Início à programação em ASP

Especificação da linguagem utilizada: VBScript ou JScript servidor. Primeiro script em ASP utilizando uma função.

Ao longo dos capítulos precedentes ficou claro que o ASP é uma linguagem orientada às aplicações em rede criada por Microsoft que funciona do lado servidor. É na verdade, o servidor quem se ocupa de executa-lo, interpreta-lo e envia-lo ao cliente (navegador) em forma de código HTML.

ASP é principalmente utilizado servindo-se da linguagem Visual Basic Script que não é mais do que uma versão light do Visual Basic. Entretanto, é possível programar páginas ASP em Java Script. O único que há de fazer é especificar na própria página que tipo de linguagem estamos utilizando.

Dado que a linguagem ASP está muito freqüentemente embebida dentro do código HTML, é importante poder marcar ao servidor que partes estão escritas em uma linguagem e quais em outra. É por isso que todas as partes do arquivo que estão escritas em ASP estarão sempre delimitadas pelos símbolos: <% e %>.

Deste modo, quando realizarmos nossos scripts, o primeiro que devemos definir é o tipo de linguagem utilizado, o qual se faz da seguinte forma:

<% @ LANGUAGE="VBSCRIPT" %> No caso de programarmos em Visual Basic Script
<% @ LANGUAGE="JSCRIPT" %> Se nos servimos do Java Script em servidor para programar em ASP

Os scripts que serão apresentados neste manual estarão baseados no VBS, o qual apresenta uma série de prestações que os tornam sem dúvida mais acessível e apto para ASP. Não é a toa que é o próprio Microsoft quem criou ambos.

Com os elementos que apresentamos até agora, já estamos em situação de poder escrever nosso primeiro programa em ASP. Vamos criar um programa que calcule o 20% de impostos que teria que acrescentar a uma série de artigos. Para concretizar o conceito de função, explicado no manual de páginas dinâmicas, vamos definir uma função "imposto" que empregaremos sucessivas vezes. O programa poderia ser algo assim:

```
<% @ LANGUAGE="VBSCRIPT" %>
<HTML>
<HEAD>
<TITLE>Função imposto</TITLE>
</HEAD>
<BODY>
<%Function imposto(preço_artigo)
preço_final=preço_artigo+preço_artigo*20/100
Response.Write preço_final
End Function%>
Um livro de $35,00 ficará em um preço de <% imposto(35) %>
<br>
Um sapato de $60,00 terá um preço final de <% imposto(60) %>
```

**
**
Um CD de música de $20,00 custaria <% imposto(20) %>
</BODY>
</HTML>

Como pode ser visto, o script contém duas partes fundamentais: Uma primeira na qual definimos a função que chamamos imposto que depende unicamente de uma variável (preço_artigo). Imposto permite acrescentar um 20% ao preço do artigo e imprimir o resultado na tela (Response.Write). Na segunda parte nos servimos da função para realizar os cálculos necessários e mostrá-los na tela, acompanhados de texto.

É muito interessante uma vez executado o script, ver o código fonte. Como se pode ver, o código HTML que mostra o browser não coincide com o que nós escrevemos. Algo que não deve nos surpreender, já que, como já explicamos, o servidor se encarrega de processa-lo e torna-lo compreensível ao navegador.

Loops e condições I

Algumas das formas mais correntes de controlar o fluxo dos programas em VBScript: Condição IF

A programação exige em muitas ocasiões a repetição de ações sucessivas ou a escolha de uma determinada seqüência e não de outra dependendo das condições específicas da execução.

Como exemplo, poderíamos fazer alusão a um script que execute uma seqüência diferente em função do dia da semana no qual nos encontramos.

Este tipo de ação pode ser realizado graças a uma paleta de instruções presentes na maioria das linguagens. Neste capítulo descreveremos sumariamente algumas delas propostas pelo VBS e que são de evidente utilidade para o desenvolvimento de páginas ASP.

Para evitar complicar o texto, nos limitaremos a introduzir as mais importantes deixando de lado outras tantas que poderão ser facilmente assimiladas a partir de exemplos práticos.

As condições: IF

Quando quisermos que o programa, chegado a um certo ponto, tome um caminho determinado em determinados casos e outro diferente se as condições de execução diferem, nos servimos do conjunto de instruções If, Then e Else. A estrutura de base deste tipo de instruções é a seguinte:

```
IF condição THEN
  Instrução 1
  Instrução 2
  ...
ELSE
  Instrução A
  Instrução B
  ...
END IF
```

Chegado a este ponto, o programa verificará o cumprimento ou não da condição. Se a condição é certa as instruções 1 e 2 serão executadas. Do contrário (Else), as instruções A e B serão realizadas.

Uma vez finalizada a estrutura, deveremos fechar com um End If.

Esta estrutura de base pode complicar-se um pouco mais, se temos em conta que não necessariamente tudo é branco ou negro e que muitas possibilidades podem se dar. É por isso que outras condições podem se colocar

dentro da condição principal. Falamos, portanto, de condições aninhadas que teriam uma estrutura do seguinte tipo:

```
IF condição THEN
  Instrução 1
  Instrução 2
  ...
ELSE
  IF condição2 THEN
    Instrução A
    Instrução B
    ...
  ELSE
    Instrução X
    ...
  END IF
END IF
```

Deste modo poderíamos introduzir tantas condições quantas quisermos dentro de uma condição principal. Neste tipo de estruturas é importante fechar corretamente cada um dos IF com seus END IF correspondentes. De grande ajuda é a instrução ELSE IF que permite em uma só linha e sem necessidade de acrescentar um END IF introduzir uma condição aninhada.

O uso desta ferramenta será claro com um pouco de prática. Colocamos um exemplo simples de utilização de condições. O seguinte programa permitiria detectar a língua empregada pelo navegador e visualizar uma mensagem em tal língua.

```
<% @ LANGUAGE="VBSCRIPT" %>
<HTML>
<HEAD>
<TITLE>Detector de Língua</TITLE>
</HEAD>
<BODY>
<%
'Antes de nada introduzimos mensagens em forma de variáveis
  espanhol="Hola"
  ingles="Hello"
  portugues="Olá"

'Agora lemos do navegador qual é a sua língua oficial
  idioma=Left(Request.ServerVariables("HTTP_ACCEPT_LANGUAGE"),2)

'Formulamos as possibilidades que podem dar
If idioma="es" Then
  Response.Write espanhol
ElseIf idioma="pt" Then
  Response.Write portugues
Else
  Response.Write ingles
End If %>
</BODY>
</HTML>
```

Para poder ver o funcionamento deste script é necessário mudar o idioma preferido o qual pode ser realizado a partir do menu de opções do navegador.

Como pode ser visto, as variáveis que contém texto são armazenadas entre aspas.

Para ler a língua aceitada pelo navegador o que fazemos é definir uma variável (idioma) que recorre as duas primeiras letras começando da esquerda do idioma aceitado pelo navegador ("HTTP_ACCEPT_LANGUAGE"). Este idioma aceitado pode ser requerido como uma variável do objeto ServerVariables. Por agora deixaremos isto tal como está, e já nos encarregaremos de vê-lo mais detalhadamente em outros capítulos.

A terceira de script se encarrega de ver se o navegador está em português (pt), espanhol (es), ou em qualquer outro idioma que não seja nenhum destes dois, e imprimir cada uma das mensagens que proceda em cada caso.

Outro ponto a comentar é o fato de poder comentar os programas. Como se pode observar, dentro do script introduzimos umas mensagens que nos serve para lê-las mais facilmente. Estas mensagens não exercem nenhuma influência no desenvolvimento do mesmo. Para introduzi-las é necessário escreve-las detrás de uma apóstrofe: '

Os comentários são de grande utilidade quando tratamos com programas muito extensos e complicados.Nestes casos, são de grande ajuda na hora de depurar falhos ou introduzir modificações. É altamente aconselhável acostumar-se a utilizá-los.

Loops e condições II

Algumas das formas mais correntes de controlar o fluxo dos programas em VBScript: Loop FOR.

Os loops FOR

Em muitas ocasiões é necessário executar um conjunto de instruções um número definido de vezes. Isto pode ser realizado a partir da instrução FOR/NEXT.

A estrutura clássica:

```
FOR contador=número inicial to número final STEP incremento
  Instrução 1
  Instrução 2
  ...
NEXT
```

A partir deste tipo de estruturas executamos as instruções contidas entre o FOR e o NEXT um certo número de vezes definido pelo número inicial, final e o incremento. O incremento é de 1 por padrão.

Colocamos um exemplo:

```
<% @ LANGUAGE="VBSCRIPT" %>
<HTML>
<HEAD>
<TITLE>Loop for/next</TITLE>
</HEAD>
<BODY>

<%For i=1 to 5%>
<font size=<%Response.Write i%>>Volta número <%Response.Write i%></font><br>
<%Next
```

```
For i=5 to 1 Step -1%>
<font size=<%Response.Write i%>>Contamos atrás: <%Response.Write i%></font><br>
<%Next%>

</BODY>
</HTML>
```

Este script composto de dois loops conta primeiro de 1 a 5. A variável i toma portanto todos os valores inteiros compreendidos entre estes dois números e pode ser utilizada dentro do loop como o fazemos neste caso para aumentar o tamanho da letra. O segundo loop realiza o processo inverso (o incremento é negativo) produzindo uma diminuição do tamanho da letra.

Se quiser ver o efeito que produz clique aqui

O que pode ser interessante para ver até que ponto o programar páginas dinâmicas pode fazer pouparmos texto em relação a mesma página programada em código HTML, é ver o código fonte da página a partir do navegador.

Loops e condições III

Algumas das formas mais correntes de controlar o fluxo dos programas em VBScript: Loops DO WHILE/LOOP e operadores lógicos.

Os loops DO WHILE/LOOP

Outra forma de realizar este tipo de seqüências loop é a partir da instrução DO WHILE. Neste caso o que especificamos para fixar a extensão do loop não é o número de voltas e sim, o que se cumpre ou não uma condição. A estrutura deste tipo de loops é análoga a dos loops FOR/NEXT:

```
DO WHILE condição
  Instrução 1
   Instrução 2
   ...
LOOP
```

O loop se dará enquanto a condição proposta seguir sendo válida.

Como será visto em exemplos posteriores, este tipo de loops é muito prático para a leitura de bases de dados.

Todo este tipo de controladores de fluxo, (condições e loops) podem ser necessários e otimizados a partir do uso de operadores lógicos. Assim, podemos escolher que sejam duas as condições que dêem para realizar um conjunto de instruções:

IF condição 1 AND condição 2 THEN ...

Também podemos requerer que seja uma das duas:

IF condição 1 OR condição 2 THEN...

Da mesma forma, é possível escolher que a condição de um loop DO seja a inversa à enunciada:

DO WHILE NOT condição

Concluindo, um conjunto de recursos básicos para controlar o desenvolvimento de programas. Sua utilidade será mais que patente e seu uso irá tornando-se intuitivo à medida que nos familiarizarmos com a linguagem.

Os objetos ASP

Introdução ao conceito de objeto e os elementos que o compõem.

O ASP é uma linguagem desenhada para a criação de aplicações na internet. Isto quer dizer que existe toda uma série de tarefas bastante correntes, as quais deve se dar um tratamento fácil e eficaz. Referimo-nos por exemplo ao envio de e-mails, acesso a arquivos, gestão de variáveis do cliente ou servidor como podem ser seu IP ou a língua aceitada...

A linguagem VB propriamente dita não dá uma solução fácil e direta a estas tarefas, e sim invoca aos denominados objetos que não são mais que uns módulos incorporados à linguagem que permitem o desenvolvimento de tarefas específicas. Estes objetos realizam de uma maneira simples toda uma série de ações de uma complexidade relevante. A partir de uma chamada ao objeto, este realizará a tarefa requerida. De certa forma, estes objetos nos poupam ter que fazer compridos programas para operações simples e habituais.

Alguns destes objetos estão incorporados no próprio ASP, outros devem ser incorporados como se se tratasse de componentes acessórios. De fato, não poderíamos executar corretamente um script no qual tivéssemos que chamar a um objeto que não estivesse integrado no servidor. Estes tipos de "plug-in" são geralmente comprados pelo servidor a empresas que os desenvolvem.

Como todo objeto do mundo real, os objetos do mundo informático têm suas propriedades que os definem, realizam um certo número de funções ou métodos e são capazes de responder de uma forma definível antes certos eventos.

Dado o nível desta obra, a descrição da totalidade de objetos com seus métodos e propriedades é certamente fora de lugar. Vamos, portanto, nos contentar com ir descrevendo os mais freqüentemente utilizados e exemplifica-los da maneira mais prática deixando a numeração exaustiva em forma de apêndice.

Objeto Request

OBJECT REQUEST I

Que funções realiza este objeto e como se passam as variáveis pela URL.

Loops e condições são muito úteis para processar os dados dentro de um mesmo script. Entretanto, em um site na internet, as páginas vistas e os scripts utilizados são numerosos. Muitas vezes necessitamos que nossos distintos scripts estejam conectados uns com outros e que se sirvam de variáveis comuns. Por outro lado, o usuário interage por meio de formulários cujos campos hão de ser processados para poder dar uma resposta. Todo este tipo de fatores dinâmicos hão de ser eficazmente regulados por uma linguagem como o ASP.

Como veremos, todo este tipo de aspectos interativos podem ser providenciados a partir do objeto Request.

O objeto Request nos devolve informações do usuário que foram enviadas por meio de formulários, por URL ou a partir de cookies (veremos de que se tratam a seguir). Também nos informa sobre o estado de certas variáveis do sistema, como pode ser a língua utilizada pelo navegador, o número IP do cliente...

Transferir variáveis por URL

Para passar as variáveis de una página a outra, podemos faze-lo introduzindo tal variável no endereço URL da página destino dentro do link hipertexto. A sintaxe seria a seguinte:

>

Para buscar a variável na página destino, devemos fazer por meio do objeto Request com o método Querystring:

Request.querystring("variavel1")
Request.querystring("variavel2")

As duas páginas seriam assim:

```
<HTML>
<HEAD>
<TITLE>Página origem.asp</TITLE>
</HEAD>
<BODY>
<a href="destino.asp?saudacao=olá&texto=Isto é uma variável texto">Passo variáveis saudação e texto à página destino.asp</a>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>destino.asp</TITLE>
</HEAD>
<BODY>
Variável saudação: <%Response.Write Request.Querystring("saludo")%><br>
Variável texto: <%Response.Write Request.Querystring("texto")%><br>
</BODY>
</HTML>
```

OBJECT REQUEST II

Como passar variáveis através dos formulários e obter mais informações práticas sobre o servidor ou o cliente.

Transferir variáveis por formulário

O processo é similar ao explicado para as URLs. Primeiramente, apresentamos uma primeira página com o formulário a preencher e as variáveis são recolhidas em uma segunda página que as processa:

```
<HTML>
<HEAD>
<TITLE>formulario.asp</TITLE>
</HEAD>
<BODY>
<FORM METHOD="POST" ACTION="destino2.asp">
Nome<br>
<INPUT TYPE="TEXT" NAME="nome"><br>
Sobrenome<br>
<INPUT TYPE="TEXT" NAME="sobrenome"><br>
<INPUT TYPE="SUBMIT">
</FORM>
</BODY>
```

```
</HTML>


<HTML>
<HEAD>
<TITLE>destino2.asp</TITLE>
</HEAD>
<BODY>
Variável nome: <%=Request.Form("nome")%><br>
Variável sobrenome: <%=Request.Form("sobrenome")%>
</BODY>
</HTML>
```

Outras utilidades de Request: as ServerVariables

O objeto Request nos dá acesso a outras informações relativas ao cliente e ao servidor, as quais podem ser de uma grande utilidade. Estas informações são armazenadas como variáveis as quais são agrupadas em uma coleção chamada ServerVariables.

Dentro desta coleção temos variáveis tão interessantes como:

HTTP_ACCEPT_LANGUAGE Informa a língua preferida pelo navegador
HTTP_USER_AGENT Indica qual é o navegador utilizado.
PATH_TRANSLATED Devolve o path físico do disco rígido do servidor no qual se encontra nosso script
SERVER_SOFTWARE Diz que tipo de software utiliza o servidor

Para visualizar na tela algumas dessas variáveis, devemos escrever algo como:

Response.write request.servervariables("nome da variável")

Uma forma rápida de visualizar todas estas variáveis é a partir de um script com esta seqüência:

```
<%
For Each elemento in Request.ServerVariables
  Response.Write elemento&" : "&Request.ServerVariables(elemento)& "<br>"
Next
%>
```

Isto nos daria por um lado o nome da variável e do outro o seu valor. Este tipo de loop For Each/Next se parece a outros já vistos. Neste caso, o loop se realiza tantas vezes como elementos que tiver a coleção (ServerVariables) que não é mais que o conjunto de elementos compreendidos na extensão do objeto (Request). Este tipo de loop é aplicável a outras coleções deste e de outros objetos como, por exemplo, os Request.Form ou Request.Querystring ou as cookies. Desta forma seríamos capazes de visualizar o nome e o conteúdo de tais coleções sem a necessidade de enuncia-las uma por uma.

Objeto Response

Que funções realiza este objeto. Como inscrever textos ou valores e como re-endereçar automaticamente para outras páginas.

Tal como vimos, o objeto Request providencia tudo que é relativo a entrada de dados ao script por parte do usuário (formulários), provenientes de outra URL, do próprio servidor ou do browser.

Resta-nos explicar como o script pode "responder" a estes estímulos, ou seja, como, depois de processar devidamente os dados, podemos imprimir estes na tela, inscreve-los nas cookies ou enviar ao internauta a uma página ou outra. Definitivamente, falta definir a forma na qual o ASP regula o conteúdo que é enviado ao navegador.

Esta função é encarregada pelo objeto Response o qual foi ligeiramente mencionado em capítulos precedentes concretamente em associação ao método Write.

Na prática, sentenças do tipo:

<%Response.Write "Uma cadeia de texto"%>

o bien,

<%Response.Write variable%>

Têm como cometido imprimir no documento HTML gerado uma mensagem ou valor de variável. Este método é geralmente tão utilizado que existe uma abreviação do mesmo de forma a facilitar sua escritura:

<% = variable %> é análogo a <%response.write variable%>

É importante ressaltar o fato que imprimir no documento HTML não significa necessariamente visualizar na tela já que poderíamos nos servir destas etiquetas para criar determinadas etiquetas HTML. Temos aqui um exemplo do que pretendemos dizer:

<% path="http://www.meusite.com/graficos/imagem.gif" %>

<img src="<% =path %>">

Este fragmento de script nos geraria um código HTML que seria recebido no navegador da seguinte forma:

<img src="http://www.meusite.com/graficos/imagem.gif">

Outro elemento interessante deste objeto Response é o método Redirect. Este método nos permite enviar automaticamente o internauta a uma página que nós tivermos decidido.

Esta prática função, pode ser empregada em scripts que enviamos ao visitante de nosso site a uma página ou outra em função do navegador que utiliza ou da língua que prefere. Também é muito útil para realizar páginas "escondidas" que realizam uma determinada função sem mostrar nem texto nem imagens e nos enviam a seguir a outra página que é na realidade a que nós recebemos no navegador.

Aqui pode ser visto uma seqüência de script que nos permitiria usar este método para o caso em que tivermos páginas diferentes para distintas versões de navegadores. Nela empregaremos um objeto que deve nos parecer familiar já que o acabamos de ver (Request.ServerVariables):

```
<%
tipo_navegador = Request.ServerVariables("HTTP_USER_AGENT")
If Instr(1, tipo_navegador, "MSIE 3", 1) <> 0 then
Response.Redirect "MSIE3.asp"
ElseIf Instr(1, tipo_navegador, "MSIE 4", 1) <> 0 then
Response.Redirect "MSIE4.asp"
ElseIf Instr(1, tipo_navegador, "MSIE 5", 1) <> 0 then
Response.Redirect "MSIE5.asp"
ElseIf Instr(1, tipo_navegador, "Mozilla/4", 1) <> 0 then
Response.Redirect "Netscape4.asp"
```

```
Else
Response.Redirect "qualquer.asp"
%>
```

Obviamente, acompanhando este script, devemos ter os correspondentes arquivos MSIE3.asp, MSIE4.asp, MSIE5.asp... Cada um deles com as particularidades necessárias para a boa visualização de nossa página em tais navegadores.

Exemplo simples de ASP

Fazemos um stop para realizar um exemplo simples em ASP que trata o envio de formulários e o uso de loops.

Vamos ver um simples exemplo realizado em ASP que serve para ilustrar o trabalho desenvolvido até o momento no manual da tecnologia. Este exemplo é muito básico, embora experimenta várias das utilidades vistas até agora, como o trabalho com loops e os objetos request e response, que servem para receber dados e imprimi-los na página.

O exemplo em concreto se trata de um gerador de tabelas de multiplicar. À princípio, quando se acessa ao arquivo, mostra-se a tabela do zero e um formulário onde podemos selecionar outro número e ver sua tabela de multiplicar. Ao enviar o formulário se acessa a mesma página, embora agora apareceria a tabela de multiplicar do número selecionado no formulário.

Formulário para selecionar um número

Vejamos agora o formulário que mostra um campo de seleção com os números do 1 ao 10. Este formulário servirá para que o visitante possa selecionar a tabela que deseja ver.

```
<form name=tb action=tb.asp method=post>
<P align=center>Selecione uma opção
<SELECT align=center name=tab style="WIDTH: 40px">
   <OPTION selected>1</OPTION>
   <OPTION >2</OPTION>
   <OPTION >3</OPTION>
   <OPTION >4</OPTION>
   <OPTION >5</OPTION>
   <OPTION >6</OPTION>
   <OPTION >7</OPTION>
   <OPTION >8</OPTION>
   <OPTION >9</OPTION>
   <OPTION >10</OPTION>
</SELECT>
<br>
<INPUT type=submit value="Ver tabela" >
</P>
</form>
```

Temos que observar que a página que vai receber o formulário chama-se tb.asp, segundo se indica no atributo action. O único campo do formulário que se envia é chamado "tab", e salva o número que tenha sido selecionado.

Código para mostrar a tabela de multiplicar correspondente

Começamos recebendo o dado do formulário que nos indica a tabela que o usuário quer visualizar. À princípio, não se recebe nenhum dado do formulário (até que não se envie o formulário não se sabe que

tabela se deseja ver e portanto, havíamos dito que se mostraria a tabela do zero). Sendo assim, se não recebo nada, inicio a zero a variável i, que salva o número da tabela de multiplicar a ser mostrada. No caso de que receba algo do formulário, se inicia a variável i ao valor recebido no campo "tab".

```
'se não se está recebendo dados do formulário
if request.form("tab")="" then
   'inicio a tabela a mostrar a zero
   i=0
else
   'inicio a tabela a mostrar ao dado recebido no formulário
   i=Request.Form ("tab")
end if
```

Agora veremos um loop que mostra a tabela de multiplicar do valor recebido por formulário. Este loop faz uma repetição desde 1 ao 10 e vão se realizando as multiplicações e mostrando os resultados.

```
'mostro a tabela do número que recebo do formulário
Response.Write "Tabela do " & i%><br><br><%
'realizo um loop do 1 ao 10 para mostrar a tabela correspondente
for a=1 to 10
   Response.Write i &" x " & a & " = " & i*a%>
   <br>
   <%
next
%>
```

Código completo

O código completo do exemplo pode ser visto a seguir. Espero que sirva de ajuda para as pessoas que começam a dar seus primeiros passos com ASP.

```
<%@ Language=VBScript %>
<HTML>
<HEAD><title>Tabelas de Multiplicar....</title>
</HEAD>
<BODY bgColor=skyblue>

<div align="center">

<form name=tb action=tb.asp method=post>
<P align=center>Selecione uma opção
<SELECT align=center name=tab style="WIDTH: 40px">
   <OPTION selected>1</OPTION>
   <OPTION >2</OPTION>
   <OPTION >3</OPTION>
   <OPTION >4</OPTION>
   <OPTION >5</OPTION>
   <OPTION >6</OPTION>
   <OPTION >7</OPTION>
   <OPTION >8</OPTION>
   <OPTION >9</OPTION>
   <OPTION >10</OPTION>
</SELECT>
<br>
<INPUT type=submit value="Ver tabela" name=submit1 >
```

```
</P>
</form>
<%

'se não se está recebendo dados do formulário
if request.form("tab")="" then
   'inicio a tabela a mostrar a zero
   i=0
else
   'inicio a tabela a mostrar ao dado recebido no formulário
   i=Request.Form ("tab")
end if

'mostro a tabela do número que recebo do formulário
Response.Write "Tabela do " & i%><br><br><%
'realizo um loop do 1 ao 10 para mostrar a tabela correspondente
for a=1 to 10
   Response.Write i &" x " & a & " = " & i*a%>
   <br>
   <%
next
%>
</div>

</BODY>
</HTML>
```

Objeto Session

A necessidade e utilidade do método session para transferir variáveis.

Nos programas que vimos até agora, utilizamos variáveis que só existiam no arquivo que era executado. Quando carregávamos outra página distinta, os valores destas variáveis se perdiam, ao menos que não nos incomodássemos de passa-los pela URL ou inscreve-los nas cookies ou em um formulário para sua posterior exploração. Estes métodos, embora úteis, não são todo práticos que poderiam em determinados casos, nos quais a variável que queremos conservar há de ser utilizada em vários scripts diferentes e distantes uns dos outros.

Poderíamos pensar que esse problema pode ser resolvido com as cookies já que se trata de variáveis que possam ser invocadas em qualquer momento. O problema, como já foi falado, é que as cookies não são aceitas nem pela totalidade dos usuários, e nem pela totalidade dos navegadores, o qual implica que uma aplicação que se servisse das cookies para passar variáveis de um arquivo a outro não seria 100% infalível.

É então necessário, o poder declarar certas variáveis que possam ser reutilizadas tantas vezes quantas quisermos dentro de uma mesma sessão. Imaginemos um site multilingue, no qual cada vez que quisermos imprimir uma mensagem em qualquer página necessitamos saber em qual idioma deve fazer. Poderíamos introduzir um script identificador da língua do navegador em cada um dos arquivos ou então, declarar uma variável que fosse válida para toda a sessão e que tivesse como valor o idioma reconhecido em um primeiro momento.

Estas variáveis que são válidas durante uma sessão e que logo são "esquecidas" são definidas com o objeto Session da seguinte forma:

Session("nome da variável") = valor da variável

Uma vez definida, a variável Session, será armazenada em memória e poderá ser empregada em qualquer script do site web.

A duração de uma sessão vem definida por padrão em 20 minutos. Isto quer dizer que se em 20 minutos não realizamos nenhuma ação, o servidor dará por finalizada a sessão e todas as variáveis Session serão abandonadas. Esta duração pode ser modificada com a propriedade Timeout:

Session.Timeout = n° de minutos que quisermos que dure

Uma forma de apagar as variáveis Session sem necessidade de esperar que passe este prazo é a partir do método Abandon:

Session.Abandon

Deste modo todas as variáveis Session serão apagadas e a sessão será finalizada. Este método pode ser prático quando estivermos fazendo provas com o script e necessitarmos reiniciar as variáveis.

O que se costuma fazer é criar um arquivo no qual se apagam as cookies e se abandona a sessão. Este arquivo será executado quando quisermos apagar e fazer uma conta nova:

```
<% @ LANGUAGE="VBSCRIPT" %>
<HTML>
<HEAD>
<TITLE>Posta a zero</TITLE>
</HEAD>
<BODY>
<%
For Each biscoito in Response.Cookies
  Biscoito=""
Next
Session.Abandon
%>
Apagar e conta nova!!<br>
<a href="url da página de início">Voltar ao princípio</a>
</BODY>
</HTML>
```

Trabalhar com bases de dados em ASP

ASP, objetos ADO e conectores ODBC. Os elementos indispensáveis para o trabalho com bases de dados.

Uma das principais vantagens que apresenta trabalhar com páginas dinâmicas é o poder armazenar os conteúdos em base de dados. Desta forma, podemos organizá-los atualizá-los e buscá-los de uma maneira muito mais simples.

ASP nos oferece uma forma muito eficaz de interagir com estas bases de dados graças ao uso do componente ADO (ActiveX Data Objects) o qual permite acessar a tal bases de uma forma simples.

Este ADO não é mais que um conjunto de objetos que, utilizados em conjunto, nos permitem explorar de uma forma muito versátil as bases de dados de nossa aplicação. Não entraremos por este momento em considerações teóricas a respeito.

Por outra parte, os scripts ASP devem estabelecer um diálogo com a base de dados. Este diálogo se realiza a partir de um idioma universal: o SQL (Structured Query Language) o qual é comum a todas as bases de dados. Esta linguagem é muito potente e fácil de aprender.

Neste manual de ASP nos limitaremos a utilizar as instruções básicas que serão aprendidas a medida que explicamos as diferentes formas de agir sobre uma base de dados a partir de páginas ASP.

A base de dados que foi utilizada nestes exemplos é MS Access. É claro que não é a única, mas é a mais corrente em pequenos PCs e é absolutamente operativa sempre que as tabelas não sejam astronomicamente grandes. Esperamos poder lhes oferecer proximamente também um pequeno curso de Access no qual explica os princípios rudimentares necessários para poder nos servir dele. No obstante, esta aplicação é suficientemente fácil e intuitiva como para poder prescindir de tal curso por este momento.

Seleções em uma tabela

Forma simples de realizar seleções dentro de uma tabela. Exemplos práticos.

Dentro de uma base de damos, organizada por tabelas, a seleção de uma tabela inteira ou de um certo número de registros é uma operação rotineira.

A partir desta seleção pode-se posteriormente efetuar toda uma série de mudanças ou então, realizar uma simples leitura.

O seguinte script nos permite realizar a leitura da tabela clientes, contida em nossa base de dados. À primeira vista tudo pode nos parecer um pouco complexo, mas nada mais longe da realidade.

```
<HTML>
<HEAD>
<TITLE>Leitura de registros de uma tabela</TITLE>
</HEAD>
<BODY>
<h1><div align="center">Leitura da tabla</div></h1>
<br>
<br>
<%
'Antes de mais nada há que instanciar o objeto Connection
Set Conn = Server.CreateObject("ADODB.Connection")

'Uma vez instanciado Connection podemos abri-lo e lhe atribuímos à base de dados onde vamos efetuar as operações
Conn.Open "Minhabase"

'Agora criamos a sentença SQL que nos servirá para falar à BD
sSQL="Select * From Clientes Order By nome"

'Executamos a ordem
set RS = Conn.Execute(sSQL)

'Mostramos os registros%>
<table align="center">
<tr>
<th>Nome</th>
<th>Telefone</th>
</tr>
<%
Do While Not RS.Eof
%>
<tr>
```

```
<td><%=RS("nombre")%></td>
<td><%=RS("telefono")%></td>
</tr>
<%
RS.MoveNext
Loop

'Fechamos o sistema de conexão
Conn.Close
%>

</table>

<div align="center">
<a href="insertar.html">Acrescentar um novo registro</a><br>
<a href="actualizar1.asp">Atualizar um registro existente</a><br>
<a href="apagar1.asp">Apagar um registro</a><br>
</div>

</BODY>
</HTML>
```

Se o que desejamos é interrogar uma base de dados, o primeiro que há que fazer é obviamente estabelecer a conexão com ela. Isto se faz a partir do objeto Connection o qual é "invocado" ou mais tecnicamente dito instanciado por meio da primeira instrução na qual o objeto toma o nome arbitrário da variable Conn .

O passo seguinte é abrir o objeto e atribuir a base de dados com a qual deve entrar em contato. Neste caso, chamamos a base de Minhabase. Este deve de ser o mesmo nome com o qual a batizamos quando configuramos os conectores ODCB, ademais, este nome não tem porque coincidir necessariamente com o nome do arquivo.

Uma vez criada a conexão a nossa base de dados, o passo seguinte é fazer nossa petição. Esta petição pode ser formulada primeiramente e armazenada em uma variável (sSQL) para, a seguir, ser executada por meio da instrução seguinte.

A petição que realizamos neste caso é a de selecionar todos os campos que existem nas tabela clientes (* é um coringa) e ordenar os resultados por ordem alfabética em relação ao campo nome.

O resultado de nossa seleção é armazenado na variável RS em forma de tabela. Para ver a tabela o que há que fazer agora é "passear" por esta tabela "virtual" RS a qual possui uma espécie de cursor que, a menos que se especifique outra coisa, aponta ao primeiro registro da seleção. O objetivo agora é fazer deslocar-se ao cursor ao longo da tabela para poder lê-la em sua totalidade. A forma de fazê-lo é a partir de um loop Do While o qual foi explicado anteriormente e que o único que faz é executar as instruções compreendidas entre o Do e o Loop sempre que a condição proposta (Not RS.Eof) seja verdadeira. Isto se traduz como "Executar este conjunto de instruções enquanto a tabela de resultados (RS) não chegar ao final" (Eof, End of File).

As instruções incluídas no loop são, por um lado, a impressão no documento dos valores de determinados campos (=RS("nome do campo")) e por outro, saltar de um registro ao outro mediante a instrução **RS.MoveNext.**

Todo este conjunto de instruções ASP vem em combinação com um código HTML que permite sua visualização em forma de tabela. Ademais, foram incluídos uns links que apontam para outra série de scripts que veremos mais adiante e que formarão em conjunto uma aplicação.

É interessante ver o código fonte resultante deste script. Neste caso o código que vê o cliente é sensivelmente mais simples.

Criação de um novo registro

Forma simples de introduzir novos elementos na tabela. Exemplos práticos.

Neste caso o que buscamos é criar, a partir dos dados recebidos de um formulário, um novo registro em nossa tabela clientes. Teremos então, dois arquivos diferentes, um que poderia ser um HTML puro no qual introduzimos o formulário a preencher e que nos envia ao segundo, um script muito parecido ao previamente visto para realizar uma seleção. Aqui estão os dois scripts:

```
<HTML>
<HEAD>
<TITLE>Inserir.html</TITLE>
</HEAD>
<BODY>
<div align="center">
<h1>Inserir um registro</h1>
<br>
<FORM METHOD="POST" ACTION="inserir.asp">
Nombre<br>
<INPUT TYPE="TEXT" NAME="nome"><br>
Teléfono<br>
<INPUT TYPE="TEXT" NAME="telefone"><br>
<INPUT TYPE="SUBMIT" value="Inserir">
</FORM>
</div>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Inserir.asp</TITLE>
</HEAD>
<BODY>

<%
'Recolhemos os valores do formulário
nome=Request.Form("nome")
telefone= Request.Form("telefone")

'Instanciamos e abrimos nosso objeto conexão
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Minhabase"

'Agora criamos a sentença SQL
sSQL="Insert Into Clientes (nome,telefone) values ('" & nome & "','" & telefone & "')"

'Executamos a ordem
set RS = Conn.Execute(sSQL)
%>

<h1><div align="center">Registro Inserido</div></h1>
```

```
<div align="center"><a href="lectura.asp">Visualizar o conteúdo da base</a></div>

<%
'Fechamos o sistema de conexão
Conn.Close
%>

</BODY>
</HTML>
```

Como pode ser visto, a forma de operar é idêntica à vista anteriormente para o display de uma tabela. Neste caso introduzimos um link a este primeiro script de leitura para ver como as mudanças se tornaram efetivas.

A construção da sentença SQL se faz por fusão dos distintos elementos constitutivos. A forma de fundi-los mediante o símbolo &. Tudo que seja texto tem que ir entre aspas. Seria interessante introduzir uma linha suplementaria em seu código para imprimir a sSQL formada. A linha seria do seguinte tipo:

```
Response.Write sSQL
```

Esta linha seria situada evidentemente depois de haver construído a sentença.

Atualização de um registro existente

Sentenças SQL para realizar atualizações na tabela.

Para mostrar como se atualiza um registro presente em nossa base de dados, vamos faze-lo a partir de um caso um pouco mais complexo para começarmos a nos familiarizar com estas operações. Realizaremos dois scripts que permitem mudar o numero de telefone das distintas pessoas presentes em nossa base. O nome destas pessoas, assim como o novo número de telefone, serão recolhidos por meio de um formulário.

O arquivo do formulário vai ser desta vez um script ASP no qual efetuaremos uma chamada a nossa base de dados para construir um menu desdobrável onde apareçam todos os nomes. A coisa ficaria assim:

```
<HTML>
<HEAD>
<TITLE>Atualizar1.asp</TITLE>
</HEAD>
<BODY>
<div align="center">
<h1>Atualizar um registro</h1>
<br>

<%
'Instanciamos e abrimos nosso objeto conexao
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Minhabase"
%>

<FORM METHOD="POST" ACTION="atualizar2.asp">
Nombre<br>
<%
'Criamos a sentença SQL e a executamos
sSQL="Select nome From clientes Order By nome"
set RS = Conn.Execute(sSQL)
```

```
%>
<select name="nome">
<%
'Geramos o menu desdobravel
Do While not RS.eof%>
  <option><%=RS("nome")%>
  <%RS.movenext
Loop
%>
</select>
<br>
Telefone<br>
<INPUT TYPE="TEXT" NAME="telefone"><br>
<INPUT TYPE="SUBMIT" value="Atualizar">
</FORM>
</div>

</BODY>
</HTML>
```

A maneira de operar para construir o menu desdobrável é a mesma que para visualizar a tabela. De novo empregamos um loop Do While que nos permite mostrar cada uma das opções.

O script de atualização será muito parecido ao de inserção:

```
<TITLE>Atualizar2.asp</TITLE>
</HEAD>
<BODY>

<%
'Recolhemos os valores do formulário
nome=Request.Form("nome")
telefone= Request.Form("telefone")

'Instanciamos e abrimos nosso objeto conexao
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Minhabase"

'Agora criamos a sentença SQL
sSQL="Update Clientes Set telefone='" & telefone & "' Where nome='" & nome & "'"

'Executamos a ordem
set RS = Conn.Execute(sSQL)
%>

<h1><div align="center">Registro Atualizado</div></h1>
<div align="center"><a href="lectura.asp">Visualizar o conteudo da base</a></div>

<%
'Fechamos o sistema de conexao
Conn.Close
%>

</BODY>
</HTML>
```

Nada a comentar a respeito, salvo a estrutura da sentença SQL que neste caso realiza um Update no lugar de um Insert. Aconselhamos, como para o caso precedente imprimir o valor de sSQL de forma a ver como fica a sentença uma vez construída.

Excluir um registro

Como apagar um registro. Scripts de exemplo.

Outra das operações fundamentais que podem ser realizadas sobre uma base de dados é o apagar um registro. Para fazê-lo, SQL nos propõe sentenças do tipo Delete. Vejamos com um exemplo, aplicado a nossa agenda. Primeiro, criaremos um menu desdobrável dinâmico como para o caso das atualizações:

```
<HTML>
<HEAD>
<TITLE>Apagar1.asp</TITLE>
</HEAD>
<BODY>
<div align="center">
<h1>Apagar um registro</h1>
<br>
<%
'Instanciamos e abrimos nosso objeto conexao
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Minhabase"
%>

<FORM METHOD="POST" ACTION="apagar2.asp">
Nome<br>
<%
'Criamos a sentença SQL e a executamos
sSQL="Select nome From clientes Order By nome"
set RS = conn.execute(sSQL)
%>
<select name="nome">
<%
'Geramos o menu desdobravel
Do While not RS.eof%>
  <option><%=RS("nome")%>
  <%RS.movenext
Loop
%>
</select>
<br>
<INPUT TYPE="SUBMIT" value="Apagar">
</FORM>
</div>

</BODY>
</HTML>
```

O seguinte passo é fazer efetiva a operação a partir da execução da sentença SQL que construímos a partir dos dados do formulário:

```
<HTML>
```

```
<HEAD>
<TITLE>Apagar2.asp</TITLE>
</HEAD>
<BODY>
<%
'Recolhemos os valores do formulario
nome=Request.Form("nome")

'Instanciamos e abrimos nosso objeto conexao
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Minhabase"

'Agora criamos a sentença SQL
sSQL="Delete From Clientes Where nome='" & nome & "'"

'Executamos a ordem
set RS = Conn.Execute(sSQL)
%>

<h1><div align="center">Registro Apagado</div></h1>
<div align="center"><a href="leitura.asp">Visualizar o conteudo da base</a></div>

<%
'Fechamos o sistema de conexao
Conn.Close
%>

</BODY>
</HTML>
```

Crie seu próprio buscador

Propomos um buscador básico para implementar em seu website, além de lhe mostrar funções para o tratamento de variáveis tipo cadeia que podem ser muito úteis para outras aplicações .

Quando trabalhamos com sites baseados em bancos de dados e nossos conteúdos começam a crescer, pode ser muito prático para o navegante poder recorrer a um formulário no qual possa introduzir palavras-chaves e operadores que lhe ajudem a matizar o elemento que estiver buscando. Este tipo de campo pode ser visto em uma infinidade de páginas e, embora sejam distintos em seu funcionamento em muitos casos, todos têm algo em comum: tratam-se de programas que permitem processar uma variável do tipo cadeia e transforma-la em uma ordem de busca para o banco de dados.

Neste artigo vamos propor duas funções que, usadas conjuntamente, permitem a criação de uma instrução SQL. O script permite especificar quais serão os campos de busca e dentro de que tabela a realizaremos.

Este programa deverá ir combinado com outro pequeno script de colheita de dados por formulário como os vistos em nosso manual de ASP a partir do qual obteríamos a variável cadeia introduzida pelo internauta.

Estas duas funções podem ser utilizadas diretamente para cada caso particular a condição de especificar os campos de busca no array campos, especificar a tabela e modificar a função gerasql para que realize a seleção dos campos que desejarmos.

Neste caso, com o objetivo de facilitar a compreensão simplificamos ao máximo as funções que o buscador pode realizar. Na verdade, o buscador só tratará campos usando o operador like. Por outro lado, empregará unicamente como operadores "+" e "-".

Todo tipo de modificações mais ou menos complexas, podem ser introduzidas de forma a melhorar sua versatilidade:

-Emprego de parênteses para maior eficácia dos operadores

-Eliminação de seqüências repetidas como "++" ou "--"

-Eliminação de operadores no princípio e final da cadeia

-Utilização de curingas...

Passamos agora a mostrar a lista para poder comentá-lo mais detalhadamente a seguir:

```
<%
function Tirar(cadeia,campos)
dim i
dim TirarAux
while InStr(cadeia,"  ")
  Cadeia=Replace(Cadeia,"  "," ")
wend
if len(cadeia)>0 then
  if InStr(cadeia," ")>0 then
    Tirar= Tirar(left(cadeia,InStr(cadeia," ")-1),campos) & " OR " & Tirar(right(cadeia,len(cadeia)-
InStr(cadeia," ")),campos)
  elseif InStr(cadeia,"+")>0 then
    Tirar=Tirar(left(cadeia,InStr(cadeia,"+")-1),campos) & " AND "& Tirar(right(cadeia,len(cadeia)-
InStr(cadeia,"+")),campos)
  elseif InStr(cadeia,"-")>0 then
    Tirar=Tirar(left(cadeia,InStr(cadeia,"-")-1),campos) & " AND NOT " &
Tirar(right(cadeia,len(cadeia)-InStr(cadeia,"-")),campos)
  else
  'observamos a sentenca
  TirarAux=""
  i=1
  TirarAux= "( " & campos(i) & " Like '%" & cadeia & "%'"
  i=i+1
  while len(campos(i))>0
    TirarAux= TirarAux & " OR " & campos(i) & " Like '%" & cadeia & "%'"
    i=i+1
  wend
  TirarAux=TirarAux & " )"
  Tirar=TirarAux
  end if
else
tirar=""
end if
end function

function GeraSql(cadeia,tabela,campos)
if len(cadeia)>0 then
  geraSql="Select * from " & tabela & " Where " & Tirar(cadeia,campos)
else
  Response.Write "Nao ha criterios"
end if
```

```
end function

dim campos(3) 'o tamanho do array deve superar em um ao numero de campos
campos(1)="nome_campo1"
campos(2)="nome_campo2"

'para mostrar qual seria o resultado...
cadeia="ola carioca+cocacola-nescau"
tabla="qualquer"
resultado=GeraSql(cadeia,tabela,campos)

Response.Write resultado
%>
```

Como dissemos, o script consta principalmente de duas funções. A primeira delas, tirar, se encarrega de tratar a cadeia para separar as palavras e, tendo em conta os operadores, construir o fragmento final da sentença SQL. A busca se realiza dentro de uma série de campos que são definidos ao exterior da função em forma de array.

Nesta função se empregam diferentes funções de tratamento de variáveis de tipo cadeia, as quais explicamos a seguir:

InStr(cadeia,subcadeia) Devolve o número das posições nas quais uma determinada sub-cadeia aparece na cadeia principal
Replace(cadeia,subcadeia1,subcadeia2) Substitui um fragmento (subcadeia1) de uma cadeia por um novo fragmento (subcadeia2)
Len(cadeia) Devolve-nos a longitude da cadeia
Left(cadeia,numero_caracteres) Seleciona uma determinada quantidade de caracteres de nossa cadeia começando pela esquerda
Right(cadeia,numero_caracteres) Seleciona uma determinada quantidade de caracteres de nossa cadeia começando pela direita

É interessante observar a técnica de recursividade que se utiliza para decompor a cadeia principal em palavras independentes. Esta técnica se baseia em que função tirar se chama a si mesma para cada uma das sub-cadeias da cadeia principal. A parte dessa pequena astúcia o resto da função é de leitura fácil.

A segunda função, gerasql, se encarrega de acrescentar ao fragmento final tabela e o tipo de seleção que queremos realizar. Neste caso foram selecionados como resultado todos os campos da tabela, mas evidentemente, isto não tem porquê ser assim.

A última parte do script consiste na especificação dos campos dentro dos quais desejamos realizar nossa busca. A observar que o array há de ter uma longitude maior ao número de campos para o correto funcionamento do programa.

Índice Programação em ASP