinámico para la web..11

Common Gateway Interface. CGI....11
Introducción....11
Utilidades de los CGIs....11
Cómo funciona un CGI....12
Programando CGIs....16
Scripts CGI en Perl....16
Ejemplos de uso....19
Referencias....22

ColdFusion....23
Introducción....23
Cómo funciona CFML (Cold Fusion Markup Language)....23
Programando Cold Fusion Markup Language (CFML) y ejemplos de uso....24
Referencias....42

Active Server Pages. ASP....43
Introducción....43
Cómo funciona ASP....43
Programando con ASP....44
Ejemplos de uso....52
Alojamiento ASP....53
Referencias....53

WebClass....55
Introducción....55
Cómo funcionan las WebClasses....55
Programando WebClass....55
Añadir plantillas HTML....57
Sustitución de etiquetas....57
Objetos WebItem....58
Direcciones de enlace....59
WebItem personalizados....60
Incluir parámetros adicionales....60
Eventos propios....60
El objeto WebClass....61
Aplicación y sesiones....61
Uso de cookies....62
Datos de formularios....63
El objeto WebItem....63
Referencias....64

Server-Side JavaScript. SSJS....65
Introducción....65
Cómo funciona Server Side JavaScript....65
Programando una aplicación SSJS....68

El comportamiento de los operadores de comparación....69
Prototipos....69
Uso....69
Compilando una Aplicación....73
Instalando una nueva Aplicación....74
Ejemplos de uso....75

Hipertext PreProcessor PHP 77
Introducción....77
Breve historia de PHP....77
Cómo funciona un PHP y el intérprete Zend....77
Configuración de Apache + PHP + mySQL....78
Programando un PHP....82
Ejemplos de uso....87
Referencias....90

Java Servlets 92
Introducción....92
Cómo funciona un servlet....92
Inicialización....93
Proceso de las peticiones....93
Terminación....93
Ciclo de vida de un servlet....93

JavaServer Pages (JSP)....100
Introducción....100
Cómo funciona JavaServer Pages (JSP)....100
Programando JavaServer Pages (JSP)....104
Directivas JSP....107
Directiva Page....107
Directiva Include....108
Declaraciones JSP....109
Scriptlets JSP....109
Expresiones JSP....111
Etiquetas JSP Avanzadas....112
Acciones JSP....112
Acciones estándar JSP....115
<jsp:useBean>....115
<jsp:setProperty>....118
<jsp:getProperty>....119
<jsp:include>....120
<jsp:forward>....120
<jsp:param>....121
<jsp:plugin>....121
Empleo de JavaBeans en páginas JSP....123
Ejemplos de uso....123
Referencias....128

FastCGI (Fast Common Gateway Interface) 129
Introducción....129
Ventajas de FastCGI....129
Roles de Fast CGI....130

Aplicaciones Responder 130
Aplicaciones Filter....130
Aplicaciones Authorizer 131
Cómo funciona FastCGI....131
Estructura del código....132
Inicialización de Variables de Entorno....132
Variables de Entorno Por-Request....132
Detalles de Implementación....133
La librería fcgi_stdio: Compatibilidad I/O....134
La librería fcgi_stdio: Compatibilidad Binary....134
Desarrollando aplicaciones FastCGI en C....135
Las librerías I/O....135
Estructura del código....135
Construcción....136
Desarrollando aplicaciones FastCGI en Perl....136
Desarrollando aplicaciones FastCGI en Java....137
Instalación sobre Apache de Scripts FastCGI....137
Ejemplos de uso....137
Referencias....140

Servidor Zope (Z Object Publishing Enviroment) 142
Introducción....142
Cómo funciona Zope....142
Zope ORB....143
La creación de un web con Zope....143
El portal de una empresa....143
Primeros pasos en la creación del web....145
Control de acceso....150
Ejemplos de uso....153

PMZ (Poor's Man Zope)....155
Introducción....155
Utilidades de PMZ....155
PMZ vs Zope....155
Cómo funciona PMZ....155
Creación de scripts CGI....157
Conclusión....157
Instalación de PMZ....157
Ejemplos de uso....158
Referencias....159

Otras tecnologías....160
DLL ISAPI....160
Max Server Pages....161
PHPNuke 5.0....161
Server Side Includes....162

Conclusiones....164

ANEXO A. Ejemplo con las diversas tecnologías....166

Explicación del ejemplo:....166

CGI.....168
Cold Fusion.....170
ASP.....170
WebClass.....172
PHP.....173
Servlets.....174
JavaServer Pages.....176

ANEXO B. Conceptos Básicos de Redes 179
Protocolo de Comunicaciones.....179
Capas de Red.....179
Cliente y Servidor.....180
IP, Internet Protocol.....181
UDP, User Datagram Protocol.....182
Dirección IP.....182
Dominios.....184
Puertos y Servicios.....185
Cortafuegos.....185
Servidores Proxy.....185
URL, Uniform Resource Locator.....186

ANEXO C.Instrucciones SQL.....187
Select.....187
Delete.....187
Insert.....187
Update.....187

ANEXO D.Diferencias con COM+.....188

ANEXO E. ASP.NET.....190
Server.Transfer.....190
Server.Execute.....191
Remove y RemoveAll.....191
Objeto ASPError.....191
Capacidades del navegador.....191
Includes con <script>.....192
VBScript y Jscript 5.0.....192
MDAC 2.5.....192

BUFFER activo por defecto.....192
IsClientConnected.....193
Los objetos COM se liberan antes.....193
HTML con extensión .ASP.....193
Presente y futuro de ASP: ASP.NET.....193
Incremento del rendimiento.....194
Web Forms.....194
Controles HTML.....195
Controles Web.....196
Servicios Web.....197
Implantación.....198
Configuración.....199
Estado de la sesión.....199
Gestión de la caché.....199
Seguridad.....199

Access.....44
Active Server Pages.....43
ActiveX.....55
Altavista.....11
Apache.....77
API…24, 92, 96, 97, 110, 112, 130, 135, 160, 168, 188, 199
aplicación.....48
Application Manager.....75
ASP.....43
ASP.NET.....194
bloqueado.....48
CFML.....23
CGI.....11, 13, 92, 96, 97, 129, 130, 131, 132, 133, 134, 135, 136, 137, 139, 156, 157, 159, 160
Chilisoft.....43
Cliente/Servidor.....9
ColdFusion.....23
COM.....24
compilador jsac.....73
cookie.....47
COPY.....6
CORBA.....99
DCOM.....24
DELETE.....6
directiva JSP.....107
DLL ISAPI.....44
DTML.....154
Esteban Trigos.....90
etiqueta SERVER.....72
Fast Common Gateway Interface.….129
FastCGI.....129
FastCGI-savy.....137
Frontpage.....44
FTP.....142
GET.....6
HEAD.....6
HTTP.....1
HTTP/1.1.....2
HttpServlet.....95
HyperText.....2
IMAP.....77
IP 2
Java.…11, 16, 53, 66, 71, 77, 78, 82, 83, 92, 96, 97, 98, 99, 100, 103, 104, 107, 109, 110, 111, 112, 113, 115, 120, 121, 123, 124, 128, 137, 138, 143, 164, 165, 168, 174, 179, 181, 182, 184, 185
JavaBean.....123
JavaMail.....99
JavaServer Pages.....100
JDBC.....11, 99
JDK 1.2.....92
JSP.....100
Lincoln D. Stein.....16
LINK.....6
LiveConnect.....70
LoopBack.....79
Max.....161
Microsoft IIS.....77
MIME.2, 7, 8, 9, 15, 16, 18, 97, 98, 108, 130, 131, 148
MOVE.....6
MySQL. .77, 90, 91, 161, 164, 166, 167, 168, 170, 171, 172, 173, 174
Naba Barkakati.....22
Netscape.....1, 65, 66, 67, 77, 122
Netscape Enterprise Server.....65
ODBC.....44
OPTIONS.....7
ORB.....143
O "Reilly.....22
PATCH.....6
Perl.....16
PERL.....16
Personal Web Server.....43
PHP.....77
PHP/FI.....77
PHPNuke.....161
PMZ.....155
POST.....6
ProcessTag.....60
PUT.....6
Python.....143
QUERY_STRING.....15
Rafael Morales.....91
Rasmus Lerdorf.....77
RFC.....10
RMI.....99, 128, 164
Robert Orfali.....9
Scriptlet.....110
Scriptlets.....111
SERVER.....70
Servlets.....92
sesión.....48
SMTP.....24
SNMP.....77
SSJS.....65

Steven Brenner.....17
Sun.....92
TCP/IP.....2
TELNET.....2
Terminal.....3
TRACE.....6
UNIX.....23
UNLINK.....6
URL. 5, 6, 7, 8, 9, 13, 14, 15, 16, 32, 33, 34, 36, 41, 43, 65, 93, 95, 96, 97, 98, 105, 108, 110, 120, 121, 122, 130, 131, 145, 146, 186
URLData.....60
URLFor.....60
WebClass.....55
WebItem.....59
WebPlantillas.....59
WRAPPED.....7
Yahoo.....11
Zend.....77
Zope.....142