Manual ASP

Crear una página ASP

Un archivo de páginas Active Server (ASP) es un archivo de texto con la extensión .asp que contiene cualquier combinación de lo siguiente:

- Texto
- Etiquetas HTML
- Secuencias de comandos del servidor

Un método rápido para crear un archivo .asp consiste en cambiar la extensión de los archivos HTML (.html o .htm) por la extensión .asp. Si el archivo no contiene funciones ASP, el servidor prescinde del proceso de secuencias de comandos ASP y envía el archivo al cliente. Como desarrollador Web, esta opción proporciona una gran flexibilidad, ya que puede asignar a los archivos la extensión .asp incluso si no piensa agregar funciones ASP hasta más adelante.

Para publicar el archivo .asp en Web, guarde el nuevo archivo en un directorio virtual de su sitio Web (asegúrese de que el directorio tenga los permisos Secuencia de comandos o Ejecución). A continuación, escriba en el explorador la dirección URL del archivo para pedirlo. (Recuerde, las páginas ASP debe enviarlas el servidor, por lo que no puede pedirlas mediante su ruta física.) Cuando el archivo se cargue en el explorador, observará que el servidor envió una página HTML. Al principio puede parecer extraño, pero recuerde que el servidor analiza y ejecuta todas las secuencias de comandos ASP del servidor antes de enviar el archivo. El usuario siempre recibe código HTML estándar.

Para crear archivos .asp, se puede utilizar cualquier editor de textos. A medida que avance, puede que encuentre más productivo utilizar un editor más orientado a ASP, como Microsoft(r) Visual InterDev™. (Para obtener más información, visite el sitio Web de Microsoft Visual InterDev en la dirección http://msdn.microsoft.com/vinterdev/.)

Agregar secuencias de comandos del servidor

Una secuencia de comandos del servidor es una serie de instrucciones que se utiliza para enviar al servidor Web comandos de forma secuencial. (Si ya desarrolló antes sitios Web, probablemente conozca las secuencias de comandos del cliente, que se ejecutan en el explorador Web.) En los archivos .asp, las secuencias de comandos se separan del texto y de las etiquetas HTML mediante delimitadores. Un *delimitador* es un carácter o una secuencia de caracteres que marca el principio o el final de una unidad. En el caso de HTML, dichos delimitadores son los símbolos menor que (<) y mayor que (>), que enmarcan las etiquetas HTML.

ASP utiliza los delimitadores <% y %> para enmarcar los comandos. Dentro de los delimitadores puede incluir cualquier comando válido dentro del lenguaje de secuencia de comandos que esté utilizando. El ejemplo siguiente muestra una página HTML sencilla que contiene un comando de secuencia de comandos:

<HTML>

<BODY>

Esta página se actualizó por última vez el <%= Now ()%>.

</BODY>

</HTML>

La función **Now()** de VBScript devuelve la fecha y la hora actuales. Cuando el servidor Web procesa esta página, reemplaza <%= Now ()%> con la fecha y la hora actuales, y devuelve la página al explorador con el siguiente resultado:

Esta página se actualizó el 1/29/99 2:20:00 p.m.

A los comandos enmarcados por delimitadores se les llama *comandos principales de secuencias de comandos*, que se procesan mediante el lenguaje principal de secuencia de comandos. Todos los comandos utilizados dentro de los delimitadores de secuencias de comandos deben ser válidos en el lenguaje principal de secuencia de comandos. De forma predeterminada, el lenguaje principal de secuencia de comandos es VBScript, pero también puede establecer un lenguaje diferente. Consulte Trabajar con lenguajes de secuencias de comandos.

Si ya conoce las secuencias de comandos del cliente, ya sabrá que la etiqueta HTML <SCRIPT> se utiliza para delimitar las secuencias de comandos y las expresiones. También puede utilizar la etiqueta <SCRIPT> para las secuencias de

comandos del cliente, siempre que necesite definir procedimientos en múltiples lenguajes en un archivo .asp. Para obtener más información, consulte Trabajar con lenguajes de secuencias de comandos.

Combinar HTML y comandos de secuencias de comandos

Dentro de los delimitadores de ASP puede incluir cualquier instrucción, expresión, procedimiento u operador que sea válido en el lenguaje principal para secuencia de comandos. Una *instrucción*, en VBScript y en otros lenguajes, es una unidad sintácticamente completa que expresa un tipo de acción, declaración o definición. La instrucción condicional **If...Then...Else** que aparece a continuación es una instrucción de VBScript muy común:

```
<%

Dim dtmHour

dtmHour = Hour(Now())

If dtmHour < 12 Then

Saludos = "Buenos días"

Else

strGreeting = "Hola"

End If

%>

<%= strGreeting %>
```

Según la hora, la secuencia de comandos asigna el valor "Buenos días" o el valor "Hola" a la variable de cadena strGreeting. La instrucción <%= strGreeting %> envía al explorador el valor actual de la variable.

De esta forma, los usuarios que vean esta secuencia de comandos antes de las 12:00 de la mañana (según la zona horaria del servidor Web) verían la siguiente línea de texto:

```
Buenos días
```

Los usuarios que vean esta secuencia de comandos después de las 12:00 de la mañana verían la siguiente línea de texto:

```
Hola
```

Puede incluir texto HTML entre las secciones de una instrucción. Por ejemplo, la secuencia de comandos siguiente, que combina HTML dentro de una instrucción **If...Then...Else**, produce el mismo resultado que la del ejemplo anterior:

```
<%

Dim dtmHour

dtmHour = Hour(Now())

If dtmHour < 12 Then

%>

Buenos días

<% Else %>

Hola
```

```
<%End If %>
```

Si la condición es verdadera, es decir, si es antes del mediodía, el servidor Web envía al explorador el código HTML que sigue a la condición ("Buenos días"); de lo contrario, envía el código HTML que sigue a **Else** ("Hola"). Esta forma de combinar HTML y comandos de secuencia de comandos es cómoda para continuar la instrucción **If...Then...Else** en varias líneas de texto HTML. El ejemplo anterior es más útil si desea presentar un saludo en varias partes de una página Web. Puede asignar el valor de la variable una única vez y después presentarla varias veces.

En lugar de mezclar texto HTML con comandos de secuencia de comandos, puede devolver texto HTML al explorador desde dentro de los comandos de secuencia de comandos. Para devolver texto al explorador, utilice el objeto integrado **Response** de ASP. El ejemplo siguiente produce el mismo resultado que las secuencias de comandos anteriores:

```
<%

Dim dtmHour

dtmHour = Hour(Now())

If dtmHour < 12 Then

Response.Write "Buenos días"

Else

Response.Write "Hola"

End If

%>
```

Response.Write envía al explorador el texto que le sigue. Utilice **Response.Write** desde una instrucción cuando desee generar de forma dinámica el texto devuelto al explorador. Por ejemplo, puede generar una cadena de texto que contenga los valores de varias variables. Aprenderá más acerca del objeto **Response**, y de los objetos en general, en Utilizar componentes y objetos y Enviar contenido al explorador. Por ahora, observe simplemente que dispone de varias maneras de insertar comandos de secuencias de comandos en las páginas HTML.

Puede incluir procedimientos escritos en su lenguaje predeterminado de secuencias de comandos dentro de delimitadores ASP. Para obtener más información, consulte Trabajar con lenguajes de secuencias de comandos.

Si va a trabajar con comandos JScript, puede insertar las llaves que indican un bloque de instrucciones directamente en sus comandos ASP, incluso aunque estén entremezclados con etiquetas y texto HTML. Por ejemplo:

```
<%

if (screenresolution == "low")

{

%>

Ésta es la versión de texto de la página.

<%

}

else

{

%>
```

Ésta es la versión multimedia de la página.

<%

}

%>

-O bien-

<%

if (screenresolution == "low")

{

Response.Write("Ésta es la versión de texto de la página.")

}

else

{

Response.Write("Ésta es la versión multimedia de la página.")

}

%>

Utilizar directivas ASP

ASP proporciona directivas que no forman parte de los lenguajes de secuencias de comandos: Dichas directivas son la directiva de resultado y la directiva de proceso.

La *directiva de resultado* de ASP <%= *expresión* %> presenta el valor de una expresión. Dicha directiva es equivalente al uso de **Response.Write** para presentar información. Por ejemplo, la expresión <%= ciudad %> envía al explorador la palabra Barcelona (el valor actual de la variable).

La *directiva de proceso* de ASP <%@ *palabra clave* %> ofrece a ASP la información que necesita para procesar un archivo .asp. Por ejemplo, la siguiente directiva establece VBScript como lenguaje principal de secuencia de comandos de la página:

<%@ LANGUAGE=VBScript %>

Las directivas de proceso deben estar en la primera línea de los archivos .asp. Para agregar más de una directiva a una página, deben incluirse en el mismo delimitador. No ponga las directivas de proceso en los archivos incluidos con la instrucción **#include**. Debe incluir un espacio en blanco entre el signo @ y la palabra clave. La directiva de proceso tiene las siguientes palabras clave:

- La palabra clave LANGUAGE establece el lenguaje principal de secuencia de comandos de la página.
- La palabra clave ENABLESESSIONSTATE especifica si un archivo ASP utiliza el estado de la sesión. Consulte Administrar sesiones.
- La palabra clave CODEPAGE establece la página de códigos (la codificación de caracteres) del archivo .asp.
- La palabra clave LCID establece el identificador de configuración regional del archivo.

- La palabra clave TRANSACTION especifica que el archivo .asp se ejecutará dentro del contexto de una transacción. Consulte Descripción de las transacciones

Importante Puede incluir más de una palabra clave en una única directiva. Los pares de palabra clave y valor deben estar separados por un espacio. No ponga espacios ni antes ni después del signo igual (=).

El ejemplo siguiente establece el lenguaje de secuencia de comandos y la página de códigos:

<%@ LANGUAGE="Jscript" CODEPAGE="932" %>

Espacio en blanco en las secuencias de comandos

Si su lenguaje principal de secuencias de comandos es VBScript o JScript, ASP quita el espacio en blanco de los comandos. En otros lenguajes, ASP conserva el espacio en blanco para que los lenguajes que interpreten la posición o la sangría puedan interpretarlo correctamente. El espacio en blanco incluye espacios, tabuladores, retornos y saltos de línea.

En VBScript y JScript puede utilizar el espacio en blanco que sigue al delimitador de apertura y que precede al de cierre para facilitar la lectura de los comandos. Todas las instrucciones siguientes son válidas:

<% Color = "Green" %>

<%Color="Green"%>

<%

Color = "Green"

%>

ASP quita el espacio en blanco que se encuentre entre los delimitadores de cierre de las instrucciones y los delimitadores de apertura de las instrucciones siguientes. Sin embargo, se recomienda utilizar espacios para mejorar la legibilidad. Si tiene que conservar el espacio en blanco entre dos instrucciones, como cuando vaya a presentar los valores de las variables en una frase, utilice el carácter de espacio de no separación de HTML ( ). Por ejemplo:

<%

'Define dos variables con valores de cadena.

strFirstName = "Juan"

strLastName = "García"

%>

<P>Esta página Web está personalizada para "<%= strFirstName %> <%= strLastName %>." </P>

Utilizar variables y constantes

Una *variable* es una ubicación de almacenamiento con nombre dentro de la memoria del equipo que contiene datos, como un número o una cadena de texto. A los datos contenidos en una variable se les llama *valor* de la variable. Las variables ofrecen una manera de almacenar, recuperar y manipular valores mediante nombres que ayuden a entender lo que hace la secuencia de comandos.

Declarar y asignar nombre a variables

Siga las reglas y recomendaciones de su lenguaje de secuencias de comandos a la hora de declarar variables y asignarles nombres. Incluso aunque no necesita declarar una variable para poder utilizarla, es conveniente hacerlo porque ayuda a prevenir errores. *Declarar* una variable significa indicar al motor de secuencias de comandos que existe

una variable con un nombre concreto, de forma que pueda hacer referencia a la variable a lo largo de una secuencia de comandos.

VBScript

VBScript no necesita declaraciones de variables, pero es conveniente declarar todas las variables antes de utilizarlas. Para declarar una variable en VBScript, utilice la instrucción **Dim**, **Public** o **Private**. Por ejemplo:

<% Dim NombreUsuario %>

Puede utilizar la instrucción **Option Explicit** de VBScript en los archivos .asp para hacer obligatoria la declaración de variables con las instrucciones **Dim**, **Private**, **Public** y **ReDim**. La instrucción **Option Explicit** debe aparecer después de las directivas ASP y antes del texto HTML o de los comandos de la secuencia de comandos. Esta instrucción sólo afecta a los comandos ASP escritos en VBScript; no afecta a los comandos escritos en JScript.

<% Option Explicit %>

<HTML>

<%

Dim strNombreUsuario

Public lngNumeroCuenta

%>

Alcance de las variables

El *alcance*, o vida, de una variable determina qué comandos de secuencia de comandos pueden tener acceso a dicha variable. Una variable declarada dentro de un procedimiento tiene *alcance local*; la variable se crea y se destruye cada vez que se ejecuta el procedimiento. No se puede tener acceso a ella desde fuera del procedimiento. Una variable declarada fuera de un procedimiento tiene *alcance global*; su valor es accesible y modificable desde cualquier comando de secuencia de comandos de una página ASP.

Nota Al limitar el alcance de la variable a un procedimiento mejorará el rendimiento.

Si declara variables, una variable local y una variable global pueden tener el mismo nombre. La modificación del valor de una de ellas no afecta al valor de la otra. Sin embargo, si no declara las variables, podría modificar inadvertidamente el valor de una variable global. Por ejemplo, los siguientes comandos de secuencia de comandos devuelven el valor 1 incluso aunque haya dos variables llamadas Y:

<%

Option Explicit

Dim Y

Y = 1

SetLocalVariable

Response.Write Y

Sub SetLocalVariable

Dim Y

Y = 2

```
End Sub

%>
```

Por el contrario, los comandos siguientes devuelven el valor 2 porque las variables no se han declarado de forma explícita. Cuando la llamada al procedimiento asigna a Y el valor 2, el motor de secuencias de comandos da por supuesto que el procedimiento pretende modificar la variable global:

```
<%

Option Explicit

Dim Y = 1

SetLocalVariable

Response.Write Y

Sub SetLocalVariable

Y = 2

End Sub

%>
```

Para evitar problemas, adquiera el hábito de declarar explícitamente todas las variables. Lo cual es especialmente importante si utiliza la instrucción **#include** para incluir archivos en su archivo ASP. La secuencia de comandos incluida está contenida en un archivo aparte, pero se trata como si formara parte del archivo contenedor. Es muy fácil olvidarse de que hay que utilizar nombres de variables diferentes en la secuencia de comandos principal y en la secuencia de comandos incluida, a menos que declare las variables.

Asignar a las variables alcance de sesión o de aplicación

Las variables globales sólo son accesibles en un mismo archivo ASP. Para hacer que una variable sea accesible en varias páginas, asigne a la variable alcance de sesión o de aplicación. Las variables con alcance de sesión están disponibles en todas las páginas de una aplicación ASP que pida un mismo usuario. Las variables con alcance de aplicación están disponibles en todas las páginas de una aplicación ASP que pida cualquier usuario. Las variables de sesión son una buena manera de almacenar información para un único usuario, como sus preferencias o el nombre o la identificación del usuario. Las variables de aplicación son una buena manera de almacenar información para todos los usuarios de una determinada aplicación, como los saludos específicos o los valores generales necesarios en la aplicación.

ASP proporciona dos objetos integrados en los que puede almacenar variables: el objeto **Session** y el objeto **Application**.

También puede crear instancias de objetos con alcance de sesión o de aplicación. Para obtener más información, consulte Establecer el alcance de los objetos.

Alcance de sesión

Para asignar alcance de sesión a una variable, almacénela en el objeto **Session** asignando un valor a una entrada con nombre del objeto. Por ejemplo, los siguientes comandos almacenan dos nuevas variables en el objeto **Session**:

```
<%

Session("Nombre") = "Juan"

Session("Apellido") = "Soto"

%>
```

Para recuperar la información del objeto **Session**, tenga acceso a la entrada con nombre mediante la directiva de resultado (<%=) o **Response.Write**. En el ejemplo siguiente se utiliza la directiva de resultado para presentar el valor actual de Session("Nombre"):

Reciba nuestra bienvenida,<%= Session("Nombre") %>

Puede almacenar las preferencias del usuario en el objeto **Session** y después tener acceso a dichas preferencias para determinar qué página hay que devolver al usuario. Por ejemplo, puede permitir que los usuarios especifiquen la versión en texto del contenido de la primera página de la aplicación y aplicar esta opción a las siguientes páginas de la aplicación que el usuario visite.

<%

strScreenResolution = Session("ScreenResolution")

If strScreenResolution = "Low" Then

%>

Ésta es la versión de texto de la página.

<% Else %>

Ésta es la versión multimedia de la página.

<%End If %>

Nota Si hace referencia a una variable con alcance de sesión más de una vez en una secuencia de comandos, piense en asignarle una variable local, como en el ejemplo anterior, para mejorar el rendimiento.

Alcance de aplicación

Para asignar alcance de aplicación a una variable, almacénela en el objeto **Application** asignando un valor a una entrada con nombre del objeto. Por ejemplo, el comando siguiente almacena en el objeto **Application** un saludo específico de una aplicación:

<% Application("Saludo") = "¡Reciba nuestra bienvenida al Departamento de ventas!" %>

Para recuperar la información del objeto **Application**, utilice la directiva de resultado de ASP (<%=) o **Response.Write** para tener acceso a la entrada con nombre desde cualquier página posterior de la aplicación. En el ejemplo siguiente se utiliza la directiva de resultado para presentar el valor de Application("Saludo"):

<%= Application("Saludo") %>

De nuevo, si hace referencia a una variable con alcance de aplicación en su secuencia de comandos repetidamente, debe de asignarle una variable local para mejorar el rendimiento.

Utilizar constantes

Una *constante* es un nombre que representa un número o una cadena. Algunos de los componentes de base que se proporcionan con ASP, como ActiveX Data Objects (ADO), definen constantes que se pueden utilizar en las secuencias de comandos. Un componente puede declarar constantes en la *biblioteca de tipos del componente*, un archivo que contiene información acerca de los objetos y los tipos aceptados por un componente COM. Después de haber declarado una biblioteca de tipos en su archivo .asp puede usar las constantes definidas en cualquier secuencia de comandos en el mismo archivo .asp. Igualmente, puede declarar una biblioteca de tipos en el archivo Global.asa para usar las constantes definidas en cualquier archivo .asp de la aplicación.

Para declarar una biblioteca de tipos, utilice la etiqueta <METADATA> en su archivo .asp o Global.asa. Por ejemplo, para declarar la biblioteca de tipos de ADO, utilice las siguientes instrucciones:

<!--METADATA NAME="Microsoft ActiveX Data Objects 2.5 Library" TYPE="TypeLib" UUID="{00000205-0000-0010-8000-00AA006D2EA4}"-->

O bien, en lugar de hacer referencia al identificador único universal (UUID) de la biblioteca de tipos, puede hacer referencia a la biblioteca de tipos mediante la ruta del archivo:

<!-- METADATA TYPE="typelib" FILE="c:\program files\common files\system\ado\msado15.dll"-->

Entonces puede usar las constantes ADO en el archivo .asp donde declaró la biblioteca de tipos o en un archivo que reside en una aplicación que contiene un archivo Global.asa con la declaración de biblioteca de tipos ADO. En el ejemplo siguiente, adOpenKeyset y adLockOptimistic son constantes ADO:

<%

'Crea y abre el objeto Recordset.

Set rsCustomersList = Server.CreateObject("ADODB.Recordset")

rstCustomerList.ActiveConnection = cnnPubs

rstCustomerList.CursorType = adOpenKeyset

rstCustomerList.LockType = adLockOptimistic

%>

En la siguiente tabla se enumeran las bibliotecas de tipos y los identificadores UUID que se utilizan con más frecuencia:

Biblioteca de tipos	**UUID**
Biblioteca de Microsoft ActiveX Data Objects 2.5	{00000205-0000-0010-8000-00AA006D2EA4}
Biblioteca de Microsoft CDO 1.2 para Windows 2000 Server	{0E064ADD-9D99-11D0-ABE5-00AA0064D470}
Biblioteca MSWC de objetos Advertisement Rotator	{090ACFA1-1580-11D1-8AC0-00C0F00910F9}
Biblioteca MSWC de objetos de registro de IIS	{B758F2F9-A3D6-11D1-8B9C-080009DCC2FA}

Interactuar con secuencias de comandos del cliente

Es posible aumentar la eficacia de ASP si se utiliza para generar o manipular secuencias de comandos del cliente. Por ejemplo, puede escribir secuencias de comandos del servidor que construyan secuencias de comandos del cliente basadas en variables específicas del servidor, el tipo del explorador o los parámetros de la petición HTTP.

Si intercala instrucciones de secuencias de comandos del servidor en las secuencias de comandos del cliente (delimitadas mediante etiquetas HTML <SCRIPT>), como se muestra en la plantilla del ejemplo siguiente, puede inicializar de forma dinámica y alterar las secuencias de comandos del cliente en el momento de realizar la petición:

<SCRIPT LANGUAGE="VBScript">

<!--

variable = <%=*valor definido por el servidor* %>

.

.

.

secuencia de comandos del cliente

<% *secuencia de comandos del servidor que se utiliza para generar una instrucción del cliente* %>

secuencia de comandos del cliente

.

.

.

-->

</SCRIPT>

Al incorporar estas funciones es posible crear algunas aplicaciones útiles e interesantes. Por ejemplo, ésta es una sencilla secuencia de comandos del servidor (escrita en VBScript) que manipula una secuencia de comandos del cliente (escrita en JScript):

<%

Dim dtmTime, strServerName, strServerSoftware, intGreeting

dtmTime = Time()

strServerName = Request.ServerVariables("SERVER_NAME")

strServerSoftware = Request.ServerVariables("SERVER_SOFTWARE")

'Genera un número aleatorio.

Randomize

GreetCondition = int(rnd * 3)

%>

<SCRIPT LANGUAGE="JScript">

<!--

//Llama a la función para mostrar el saludo

showIntroMsg()

function showIntroMsg()

{

switch(<%= intGreeting %>)

{

case 0:

msg = "Este es el servidor Web <%= strServerName%> con <%= strServerSoftware %>."

```
break



case 1:

msg = "Reciba nuestra bienvenida al servidor Web <%= strServerName%>. La hora local es <%= dtmTime %>."

break

case 2:

msg = "Este servidor utiliza <%= strServerSoftware %>."

break

}

document.write(msg)

}

-->

</SCRIPT>
```

Escribir procedimientos

Un *procedimiento* es un grupo de comandos de secuencia de comandos que realizan una tarea específica y puede devolver un valor. Puede definir sus propios procedimientos e invocarlos repetidamente desde sus secuencias de comandos.

Puede poner las definiciones de los procedimientos en el mismo archivo .asp que llama a los procedimientos o bien puede poner los procedimientos utilizados con más frecuencia en un archivo .asp compartido y utilizar la directiva #include para incluirlo en otros archivos .asp que llamen a los procedimientos. Como alternativa, puede encapsular dicha funcionalidad en un componente COM.

Definir procedimientos

Las definiciones de los procedimientos pueden encontrarse dentro de etiquetas <SCRIPT> y </SCRIPT> y deben seguir las reglas del lenguaje de secuencias de comandos. Utilice el elemento <SCRIPT> para los procedimientos escritos en lenguajes distintos del lenguaje principal para secuencias de comandos. Sin embargo, utilice los delimitadores de las secuencias de comandos (<% y %>) en los procedimientos escritos en el lenguaje principal de secuencias de comandos.

Cuando utilice la etiqueta HTML <SCRIPT>, debe emplear dos atributos para asegurar el proceso de la secuencia de comandos por parte del servidor. La sintaxis de la etiqueta <SCRIPT> es la siguiente:

```
<SCRIPT LANGUAGE=JScript RUNAT=SERVER>

definición del procedimiento

</SCRIPT>
```

El atributo RUNAT=SERVER indica al servidor Web que procese la secuencia de comandos en el servidor. Si no establece este atributo, la secuencia de comandos la procesará el explorador del cliente. El atributo LANGUAGE determina el lenguaje de secuencia de comandos utilizado en este bloque. Puede especificar cualquier lenguaje para el que haya instalado un motor de secuencias de comandos en el servidor. Para especificar VBScript, use el valor VBScript. Para especificar JScript, use el valor JScript. Si no asigna el atributo LANGUAGE, el bloque de la secuencia de comandos se interpretará en el lenguaje principal de secuencia de comandos.

Los comandos del bloque de la secuencia de comandos deben componer uno o varios procedimientos completos en el lenguaje de secuencia de comandos elegido. Por ejemplo, los comandos siguientes definen el procedimiento JScript **MiFuncion**.

```
<HTML>

<SCRIPT LANGUAGE=JScript RUNAT=SERVER >

function MiFuncion()

{

Response.Write("Llamó a MiFuncion().")

}

</SCRIPT>
```

Importante No incluya en las etiquetas <SCRIPT> del servidor comandos de secuencias de comandos que no formen parte de procedimientos completos. Los comandos que no formen parte de un procedimiento pueden provocar resultados impredecibles puesto que deben ejecutarse en un orden determinado. Además, no puede utilizar la directiva de resultado de ASP <%=%> dentro de los procedimientos. En su lugar, utilice **Response.Write** para enviar contenido al explorador.

Llamar a procedimientos

Para llamar a procedimientos, incluya el nombre de los procedimientos en un comando. Si va a llamar a procedimientos JScript desde VBScript, debe utilizar paréntesis después del nombre del procedimiento; si el procedimiento no tiene argumentos, utilice paréntesis vacíos. Si va a llamar a procedimientos VBScript o JScript desde JScript, utilice siempre paréntesis después del nombre del procedimiento.

En VBScript también puede utilizar la palabra clave **Call** para llamar a un procedimiento. Sin embargo, si el procedimiento al que llama requiere argumentos, la lista de argumentos debe aparecer entre paréntesis. Si omite la palabra clave **Call**, también debe omitir los paréntesis en los que se incluye la lista de argumentos. Si utiliza la sintaxis **Call** para llamar a una función integrada o definida por el usuario, se descartará el valor devuelto por la función.

El ejemplo siguiente ilustra la creación y la llamada a procedimientos en dos lenguajes para secuencias de comandos diferentes (VBScript y JScript).

```
<%@ LANGUAGE=VBScript %>

<HTML>

<BODY>

<% Echo %>

<BR>

<% printDate() %>

</BODY>

</HTML>

<%

Sub Echo

Response.Write "<TABLE>" & _

"Response.Write "<TR><TH>Nombre</TH><TH>Valor</TH></TR>"
```

```
Set objQueryString = Request.QueryString

For Each strSelection In objQueryString

Response.Write "<TR><TD>" & p & "</TD><TD>" & _

FormValues(strSelection) & "</TD></TR>"

Next

Response.Write "</TABLE>"

End Sub

%>

<SCRIPT LANGUAGE=JScript RUNAT=SERVER>

function PrintDate()

{

var x

x = new Date()

Response.Write(x.toString())

}

</SCRIPT>
```

Nota Las llamadas de VBScript a las funciones JScript *no* distinguen entre mayúsculas y minúsculas.

Pasar matrices a procedimientos

Para pasar una matriz entera a un procedimiento en VBScript, utilice el nombre de la matriz seguido de paréntesis vacíos; en JScript, utilice corchetes vacíos.

Procesar los datos proporcionados por el usuario

Mediante el objeto **Request** de ASP puede crear sencillas y eficaces secuencias de comandos para recopilar y procesar datos obtenidos de formularios HTML. En este tema no sólo aprenderá a crear secuencias de comandos básicas para procesar formularios, sino que también conocerá técnicas útiles para validar los datos de los formularios, tanto en su servidor Web como en el explorador de Web del usuario.

Acerca de los formularios HTML

Los formularios HTML, el método más común para recopilar información desde el Web, consiste en un conjunto de etiquetas HTML especiales que presentan elementos de interfaz de usuario en una página Web. Los cuadros de texto, los botones y las casillas de verificación son ejemplos de elementos que permiten que los usuarios intercalen con una página Web y envíen información a un servidor Web.

Por ejemplo, las siguientes etiquetas HTML generan un formulario en el que un usuario puede escribir su nombre, apellido y edad, e incluye un botón para enviar la información a un servidor Web. El formulario también contiene una etiqueta de entrada oculta (no presentada por el explorador Web) que puede utilizar para pasar información adicional al servidor Web.

```
<FORM METHOD="Post" ACTION="Profile.asp">
```

<INPUT TYPE="text" NAME="Nombre">

<INPUT TYPE="text" NAME="Apellido">

<INPUT TYPE="text" NAME="Edad">

<INPUT TYPE="Hidden" NAME="EstadoUsuario" VALUE="Nuevo">

<INPUT TYPE="Submit" VALUE="Entrar">

</FORM>

Presentar con detalle el conjunto completo de etiquetas HTML para formularios se sale del ámbito de este tema, sin embargo, hay numerosas fuentes de información que puede utilizar para aprender a crear y utilizar formularios HTML. Por ejemplo, puede utilizar la posibilidad que ofrece su explorador para ver el código fuente con el fin de examinar cómo se crean los formularios HTML en otros sitios Web. También puede visitar el sitio Web de MSDN de Microsoft en la dirección http://msdn.microsoft.com/ para conocer las técnicas más recientes para utilizar los formularios HTML con otras tecnologías de Internet.

Procesar datos de formularios con ASP

Después de crear el formulario HTML tendrá que procesar los datos proporcionados por el usuario, lo que implica enviar la información a un archivo .asp para que la analice y manipule. De nuevo, examine el código HTML del ejemplo anterior. Observe que el atributo ACTION de la etiqueta <FORM> hace referencia a un archivo llamado Profile.asp. Cuando el usuario envía información HMTL, el explorador utiliza el método POST para enviarla a un archivo .asp del servidor, en este caso Profile.asp. Este archivo .asp puede contener archivos de comandos que procesen la información e interactúen con otras secuencias de comandos, componentes COM o recursos, como una base de datos.

Existen tres formas básicas de recopilar información de un formulario HTML mediante ASP:

- Un archivo .htm estático puede contener un formulario que envíe sus valores a un archivo .asp.
- Un archivo .asp puede crear un formulario que envíe información a otro archivo .asp.
- Un archivo .asp puede crear un formulario que se envíe información a sí mismo; es decir, al archivo .asp que contiene el formulario.

Los dos primeros métodos funcionan igual que los formularios que interactúan con otros programas de servidor Web, con la única diferencia de que con ASP se simplifica mucho la tarea de recuperar y procesar la información. El tercer método es especialmente útil y se muestra en la sección Validar los datos de los formularios.

Obtener datos de los formularios

El objeto **Request** de ASP proporciona dos colecciones que facilitan la tarea de recuperar información de los formularios enviados con las peticiones URL.

La colección QueryString

La colección **QueryString** recupera los valores del formulario pasados al servidor Web como texto a continuación del signo de interrogación de la dirección URL de la petición. Los valores del formulario se pueden anexar a la dirección URL de la petición mediante el método GET de HTTP o, manualmente, si se agregan los valores del formulario a la dirección URL.

Si el ejemplo del formulario anterior utilizara el método GET (METHOD = "GET") y el usuario escribiera *Juan*, *Soto* y *30*, se enviaría la siguiente petición URL al servidor:

http://Workshop1/Painting/Profile.asp?Nombre=Juan&Apellido=Soto&Edad=30&EstadoUsuario=Nuevo

El archivo Profile.asp podría contener la siguiente secuencia de comandos para procesar formularios:

Hola, <%= Request.QueryString("nombre") %> <%= Request.QueryString("apellido") %>.

Tiene <%= Request.QueryString("edad") %> años.

<%

```
If Request.QueryString("EstadoUsuario") = "Nuevo" Then

Response.Write"Ésta es su primera visita a este sitio Web"

End if

%>
```

En este caso, el servidor Web devolvería el texto siguiente al explorador Web del usuario:

Hola Juan Soto. Tiene 30 años. Ésta es su primera visita a este sitio Web

La colección **QueryString** también tiene un parámetro opcional que puede utilizar para tener acceso a uno de los múltiples valores que se encuentran en el cuerpo de la petición URL (con el método GET). También puede utilizar la propiedad **Count** para contar el número de veces que aparece un determinado tipo de valor.

Por ejemplo, un formulario que contenga un cuadro de lista con varios elementos puede generar la siguiente petición:

http://OrganicFoods/list.asp?Comida=Manzanas&Comida=Aceitunas&Comida=Pan

Podría utilizar el siguiente comando para contar los diferentes valores:

```
Request.QueryString("Comida").Count
```

Para presentar los tipos de valores, Lista.asp podría contener la secuencia de comandos siguiente:

```
<%

lngTotal = Request.QueryString("Comida").Count

For i = 1 To lngTotal

Response.Write Request.QueryString("Comida")(i) & "<BR>"

Next

%>
```

La secuencia de comandos anterior mostraría:

Manzanas

Aceitunas

Pan

También puede mostrar la lista completa de valores como una cadena delimitada por comas, del modo siguiente:

```
<% Response.Write Request.QueryString("Item") %>
```

Aparecería la cadena siguiente:

Manzanas, Aceitunas, Pan

Colección Form

Cuando utiliza el método GET de HTTP para pasar a un servidor Web valores de un formulario grande y complejo, corre el riesgo de perder información. Algunos servidores Web tienden a restringir el tamaño de la cadena de petición de URL, por lo que los valores de los formularios grandes pasados con el método GET podrían quedar truncados. Si necesita enviar una gran cantidad de información desde un formulario a un servidor Web, debe utilizar el método POST de HTTP. El método POST, que envía los datos de los formularios en el cuerpo de la petición HTTP, puede enviar un número casi

ilimitado de caracteres a un servidor. Puede utilizar la colección **Form** del objeto **Request** de ASP para recuperar los valores enviados mediante el método POST.

La colección **Form** almacena valores de manera similar a la colección **QueryString**. Por ejemplo, si un usuario completa un formulario escribiendo una larga lista de nombres, se podrían leer los nombres con la secuencia de comandos siguiente:

```
<%

lngTotal = Request.Form("Comida").Count

For i = 1 To lngTotal

Response.Write Request.Form("Comida")(i) & "<BR>"

Next

%>
```

Validar los datos de los formularios

Un formulario Web bien diseñado suele incluir una secuencia de comandos del cliente que valida los datos proporcionados por el usuario antes de enviar la información al servidor. Las *secuencias de comandos de validación* pueden comprobar si el usuario escribió un número válido o si un cuadro de texto está en blanco. Imagine que su sitio Web incluye un formulario que permite calcular la tasa de retorno de una inversión. Probablemente querrá comprobar si un usuario realmente escribió texto o números en los campos apropiados del formulario, antes de enviar al servidor información que podría no ser válida.

En general, lo mejor es realizar en el cliente tantas comprobaciones como sea posible. Además, de poder preguntar antes al usuario por los errores, la validación en el cliente mejora el tiempo de respuesta, reduce la carga del servidor y libera ancho de banda para otras aplicaciones.

La siguiente secuencia de comandos del cliente valida los datos escritos por el usuario (en este caso, la secuencia de comandos determina si el número de cuenta que escribió el usuario es realmente un número) antes de enviarlos al servidor:

```
<SCRIPT LANGUAGE="JScript">

function ComprobarNumero()

{

if (isNumeric(document.UserForm.AcctNo.value))

return true

else

{

alert("Escriba un número de cuenta válido.")

return false

}

}

//Función para determinar si el valor del formulario es un número.

//Nota: El método isNaN JScript es un método más elegante para determinar si
```

```
//un valor no es un número. Sin embargo, algunos exploradores no admiten este método.

function isNumeric(str)

{

for (var i=0; i < str.length; i++)

{

var ch = str.substring(i, i+1)

if( ch < "0" || ch>"9" || str.length == null)

{

return false

}

}

return true

}

</SCRIPT>

<FORM METHOD="Get" ACTION="balance.asp" NAME="FormularioUsuario" ONSUBMIT="return CheckNumber()">

<INPUT TYPE="Text" NAME="NumCuen">

<INPUT TYPE="Submit" VALUE="Enviar">

</FORM>
```

Sin embargo, si la validación de un formulario requiere el acceso a una base de datos debe considerar la utilización de la validación del formulario en el servidor. Una forma muy útil de realizar la validación en el servidor es crear formularios que se envíen la información a sí mismos. Es decir, el archivo .asp contiene en realidad el formulario HTML que recibe los datos del usuario. (Recuerde, puede utilizar ASP para interactuar con secuencias de comandos del cliente y código HTML. Para obtener más información, consulte Interactuar con secuencias de comandos del cliente.) Los datos escritos vuelven al mismo archivo, que se encarga de validar la información y avisa al usuario si éstos no son correctos.

Mediante este método se pueden mejorar las características de uso y la respuesta de los formularios basados en Web. Por ejemplo, si se incluye información del error junto al campo del formulario en el que se escribió la información incorrecta, será más fácil para el usuario descubrir el origen del error. (Normalmente, los formularios basados en Web reenvían las peticiones a una página Web independiente que contiene información del error. Los usuarios que no comprendan esta información podrían sentirse frustrados.)

Por ejemplo, la secuencia de comandos siguiente determina si un usuario escribió un número de cuenta válido, para lo que se envía la información a sí misma (Verify.asp) y llama a una función definida por el usuario que realiza una consulta a una base de datos:

```
<%

strAcct = Request.Form("Cuenta")

If Not AccountValid(strAcct) Then

ErrMsg = "<FONT COLOR=Red>El número de cuenta que ha escrito no es válido.</FONT>"

Else
```

Procesa los datos del usuario

.

.

.

Server.Transfer("Complete.asp")

End If

Function AccountValid(strAcct)

Aquí se incluirá una secuencia de comandos o una llamada a un método de un componente de conexión con una base de datos.

End Function

%>

<FORM METHOD="Post" ACTION="Verify.asp">

Número de cuenta: <INPUT TYPE="Text" NAME="Cuenta"> <%= ErrMsg %>

<INPUT TYPE="Submit">

</FORM>

En este ejemplo, la secuencia de comandos se encuentra en un archivo llamado Verify.asp, el mismo archivo que contiene el formulario HTML y, para enviarse la información a sí mismo, especifica Verify.asp en el atributo ACTION.

Importante Si utiliza JScript para la validación en el servidor, asegúrese de colocar un par de paréntesis vacíos detrás del elemento de la colección **Request** (**QueryString** o **Form**) al asignar la colección a una variable local. Sin el paréntesis la colección devolverá un objeto en lugar de una cadena. La secuencia de comandos siguiente muestra el método correcto para asignar variables con JScript:

<%

var Nombre = Request.Form("Nombre")();

var Contraseña = Request.Form("Contraseña")();

if(Nombre > "")

{

if(Nombre == Contraseña)

Response.Write("El nombre y la contraseña son iguales".)

else

Response.Write("El nombre y la contraseña son diferentes".);

}

%>

VBScript presenta el mismo comportamiento si la colección contiene varios valores separados por comas o con los que se pueda crear un índice. Esto significa que tanto para VBScript como para JScript, además de colocar un par de paréntesis detrás del elemento de la colección **Request**, necesitará especificar el índice del valor deseado. Por ejemplo, la siguiente línea en JScript devuelve sólo el primero de los varios valores de un elemento de un formulario:

var Nombre = Request.Form("Nombre")(1);

Acceso al origen de datos

ActiveX Data Objects (ADO) es una tecnología ampliable y de fácil uso para agregar a sus páginas Web acceso a bases de datos. Puede utilizar ADO para escribir secuencias de comandos compactas y escalables que le permitan conectarse a orígenes de datos compatibles con OLE DB, como bases de datos, hojas de cálculo, archivos de datos secuenciales o directorios de correo electrónico. OLE DB es una interfaz de programación de nivel de sistema que proporciona un conjunto estándar de interfaces COM para que permitan exponer las funciones del sistema de administración de bases de datos. Con el modelo de objetos ADO es fácil tener acceso a estas interfaces (mediante lenguajes de secuencias de comandos, como VBScript o JScript) para agregar funciones de bases de datos a las aplicaciones Web. Además, también puede utilizar ADO para tener acceso a bases de datos compatibles con la Conectividad abierta de bases de datos (ODBC, Open Database Connectivity).

Si no tiene mucha experiencia en conectividad con bases de datos, encontrará que la sintaxis de ADO es sencilla y fácil de utilizar. Si es un programador experimentado, agradecerá el acceso escalable de alto rendimiento que proporciona ADO para una gran variedad de orígenes de datos.

Crear una cadena de conexión

El primer paso en la creación de una aplicación de datos en Web consiste en proporcionar un método para que ADO encuentre e identifique el origen de datos. Para ello se utiliza una *cadena de conexión*, una serie de argumentos separados mediante un punto y coma que definen parámetros como el proveedor del origen de datos y la ubicación del mismo. ADO utiliza la cadena de conexión para identificar el *proveedor* OLE DB y para dirigir al proveedor al origen de datos. El proveedor es un componente que representa el origen de datos y que expone la información en la aplicación en forma de conjuntos de filas.

En la tabla siguiente se enumeran las cadenas de conexión de OLE DB para varios orígenes de datos habituales:

Origen de datos	**Cadena de conexión OLE DB**
Microsoft(r) Access	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=*ruta física de acceso al archivo .mdb*
Microsoft SQL Server	Provider=SQLOLEDB.1;Data Source=*ruta de acceso a la base de datos del servidor*
Oracle	Provider=MSDAORA.1;Data Source=*ruta de acceso a la base de datos del servidor*
Microsoft Indexing Service	Provider=MSIDXS.1;Data Source=*ruta de acceso al archivo*

Para proporcionar compatibilidad con versiones anteriores, el proveedor OLE DB para ODBC admite la sintaxis de las cadenas de conexión ODBC. En la tabla siguiente se enumeran las cadenas de conexión ODBC que se utilizan habitualmente:

Controlador del origen de datos	**Cadena de conexión ODBC**
Microsoft Access	Driver={Microsoft Access Driver (*.mdb)};DBQ=*ruta física de acceso*

	al archivo .mdb
SQL Server	DRIVER={SQL Server};SERVER=*ruta de acceso al servidor*
Oracle	DRIVER={Microsoft ODBC for Oracle};SERVER=*ruta de acceso al servidor*
Microsoft Excel	Driver={Microsoft Excel Driver (*.xls)};DBQ=*ruta física de acceso al archivo .xls*; DriverID=278
Microsoft Excel 97	Driver={Microsoft Excel Driver (*.xls)};DBQ=*ruta física de acceso al archivo .xls*;DriverID=790
Paradox	Driver={Microsoft Paradox Driver (*.db)};DBQ=*ruta física de acceso al archivo .db*;DriverID=26
Texto	Driver={Microsoft Text Driver (*.txt;*.csv)};DefaultDir=*ruta física de acceso al archivo .txt*
Microsoft Visual FoxPro(r) (con un contenedor de bases de datos)	Driver={Microsoft Visual FoxPro Driver};SourceType=DBC;SourceDb=*ruta física de acceso al archivo .dbc*
Microsoft Visual FoxPro (sin un contenedor de bases de datos)	Driver={Microsoft Visual FoxPro Driver};SourceType=DBF;SourceDb=*ruta física de acceso al archivo .dbf*

Conectarse al origen de datos

ADO proporciona el objeto **Connection** para establecer y administrar las conexiones entre las aplicaciones y los orígenes de datos compatibles con OLE DB o las bases de datos compatibles con ODBC. El objeto **Connection** incorpora propiedades y métodos que se pueden utilizar para abrir y cerrar conexiones con bases de datos, y para enviar consultas de actualización de la información.

Para establecer una conexión con una base de datos, cree primero una instancia del objeto **Connection**. Por ejemplo, la secuencia de comandos siguiente crea una instancia del objeto **Connection** y procede a abrir una conexión:

```
<%

'Crea un objeto Connection.

Set cn = Server.CreateObject("ADODB.Connection")

'Abre una conexión mediante la cadena de conexión OLE DB.

cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\DatosMercado\VentasPrevistas.mdb"

%>
```

Nota La cadena de conexión no contiene espacios en blanco ni antes ni después del signo igual (=).

En este caso, el método **Open** del objeto **Connection** se refiere a la cadena de conexión.

Ejecutar consultas SQL con el objeto Connection

Con el método **Execute** del objeto **Connection** puede emitir comandos al origen de datos, como consultas de SQL (Lenguaje de consulta estructurado). (SQL, lenguaje estándar para comunicarse con bases de datos, define comandos para recuperar y actualizar información.) El método **Execute** acepta parámetros que especifiquen el comando (o la consulta), el número de registros de datos a los que afecta y el tipo de comando que se utiliza.

La siguiente secuencia de comandos utiliza el método **Execute para enviar una consulta con un comando INSERT de SQL, que inserta datos en una tabla concreta de la base de datos. En este caso, el bloque de la secuencia de comandos inserta el nombre *José Lugo* en una tabla de la base de datos llamada *Customers*.**

```
<%
```

```
'Define la cadena de conexión OLE DB.

strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Datos\Empleados.mdb"

'Crea la instancia del objeto Connection y abre una conexión con la base de datos.

Set cn = Server.CreateObject("ADODB.Connection")

cnn.Open strConnectionString

'Define la instrucción SELECT de SQL.

strSQL = "INSERT INTO Customers (FirstName, LastName) VALUES ('José','Lugo')"

'Utiliza el método Execute para enviar una consulta SQL a la base de datos.

cnn.Execute strSQL,,adCmdText + adExecuteNoRecords

%>
```

Observe que se especifican dos parámetros en la instrucción que se utiliza para ejecutar la consulta: **adCmdText** y **adExecuteNoRecords**. El parámetro opcional **adCmdText** especifica el tipo de comando e indica que el proveedor debe evaluar la instrucción de consulta (en este caso, una consulta SQL) como una definición textual de un comando. El parámetro **adExecuteNoRecords** indica a ADO que no debe crear un conjunto de registros de datos si no se devuelven resultados a la aplicación. Este parámetro sólo funciona con los tipos de comandos definidos como texto, como las consultas SQL, o con procedimientos almacenados de bases de datos. Aunque los parámetros **adCmdText** y **adExecuteNoRecords** son opcionales, debe especificarlos al utilizar el método **Execute** para mejorar así el rendimiento de la aplicación de datos.

Importante Los parámetros ADO, como **adCmdText**, deben estar definidos para poder utilizarlos en una secuencia de comandos. Un método cómodo para definir los parámetros consiste en utilizar una *biblioteca de tipos de componentes*, que es un archivo que contiene definiciones para todos los parámetros ADO. Para implementar una biblioteca de tipos de componentes debe declararla antes. Agregue la etiqueta siguiente <METADATA> al archivo .asp o a Global.asa para declarar la biblioteca de tipos ADO:

```
<!--METADATA NAME="Microsoft ActiveX Data Objects 2.5 Library" TYPE="TypeLib" UUID="{00000205-0000-0010-8000-00AA006D2EA4}"-->
```

Si desea obtener más detalles acerca de cómo implementar las bibliotecas de tipos de componentes, consulte la sección Utilizar constantes del tema Utilizar variables y constantes.

Además del comando **INSERT** de SQL, puede utilizar los comandos **UPDATE** y **DELETE** de SQL para modificar y quitar información de la base de datos.

Con el comando **UPDATE** de SQL puede modificar los valores de los elementos de una tabla de la base de datos. La siguiente secuencia de comandos usa el comando **UPDATE** para cambiar todos los campos FirstName de la tabla Customers a Juan en todas las filas cuyo campo LastName contenga el apellido Soto.

```
<%

Set cn = Server.CreateObject("ADODB.Connection")

cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Datos\Empleados.mdb"

cn.Execute "UPDATE Customers SET FirstName = 'Juan' WHERE LastName = 'Soto' ",,adCmdText + adExecuteNoRecords

%>
```

Para quitar determinados registros de una tabla de la base de datos, utilice el comando **DELETE** de SQL. La siguiente secuencia de comandos quita todas las filas de la tabla Customers cuyo apellido sea Soto:

```
<%

Set cn = Server.CreateObject("ADODB.Connection")

cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Datos\Empleados.mdb"

cn.Execute "DELETE FROM Customers WHERE LastName = 'Soto'",,adCmdText + adExecuteNoRecords

%>
```

Nota Debe tener mucho cuidado al utilizar el comando **DELETE** de SQL. Un comando **DELETE** que no vaya acompañado de una cláusula **WHERE** eliminará todas las filas de la tabla. Asegúrese de incluir la cláusula **WHERE** de SQL, que especifica las filas exactas que se van a eliminar.

Utilizar el objeto Recordset para manipular los resultados

Para recuperar datos, examinar resultados y modificar su base de datos, ADO proporciona el objeto **Recordset**. El objeto **Recordset** tiene las funciones necesarias para, dependiendo de las restricciones de las consultas, recuperar y presentar un conjunto de filas, o *registros*, de una base de datos. El objeto **Recordset** mantiene la posición de cada registro devuelto por la consulta, lo que permite recorrer los resultados de uno en uno.

Recuperar un conjunto de registros

Las buenas aplicaciones de datos Web emplean el objeto **Connection** para establecer un vínculo, y el objeto **Recordset** para manipular los datos devueltos. Al combinar las funciones especializadas de ambos objetos puede desarrollar aplicaciones de bases de datos que realicen casi cualquier tarea de tratamiento de datos. Por ejemplo, la siguiente secuencia de comandos del servidor utiliza el objeto **Recordset** para ejecutar un comando **SELECT** de SQL. El comando **SELECT** recupera un conjunto específico de información basándose en las restricciones de la consulta. La consulta también contiene una cláusula **WHERE** de SQL, que se utiliza para establecer el criterio de selección de la consulta. En este ejemplo, la cláusula **WHERE** limita la consulta a todos los registros que contengan el apellido *Soto* en la tabla *Customers* de la base de datos.

```
<%

'Establece una conexión con un origen de datos.

strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Datos\Empleados.mdb"

Set cn = Server.CreateObject("ADODB.Connection")

cnn.Open strConnectionString

'Crea una instancia de un objeto Recordset.

Set rsCustomers = Server.CreateObject("ADODB.Recordset")

'Abre un objeto Recordset con el método Open

'y utiliza la conexión establecida por el objeto Connection.

strSQL = "SELECT FirstName, LastName FROM Customers WHERE LastName = 'Soto' "

rstCustomers.Open strSQL, cnn

'Recorre el conjunto de los registros y presenta los resultados

'e incrementa la posición del registro con el método MoveNext.

Set objFirstName = rstCustomers("Nombre")

Set objLastName = rstCustomers("Apellido")
```

```
Do Until rstCustomers.EOF

Response.Write objFirstName & " " & objLastName & "<BR>"

rstCustomers.MoveNext

Loop

%>
```

Observe que en el ejemplo anterior, el objeto **Connection** estableció la conexión con la base de datos y que el objeto **Recordset** utilizó la misma conexión para recuperar resultados de la base de datos. Este método es útil cuando tenga que configurar con precisión la forma en que se establece el vínculo con la base de datos. Por ejemplo, si necesitara especificar el tiempo de espera antes de anular un intento de conexión, tendría que utilizar el objeto **Connection** para establecer dicha propiedad. Sin embargo, si sólo desea establecer una conexión con las propiedades de conexión predeterminadas de ADO, podría utilizar el método **Open** del objeto **Recordset** para establecer el vínculo:

```
<%

strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Datos\Empleados.mdb"

strSQL = "SELECT FirstName, LastName FROM Customers WHERE LastName = 'Soto' "

Set rsCustomers = Server.CreateObject("ADODB.Recordset")

'Abre una conexión con el método Open

'y utiliza la conexión establecida por el objeto Connection.

rstCustomers.Open strSQL, strConnectionString

'Recorre el conjunto de registros, presenta los resultados

'e incrementa la posición del registro con el método MoveNext.

Set objFirstName = rstCustomers("Nombre")

Set objLastName = rstCustomers("Apellido")

Do Until rstCustomers.EOF

Response.Write objFirstName & " " & objLastName & "<BR>"

rstCustomers.MoveNext

Loop

%>
```

Cuando establece una conexión con el método **Open** del objeto **Recordset**, está utilizando implícitamente el objeto **Connection** para proteger el vínculo. Para obtener más información, consulte la documentación acerca de Microsoft ActiveX Data Objects (ADO), disponible en el sitio Web de Microsoft Universal Data Access en la dirección http://www.microsoft.com/data/.

Nota Para mejorar de forma significativa el rendimiento de las aplicaciones ASP de base de datos, piense en la posibilidad de cambiar el estado del conjunto de registros a **Application**. Para obtener más información, consulte Guardar datos en la memoria caché.

A menudo resulta útil contar el número de registros que se devuelven en un conjunto de registros. El método **Open** del objeto **Recordset** permite especificar un parámetro opcional, *cursor*, que determina cómo recupera y recorre el conjunto de registros el proveedor subyacente. Al agregar el parámetro de cursor **adOpenKeyset** a la instrucción que se utiliza para ejecutar la consulta, permite que la aplicación cliente recorra todo el conjunto de registros. Como resultado, la

aplicación puede utilizar la propiedad **RecordCount** para calcular con precisión el número de registros del conjunto. Vea el siguiente ejemplo:

```
<%

Set rsCustomers = Server.CreateObject("ADODB.Recordset")

rs.Open "SELECT * FROM NewOrders", "Provider=Microsoft.Jet.OLEDB.3.51;Data Source='C:\PedidosClientes\Pedidos.mdb'", adOpenKeyset, adLockOptimistic, adCmdText

'Utiliza la propiedad RecordCount del objeto Recordset para realizar la cuenta.

If rs.RecordCount >= 5 then

Response.Write "Recibimos estos " & rs.RecordCount & " nuevos pedidos<BR>"

Do Until rs.EOF

Response.Write rs("NombreCliente") & " " & rs("ApellidoCliente") & "<BR>"

Response.Write rs("NumeroCuenta") & "<BR>"

Response.Write rs("Cantidad") & "<BR>"

Response.Write rs("FechaEntrega") & "<BR><BR>"

rs.MoveNext

Loop

Else

Response.Write "Hay menos de " & rs.RecordCount & " nuevos pedidos."

End If

rs.Close

%>
```

Combinar formularios HTML y el acceso a bases de datos

Las páginas Web que contienen formularios HTML pueden permitir que los usuarios consulten de forma remota una base de datos y recuperen información concreta. Con ADO puede crear secuencias de comandos sorprendentemente sencillas que recopilen información del formulario del usuario, creen una consulta personalizada para la base de datos y devuelvan información al usuario. Mediante el objeto **Request** de ASP puede recuperar la información escrita en los formularios HTML e incorporar dicha información a sus instrucciones SQL. Por ejemplo, el siguiente bloque de secuencia de comandos inserta en una tabla la información suministrada por un formulario HTML. La secuencia de comandos recopila información del usuario con la colección **Form** del objeto **Request**.

```
<%

'Abre una conexión mediante el objeto Connection. El objeto Command

'no tiene un método Open para establecer la conexión.

strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\Catálogo\Semillas.mdb"

Set cn = Server.CreateObject("ADODB.Connection")

cnn.Open strConnectionString
```

```
'Crea una instancia del objeto Command

'y utiliza la propiedad ActiveConnection para adjuntar

'la conexión al objeto Command.

Set cm= Server.CreateObject("ADODB.Command")

Set cmn.ActiveConnection = cnn

'Define la consulta SQL.

cmn.CommandText = "INSERT INTO MySeedsTable (Type) VALUES (?)"

'Define la información de configuración de los parámetros de la consulta.

cmn.Parameters.Append cmn.CreateParameter("type",adVarChar, ,255)

'Asigna el valor de la entrada y ejecuta la actualización.

cmn("type") = Request.Form("SeedType")

cmn.Execute ,,adCmdText + adExecuteNoRecords

%>
```

Para obtener más información acerca de los formularios y el uso del objeto **Request** de ASP, consulte Procesar los datos introducidos por el usuario.

Transferencia entre archivos .ASP

Para utilizar **Response.Redirect** para redirigir un explorador se necesita un *viaje de ida y vuelta*, lo que significa que el servidor envía al explorador una respuesta HTTP en la que indica la ubicación de la nueva dirección URL. El explorador abandona automáticamente la cola de la petición del servidor y envía una nueva petición HTTP para la dirección URL. Después, el servidor agrega esta petición a la cola de peticiones, junto con las peticiones que mientras tanto llegan desde otros clientes. En un sitio Web con mucha carga, este sistema puede desperdiciar ancho de banda y reducir el rendimiento del servidor, especialmente si se redirige el explorador a un archivo que se encuentre en el mismo servidor.

Puede utilizar el método **Server.Transfer** para realizar la transferencia de un archivo .asp a otro que se encuentre en el mismo servidor, en lugar del método **Response.Redirect**. Con **Server.Transfer** puede transferir directamente las peticiones de archivos .asp sin necesidad de abandonar la cola de peticiones, lo que elimina viajes de ida y vuelta.

Por ejemplo, la secuencia de comandos siguiente muestra cómo podría utilizar **Server.Transfer** para saltar entre las páginas de una aplicación según la información de estado:

```
<%

If Session("blnSaleCompleted") Then

Server.Transfer("/Pedido/Gracias.asp")

Else

Server.Transfer("/Pedido/MasInfo.asp")

End if

%>
```

Server.Transfer envía peticiones desde un archivo .asp en ejecución a otro archivo. Durante la transferencia, el archivo .asp que se pidió inicialmente finaliza inmediatamente la ejecución sin vaciar el búfer de salir (para obtener más información, consulte la sección Almacenamiento en búfer del contenido). La petición de información se pone a

disposición del archivo de destino cuando éste comienza la ejecución. Durante la ejecución, el archivo tiene acceso al mismo conjunto de objetos intrínsecos (**Request**, **Response**, **Server**, **Session** y **Application**) que el archivo que se pidió inicialmente.

También se puede utilizar **Server.Transfer** para realizar una transferencia entre archivos .asp que se encuentren en diferentes aplicaciones. Sin embargo, al hacerlo el archivo se comportará como si formara parte de la aplicación que inició la transferencia (es decir, el archivo sólo tendrá acceso a las variables con alcance en la aplicación inicial, no en la aplicación en la que realmente reside el archivo). Por ejemplo, si realiza una transferencia desde un archivo que se encuentra en la aplicación Ventas a otro que se encuentra en Personal, la aplicación Ventas tomará prestado el archivo de la aplicación Personal y lo ejecutará como si fuera parte de ella.

ASP proporciona también el comando **Server.Execute** que permite transferir un archivo, ejecutar el contenido y volver al archivo que inició la transferencia. Si tiene conocimientos de VBScript, le ayudará pensar en **Server.Execute** como análogo a una llamada a un procedimiento, con la diferencia de que en lugar de ejecutar un procedimiento se ejecuta un archivo .asp completo.

Por ejemplo, la secuencia de comandos siguiente muestra cómo podría utilizar **Server.Execute** para incluir archivos .asp de forma dinámica:

```
<%

.

.

If blnUseDHTML Then

Server.Execute("DHTML.asp")

Else

Server.Execute("HTML.asp")

End If

.

.

%>
```

Mientras el archivo de destino pertenezca a una aplicación del mismo servidor, la aplicación inicial se transferirá a este archivo, ejecutará su contenido y continuará con la ejecución del archivo que inició la transferencia. Igual que sucede con **Server.Transfer**, al ejecutar un archivo .asp éste se comporta como si formara parte de la aplicación inicial. Sin embargo, Server.Execute, *no* funciona cuando los servidores son distintos. Para obtener más información, consulte **Server.Execute**.

Enviar contenido al explorador

A medida que se procesa una secuencia de comandos ASP, el texto y los gráficos que no se encuentren entre delimitadores ASP o etiquetas <SCRIPT> se devuelve directamente al explorador. También puede enviar explícitamente contenido al explorador mediante el objeto **Response**.

Enviar contenido

Para enviar contenido al explorador desde delimitadores ASP o desde un procedimiento, utilice el método **Write** del objeto **Response**. Por ejemplo, la instrucción siguiente envía un saludo diferente al usuario dependiendo de si el usuario ha visitado la página con anterioridad o no:

```
<%

If blnFirstTime Then
```

```
Response.Write "<H3 ALIGN=CENTER>Reciba nuestra bienvenida a la página de introducción</H3>"

Else

Response.Write "<H3 ALIGN=CENTER>Gracias por volver a la página de introducción</H3>"

End If

%>
```

Fuera de un procedimiento, no tiene que utilizar **Response.Write** para devolver contenido al usuario. El contenido que no se encuentra dentro de delimitadores de secuencia de comandos se envía directamente al explorador, que da formato y presenta este contenido. Por ejemplo, la secuencia de comandos siguiente produce el mismo resultado que la secuencia de comandos anterior:

```
<H3 ALIGN=CENTER>

<% If blnFirstTime Then %>

Reciba nuestra bienvenida a la página de introducción.

<% Else %>

Gracias por volver a la página de introducción.

<%End If %>

</H3>
```

Intercale secuencias de comandos y código HTML cuando tenga que devolver el resultado una vez o cuando sea más cómodo agregar instrucciones a texto HTML existente. Utilice **Response.Write** cuando no desee dividir una instrucción con delimitadores o cuando desee generar la cadena de texto que vaya a devolver al explorador. Por ejemplo, podría generar una cadena de texto que creara una fila de una tabla con los valores enviados por un formulario HTML:

```
Response.Write "<TR><TD>" & Request.Form("Nombre") _

& "</TD><TD>" & Request.Form("Apellido") & "</TD></TR>"
```

Request.Form devuelve los valores enviados por un formulario HTML (consulte Procesar los datos introducidos por el usuario).

Nota El carácter & es el carácter de continuación de cadenas de VBScript. El carácter de subrayado (_) es el carácter de continuación de línea de VBScript.

Secuencias de comandos sencillas

La siguiente secuencia de comandos ilustra las técnicas básicas que se utilizan en las secuencias de comandos ASP. Si no tiene experiencia en el desarrollo de aplicaciones o nunca ha escrito secuencias de comandos, éste es un buen lugar para empezar.

Elija un ejemplo en la siguiente lista:

- Variables: Muestra cómo crear y manipular variables en una secuencia de comandos ASP.
- Bucles: Proporciona un ejemplo de las tres construcciones más comunes para crear bucles, **For ... Next**, **Do ... Loop** y **While ... Wend**.
- Operadores condicionales: Ilustra el uso de los operadores condicionales, como **If ... Then**, en las secuencias de comandos ASP.
- Matrices: Muestra cómo crear, administrar y tener acceso a matrices.
- Archivos de inclusión del servidor: Muestra el uso de los archivos de inclusión del servidor.
- Funciones y procedimientos: Muestra cómo crear y utilizar funciones y procedimientos en una secuencia de comandos ASP.

Variables

Introducción

Todas las aplicaciones escritas a lo largo de la historia, independientemente del lenguaje de programación empleado, han utilizado variables de algún tipo y las secuencias de comandos ASP no son una excepción. Tanto VBScript como JScript permiten crear y administrar variables de forma fácil y sencilla.

Cada lenguaje realiza de forma diferente la declaración de variables. JScript y VBScript son bastante flexibles en lo que respecta a las variables y su declaración. En VBScript, cualquier variable se considera automáticamente de tipo **Variant** si se declara inicialmente con la instrucción **Dim**. A cada variable se le asigna un subtipo, como **Numeric** y **Array**. JScript es parecido; la variable se declara inicialmente con la instrucción **var**. En general, ambos lenguajes tienden a realizar automáticamente gran parte de la administración de tipos de datos, incluida la conversión de tipos. De hecho, ni siquiera es necesario emplear las instrucciones **Dim** o **var** para utilizar una nueva variable; en sus respectivos lenguajes son opcionales.

Paseo por el código

Este ejemplo declara varios tipos diferentes de variables, realiza algunas operaciones sencillas en ellas y las muestra al explorador cliente con los delimitadores especiales de secuencias de comandos <% = ...%>. Se asigna un entero a la variable *intVariable*, se suma a sí mismo y se envía el resultado al explorador cliente. A la variable *StrVariable* se le asigna el nombre, se agrega a Soto y se envía al explorador cliente. Del mismo modo se declaran o crean, inicializan, manipulan y muestran los valores booleanos y las fechas.

Observaciones

El último paso de la demostración de la variable de fecha es especialmente interesante. En VBScript, primero se asigna la variable a una cadena de fecha literal y después se muestra. Después se restablece y se asigna el valor devuelto por la función **Now** de VBScript, que devuelve la hora actual del sistema. El ejemplo de JScript utiliza la función **Date** de JScript para establecer el literal inicial, para lo que pasa parámetros a la función, y establecer después la fecha actual del sistema en la variable, sin pasar parámetros a la función.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<HTML>

<HEAD>

<TITLE>Variable Sample</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>Variable Sample</B></FONT><BR>

<HR>

<H3>Integer Manipulation</H3>

<%

'Declare variable.
```

```
Dim intVar

'Assign the variable an integer value.

intVar = 5

%>

<P><%= intVar %> + <%= intVar %> = <%= intVar + intVar %></P>

<HR>

<H3>String Manipulation</H3>

<%

'Declare variable.

Dim strVar

'Assign the variable a string value.

strVar = "Jemearl"

%>

<P>This example was done by <%= strVar + " Smith" %></P>

<HR>

<H3>Boolean Manipulation</H3>

<%

'Declare variable.

Dim blnVar

'Assign the variable a Boolean value.

blnVar = true

'Output message based on value.

If (blnVar) Then

Response.Write "<P>The Boolean value is True.</P>"

Else

Response.Write "<P>The Boolean value is False.</P>"

End If

%>

<HR>

<H3>Date and Time</H3>

<%
```

```
'Declare variable.

Dim dtmVar

'Assign the variable a value.

dtmVar = #08 / 27 / 97 5:11:42pm#

%>

<P>The date and time is <%= dtmVar %>

<%

'Set the variable to the web server date and time.

dtmVar = Now()

%>

<P>The <STRONG>system</strong> date and time is <%= dtmVar %></P>

</BODY>

</HTML>
```

Bucles

Introducción

Los bucles representan uno de los mecanismos más importantes de control de flujo en un lenguaje de programación. Las construcciones en bucle proporcionan la base de cualquier aplicación que deba realizar una tarea de forma repetitiva, como sumar 1 a una variable, leer un archivo de texto o procesar y enviar un mensaje de correo electrónico.

Paseo por el código

VBScript y JScript proporcionan varios mecanismos para realizar bucles. Este ejemplo demuestra las tres instrucciones más comunes para realizar bucles, **For ... Next**, **Do ... Loop** y **While ... Wend**. Estas tres instrucciones son ligeramente diferentes y la situación indicará cuál de las tres es la más indicada. Sin embargo, para este ejemplo, cada tipo de instrucción de bucle se utiliza para realizar la misma tarea: imprimir un saludo cinco veces, cada una de ellas con una fuente mayor. En cada instrucción de bucle se inicializa la variable *i* y se define la condición de prueba, de forma que *i* nunca sea mayor que 5. La variable se incrementa en 1 unidad en cada iteración del bucle.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<HTML>

<HEAD>

<TITLE>Looping</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>Looping with ASP</B></FONT><BR>
```

```
<HR SIZE="1" COLOR="#000000">

<!-- Looping with a For loop. -->

<%

Dim intCounter

For intCounter = 1 to 5 %>

                        <FONT SIZE=<% = intCounter %>>

Hello World with a For Loop!<BR>

</FONT>

                        <% next %>

<HR>

<!-- Looping with a While...Wend loop. -->

<%

intCounter = 1

While(intCounter < 6) %>

<FONT SIZE=<% = intCounter %>>

Hello World with a While Loop!<BR>

</FONT>

<% intCounter = intCounter + 1 %>

<% wend %>

 <HR>

<!-- Looping with a Do...While loop. -->

<%

intCounter = 1

Do While(intCounter < 6) %>

<FONT SIZE=<% =intCounter %>>

Hello World with a Do...While Loop!<BR>

</FONT>

<% intCounter = intCounter+1 %>

<% loop %>

</BODY>

</HTML>
```

Operadores condicionales

Introducción

Los operadores condicionales, junto con las variables y las construcciones de bucle, forman los pilares básicos de los lenguajes de programación y, por tanto, de las aplicaciones. Las aplicaciones basadas en Web que se implementan mediante secuencias de comandos ASP pueden aprovechar el control de flujo que proporcionan los operadores condicionales, así como la interactividad y la sofisticación de HTML.

Paseo por el código

Este ejemplo demuestra las instrucciones **If ... Then** o **if ... else** en VBScript y JScript, así como las instrucciones **Select ... Case** y **switch ... case**, más complejas. La demostración de cada una de estas instrucciones realiza la misma tarea: enviar una página al explorador cliente con la fecha y la hora actuales y un saludo. El texto del saludo será "Buenos días" o "Buenas tardes", dependiendo de si en el reloj del sistema aparece a.m. o p.m.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<HTML>

<HEAD>

<TITLE>Conditional Operator Sample</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>Conditional Operator Sample</B></FONT><BR>

<HR SIZE="1" COLOR="#000000">

<!-- If...Then example -->

<%

Dim varDate

varDate = Date()

%>

<P>The date is: <%= varDate %></P>

<%

'Select Case statement to display a message based on the day of the month.

Select Case Day(varDate)

Case 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Response.Write("<P>It's the beginning of the month.</P>")

Case 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
```

```
Response.Write("<P>It's the middle of the month.</P>")

Case Else

Response.Write("<P>It's the end of the month.</P>")

End Select

%>

<P>The time is: <%= Time %></P>

<%

'Check for AM/PM, and output appropriate message.

If (Right(Time,2)="AM") Then

Response.Write("<P>Good Morning</P>")

Else

Response.Write("<P>Good Evening</P>") End If

%>

</BODY>

</HTML>
```

Funciones y procedimientos

Introducción

Las funciones y los procedimientos proporcionan un método para evitar tener que escribir varias veces un mismo bloque de código cada vez que desee realizar una tarea determinada. Tanto VBScript como JScript permiten llamar a una función o a un procedimiento desde cualquier lugar de una secuencia de comandos. Este ejemplo demuestra cómo puede crear y utilizar estas herramientas en secuencias de comandos ASP.

Si no tiene ninguna función en la página ASP, el motor ASP se limita a procesar el archivo completo, de principio a fin, cada vez que lo pide un explorador cliente. Sin embargo, las funciones y los procedimientos se ejecutan sólo cuando se les llama, no con el resto del código.

En VBScript o JScript las funciones y los procedimientos se indican mediante la instrucción **Function**. Además, VBScript establece una diferencia entre una función que devuelve un valor y otra que no lo hace; la primera de ellas se indica con la instrucción **Sub**, que la identifica como una subrutina.

Paseo por el código

Este ejemplo define una función, **PrintOutMsg**, que toma como parámetros un mensaje y un número que especifica cuántas veces se escribirá el mensaje en el explorador cliente mediante el método **Response.Write**. Para este ejemplo, la función se limita a devolver al explorador cliente el número de veces que se imprimió el mensaje.

Observaciones

Es importante tener en cuenta el atributo RUNAT de la etiqueta <SCRIPT>. SI no se incluye, ASP asumirá que se trata de una secuencia de comandos del cliente y devolverá el código al explorador para que lo procese. Esto haría que ASP no reconociera la llamada a la función **PrintOutMsg**, devolviera un error y anulara la ejecución.

```
<%@ LANGUAGE = VBScript %>
```

```
<% Option Explicit %>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>

'Define Server Side Script Function.

Function PrintOutMsg(strMsg, intCount)

Dim i

'Output Message count times.

For i = 1 to intCount

Response.Write(strMsg & "<BR>")

Next

'Return number of iterations.

PrintOutMsg = intCount

End Function

</SCRIPT>

<HTML>

<HEAD>

<TITLE>Functions</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>Server Side Functions</B></FONT><BR>


<P>

The function "PrintOutMsg" prints out a specific message a set number of times.<P>

<%

'Store number of times function printed message.

Dim intTimes

'Call function.

intTimes = PrintOutMsg("This is a function test!", 4)

'Output the function return value.

Response.Write("<p>The function printed out the message " & intTimes & " times.")
```

%>

</BODY>

</HTML>

Datos introducidos por el usuario en un formulario con POST

Introducción

Probablemente, la forma más básica de interactividad Web es el formulario HTML. Es importante tener en cuenta que ASP no sustituye a los formularios, sino que los mejora y hace que sea más fácil implementarlos y administrarlos.

La etiqueta HTML <FORM> especifica qué método utilizará el formulario para comunicar la información a la secuencia de comandos que la procesará. El atributo del método **POST** indica que la información del formulario pasará a través de una conexión HTTP independiente a la secuencia de comandos o al programa que la procesará. La secuencia de comandos o el programa pueden analizar la información y realizar cualquier tarea que sea necesaria, y devolver el resultado al explorador cliente.

Paseo por el código

Este ejemplo muestra cómo implementar un formulario sencillo mediante el atributo del método **POST** de HTTP, así como una de las principales ventajas de la creación de formularios mediante ASP: la posibilidad de combinar el formulario y el código de proceso real en un mismo archivo. Este ejemplo crea un pequeño formulario con dos cuadros de entrada de texto, uno para el nombre del usuario (*fname*) y otro para el apellido (*lname*). Para obtener el valor de las variables *fname* y *lname* en la petición se tiene acceso a la colección **Request.Forms** y después se muestran los resultados en la parte inferior de la página.

La primera vez que se ejecuta la secuencia de comandos no aparece ningún texto debajo de la línea horizontal. Esto se debe a que no había información disponible para pasarla a la secuencia de comandos cuando se inició y ASP pasa por alto las búsquedas de **Request.Forms** si no existe información. Sin embargo, si hace clic en el botón Enviar, se cargará de nuevo la página y la secuencia de comandos ya dispondrá de la información que escribió en los cuadros de texto.

<%@ Language = VBScript %>

<% Option Explicit %>

<HTML>

<HEAD>

<TITLE>Form Posting</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>Form Posting</B></FONT>

<HR>

<P>This page will take the information entered in the form fields, and use the POST method to send the data to an ASP page.

<FORM NAME=Form1 METHOD=Post ACTION="Form_VBScript.asp">

First Name: <INPUT TYPE=Text NAME=fname><P>

Last Name: <INPUT TYPE=Text NAME=lname><P>

<INPUT TYPE=Submit VALUE="Submit">

</FORM>

<HR>

<% Response.Write Request.form("fname")%>

<% Response.Write Request.form("lname")%>

</BODY>

</HTML>

Llenar los campos

Introducción

Puede utilizar formularios para recopilar datos de los usuarios, pero también puede utilizarlos para mostrar información. Por ejemplo, si un explorador cliente tiene acceso al motor de búsqueda de su directorio de teléfonos, querrá mostrar los resultados de la búsqueda. La secuencia de comandos de búsqueda (que también puede implementar mediante ASP) acepta los datos introducidos por el usuario, tiene acceso a la base de datos y envía el resultado al formulario de presentación en una cadena de consulta. Este ejemplo es una demostración sencilla de cuál sería la apariencia del formulario.

Paseo por el código

Para este ejemplo, los datos se incluyen en el código de la secuencia de comandos pero, obviamente, la información podría provenir de un formulario interactivo, de una base de datos o de un archivo de texto. Al iniciarse el ejemplo se inicializan las variables. Después, crea un formulario con las etiquetas HTML <FORM> y define cuatro cuadros de texto. Por último, con los delimitadores de secuencias de comandos del servidor <%= ... %>, la secuencia de comandos llena los cuadros de texto con los valores establecidos en la inicialización.

<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<%

'Create and set variables that will be used in populating 'the form. In a typical application, these values would come from a database or text file.

Dim strFirstName

Dim strLastName

Dim strAddress1

Dim strAddress2

Dim blnInfo

```
strFirstName = "John"

strLastName = "Doe"

strAddress1 = "1 Main Street"

strAddress2 = "Nowhere ZA, 12345"

%>

<HTML>

<HEAD>

<TITLE>PopulateForm Sample</TITLE>

</HEAD>

<BODY BGCOLOR="White" TOPMARGIN="10" LEFTMARGIN="10">

<!-- Display header. -->

<FONT SIZE="4" FACE="ARIAL, HELVETICA">

<B>PopulateForm Sample</B></FONT><BR>

<HR SIZE="1" COLOR="#000000">

<FORM ACTION="">

<!-- Use ASP variables to fill out the form. -->

<P>First Name: <INPUT TYPE="TEXT" NAME="FNAME" VALUE="<%= strFirstName %>"></P>

<P>Last Name: <INPUT TYPE="TEXT" NAME="LNAME" VALUE="<%= strLastName %>"></P>

<P>Street: <INPUT TYPE="TEXT" NAME="STREET" VALUE="<%= strAddress1%>"></P>

<P>City State, Zip: <INPUT TYPE="TEXT" NAME="FNAME" VALUE="<%= strAddress2 %>"></P>

</BODY>

</HTML>
```

Conectividad con bases de datos

Si va a crear una aplicación basada en Web, lo más probable es que tenga formularios. Posiblemente, también tendrá que conectarla de alguna forma con una base de datos. ActiveX® Data Objects (ADO) proporciona un conjunto de eficaces herramientas que le permitirán tener acceso a orígenes de datos y manipularlos.

Los ejemplos de esta sección ilustran las técnicas necesarias para utilizar ADO de forma efectiva y cómo puede utilizar mejor estas funciones en una aplicación basada en Web.

- Consulta sencilla: Muestra cómo utilizar ADO y ASP para realizar consultas sencillas en una base de datos.
- Limitar los resultados de una consulta: Muestra cómo es posible utilizar ASP y ADO en las secuencias de comandos para limitar el número de filas devueltas en un conjunto de registros.
- Consulta desplazable: Muestra cómo realizar con ADO una consulta desplazable multidireccional.

- Agregar o eliminar registros: Presenta las técnicas necesarias para agregar y eliminar registros de un origen de datos mediante ASP y ADO.
- Actualizar registros: Muestra cómo utilizar ADO en una aplicación para actualizar los registros existentes.
- Ejecutar procedimientos almacenados: Muestra cómo utilizar ADO con las secuencias de comandos ASP para ejecutar procedimientos almacenados de la base de datos.

Para obtener más información acerca de ADO y de las herramientas de acceso a datos de Microsoft en general, consulte la documentación de **Microsoft Data Access**.

Consulta sencilla

Introducción

Aunque una base de datos puede ser un sistema muy complicado y las herramientas de acceso a datos deben ser eficaces y sensibles, es igualmente importante que las tareas sencillas de acceso a bases de datos sean fáciles de realizar. Este ejemplo demuestra cómo ADO proporciona un método sencillo para realizar este tipo de tareas.

Paseo por el código

El objetivo de esta aplicación de ejemplo es obtener un pequeño conjunto de registros de una base de datos de Microsoft® Access e imprimir el resultado. El primer paso consiste en crear una instancia del objeto **Connection** mediante el método **Server.CreateObject**. El ejemplo utiliza la instancia del objeto **Connection** para abrir el proveedor de datos OLE DB y, después, para ejecutar un comando **SELECT** de SQL y así obtener todos los registros de la tabla Autores. Para terminar, la secuencia de comandos recorre la colección del conjunto de registros obtenido y muestra los resultados. Después se cierran el conjunto de registros y la conexión con el origen de datos OLE DB.

Importante OLE DB debe estar correctamente configurado en el servidor para que este ejemplo funcione correctamente.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<HTML>

<HEAD>

<TITLE>Simple ADO Query</TITLE>

</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">

<b>Simple ADO Query with ASP</b></font><br>

<hr size="1" color="#000000">

Contacts within the Authors Database:<br><br>

<%

Dim oConn

Dim oRs
```

```
Dim filePath

Dim Index

' Map authors database to physical path

filePath = Server.MapPath("authors.mdb")

' Create ADO Connection Component to connect

' with sample database

Set oConn = Server.CreateObject("ADODB.Connection")

oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & filePath

' Execute a SQL query and store the results

' within recordset

Set oRs = oConn.Execute("SELECT * From Authors")

%>


  <TABLE border = 1>

<%

Do while (Not oRs.eof) %>

<tr>

<% For Index=0 to (oRs.fields.count-1) %>

<TD VAlign=top><% = oRs(Index)%></TD>

<% Next %>

</tr>

<% oRs.MoveNext

Loop

%>

</TABLE>

<%

oRs.close

oConn.close

%>

</BODY>

</HTML>
```

Agregar o eliminar registros

Introducción

Este ejemplo muestra las técnicas que necesita conocer para utilizar ASP y ADO con el fin de agregar y eliminar registros de una base de datos.

Paseo por el código

Primero se utiliza **CreateObject** para crear una instancia del objeto **Connection**, que a su vez se utiliza para abrir una conexión con el proveedor de datos OLE DB. Se utiliza de nuevo **CreateObject** para crear un objeto **Recordset** vacío. Se configura la propiedad **ActiveConnection** para hacer referencia al nuevo objeto **Connection**.

Aunque el objeto **Recordset** de ADO proporciona el método **AddNew** para agregar nuevos registros a una base de datos, puede que la escalabilidad mejore si envía comandos INSERT de SQL directamente al motor de base de datos. Este ejemplo utiliza el comando **Recordset.Execute**, junto con la cadena de comandos apropiada de SQL, para insertar información acerca de un nuevo autor.

En este momento se crea otra instancia del objeto **Recordset** y se abre con otro comando SQL. Se selecciona el registro recién agregado y se elimina, para lo cual se pasa el comando DELETE de SQL directamente al motor de base de datos. Por último, finaliza la secuencia de comandos.

Importante OLE DB debe estar correctamente configurado en el servidor para que este ejemplo funcione correctamente.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<% Response.Expires= -1 %>

<!--METADATA TYPE="typelib"

uuid="00000205-0000-0010-8000-00AA006D2EA4" -->

<HTML>

<HEAD>

<TITLE>Add/Delete Database Sample</TITLE>

</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">

<b>Add/Delete Database Sample</b></font><br>

<hr size="1" color="#000000">

<%
```

```
Dim oConn

Dim oRs

Dim filePath

' Map authors database to physical path

filePath = Server.MapPath("authors.mdb")

' Create ADO Connection Component to connect with sample database

Set oConn = Server.CreateObject("ADODB.Connection")

oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & filePath

' To add, delete and update recordset, it is recommended to use

' direct SQL statement instead of ADO methods.

oConn.Execute "insert into authors (author, YearBorn) values ('Paul Enfield', 1967)"

' Output Result

Set oRs = oConn.Execute (" select * from authors where Author= 'Paul Enfield' and YearBorn =1967 " )


Response.Write("<p>Inserted Author: " & oRs("Author") & "," & oRs("YearBorn"))

' Close Recordset

oRs.Close

Set oRs= Nothing

' Delete the inserted record

oConn.Execute "Delete From authors where author='Paul Enfield' and YearBorn = 1967 "

' Output Status Result

Response.Write("<p>Deleted Author: Paul Enfield, 1967")

%>

</BODY>

</HTML>
```

Actualizar registros

Introducción

Este ejemplo muestra cómo utilizar ADO en una aplicación para actualizar los registros existentes.

Paseo por el código

Primero se utiliza **CreateObject** para crear una instancia del objeto **Connection**, que a su vez se utiliza para abrir una conexión con el proveedor de datos OLE DB. Se utiliza de nuevo **CreateObject** para crear un objeto **Recordset** vacío. Se configura la propiedad **ActiveConnection** para hacer referencia al nuevo objeto **Connection**.

Después se configura el nuevo conjunto de registros. El método **Recordset.Execute** utiliza como parámetro una cadena de comandos SQL. Este ejemplo utiliza una cadena de comandos UPDATE de SQL para realizar la actualización en los registros adecuados de la base de datos.

```
<%@ LANGUAGE = VBScript %>

<% Option Explicit %>

<!--METADATA TYPE="typelib"

uuid="00000205-0000-0010-8000-00AA006D2EA4" -->

<HTML>

<HEAD>

<TITLE>Update Database</TITLE>

</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">

<b>Update Database</b></font><br>

<hr size="1" color="#000000">

<%

Dim oConn ' object for ADODB.Connection obj

Dim oRs ' object for output recordset object

Dim filePath ' Directory of authors.mdb file

' Map authors database to physical path

filePath = Server.MapPath("authors.mdb")

' Create ADO Connection Component to connect with sample database

Set oConn = Server.CreateObject("ADODB.Connection")

oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & filePath

' To add, delete and update recordset, it is recommended to use

' direct SQL statement instead of ADO methods.

oConn.Execute "Update Authors Set Author ='Scott Clinton'" _

& "where Author='Scott Guthrie' "

Set oRs = oConn.Execute ( "select * from Authors where author" _
```

```
& "= 'Scott Clinton'" )

%>

Changed Author: <%= oRs("Author") %>, <%= oRs("Yearborn") %> <P>

<%

oConn.Execute "Update Authors Set Author ='Scott Guthrie'" _

& "where Author='Scott Clinton' "

Set oRs = oConn.Execute ( "select * from Authors where author" _

& "= 'Scott Guthrie'" )

%>

Changed Author: <%= oRs("Author") %>, <%= oRs("Yearborn") %>

</BODY>

</HTML>
```

MapPath

El método **MapPath** asigna al directorio físico correspondiente del servidor la ruta virtual o relativa especificada.

Sintaxis

Server.MapPath(*Ruta*)

Parámetros

Ruta

Especifica la ruta virtual o relativa que se asignará a un directorio físico. Si *Ruta* comienza con una barra diagonal (/) o con una barra diagonal inversa (\), el método **MapPath** devuelve la ruta como si *Ruta* fuera una ruta virtual completa. Si *Ruta* no comienza con una barra diagonal, el método **MapPath** devuelve una ruta relativa al directorio del archivo .asp que se procesa.

Observaciones

El método **MapPath** no comprueba si la ruta que devuelve existe en el servidor o si es válida.

Como el método **MapPath** asigna la ruta independientemente de si los directorios especificados existen o no, puede utilizarlo para asignar una ruta a una estructura física de directorios y, después, pasarla a un componente que cree en el servidor el directorio o el archivo especificado.

Puede utilizar la sintaxis de ruta relativa para el parámetro *Ruta* si el valor de la propiedad AspEnableParentPaths es TRUE (que es el valor predeterminado). Si le preocupa permitir que las secuencias de comandos tengan acceso a la estructura física de directorios, puede deshabilitar esta característica si asigna el valor FALSE a la propiedad **AspEnableParentPaths**. Para ello puede utilizar el complemento Servicios de Internet Information Server o una secuencia de comandos.

Ejemplo

Para los siguientes ejemplos, el archivo data.txt se encuentra en el directorio C:\Inetpub\Wwwroot\Script, al igual que el archivo test.asp, que contiene las siguientes secuencias de comandos. El directorio C:\Inetpub\Wwwroot se establece como directorio particular del servidor.

El siguiente ejemplo utiliza la variable de servidor PATH_INFO para asignar la ruta física al archivo actual. La siguiente secuencia de comandos

<%= server.mappath(Request.ServerVariables("PATH_INFO"))%>

produce el resultado

c:\inetpub\wwwroot\script\test.asp

Como los parámetros de la ruta de los siguientes ejemplos no empiezan con un carácter de barra diagonal, se asignan de forma relativa al directorio actual, en este caso C:\Inetpub\Wwwroot\Script. Las siguientes secuencias de comandos

<%= server.mappath("data.txt")%>

<%= server.mappath("script/data.txt")%>

producen el siguiente resultado

c:\inetpub\wwwroot\script\data.txt

c:\inetpub\wwwroot\script\script\data.txt

Los dos ejemplos siguientes utilizan el carácter de barra diagonal para especificar que la ruta que se obtiene debe buscarse como ruta virtual completa del servidor. Las siguientes secuencias de comandos

<%= server.mappath("/script/data.txt")%>

<%= server.mappath("\script")%>

producen el siguiente resultado

c:\inetpub\wwwroot\script\data.txt

c:\inetpub\wwwroot\script

Los siguientes ejemplos muestran cómo puede utilizar una barra diagonal (/) o una barra diagonal inversa (\) para obtener la ruta física al directorio particular. Las siguientes secuencias de comandos

<%= server.mappath("/")%>

<%= server.mappath("\")%>

producen el siguiente resultado

c:\inetpub\wwwroot

c:\inetpub\wwwroot

El siguiente documento es un manual de ASP, trata los principales comandos de ASP, desde lo mas sencillo o lo intermedio, comandos y utilidades basicas para el desarrollo de paginas Web interactivas, haciendo uso del Microsoft Internet Information Service.
Dicho manual fue tomado de los principales puntos de la ayuda de Windows 2000 Professional. P.P/p.p.

Trabajo enviado por:
ppadron@telcel.net.ve

Volver al inicio | **Volver arriba**

Términos y Condiciones - Haga Publicidad en Monografias.com - Contáctenos
© 1997 Lucas Morea / Sinexi S.A.