FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA
"Tres décadas de orgullosa excelencia" 1971 - 2001

CURSOS INSTITUCIONALES

ACTIVE SERVER PAGES

Del 13 AL 24 de Agosto de 2001

APUNTES GENERALES

Yazmin Vega Hernández
Jonathan Gamboa Beltrán
Palacio de Minería
Agosto /2001

`CSng()`: Convierte el argumento en un tipo Single, como en los casos anteriores el argumento debe comprender un rango entre –3,402823 E38 y –1,401298 E-45 para números negativos y entre 1,401298 E-45 y 3,402823 E38 para números positivos.

`CStr()`: Convierte el argumento en tipo String. En este caso tiene diferentes salidas, dependiendo del tipo de dato que reciba:

- Cualquier tipo numérico devolverá el número escrito en String.
- Boolean devuelve "True" o "False" escrito como cadena.
- Date regresa la fecha escrita en una cadena en formato de fecha corta: "dd/mm/aa"
- Error devolverá un String con la palabra Error seguido del número de error.
- Empty devuelve la cadena ""
- Null causa error.

2.1.7 Matrices.

Las matrices son variables que almacenan elementos de un mismo tipo y a los que se puede acceder a través de un índice. Una de las ventajas de VBS es que además de las matrices que nos ofrecen otros lenguajes, las unidemencionles y n-dimesionales están las matrices de longitud variable lo cual hace el uso de estas mucho mas flexibles.

- **Matrices unidimensionales**

  Se declaran de la misma forma que las variables vistas anteriormente, con la única diferencia que se le agrega el tamaño del arreglo entre paréntesis después del nombre.

```
Dim nombre_de_la_matriz(rango)
```

Es importante no perder de vista que el rango de la matriz deberá ser el número de elementos menos uno ya que esta empezará a numerar desde cero. De esta forma la siguiente matriz tendrá 4 elementos:

```
Dim gatos(3)
```

- **Matrices n-dimensionales**

Además de las matrices unidimensionales es posible trabajar con matrices de hasta 60 dimensiones, aunque no es muy recomendable una matriz tan grande ya que ocupa un espacio muy grande en memoria y resulta difícil de comprender lo que son 60 dimensiones por lo que su manejo quizá no sería tan sencillo.

Así que para declarar una matriz de mas de una dimensión la sintaxis será muy parecida a la de una unidimensional, pero agregando cada dimensión nueva dentro del mismo paréntesis separada por una coma de la anterior.

```
Dim matriz(dimension_x,dimension_y,dimension_z,.....)
```

Por ejemplo nuestra matriz será de 2x3

```
Dim tablas(1,2)
```

- **Matrices Dinámicas**

Como comentamos al principio en VBS se nos da la facilidad de trabajar con matrices dinámicas para esto necesitamos declarar una matriz como lo hacemos en el caso de las anteriores, sin embargo no se le **debe** asignar ninguna dimensión.

```
Dim matriz()
```

Sin embargo antes de poder utilizar nuestra matriz será necesario asignarle su dimensión (es) utilizando la palabra REDIM.

```
REDIM matriz(x,y,z,w,...)
```

Esta instrucción la podremos utilizar cuantas veces queramos y en los lugares que necesitemos, para cambiar la dimensión de nuestra matriz, sin embargo es importante considerar que cada vez que la ocupemos los

elementos almacenados en ella se perderán a menos que la redimensionemos con la instrucción PRESERVE.

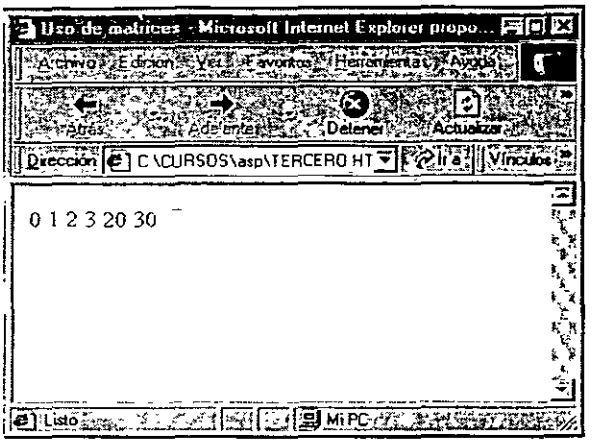
```
REDIM PRESERVE matriz (x,y,z,w,..)
```

En este caso no se pierden los datos que queden en la matriz en el rango establecido, los que se vean afectados se perderán.

Ejemplo.

```
<HTML>
    <HEAD><TITLE>Uso de matrices </TITLE></HEAD>
    <BODY>
        <SCRIPT LANGUAGE="VBScript">
        <!-
        DIM NUMEROS()
        REDIM NUMEROS(3)
        FOR I=0 TO 3
            NUMEROS(I)=I
        NEXT
        REDIM PRESERVE NUMEROS(5)
        NUMEROS(4)=20
        NUMEROS(5)=NUMEROS(4)+10
        FOR EACH I IN NUMEROS
            Document.Write I & " "
        NEXT
        -- >
        </SCRIPT>
    </BODY>
</HTML>
```

El resultado sería el siguiente:



Nota: El código de VBScript se encuentra entre <!-- y --> por precaución. Si algún navegador viejo abre nuestra pagina no intentará interpretar el código de VBScript, ya que lo tomara como un comentario, y de esta forma evitaremos que marque errores.

2.1.8 Constantes.

Puede ser que necesitemos tener una variable que no cambie su valor durante toda la ejecución del programa para ello tenemos a las constantes, si en alguna parte del código se intenta alterar ese valor generará un error, para declarar una constante se hace de la siguiente forma:

```
CONST nombre_de_la_constante
```

Una vez declarada la constante le podremos asignar un valor una sola vez.

```
Nombre_de_la_constate=valor
```

2.2 Control de flujo.

2.2.1 If..then.

Algunas veces es necesario tomar decisiones en el flujo de nuestro programa, dependiendo de las acciones del usuario. Cuando solamente tenemos dos caminos a elegir podemos utilizar la sentencia If... then que nos permite elegir un camino de ejecución dependiendo del valor de una cierta condición booleana.

Entonces su sintaxis quedaría como:

```
If condición then
     Instrucciones si la
     Condición es verdadera
End if
```

2.2.2 If... then...else.

Con esta instrucción tenemos la opción de tomar por dos caminos diferentes, es decir si se cumple determinada condición booleana entonces se llevará a cabo lo que esta después del then y antes del else, en caso contrario se ejecutará lo que esta después del else y antes del end if.

Entonces su sintaxis será:

```
If condición then
     Instrucciones si la
     Condición es verdadera
Else
     Instrucciones si la
     Condición es falsa
End if
```

En el ejemplo de los msgbox se maneja un ciclo de control If..then..else.

```
If I=1 then
Document.Write "Ha seleccionado <B>Aceptar</B>"
Else
Document.Write "Ha seleccionado <B>Cancelar</B>"
End if
```

2.2.3 Select...Case.

Como se puede ver, si tenemos varias opciones el uso de If resulta muy engorroso es por esto que existe la sentencia Select-Case.

La estructura de esta expresión es la siguiente:

```
Select Case expresión
     Case valor1
          Instrucciones para el valor1
     Case valor2
          Instrucciones para el valor1
     .
     .
     Case Else
          Instrucciones para el Else
End Select
```

Como se puede ver en esta estructura también hay una opción para el caso de que no se cumpla con alguno de los valores que esperamos, el caso Else.

2.2.4 For..Next.

La instrucción For..Next nos será de utilidad cuando necesitemos que una secuencia de instrucciones se repita por un numero determinado de veces, también encontraremos que este ciclo es uno de los mas útiles cuando tenemos que trabajar con matrices.

La sintaxis para esta instrucción será:

```
For contador=valor_inicial to valor_final step incremento
     Intrucciones
Next
```

Como se puede observar para determinar el numero de veces que se realizara la iteración se requiere un contador, este contador nosotros podemos decidir en que numero empieza, en cual termina, y en que cantidad necesitamos incrementar o

decrementar ese contador. Entonces se repetirán las instrucciones que se encuentren entre el For y el Next.

En el ejemplo de las matrices se ve el uso de esta estructura:

```
FOR I=0 TO 3
     NUMEROS(I)=I
NEXT
```

En el ejemplo podemos observar que no es necesario emplear la palabra Step ya que si no se usa se entiende que deseamos que el incremento sea de uno en uno.

2.2.5 For..Each.

VBS proporciona una instrucción de iteración para hacer mas fácil el trabajo con colecciones o matrices. Esta instrucción va a recorrer la colección o matrices con un contador, de manera similar que el For..Next de modo que por cada elemento de la misma se realicen las instrucciones que se indican en el bloque.

La sintaxis es la siguiente:

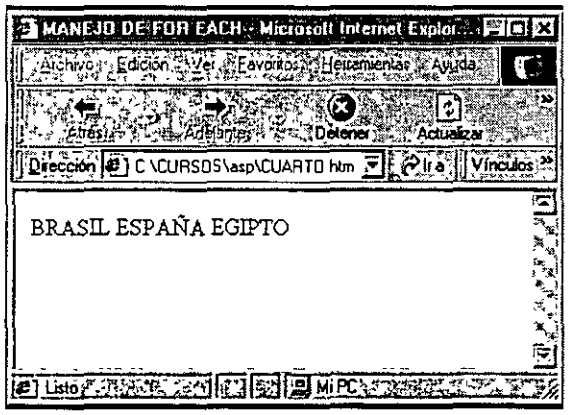
```
For Each contador In colección
     Instrucciones
Next
```

El uso de el contador en esta estructura es un poco diferente y muy útil ya que el contador en cada momento contiene el elemento actual.

Un ejemplo para el uso de esta estructura lo vemos a continuación:

```
<HTML>
<HEAD><TITLE>MANEJO DE FOR EACH</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
```

```
    <! - -
    DIM PAISES(2)
    PAISES(0)="BRASIL"
    PAISES(1)="ESPAÑA"
    PAISES(2)="EGIPTO"
    FOR EACH PAIS IN PAISES
        Document.write pais & " "
    NEXT
    -->
</SCRIPT>
</BODY>
</HTML>
```



2.2.6 Do While... Loop.

Algunas veces necesitaremos realizar un ciclo sin embargo no conocemos el número de veces que este se llevara a cabo, para esto existe la instrucción Do While... Loop la cual indica que las instrucciones que se encuentren entre Do While y Loop se repetirán mientras cierta condición booleana sea verdadera.

La sintaxis para la estructura de control es:

```
Do While condición
    Instrucciones
Loop
```

En este caso la condición será evaluada antes de ejecutar por primera vez las instrucciones dentro del bloque. También existe otra forma de escribir la sentencia de tal forma que garanticemos que el bloque se ejecutará por lo menos una vez.

```
Do
    Instrucciones
Loop While condición
```

En este caso el bloque se realiza una vez si la condición es verdadera, al llegar al final de este, entonces el bloque se repite, hasta que sea falsa.

2.2.7 Do Until... Loop.

También puede ser que necesitemos que el bloque se ejecute hasta que una condición sea verdadera, el caso contrario al While, y también presenta dos formas de escribirlo, que se ejecute o no al menos una vez.
Si la condición desde un principio es verdadera el bloque no se realiza ni una sola vez a menos que se encuentre indicado así.
La sintaxis será:

```
Do Until condición
    Instrucciones
Loop
```

O bien:

```
Do
    Instrucciones
Loop While condición
```

3. FUNDAMENTOS DE ASP.

3.1 Funcionamiento básico de ASP.

El código de ASP intercalado en una página se debe de encerrar entre las etiquetas <% %> , separando cada sentencia en un par de ellas o encerrando bloques enteros en un solo par, este código no será escrito en HTML sino en el lenguaje de guiones que se haya decido usar, en este curso se utilizara exclusivamente VBScript .

Ejemplo:

Código ASP.

```
<%@ LANGUAGE=``VBScript'' %>
<HTML>
<HEAD>
<TITLE>EJEMPLO</TITLE>
</HEAD>
<BODY>
<%  for  i=1 to 3 %>
Este es el primer ejemplo
tr<BR>
<% next %>
</BODY>
</HTML>
```

Código HTML generado.

```
<HTML>
<HEAD>
<TITLE> EJEMPLO</TITLE>
</HEAD>
<BODY>
Este es el primer ejemplo
<BR>
Este es el primer ejemplo
<BR>
Este es el primer ejemplo
<BR>
</BODY>
</HTML>
```

En este ejemplo se utiliza la directiva `<%@ LANGUAGE=``VBScript'' %>` para indicar el lenguaje en el que se encuentra el código ASP almacenado en la pagina, en este caso es posible omitir esta directiva siempre que se utilice a VBScript como lenguaje, en caso contrario deberá de utilizarse para indicar el lenguaje usado.

Como se puede apreciar en este ejemplo se intercala el código de HTML con el de ASP para conseguir un resultado bastante interesante, se realiza una iteración entre un par de líneas de ASP separadas por código estático, este será tratado como parte del proceso de la iteración, y por lo tanto se repetirá sobre el documento tres veces, apareciendo el resultado como parte del código generado para ser enviado al cliente.

Este primer acercamiento a ASP nos permite comprender en parte su lógica y estructuración; El resultado de la ejecución del documento ASP en el servidor se escribe como parte del documento estático que será enviado al cliente.

3.1.1 El uso de variables en ASP.

Como en todos los sistemas el manejo de los datos es la parte central del uso de ASP, estos datos se operaran mediante constantes y variables, manipuladas según el lenguaje que se haya seleccionado para trabajar, solo que en este caso el alcance de las mismas presenta una fundamental diferencia de los programas comunes, hay que tomar en cuenta que las variables contenidas en el documento solo existen mientras el documento de respuesta esta siendo generado en el servidor, esto quiere decir que si nuestra aplicación ASP se encuentra fragmentada en diferentes documentos, los datos almacenados en las variables de cada uno de ellos no estarán disponibles en los demás. Para poder utilizar variables mas halla de la generación del documento se hará uso de los objetos integrados, mas adelante en el capitulo 4.

3.1.2 Escribiendo variables desde el código ASP.

Además de manipular el código estático desde ASP es posible escribir el contenido de variables, o expresiones en el documento que se enviara al cliente haciendo uso del operador = de la siguiente forma <%=expresion %>, esto puede ser de mucha utilidad para generar código de forma automática:

Ejemplo:

Código ASP.

```
<HTML>
<HEAD>
<TITLE>EJEMPLO 2</TITLE>
</HEAD>
<BODY>
<%  for  i=1 to 3 %>
<FONT SIZE=<%=i%>> Crece
</FONT>
<% next %>
 </BODY>
</HTML>
```

Código HTML generado.

```
<HTML>
<HEAD>
<TITLE> EJEMPLO</TITLE>
</HEAD>
<BODY>
<FONT SIZE=1> Crece
</FONT>
<FONT SIZE=2> Crece
</FONT>
<FONT SIZE=3> Crece
</FONT>
</BODY>
</HTML>
```

En este ejemplo además de generar el código mediante ASP, modificamos atributos dentro de las etiquetas HTML, escribiendo valores para ellos en tiempo de ejecución en el servidor.

La siguiente tabla resume las diferentes combinaciones en que se pueden utilizar los valores provenientes de variables o expresiones:

Consideremos <% contador = 10 %>

Sintaxis.	Resultado generado.
<% =contador %>	10
<% ="contador " %>	Contador
<% = "contador " & contador %>	contador 10

Tabla 3.1

Es importante hacer notar que cuando escribimos expresiones sobre el documento con esta técnica, lo que estamos haciendo es incrustar directamente código estático sobre el, la consecuencia mas importante de esto es que si nosotros incluimos etiquetas HTML en las cadenas que se escriban, estas pasaran a formar parte de la estructura del documento.

Ejemplo:

Código ASP.

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>EJEMPLO</TITLE>
</HEAD>
<BODY>
<%  for  i=1 to 3 %>
<%="Este  es  el  tercer
ejemplo <BR>"%>
<% next %>
</BODY>
</HTML>
```

Código HTML generado.

```
<HTML>
<HEAD>
<TITLE> EJEMPLO</TITLE>
</HEAD>
<BODY>
Este es el tercer ejemplo
<BR>
Este es el tercer ejemplo
<BR>
Este es el tercer ejemplo
<BR>
</BODY>
</HTML>
```

En este ejemplo se puede apreciar lo expuesto anteriormente, las cadenas que se escriben en el documento desde el código ASP forman parte de la

estructura de la página proporcionada al cliente, y las etiquetas contenidas en ellas serán interpretadas como parte de la estructura.

3.1.3 Utilizando funciones de VBScript en el código ASP.

Para el desarrollo de nuestras paginas Web dinámicas con ASP es posible utilizar todas las funciones que conforman la librería de VBScript como lo haríamos en cualquier aplicación. En el siguiente ejemplo se ilustra el uso de estas funciones :

Ejemplo:

Código ASP.

```
<HTML>
<HEAD>
<TITLE>EJEMPLO 3</TITLE>
</HEAD>
<BODY>
<%=''La hora es '' & Time %>
</BODY>
</HTML>
```

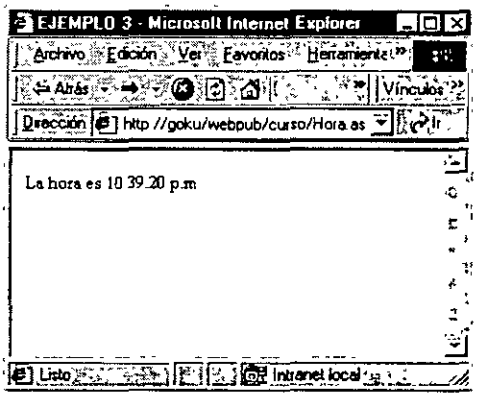


fig. 3.1 Resultado del ejemplo 3.

En este ejemplo se hace uso de la función `Time` que regresa la hora actual en el equipo sobre el que se ejecuto, en nuestro caso; el servidor.

En la figura 3.1 podemos apreciar el resultado de la ejecución de `Hora.asp`.

3.1.4 Utilizando Objetos y Componentes ASP.

Hasta el momento hemos creado aplicaciones utilizando exclusivamente las herramientas disponibles en el lenguaje de guiones seleccionado, como puede apreciarse, la funcionalidad de la tecnología ASP seria mínima si esta fuera la única opción disponible, como se mencionaba en capítulos anteriores, mas halla del lenguaje seleccionado para la creación de los documentos es posible utilizar objetos y componentes integrados a ASP, como parte de su librería para ampliar la funcionalidad de las aplicaciones, entre los ejemplos mas contundentes de la necesidad de ampliar la funcionalidad de ASP se cuentan los componentes para el acceso a archivos o bases de datos.

Estos componentes y objetos ASP son componentes ActiveX, la diferencia entre ellos es la integración con la plataforma de ASP que presentan:

Los **objetos** podrán ser invocados directamente en el código de la aplicación sin necesidad de crearlos explícitamente, entre estos objetos integrados se encuentran; **Application, Session, Request, Response y Server** .

Los **componentes** son librerías DLL no integradas dentro del entorno inmediato de ASP, estos se necesitan instanciar para su uso y pueden ser generados en cualquier lenguaje de programación, son utilizados para manejar correo electrónico, acceder a bases de datos, etc. Algunos de los componentes mas usados son **Ad Rotator** y **Content Linking**.

En este capitulo solo utilizaremos de forma inicial uno de los objetos mencionados anteriormente, algunos otros entre los mas útiles se examinaran a fondo en el capitulo 4.

3.2 Obtención de información enviada por los formularios.

Con ASP es posible procesar en diferentes formas la información proporcionada por el usuario, casi todas se basadas en el uso de formularios, el proceso puede realizarse bajo los siguientes esquemas:

3.2.1 Esquema de dos documentos.

En este esquema como su nombre lo indica intervienen dos documentos diferentes uno el primero que será enviado al usuario, donde se recabara la información en un formulario, y otro donde se procesara esa información para generar una pagina de respuesta.

Las ventajas de este sistema se aprecian en que para la primera parte del procedimiento el motor de ASP no intervendrá en la operación, con el consiguiente ahorro de recursos, pues no será hasta que se reciban los datos que se generara un nuevo documento.

En este esquema ocurre lo siguiente:

a) El usuario recibe una pagina HTML normal, la cual cuenta con un formulario para recabar información. El usuario rellena el formulario y pulsa el botón de submit.

b) Se envían los datos recabados del usuario al servidor, y que son recibidos por un documento ASP que los procesa para generar el documento que recibirá el usuario, o para registrarlo en una base de datos, etc.

c) Se regresan los resultados al cliente. (Una nueva página)

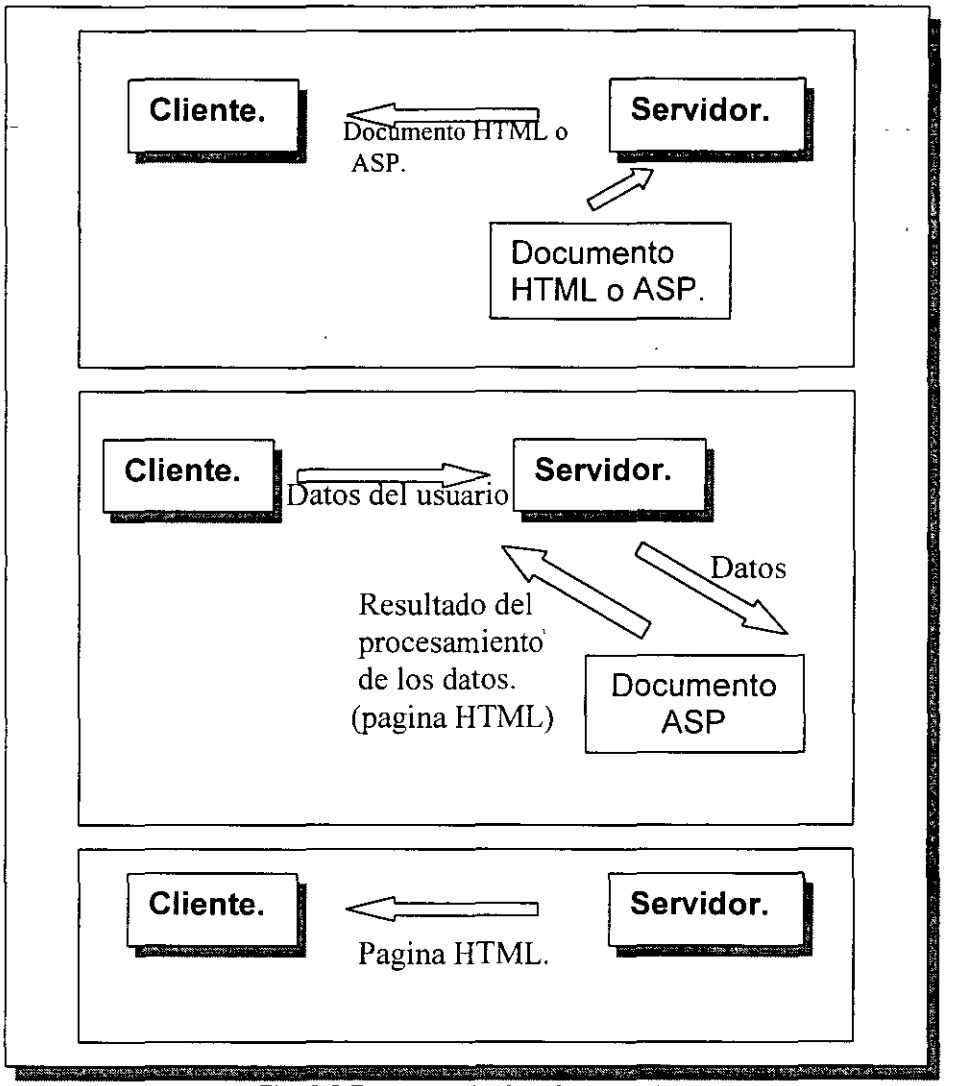


Fig. 3.2 Esquema de dos documentos.

3.2.2 Esquema de un solo documento.

En este caso existe un solo documento, en el cual se recabaran los datos y se procesaran, para generar una respuesta .

En este otro esquema ocurre lo siguiente:

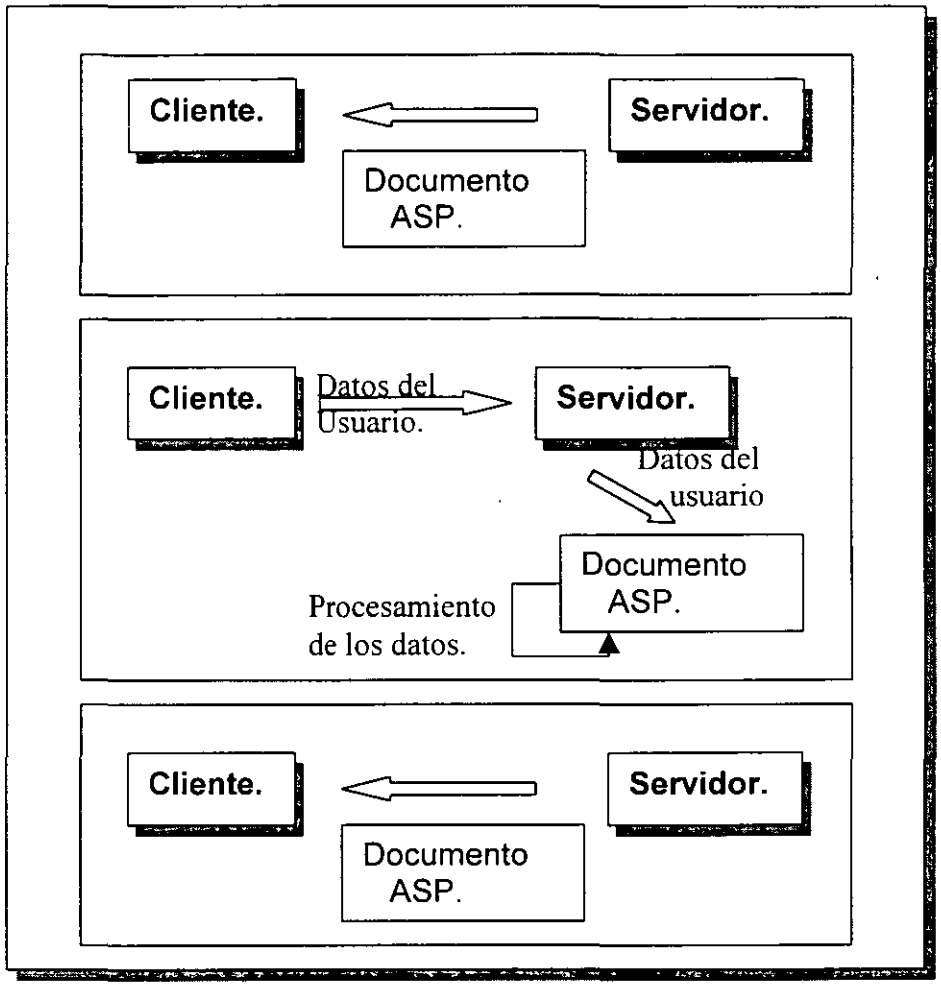


Fig. 3.3 Esquema de un solo documento.

a) El usuario recibe una pagina ASP, la cual como en el caso anterior cuenta con un formulario para recabar información. El usuario lo rellena y pulsa el botón de submit.

b) Se envían los datos recabados del usuario al servidor, y que son recibidos por el mismo documento ASP que la procesa para generar el documento que recibirá el usuario, o para registrarlo en una base de datos, etc.

c) Se regresa el documento generado al cliente.

Este par de esquemas basados en formularios para la interacción con el usuario resultan de bastante utilidad, y con un poco de imaginación se pueden conseguir resultados impresionantes, tales como encuestas en tiempo real, actualización y consulta de bases de datos, o personalización de contenidos de las páginas por los usuarios, etc.

3.2.1 El objeto Request.

Los dos esquemas presentados en la sección anterior se basan en un formulario y en un objeto integrado en ASP de los mencionados antes en este capitulo: **Request** , este objeto nos permitirá mediante sus métodos y atributos manipular la información proporcionada en los formularios de una página, en general el objeto **Request** representa la mitad de la interacción entre el cliente y el servidor, pues contiene los datos enviados en la petición del cliente, la otra mitad la representa el objeto **Response** que será tratado en el capitulo 4. Para recabar los datos que serán enviados al servidor Web se utiliza un formulario construido con las etiquetas <FORM>, donde se indica el tipo de método empleado para enviar la información mediante el uso del atributo METHOD, cuyos valores podrán ser POST o GET, (ambos métodos http), en el primero, se encapsulan los datos y se envían al servidor, mientras que con el segundo se envía la URL de la página, script o programa que procesara la información, con la misma, contenida dentro de la cadena que lo conforma, la selección entre alguno de estos dos métodos se basa en la cantidad de información que se requiere enviar, pues al usar GET, corremos el riesgo de encontrarnos con restricciones en el tamaño de la cadena del URL por parte del servidor . Por otro lado tenemos el atributo ACTION segunda parte vital para el uso de los formularios, pues precisamente en él se coloca la URL del documento, programa, etc que procesara la información.

A continuación se realizara un ejemplo del uso de los formularios con ASP bajo el primer esquema mostrado en este capitulo:

Ejemplo. Formulario.html

```
<HTML>
<FORM METHOD=''post'' ACTION=''ejemplo.asp''>
<INPUT  TYPE=''text'' name=''nombre''>
<INPUT  TYPE=''submit'' NAME=''enviar''
value=''EJECUTAR''>
</FORM>
</HTML>
```

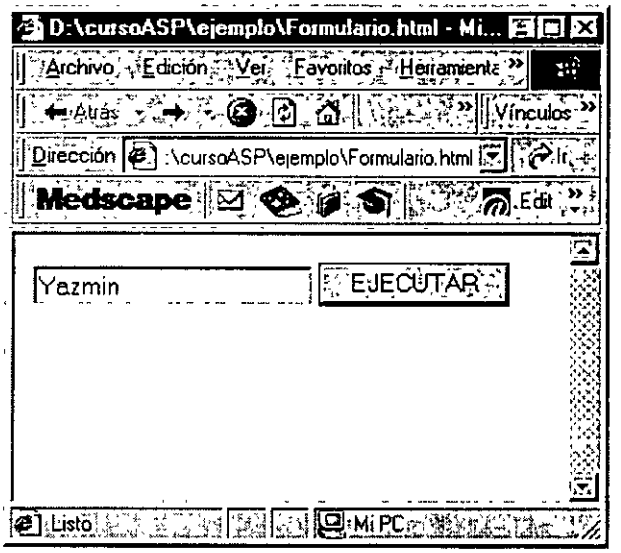


fig. 3.4 Formulario

La figura 3.4 muestra el documento generado mediante el código anterior, consiste de un formulario muy simple con un botón y un campo de texto. Cuando el usuario pulse el botón enviar, los datos contenidos en el formulario serán enviados al documento `ejemplo.asp`, donde intervendrá el objeto **Request**. El código se muestra a continuación:

`ejemplo.asp`

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>DATOS DEL FORMULARIO</TITLE>
</HEAD>
<BODY>
Usted escribio  <%= Request.Form(``nombre'')%>
</BODY>
</HTML>
```

Este es un ejemplo muy simple de lo que puede conseguirse con el uso de los formularios, hemos tomado los datos proporcionados por el usuario y los hemos colocado en la pagina de respuesta generada para el usuario, el resultado se demuestra en la figura 3.5.

fig. 3.5 Resultado.

Examinando el código del documento podemos encontrar que la única línea de código en ASP es `<%= Request.Form(``nombre'')%>` en ella se invoca al objeto **Request**, y de este la colección `Form` que nos permite traer el valor de cualquiera de los campos del formulario, invocándolo por su nombre, cuando en la página de inicio se halla usado el método `post`, por lo tanto en este caso mediante esa sentencia obtenemos el texto tecleado por el usuario en `nombre`.

Cuando se utilice el método GET en el formulario no será posible utilizar la colección `Form` del objeto **Request**, en su lugar se deberá de utilizar `QueryString` que funciona de manera similar a lo explicado para `Form`.

Para implementar el esquema de un solo documento expuesto al principio de esta sección es necesario hacer uso de estructuras de control para decidir el contenido del documento que recibirá el cliente,

Ejemplo.

```
<HTML>
<HEAD>
```

```
<TITLE>DATOS DEL FORMULARIO</TITLE>
</HEAD>
<BODY>
<%if(Request.Form.count = 0) then%>
<%=''<FORM NAME=formualrioInfo METHOD=POST
ACTION=ejemplo.asp>'' %>
<% ="<INPUT  TYPE=text name=nombre>"%>
<%="<INPUT  TYPE=submit NAME=enviar value=EJECUTAR>"%>
<%="</FORM>"%>
<%else%>
Usted escribio  <%= Request.Form("nombre")%>
<%end if%>
</BODY>
</HTML>
```

Para el uso de un solo documento se realiza un proceso de selección del código que se enviara al usuario mediante la sentencia:

```
<%if(Request.Form.count = 0) then%>
```

en esta se verifica que no haya datos dentro de la colección que representa al formulario, mediante el atributo `count`, que contiene el numero de elementos dentro de la colección, de ser así se colocará el código del formulario en el documento de respuesta:

```
<%=''<FORM NAME=formualrioInfo METHOD=POST ACTION=ejemplo.asp
>''%>
<% ="<INPUT  TYPE=text name=nombre>"%>
<%="<INPUT  TYPE=submit NAME=enviar value=EJECUTAR>"%>
<%="</FORM>"%>
```

como puede apreciarse, la totalidad del formulario es colocada en esta sección del código. Finalmente si ya se ha recibido la información, en lugar del formulario, se colocara la frase del ejemplo anterior. El uso de un solo documento para la captura

y proceso de los datos requiere un uso mas intensivo de la programación, y un entendimiento mayor de los alcances de ASP.

A continuación se ejemplificara el uso de la colección `QueryString`.

Ejemplo.

`Ligas.html`

```
<HTML>
<HEAD>
<TITLE>ENVIO DE DATOS MEDIANTE LIGAS</TITLE>
</HEAD>
<BODY>
<A HREF=''ProcesaDatos.asp?nombre=Yazmin''>Envia Datos
</A>
</BODY>
</HTML>
```

Para le demostración del uso de `QueryString` se ha decidido no utilizar un formulario, en su lugar hemos colocado una liga dentro del documento original, en la cual pasamos datos mediante el uso del símbolo `?`, este procedimiento es análogo al que se realiza mediante el método GET en el caso de un formulario. El código del documento que procesara la cadena con los datos se muestra a continuación.

`ProcesaDatos.asp`

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>DATOS DE LA LIGA</TITLE>
</HEAD>
<BODY>
```

```
Usted escribio  <%= Request.QueryString(``nombre'')%>
</BODY>
</HTML>
```

Este ejemplo se basa en el primero visto de procesamiento de formulario, es mas el uso de la `QueryString` es tan similar al explicado para el cado de Form que lo único que ha sido necesario realizar es el cambio entre el nombre de una, por el de la otra.

4.Objetos y componentes más importantes integrados en ASP.

Como se ha explicado en el capitulo anterior, es posible utilizar objetos y componentes en ASP, a continuación se presentarán algunos de los más importantes de ellos. ASP incluye seis objetos integrados:

- Server.
- OjectContext.
- Application.
- Session.
- Request.
- Response.

En el capitulo anterior ya se a tratado al objeto Request, el resto serán explicados en este capitulo

4.1 El objeto Response.

Este objeto es la contraparte del objeto **Request**, y es el encargado de generar la respuesta que es enviada al usuario, aunque no lo veamos este objeto interviene siempre que escribimos contenido, por ejemplo cuando utilizamos el operador = para escribir expresiones desde ASP en el código, estamos usando un equivalente a `Respose.Write`

Ejemplo.

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>EJEMPLO</TITLE>
```

```
</HEAD>
<BODY>
<%Response.Write "La hora es " & Time%>
</BODY>
</HTML>
```

Como se ilustra en el ejemplo es posible sustituir el uso del operador igual. Por la función Write del objeto **Reponse**. Esta es tan solo una demostración del uso de este, mas adelante se utilizara en otros casos, como en el de las cookies.

Existen otros métodos y propiedades útiles disponibles dentro del objeto **Response**, algunos se listan a continuación:

- ***Expires***. Esta propiedad determina el tiempo en minutos que permanecerá almacenada en la memoria caché del cliente la página de respuesta, para fijar este valor se utilizarán números enteros.
- ***ExpiresAbsolute***. Esta propiedad establece una caducidad en fecha y hora para el almacenamiento de la página, para inicializarse se utilizara una cadena con el siguiente formato: #mm/dd/aaaa hh:mm:ss#.
- ***CacheControl***. Esta propiedad determinara si la página se almacenara en el caché del servidor proxy, así será cuando se le asigne el valor de *public*, para el caso contrario se utilizara *private*.
- ***Redirect***. Este método nos permitirá redireccionar al usuario desde un documento ASP a otro documento diferente, esto permite por ejemplo contar con sitios en diferentes idiomas, para proporcionar el adecuado a cada cliente.
- ***Buffer***. Este método determina el medio .en que se genera y envía el documento de respuesta al usuario, si su valor es *True*, el documento es enviado conforme se va generando, en caso contrario *False* el documento se almacena al servidor y es enviado cuando se ha terminado de procesar.
- ***CharSet***. Este método indica el conjunto de caracteres utilizado para el texto.

- ***Flush***. Con este método se envía el contenido actual del buffer al cliente, y el resto sigue ejecutandose para que este método funcione la propiedad buffer debera de ser fijada a True.
- ***End***. Con este método se termina el proceso de generación de la respuesta al cliente y se le envía el contenido actual del buffer.
- ***Clear***. Mediante este método se elimina el contenido del buffer.

En este objeto también se trabajan las cookies para escritura, este uso será estudiado en el capitulo 6.

4.2 Objeto Server.

El objeto Server, es pequeño por el numero de métodos y propiedades que contiene en comparación con los otros integrados en ASP, sin embargo es de crucial importancia, pues solo con el se podrán crear objetos externos, entre otras cosas, a continuación se enlistan sus métodos y atributos mas importantes:

- ***ScriptTimeout***. Esta es su única propiedad, determina el numero de segundos que puede ejecutarse el script del lado del servidor, después de los cuales se vera forzado a terminar. Su valor por omisión es de 90.
- ***CreateObject***. Este método es el que nos permitirá generar Componentes externos de ASP

4.3 Los componentes de Acceso a Archivos.

Como se ha dicho anteriormente, VBScript no cuenta con capacidades para el manejo de archivos, pero dentro de los componentes que es posible utilizar desde ASP se encuentran los objetos para poder manipularlos en el servidor.

El componente **File Acces** forma parte de la librería de scripting de Microsoft, en él se encuentran los objetos **FileSystem**, y **TextStream**, el primero nos permitira acceder a los archivos en el Servidor, y el segundo establecer flujos de escritura o lectura en un archivo particular. **FileSystem** cuenta con los siguientes métodos:

- ***CreateTextFile***. Este método nos permitirá crear un nuevo archivo, lo que nos regresa es el flujo de salida / escritura al archivo.
- ***OpenTextFile***. Este método nos permitirá acceder a un archivo de texto en el servidor, lo que nos regresa es un flujo de entrada / lectura de un archivo. Este método recibe como parámetros :

OpenTextFile(Nombre, [tipo de flujo],[Crear])

El nombre será el nombre del archivo a abrir, el tipo de flujo podrá ser alguno de los siguientes:

Valor	Descripción
1	Abre un archivo para lectura, no es posible escribir en el
2	Abre un archivo exclusivamente para escritura.
8	Abre un archivo para escribir al final de su contenido.

Tabla 4.1

Y el ultimo atributo podra recibir los valores de False o True, en el caso de que se desee crear o no el archivo en caso de no encontrarlo.

El objeto **TextStream** contiene los siguientes métodos y atributos:

- ***Close***. Este método cierra el flujo al archivo, esto se realiza cuando se ha terminado de utilizar el archivo, o cuando se necesita cambiar el tipo de flujo del que se trata.

- ***Read***. Regresa como cadena, el número de caracteres especificados, provenientes de un archivo.
- ***ReadAll***. Lee todo el contenido del archivo asociado al flujo
- ***ReadLine***. Lee la línea del archivo asociado al flujo en la que se encuentre situado el cursor, en el momento de ser invocado, y la regresa en forma de cadena.
- ***Skip***. Desplaza el cursor en el archivo, el número de caracteres que se le pasen como argumento. (en forma de entero)
- ***SkipLine***. Desplaza el cursor una línea completa
- ***Write***. Escribe una cadena en el archivo asociado al flujo de salida.
- ***WriteLine***. Escribe una cadena en el archivo, y continuación el carácter de nueva línea.
- ***WriteBlankLines***. Escribe el número determinado de líneas en blanco.
- ***AtendofStream***. Este método sirve para determinar cuando se halla alcanzado el final de un archivo.
- ***AtendofLine.*** Su funcionamiento es similar al del anterior solo que para detectar el final de una linea.

A continuación se ejemplificara el uso de estos objetos y sus interfaces.

Ejemplo.

```
<HTML>
<HEAD>
<TITLE>Prueba de Archivos</TITLE>
</HEAD>
<BODY>
<%
Dim SistArch, MiArchivo
Set SistArch=CreateObject("Scripting.FileSystemObject")
```

```
Set MiArchivo=SistArch.CreateTextFile("c:\prueba.txt", True)
MiArchivo.WriteLine("Esto es una prueba.")
MiArchivo.Close
Set MiArchivo=SistArch.OpenTextFile("c:\prueba.txt",1, True)
Contenido=MiArchivo.Read(5)
%>
Hemos leido del archivo:<BR>
<B>
<%=Contenido%>
</B>
</BODY>
</HTML>
```

Los resultados de este ejemplo se muestran en la siguiente figura:

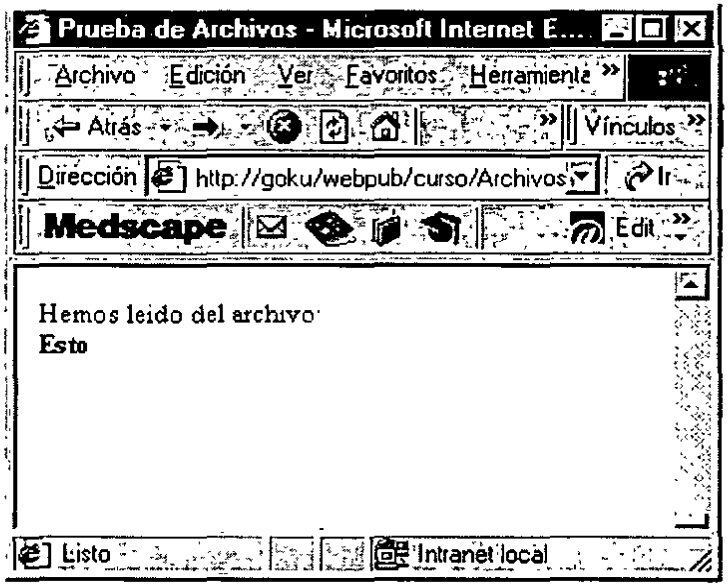


fig. 4.1 Manejo de Archivos

En este ejemplo utilizamos algunos de los métodos de los objetos de acceso a archivos, declaramos dos variables `SistArch`, `MiArchivo` la primera la inicializamos mediante el método `CreateObject` de **Server**, a continuación utilizando `CreateTextFile` creamos en el servidor un archivo llamado *prueba.txt*, escribimos en el una frase mediante `WriteLine` después cerramos el

flujo al archivo para poder abrir uno nuevo de lectura, esto por que no es posible establecer mas de un flujo a un mismo archivo. Se establece el flujo de entrada / lectura, mediante OpenTextFile pasándole el mismo archivo que generamos en la ocasión anterior, leemos cinco caracteres de él, y luego los vaciamos en la página que esta siendo generada.

Dentro del conjunto de objetos disponibles para el manejo de archivos existen otros diferentes a los ya vistos que cumplen un fin mas descriptivos sobre el estado de un sistema de archivos, y la realización de modificaciones en su estructura, estos son:

- **Drive.** Representa una unidad lógica dentro de un sistema de archivos.A continuación se listan algunas de sus propiedades, como se ha explicado, por su carácter informativo no tiene métodos:

    - ***AvalaibleSpace***. Contiene la cantidad de espacio disponible en la unidad.
    - ***DriveLetter***. Contiene la letra con la que se designa a la unidad. Solo lectura
    - ***FreeSpace***. Contiene la cantidad de espacio disponible en la unidad. Solo Lectura
    - ***IsReady***. Regresa falso o verdadero según se encuentre no disponible o disponible la unidad
    - ***Path***. Contiene la ruta de acceso.
    - ***RootFolder***. Contiene un objeto del tipo fólder que representa el directorio principal de la unidad.
    - ***SerialNumber***. Contiene el numero de serie de la unidad.
    - ***TotalSize***. Contiene en bytes el espacio total de una unidad.
    - ***VolumeName***. Contiene el nombre asignado a una unidad.

- **Folder.** Representa un directorio dentro de un sistema de archivos. A continuación se listan sus principales atributos y métodos:

  Atributos:

  - ***DateCreated.*** Contiene la fecha y hora de la creación del archivo.
  - ***DateLastAccessed.*** Contiene la fecha y hora del ultimo acceso al archivo.
  - ***DateLastModified.*** Contiene la fecha y hora de la ultima modificación realizada al archivo.
  - ***Drive.*** Contiene la letra del drive donde el archivo o directorio reside.
  - ***Files.*** Contiene una colección de objetos del tipo **File**, que representan a todos los archivos contenidos en el directorio
  - ***IsRootFolder.*** Contiene **True** en caso de que el directorio representado por el objeto sea el directorio principal de la unidad, o **False** en caso contrario.
  - ***Name.*** Contiene y permite modificar el nombre del archivo o directorio.
  - ***ParentFolder.*** Contiene un objeto del tipo **Folder** que representa el directorio inmediato en el cual reside el archivo o directorio.
  - ***Path.****
  - ***Size.*** Devuelve el tamaño en bytes del archivo o en el caso de los directorios de la suma de su contenido.
  - ***SubFolders.*** Contiene una colección de objetos del tipo **Folder** que representan a todos los subdirectorios.
  - ***Type.*** Regresa el tipo de archivo o de directorio del que se trate.

Metodos:

- ***Copy.*** Copia un directorio o archivo a la dirección que se le especifique como parámetro.
- ***Delete***. Borra el archivo o directorio.
- ***Move***. Mueve un directorio o archivo a la ruta especificada como parámetro.
- ***CreateTextFile.****

- **File.** Representa un archivo dentro de un sistema de archivos. A continuación se listan sus principales atributos y métodos

  - ***DateCreated.****
  - ***DateLastAccessed.****
  - ***DateLastModified.****
  - ***Drive.****
  - ***Name.****
  - ***ParentFolder.****
  - ***Path.****
  - ***Size.****
  - ***Type.****

Metodos:

- ***Copy.****
- ***Delete.****
- ***Move.****
- ***OpenAsTextStream.*** Cumple la misma función que el método **OpenTextFile** del objeto **FileSystem**

Cada uno de estos objetos se instancia mediante un método del **FileSystem**, a saber:

- **GetDrive**. Recibe como parámetro la ruta de la unidad que representara el objeto.
- **GetFile**. Recibe como parámetro la ruta y nombre del archivo que representara el objeto.
- **GetFolder**. Recibe como parámetro la ruta y nombre del directorio que representara el objeto.

4.4 El objeto Application.

Como se vio en el capitulo 3, en las aplicaciones de ASP las variables y los valores que almacenan, existen solo en el tiempo en el que se genera el documento, el objeto **Application** nos permite romper con estas limitaciones. Con este se podrán generar variables globales a toda una aplicación que serán compartidas por todos aquellos clientes que visiten una página de la misma, este objeto se creara desde el momento en el que primer usuario visita un pagina de la aplicación, y persistirá hasta que se detenga el servidor Web, o se apague la maquina que lo alberga. Los limites de la aplicación ASP son los subdirectorios, es decir una aplicación esta conformada de manera funcional, por los contenidos de un directorio, esto no quiere decir que no se puedan incluir documentos de sitios y hasta de dominios diferentes en una sola carpeta, pero si que su manipulación como un conjunto coherente, será mucho mas difícil e ineficiente. A continuación se enlistan las partes mas importantes del objeto **Application**:

- ***Lock***. El método Lock bloquea las variables que se invoquen después de el, para evitar inconsistencias, por el uso recurrente de los datos por parte de diferentes clientes.

- ***Unlock***. Con este método se desbloquearan los datos bloqueados mediante el uso del método Lock.
- ***Value***. Esta propiedad contiene el valor asociado a cada una de las variables almacenadas en el objeto Application, este atributo puede omitirse utilizando en su lugar directamente el nombre del objeto.
- ***OnStart***. Este es un evento, como veremos mas adelante, este evento es se genera cuando se inicializa la aplicación (se accede por primera vez a una página de la aplicación).
- ***OnEnd***. Este es el evento asociado a la finalización de la aplicación. Este evento y el anterior no se manejaran directamente en las aplicaciones ASP, se manejaran desde un archivo independiente.

Ejemplo.

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>DATOS DE APLICACION</TITLE>
</HEAD>
<BODY>
ESTA PAGINA FUE ACCEDIDA POR ULTIMA VEZ EN
<%= Application(``Fecha'')%>
<% Application.Lock%>
<% Application(``Fecha'')=Time%>
<% Application.Unlock%>
</BODY>
</HTML>
```

En este ejemplo estamos generando una variable de nombre Fecha, la cual modificamos desde esta página cada vez que se tenga que generar colocando en ella la hora en que se visito, cada usuario que acceda a la página sabrá cuando fue la ultima vez que esta fue visitada. Si varios clientes intentaran acceder al mismo tiempo a esta página, el segundo no podría hacer la modificación

correspondiente al valor de Fecha hasta que no se haya terminado de procesar esa parte del documento para el primero, y así sucesivamente.

4.4.1 El archivo GLOBAL.ASA.

En el ejemplo anterior la primera vez que se tiene acceso a al página se le encuentra sin el valor esperado de la fecha, esto por que no hay un valor de inicio que se le pueda proporcionar desde un documento de la aplicación, es en este caso cuando aparece en acción el archivo GLOBAL.ASA, este se encontrara en cada subdirectorio que represente una o mas aplicaciones, en él se manejaran los eventos de inicio y fin de objetos como por ejemplo el **Application**. En el siguiente ejemplo se muestra el uso del GLOBAL.ASA completando la función del anterior.

Ejemplo.

```
<SCRIPT LANGUAGE=VBScript RUNAT="Server">
Sub Application_OnStart
     Application("Fecha")=Now
End Sub
</SCRIPT>
```

El código de este ejemplo completaría al anterior para un buen funcionamiento, en el se codifica al estilo de VBScript el evento OnStart del objeto **Application**, también será posible codificar otros eventos de otros objetos, como se verá mas adelante.

Una consideración de peso en el uso del GLOBAL.ASA es que como se ha explicado antes, existirá uno por cada directorio donde se deseen utilizar objetos como el **Application**, y el **Session**, por lo tanto cualquier modificación en su contenido afectara el comportamiento de las páginas contenidas en el mismo directorio, es por esto que se recomienda utilizar un directorio diferente para cada conjunto de páginas que constituyan una aplicación. Una ultima aclaración

importante sobre las Aplicaciones construidas con el objeto **Application** y el archivo GLOBAL.ASA es que la aplicación **NO** se inicia cuando se accede a una página HTM o HTML contenida en el mismo directorio, aunque de forma lógica y estructural esta forme parte de la Aplicación, solo iniciara cuando se acceda por primera vez a un documento ASP.

4.5 Objeto Session.

Asi como el objeto **Applicaction** nos permite mantener una Aplicación con variables globales, trabajando, existe un objeto, el **Session**, que en conjunto con el archivo GLOBAL.ASA nos permite mantener información dentro de la sesión de cada uno de los usuarios. Cada vez que un usuario acceda a una de las paginas de una aplicación se creara un nuevo objeto **Session** único , que persistirá hasta que ese usuario la abandone, este hecho se desencadenara en función del tiempo que el usuario tenga sin enviar una petición al servidor. El objeto **Session** cuenta con los mismos atributos, métodos y eventos que el Application y uno mas:

- ***Timeout***. Esta propiedad fija el tiempo en minutos que esperara el servidor para dar por terminada la sesión de un usuario, y destruir su respectivo objeto **Session**. Su valor por omisión es de 20, y se modifica pasándole valores enteros.

El modelo de las variables de la aplicación y de la sesión, y su interaccion con los usuarios, se esquematiza en la siguiente figura:

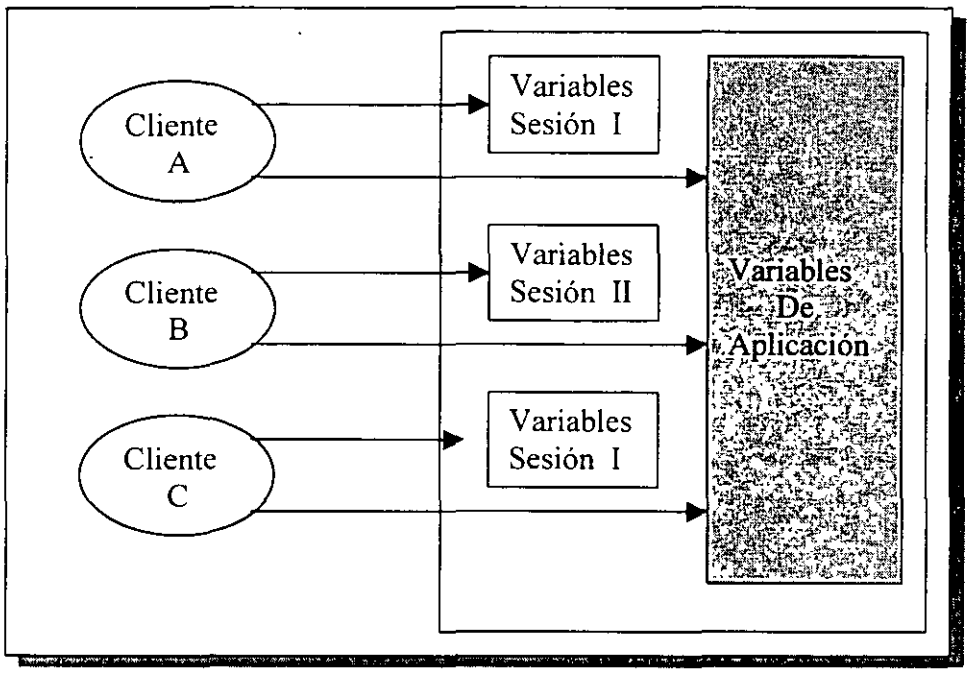


Fig. 4.2 Funcionamiento conjunto de Session y Application.

4.6 AdRotator.

Como un ejemplo mas de componentes ASP, tenemos el ADRotator, este permite generar banners rotatorios, es decir que la publicidad mostrada al cliente no será siempre la misma, para generar los anuncios este componente se vale de dos elementos:

- El Objeto **AdRotator**.
- El archivo guía o de programación.

El objeto **AdRotator** se generara de la misma forma en que se generaba el objeto **FileSystem**, haciendo uso del método CreateObject , a continuación se mostrara la sintaxis de los otros archivos necesarios:

El archivo de programación o guía tendrá entradas de cuatro renglones, del siguiente tipo:

1. Ruta de la imagen a cargar.
2. Dirección del anunciante.
3. Texto alterno al anuncio.
4. Ponderación del anuncio.

El primer renglón corresponde a la ruta de la imagen del anuncio, el segundo será la URL de la pagina del anunciante, el tercero será el texto de sustitución de la imagen, o tooltiptext, y el cuarto la frecuencia relativa con la que se mostrara el anuncio, a continuación se muestra el siguiente ejemplo:

Ejemplo. **AdRotrguia.txt**

/anuncios/librería.gif
http://elsotano.com
El sótano su mejor librería
3
/anuncios/cafeteria.gif
http://elcafetal.com
El mejor café del mundo.
2

Si utilizáramos un objeto **AdRotator** con el archivo mostrado arriba tendríamos dos anuncios posibles, de los cuales el primero se mostraría 3 de cada cinco veces, y el segundo dos por cada cinco, ahora para cargar este rchivo y mostrar la publicidad el procedimiento seria el siguiente:

Ejemplo. Publicidad.asp

```
<%
        set AdObjeto = CreateObject(``MSWC.AdRotator'')
        letrero=AdObjeto.GetAdvertisement(``AdRotrguia.txt'')
        Response.Write letrero
%>
```

Con este pequeño código estamos creando el objeto, después cargando el archivo de publicidad en el objeto y generando con el las líneas necesarias para colocar el banner en la página. Con este sencillo procedimiento estaremos obteniendo una eficiente publicidad rotatoria, y sin sobrecargar la pagina enviada al cliente pues esta será simple código HTML.

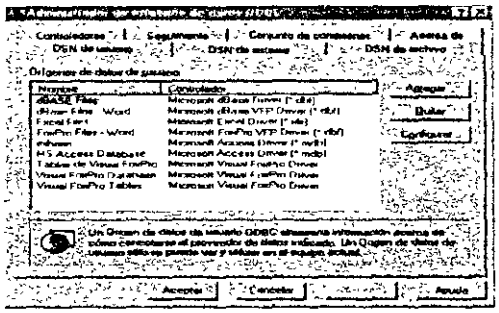
5.0 Acceso a Bases de Datos

5.1 Conexión ODBC

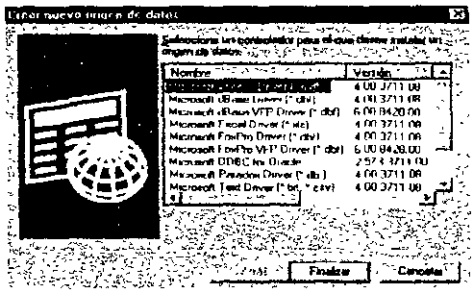
Para poder utilizar una base de datos desde ASP esta debe estar dada de alta para emplearla con controladores ODBC.

Vamos a ver como hacer esta conexión paso a paso.

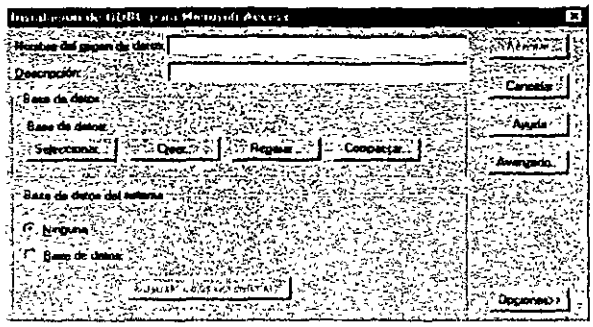
Lo primero será ir a **Inicio -> Configuración -> Panel de Control ->Fuentes ODBC.** Entonces estaremos en una pantalla como la siguiente:



En esta pantalla se encuentran algunos de los controladores para los diferentes tipos de bases de datos que soportan controladores ODBC. Estando ahí damos clic en el botón Agregar..., y se nos mostrará una pantalla como la siguiente:



En ella seleccionamos el tipo de base de datos que tenemos, es decir Acces, FoxPro, Oracle, etc. Y damos clic en el botón Finalizar, entonces estaremos en la siguiente pantalla. En ella tendremos que hacer clic en el botón que dice Seleccionar, en el recuadro que dice Base de Datos. Para que se nos muestre una ventana para seleccionar la base de datos que deseamos agregar.



Después le damos un nombre, en la primera caja de dialogo donde dice: Nombre del origen de datos, con este nombre nosotros haremos la conexión a la base de datos, no tiene que ser el mismo nombre de la base de datos.

Entonces damos clic en Aceptar, y nos regresara a la primera pantalla en ella ahora podremos ver el nombre del origen de datos que le dimos a la conexión acompañado de los controladores para el tipo de base de datos que tenemos. Ahora nuestra base de datos esta lista para ser utilizada desde ASP. Si ahora vamos al explorador de Windows podremos ver que un hay un icono que nos indica que la conexión se realizo satisfactoriamente. En la misma carpeta donde se encuentre la base de datos sobre la que vamos a trabajar encontraremos un icono parecido a esto:

db2.mdb

5.2 Objetos ADO

El modelo de bases de datos basado en Actives Data Objects, está formado por objetos, estos objetos proporcionan una serie de métodos y propiedades con los que podemos acceder fácilmente a las bases de datos .

Para manejar este tipo de bases de datos tenemos siete objetos. De estos siete objetos hay que ver con especial cuidado a tres de ellos que son los mas importantes Connection, Recordset y Command. El resto de los objetos son Field, Parameter, Property y Error, sin embargo para poder emplear estos últimos necesitamos de los primeros.

La descripción de cada uno de los objetos seria:

- Connection: Representa la conexión con una base de datos. Este objeto lo utilizaremos para crear un enlace directo entre nuestra página Web y el servidor de bases de datos. Mientras dure la conexión podremos realizar todas las operaciones que deseemos sobre la base de datos. La conexión terminará cuando nosotros así lo indiquemos con el método Close del mismo objeto.
- RecordSet: representa una tabla de datos. En este objeto serpá donde almacenemos las consultas realizadas a la base de datos a la que estemos conectados. Estará formada por filas, y por columnas a los que podremos acceder para exponer la información adquirida.

- Command: Representa un comando SQL. Con este objeto podremos ejecutar sentencias SQL sobre la base de datos a la que estemos conectados.

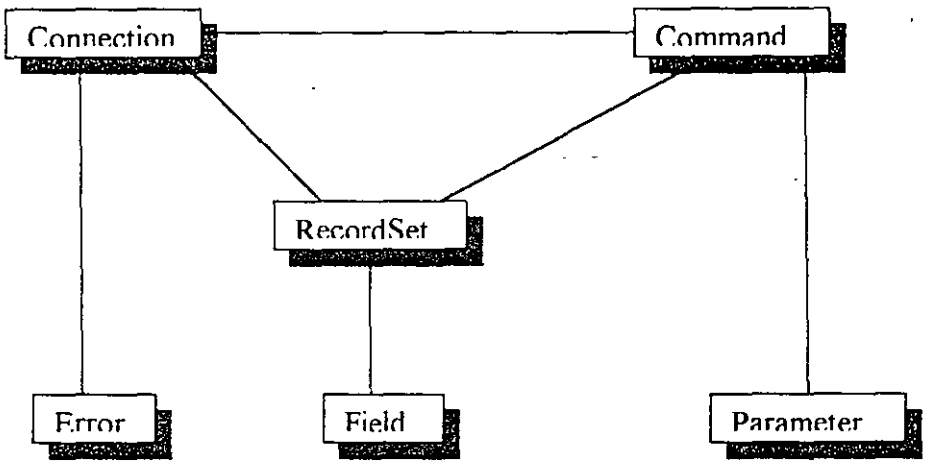
- Field: Representa un campo de un objeto RecordSet. Este objeto solo existirá si existe el correspondiente objeto RecordSet. El objeto RecordSet lleva implícito la colección Fields que representa todos sus campos. Cada elemento de esa colección es un objeto Field.

- Parameter: Representa un parámetro de un procedimiento o cuestión. Nos será de gran ayuda al utilizar el objeto Command para lanzar procedimientos o cuestiones sobre una base de datos.

- Error: Representa un error ADO. Se puede producir un error al realizar la conexión sobre la base de datos. Esto quiere decir que este objeto solo existirá si previamente se ha intentado conectar con una base de datos erróneamente.

- Property: Representa una propiedad específica de un proveedor de datos. Este objeto se encuentra un poco al margen de lso demás, ya que no tiene relación alguna con ellos.

Como se puede suponer entre estos objetos existe cierta relación.



Relaciones:

Connection-RecordSet: Una cualidad que tiene el objeto Connection es la de poder ejeutar comandos SQL. Al hacer esto pueda darse el caso de que el comando devuelva cierto resultado, por ejemplo, si el comando realiza una consulta sobre la base de datos. En tal caso el resultado vendrá dado dentro de un objeto RecordSet.

Command-RecordSet: Al igual que en el caso anterior, al ejecutar una consulta con el objeto Command sobre una base de datos el resultado viene dado en un objeto RecordSet.

Command-Connection: Cuando creamos un objeto Command no será utilizable hasta que no lo relacionamos con una base de datos. Para esto es necesario el objeto Connection. Con este objeto abriremos una sesión con una base de datos y entonces se lo asignaremos al objeto Command como conexión activa. Solo de este modo podremos ejecutar los camandos de este objeto sobre una base de datos.

Connection-Error: El objeto Error solo aparece cuando se produce una conexión errónea a una base de datos. Con este objeto y sus métodos y propiedades podremos determinar la causa del error.

RecordSet-Field: Todo objeto RecordSet tiene entre sus atributos la colección Fields. Esta colección esta formada por objetos del tipo Field. Estos objetos representan ada campo de cada registro del objeto RecordSet. El objeto Field nos permitirá un acceso sencillo a la información del objeto RecordSet.

Command-Parameter: Un comando SQL puede tener parámetros. Esos parámetros pertenecerán al objeto Parameter. Además no sólo podemos ejecutar sentencias simples de SQL con el objteo Command, sino también procedimientos que probablemente llevarán parámetros, tanto de entrada como de salida.

5.2 Uso del objeto connection

Para ejemplificar el uso del objeto connection vamos a ver un ejemplo. Para que este ejemplo funcione ya dimos de alta en los controladores ODBC la base de datos "mibase".

```
1 <html>
2 <head></head>
3 <body>
4 <! - - #INCLUDE File="ADOVBS.inc" - - >
5 <%Set con= Server.CreateObject ("ADODB.Connection")
6       con.Open "mibase"
7       Set res=con.Execute("SELECT * FROM Tabla1")%>
8 <table border=1>
9 <tr>
10 <th>numero</th>
11 <th>pregunta</th>
12 <th>a</th>
```

```
13 <th>b</th>
14 <th>c</th>
15 <th>respuesta</th>
16 </tr>
17 <% do until res.Eof%>
18 <tr>
19 <td><%=res("Numero")%></td>
20 <td><%=res("Pregunta")%></td>
21 <td><%=res("a")%></td>
22 <td><%=res("b")%></td>
23 <td><%=res("c")%></td>
24 <td><%=res("Respuesta")%></td>
25 </tr>
26 <% res.MoveNext
27    Loop
28  res.Close
29  con.Close %>
30 </table>
31 </body>
32 </html>
```

Ahora hagamos un análisis del código señalando las líneas que son representativas para el uso de una base de datos.

En la línea 4 encontramos la primera novedad con esa sentencia estamos llamando al archivo que nos servirá para utilizar el objeto ADO, es decir, instanciar el objeto para la conexión. Será necesario llamar este archivo en todos los documentos que necesitemos instanciar un objeto para conexión.

En la línea 5 estamos instanciando un objeto para la conexión lo hacemos con la palabra reservada Set como lo hacemos cada vez que instanciamos un objeto. En la línea 6 abrimos la base de datos "mibase" con ayuda de el objeto "con" que acabamos de crear y su método Open. La línea 7 es una de las mas importantes. Con el método Execute del objeto "con" que creamos le indicamos

que tipo de consulta vamos a hacer sobre la base de datos, es una consulta SQL entre el paréntesis se le indican las condiciones de la consulta con la palabra reservada **SELECT** le indicamos la condición que deben cumplir los datos que extraiga de la tabla, en este caso con el asterisco le indicamos que debe traerse todos, y con la palabra reservada **FROM** le indicamos de que tabla debe traerlos. Como esta búsqueda nos da por resultado, un objeto del tipo **RecordSet** el resultado lo asociamos con la palabra Set que es como crear y al mismo tiempo asignar un objeto. Entonces en res ahora tenemos la tabla completa.

En la linea 17 vemos el uso de el metodo **Eof** del objeto **RecorSet** que nos devolverá verdadero cuando se haya llegado al final de la tabla.

En las líneas 19 a la 24 encontramos una forma de acceder a las columnas de las tablas con el uso del objeto "res" que creamos, y el nombre de cada uno de los campos, o columnas de nuestra base e datos.

En la línea 26 vemos como nos movemos al siguiente registro de nuestra tabla, resultado de la consulta.

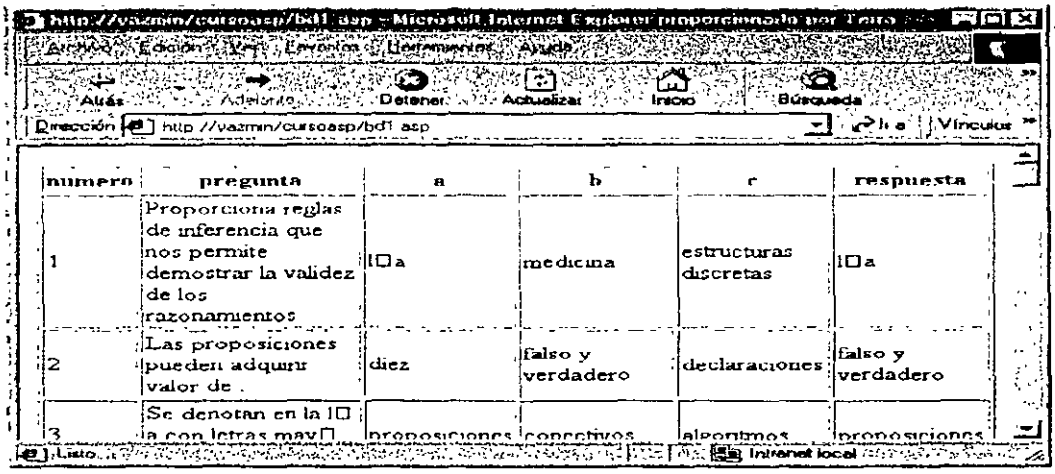
En las líneas 28 y 29 vemos como es necesario cerrar los objetos "res" y "con" el primero para liberar la memoria que ocupaba la tabla resultante, y el segundo por seguridad, y para no provocar posibles errores en el sistema.

A continuación veremos dos imágenes, la primera es el estado de la tabla en la base de datos, y la segunda es el resultado de nuestro código. Si se observa con cuidado veremos que en el resultado del código hay algunas alteraciones y esto se debe a que en los acentos no los interpreta adecuadamente así como todos los caracteres con los que hay que tener cuidado en el código html.

Tabla1 : Tabla

Numero	Pregunta	a	b	c	Respue
1	Proporciona reglas de inferencia que lógica		medicina	estructuras discret	lógica
2	Las proposiciones pueden adquirir v: diez		falso y verdade	declaraciones	falso y verdadero
3	Se denotan en la lógica con letras m proposicior		conectivos	algoritmos	proposiciones
4	¿Cuáles son las proposiciones que i binario		atómicas	compuestas	atómicas
5	Se conocen como proposiciones se: compuesta		lógicas	atómicas	compuestas
6	¿Cuáles son los elementos que rela conectivos		proposiciones	lógica	conectivos
7	Son conectivos que relacionan o mo binario		unario	compuestos	unario
8	¿Qué conectivos relacionan o modifi binario		unario	atómicas	binario
9	¿Cómo se representa el conectivo d 7		V	->	7
10	¿Cómo se representa el conectivo d 7		V	^	^

Registro: 1 de 28



numero	pregunta	a	b	c	respuesta
1	Proporciona reglas de inferencia que nos permite demostrar la validez de los razonamientos	l□a	medicina	estructuras discretas	l□a
2	Las proposiciones pueden adquirir valor de .	diez	falso y verdadero	declaraciones	falso y verdadero
3	Se denotan en la l□ a con letras may□	proposiciones	conectivos	algoritmos	proposiciones

Con el ejemplo anterior vemos la manera de obtener los datos de una tabla pero solamente.

El resto de las operaciones se realizan mediante consultas SQL comunes.

6. PERSONALIZACIÓN DEL SITIO MEDIANTE COOKIES.

6.1 Introducción a las cookies.

Las cookies son la primera opción que se genero comercialmente para acercar el concepto de sesión a la Web, en la Web el usuario puede visitar una o mas paginas pertenecientes al mismo dominio, y sin embargo no es posible almacenar información sobre las preferencias, intereses y necesidades del usuario, o corrigiendo, esto no era posible antes de la aparición de las cookies, pues no había un medio para almacenar datos, por ejemplo de la ultima visita de los usuarios. Esto es precisamente lo que hacen las cookies, almacenan información sobre las preferencias del usuario en su maquina, se trata de un pequeño archivo de texto, organizado por pares de nombre-valor. Este esquema resulta atractivo desde muchos puntos de vista, las cookies no ocupan espacio en el servidor, pues se almacenan en la maquina del cliente, proporcionan un medio seguro para el cliente pues no se trata de ejecutables, por lo tanto no podrán contener virus u otros programas nocivos, etc.

La pregunta que surge a continuación es: ¿Como funcionan la cookies?

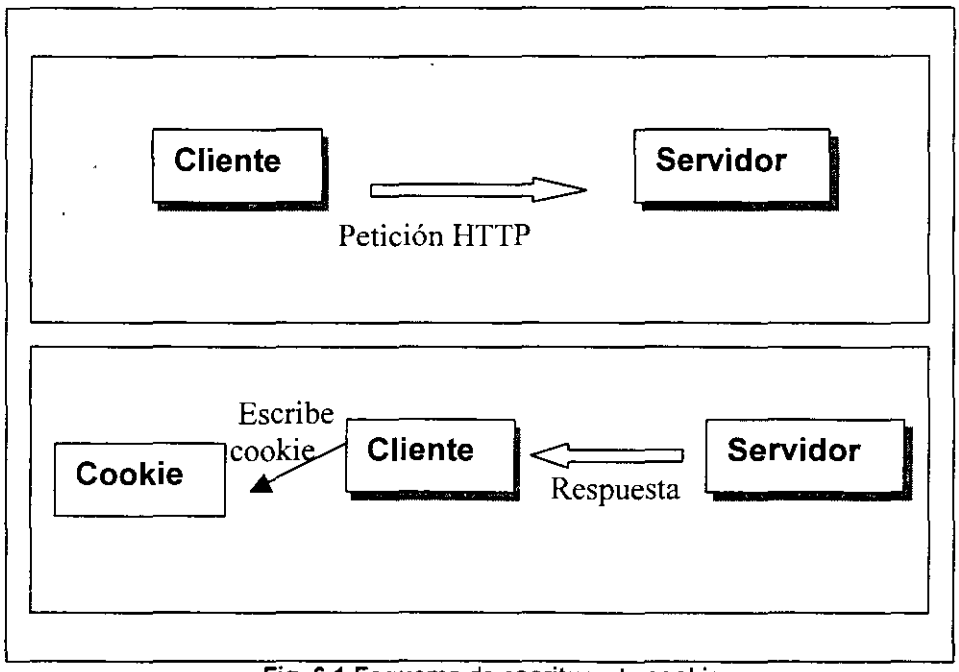


Fig. 6.1 Esquema de escritura de cookies.

Las cookies se escriben en la maquina del cliente la primera vez que visita la página o si ha caducado la cookie anterior, o simplemente si necesitamos actualizar los datos almacenados.

Una vez que un sitio a escrito una cookie en el cliente, esta formara parte de las peticiones que ese cliente realice al servidor, y por lo tanto modificara el tipo de respuesta que obtenga el usuario.

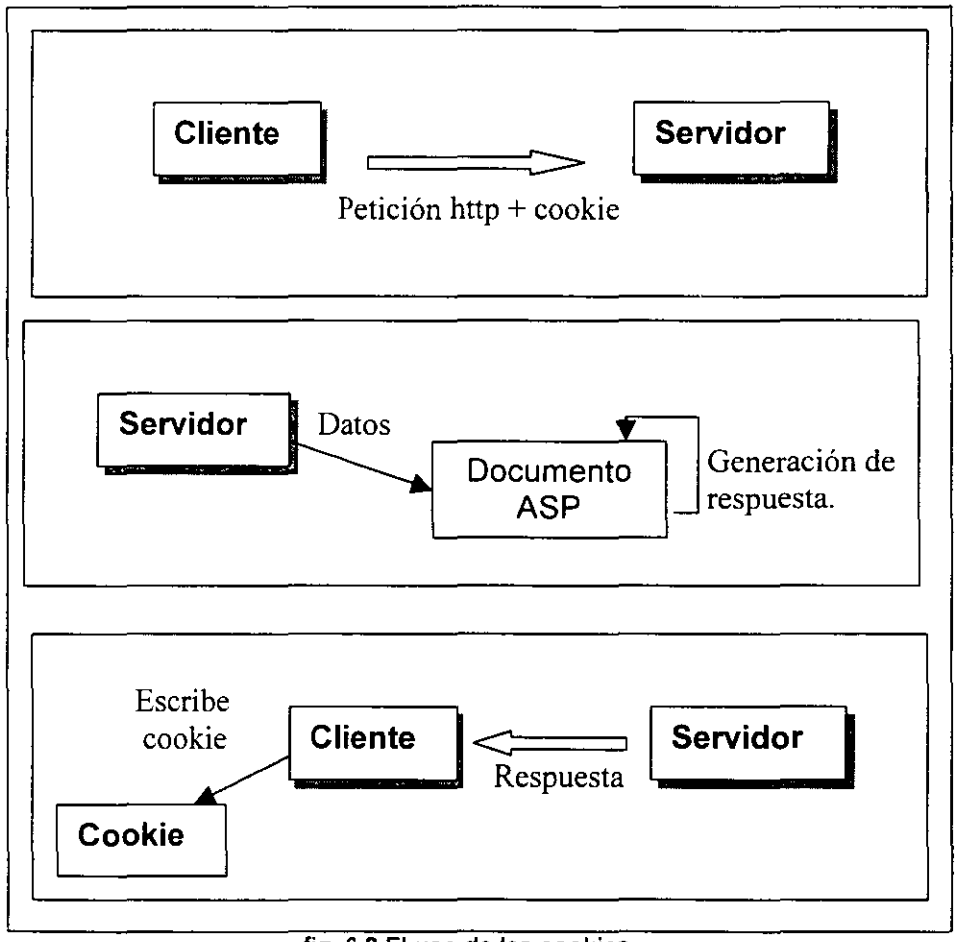


fig. 6.2 El uso de las cookies.

Las cookies tienen muchas aplicaciones posibles en el funcionamiento de sitios Web:

- **Mantener la identidad del usuario.** Un sitio puede conocer la identidad de un usuario almacenándola en una cookie, y reconociéndola automáticamente en cada visita.

- **Personalización del aspecto de la pagina.** Con este uso el usuario puede seleccionar preferencias en contenidos, y aspectos de diseño, que serán almacenados e una cookie para uso en la generación de la pagina.
- **Rastreo de la Navegación en el sitio.** Mantener una relación mediante cookies de las paginas visitadas por el usuario dentro del sitio. (Algunos usuarios suelen molestarse al descubrir que un sitio realiza esta práctica)
- **Publicidad especifica.** Mediante la información de las cookies se puede determinar el tipo especifico de publicidad que es útil para cada usuario.
- **Compras en línea.** Con la cookies se puede mantener una lista de compras en un carrito.

6.2 Cookies en acción. (Cookies y colecciones.)

Las cookies pertenecientes a una página se encuentran organizadas en una colección llamada `cookies` donde la llave de cada elemento será el nombre dado a la cookie, como se ha visto hasta ahora, las cookies funcionan de manera bidireccional, por lo tanto serán manipuladas al ser recibidas por medio del objeto **Request**, y al ser escritas por medio del objeto **Response**, por lo tanto cada uno de estos objetos accederá a la colección de cookies, para leerlas en el primer caso, y para escribirlas en el segundo . Es importante tomar en cuenta que cuando se vayan a escribir cookies, será lo primero que deberá de realizarse en el envío de la respuesta al cliente, pues de otro modo se recibirá un error al intentar escribir la pagina, pues estas deben ser parte de la cabecera de la petición del documento, y si el documento ya esta siendo escrito, NO será posible establecer mas cookies.

Ejemplo.

```
<%@ LANGUAGE="VBScript" %>
<%if(Request.cookies.count = 0) then
     visitas = 0
     Response.cookies(``visitas'')=1
Else
     visitas = Request.cookies(``visitas'')
     Response.cookies(``visitas'')=1 +
Request.cookies(``visitas'')
end if%>
<HTML>
<HEAD>
<TITLE>COOKIES</TITLE>
</HEAD>
<BODY>
<% if visitas=0 then%>
<%=''Bienvenido Visitante, es su primera vez aqui''%>
<% else %>
<%=''Bienvenido Visitante, usted nos ha visitado ''& visitas
& `` veces''%>
<%end if%>
</BODY>
</HTML>
```

En la imagen se muestra el resultado de visitar por primera vez cookies.asp, al realizar la verificación de arreglo de cookies se encuentra que este esta vacío, y se asigna a visitas el valor de 0, en caso contrario se extrae el valor actual , y después se incrementa el valor de la cookie, el valor asignado a variables es usado para decidir que se le mostrara al usuario, si una bienvenida de primera vez como la mostrada en la figura 3.3, o el número de veces que ha ingresado a la misma como en la figura 3.4.

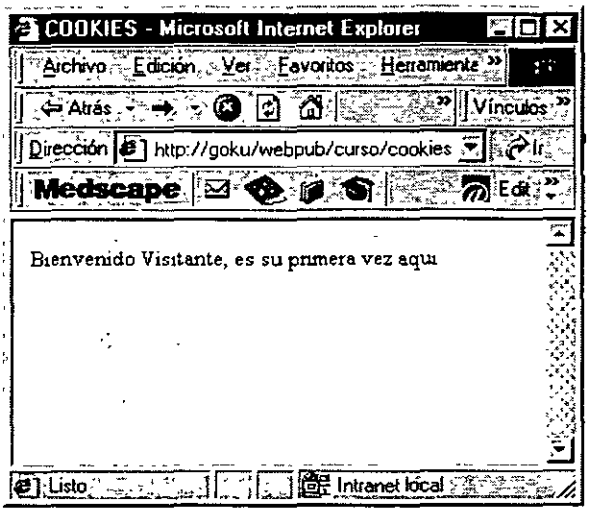


fig. 3.3 La primera vez que se accede a la página.

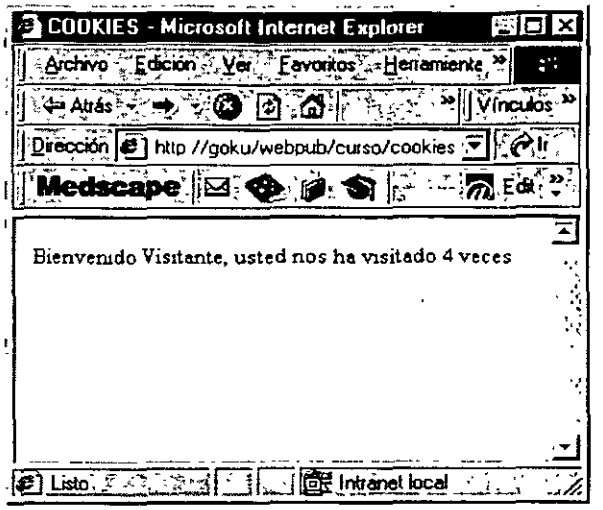


fig. 3.4 El estado de la pagina tras varias visitas.

Como puede apreciarse en el ejemplo, el juego entre los objetos Request y Response, es lo que nos permite extraer toda su utilidad a las cookies.

6.3 Propiedades de las cookies.

Las cookies tienen algunas propiedades interesantes que es importante tomar en cuenta para el buen desempeño de los sitios en los que se utilicen; por ejemplo la caducidad.

Cada una de las cookies establecidas para una pagina tendrá como duración predeterminada la sesión de navegación del cliente, después de la serán borradas, para modificar este hecho existe la propiedad **expires** en esta se fijara la fecha y hora de expiración de cada una de las cookies, el procedimiento se muestra a continuación:

```
Response.cookies(``nombrecookie'').expires=#mes dia,año hora
```

Después de que se haya alcanzado la fecha de expiración se borrara la cookie de la maquina del usuario y la próxima vez que ingrese al sitio esta se grabara de nuevo.

Otra de las propiedades es el dominio al que pertenece la cookie fijada, esta propiedad se establece mediante la sentencia **domain** , si el dominio no es definido solo el servidor de este desde el cual se estableció podrá acceder a ella:

```
Response.cookies(``nombrecookie'').domain=''nombre dominio''
```

Una mas seria la de la ruta de la cookie, esta representa el lugar desde el servidor en el que se podrá tener acceso a la cookie, esta ruta se establece mediante la propiedad **path** :

```
Response.cookies(``nombrecookie'').path =''ruta''
```

Por ultimo mencionaremos la propiedad **secure** que de ser establecida a **true** enviara la cookie por medio de una conexión segura.