CAPÍTULO I – CONCEITOS INTRODUTÓRIOS

Aplicações cliente-servidor

Consiste na divisão de processos entre estações clientes e servidores, com a finalidade de buscar melhor performance, menor tempo de resposta e maior facilidade de manutenção.

Active Server Pages – O que são?

Active Server Pages são páginas web que possuem conteúdo dinâmico. Tais páginas consistem em arquivos de extensão .asp que contêm combinações de Server-Side scripts e tags HTML.
Todo o código de programação existente em páginas Asp é executado no servidor, e este retorna ao cliente somente respostas em HTML padrão – o que faz com que aplicações Asp possam ser acessadas por qualquer browser existente no mercado. Uma aplicação feita em Asp pode ainda conter linhas de Client-Side script, que serão executados na estação cliente. Essas páginas devem estar hospedadas num servidor Microsoft Information Server.

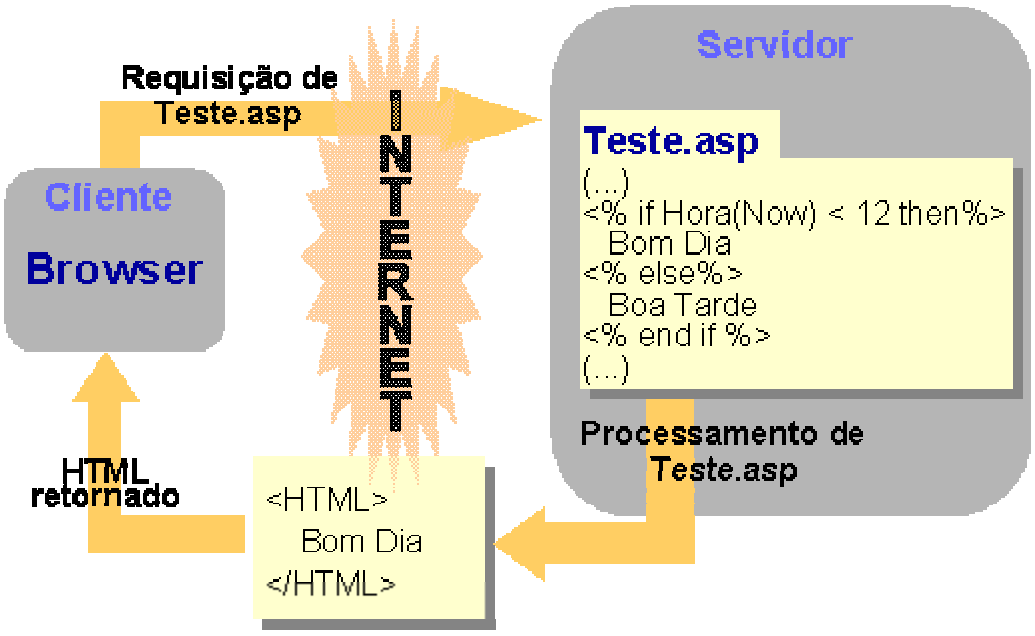
Client Side scripts

São códigos de programa que são processados pela estação cliente. Geralmente em aplicações voltadas à Internet, o código que é executado no cliente cuida apenas de pequenas consistências de telas e validações de entrada de dados
Em se tratando de páginas web, os client-side scripts terão de ser processados por um browser. O maior problema de se utilizar este tipo de artifício em uma aplicação é a incompatibilidade de interpretação da linguagem entre os browsers. O Microsoft Internet Explorer, por exemplo, é capaz de interpretar o Visual Basic Script, porém o Netscape não o faz sem o auxílio de um plug in (que foi desenvolvido por terceiros). Há ainda o problema de versões muito antigas de navegadores, que não conseguem interpretar nenhum script.
Em grande parte das situações, não é possível exigir que o usuário final disponha de determinado produto para acessar a aplicação. Portanto é importante pesar todos estes fatores ao planejar alguma aplicação com client-side scripts.
A linguagem script mais indicada para se construir client-side scripts é o JavaScript, devido a sua compatibilidade com os dois browsers (Netscape e Microsoft Internet Explorer, que devem ser de versões iguais ou superiores a 3.0 e 4.0 respectivamente).

Server Side scripts

São códigos de programa que são processados no servidor. Devido a este fato, não é necessário preocupar-se com a linguagem que o código foi criado: o servidor é quem se encarrega de interpretá-lo e de devolver uma resposta para o cliente. Em páginas Asp, são esses códigos os maiores responsáveis pelos resultados apresentados, e a linguagem default utilizada é o Visual Basic Script.

Como funciona uma página Asp? - Esquema



Ambiente de desenvolvimento de páginas Asp

Como os arquivos asp são arquivos do tipo texto (ASCII), eles podem ser escritos em um editor de textos comum – Edit ou Notepad, por exemplo. Existe também o MS-Visual Interdev, que proporciona um ambiente mais agradável de desenvolvimento, mas exige os mesmos conhecimentos do programador.

Pré-Requisitos de funcionamento

Páginas asp necessitam ser hospedadas no servidor Web da Microsoft: o Internet Information Server (IIS) na versão 3 ou superior. Este servidor deve ser instalado numa máquina NT Server 4. Para o IIS 3, ainda é preciso instalar um pacote adicional do Asp para que as aplicações funcionem. A partir da versão 4 este pacote já vem incorporado ao IIS.

Resumo

Neste capítulo aprendemos que...

- Active Server Pages: São páginas web dinâmicas que combinam HTML, server-side scripts e podem também conter client-side scripts.
- Client-side scripts: São códigos de programação que rodam na estação cliente. Geralmente são responsáveis por pequenas validações e consistências. Seu funcionamento é dependente do browser utilizado.
- Server-side scripts: São códigos de programação que rodam no servidor, sendo assim, independentes do browser. Em programas Asp, são os grandes responsáveis pela atividade da aplicação.
- Funcionamento: Ao atender um pedido por uma página, o servidor processa o código script da mesma e retorna ao cliente solicitante uma resposta HTML.
- Requisitos: As páginas Asp devem ser hospedadas em um servidor Windows NT Server 4 com o Internet Information Server (versão 3 ou superior).

Dicas

- Você pode tornar suas aplicações muito mais acessíveis através da Internet. Seus clientes ficarão muito mais satisfeitos ao encontrar uma aplicação disponível em qualquer parte do mundo, em qualquer hora, em qualquer microcomputador.

- Pense bem antes de adicionar client-side scripts em suas páginas. Você poderá tornar uma aplicação perfeita em algo inacessível para alguns usuários.

Que browsers suportam ASP?
Se a aplicação não possuir client-side script, todos os browsers suportam ASP. Isto acontece pelo fato das páginas ASP serem processadas pelo servidor. O que o cliente recebe é somente código HTML padrão.

Terei meu código fonte protegido?
Sim. Como o servidor retorna somente o resultado HTML, o código fonte (lógica) fica preservado. Se, no browser, visualizarmos a fonte da página, veremos somente código HTML.

Meu NT Server já veio com o IIS 2. Posso utilizá-lo para Active Server Pages?
Não. A tecnologia Asp foi incorporada somente a partir da versão 3.

Capítulo II - O Internet Information Server

O **Microsoft Internet Information Server** (IIS) é o servidor web da Microsoft. É nele que devemos configurar os alicerces das nossas aplicações ASP, criando diretórios virtuais, definindo permissões de acesso e disponibilizando as aplicações. É através do **Management Console** do IIS que faremos estas configurações.

Instalação do IIS

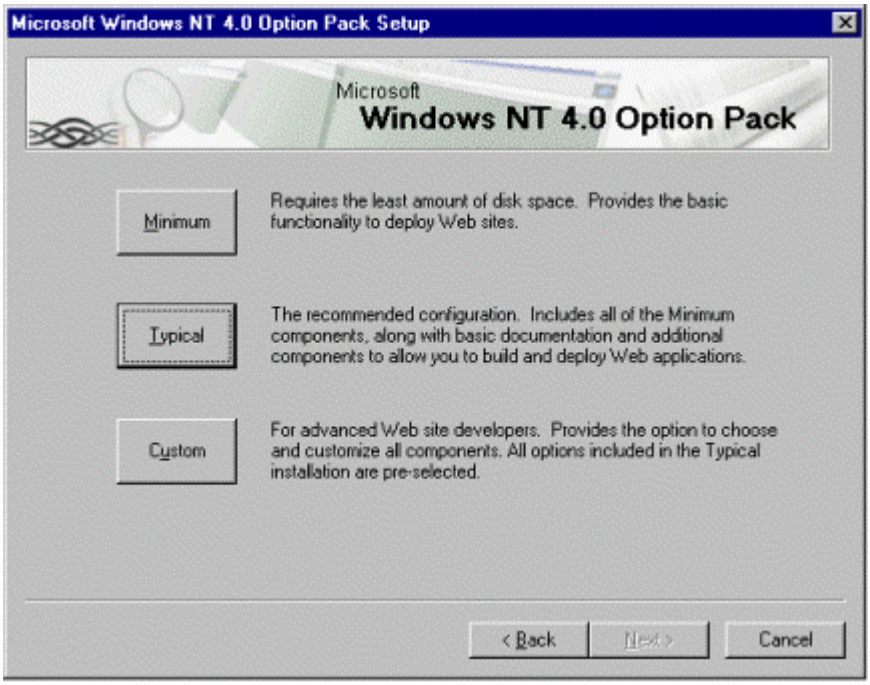
Antes de detalharmos a instalação do IIS, é preciso saber quais são seus pré-requisitos:

- Windows NT Server 4.0
- Internet Explorer 4.0 ou superior.
- Option Pack 4.0

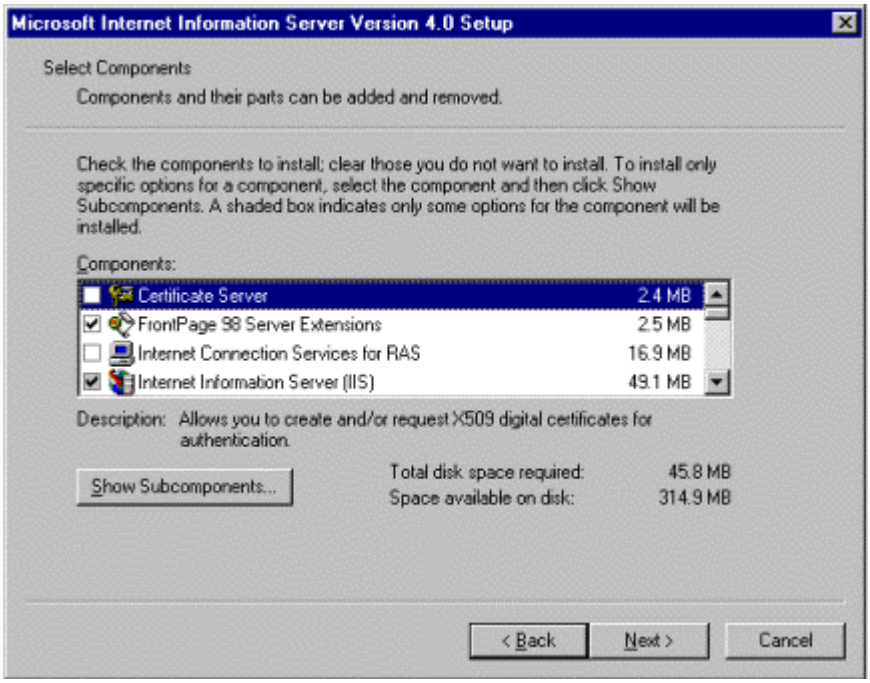
A partir destes softwares e pacotes, podemos iniciar a instalação do IIS que na verdade, faz parte do pacote Option Pack 4.0 da Microsoft.

Ao iniciar a instalação, a primeira tela a ser apresentada é a seguinte:

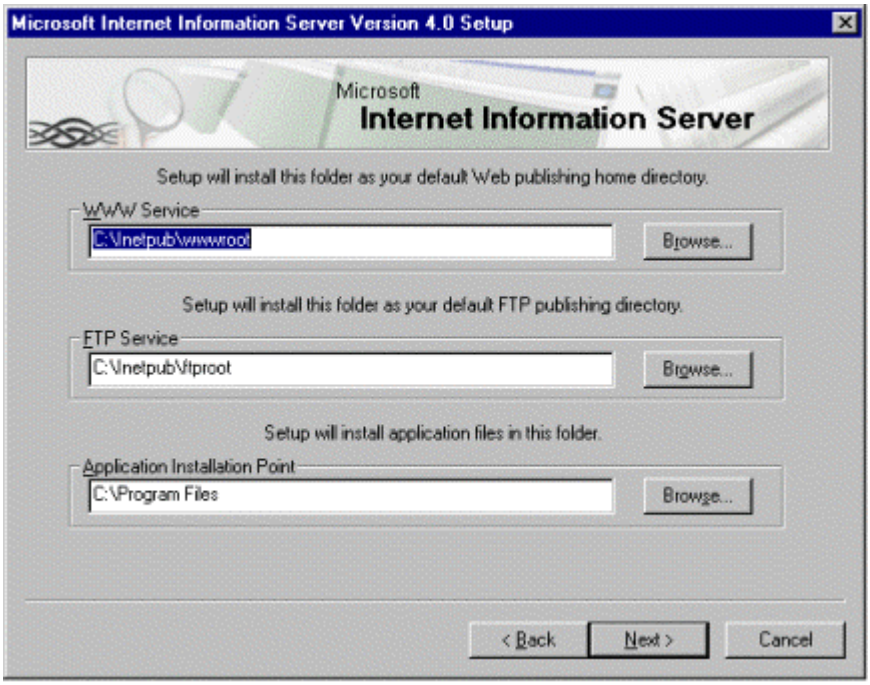
Se você observar o conteúdo desta tela, verá que um dos itens que será instalado é o Internet Information Server (IIS).
Clique em **Next**.



Especifique o tipo de instalação a ser feita. Como exemplo, estaremos utilizando a instalação customiza, onde podemos escolher os componentes a serem instalados.



Nesta tela, selecione os componentes a serem instalados.

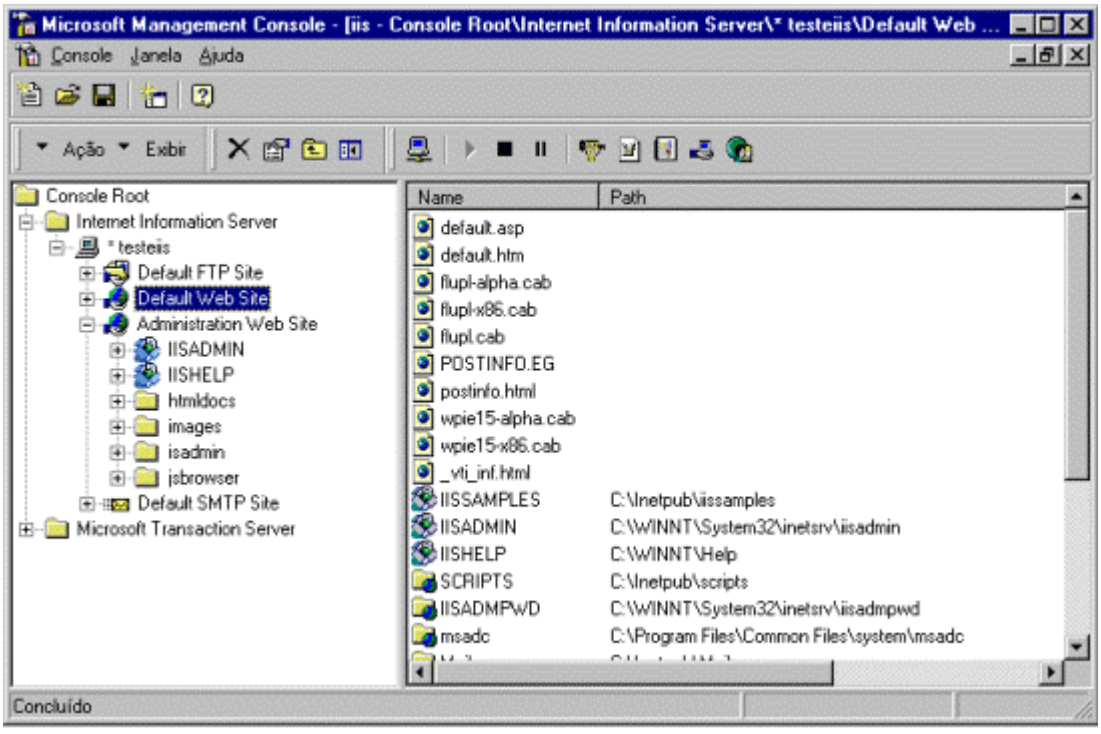


Depois de selecionados os componentes, será solicitado ao administrador, o local onde os pacotes devem ser instalados. Como indica a figura ao lado.

Estes são os passos para a instalação do IIS. O próximo item a vermos será como utilizar o Management Console para administrar os sites criados.

O Management Console

O Management Console é o painel de administração geral do Microsoft Internet Information Server. É através dele que são feitas as configurações de todos os sites e aplicações hospedados no servidor.

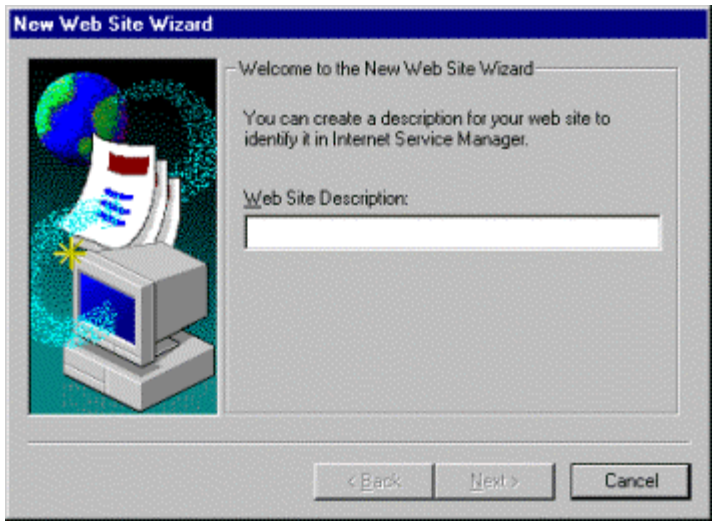


Gerenciando Web Sites

Em versões anteriores do IIS, não era possível criar vários Web Sites com o mesmo IP. Nesta versão já é possível criar vários Web Sites com o mesmo IP e ainda administrá-los de forma diferente.

Criando novos Sites

Para criar um novo site, clique com o botão direito do mouse sobre o nome da máquina na qual deve residir este site, selecione a opção **New → New Site**. A tela abaixo será mostrada:

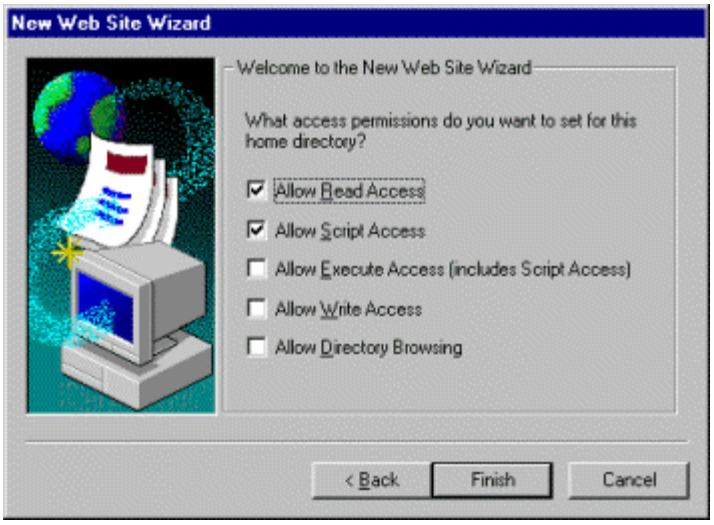


Nesta tela, indique uma descrição do novo site a ser criado.

Você deve indicar para o Setup, o IP da máquina e o número da porta do servidor Web.

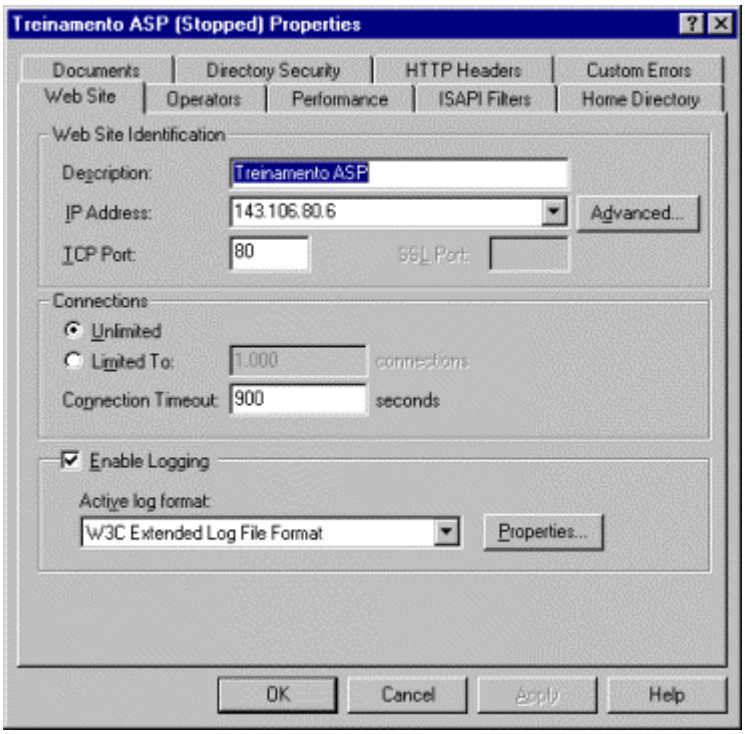
Geralmente, as portas de FTP e HTTP são 21 e 80, respectivamente.

Nesta parte do Setup indique o caminho do diretório Home.



Para finalizar, configure as permissões de acesso, e selecione as permissões de Script

Depois de criado o novo site, você pode alterar as configurações. A tela baixo é mostrada quando você executa as propriedades de um Web Site, para isso, clique com o botão direito do mouse no Web Site e selecione a opção Properties.



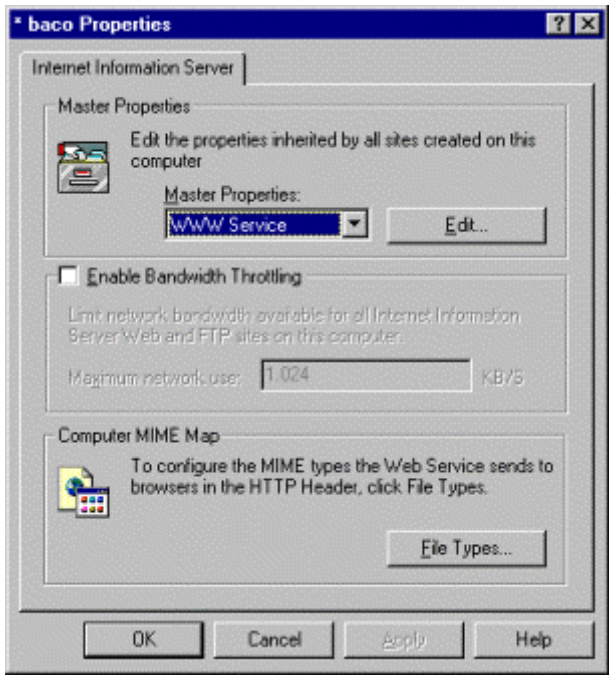
Cada Web Site criado possui suas próprias configurações. Estas configurações são independentes de outros site, ou seja, as configurações feitas para um site, não serão as mesmas para outro, a não ser que você faça as mesmas configurações.

A tela de propriedades dos Web Sites permite que você configure contas de segurança, performance, filtros ISAPI, diretório Home, documentos, cabeçalhos HTTP, mensagens de erros customizadas.

Item	Descrição
Web Site	Configuração do IP, portas de conexão e capacidades de login.
Security Accounts	Configuração das permissões de usuários.
Performance	Estabele performance, largura de banda e configurações de conexão.
ISAPI Filters	Gerenciamento dos filtros ISAPI.
Home Directory	Configuração de permissão de acesso e configurações das aplicações.
Documents	Configuração de documentos e rodapés padrões.
Directory Security	Configuração de autenticação da senha, segurança de comunicação e restrições TCP/IP.
Custom Erros	Definição e configuração das mensagens de erro do HTTP.

Alterando os valores padrão do Site

Você também pode alterar as propriedades padrão (Default) para todos os Sites criados. Para isso, selecione o nome da máquina onde você deve criar os Web Sites. A tela a seguir será mostrada:



Para alterar os valores, clique em Edit ...

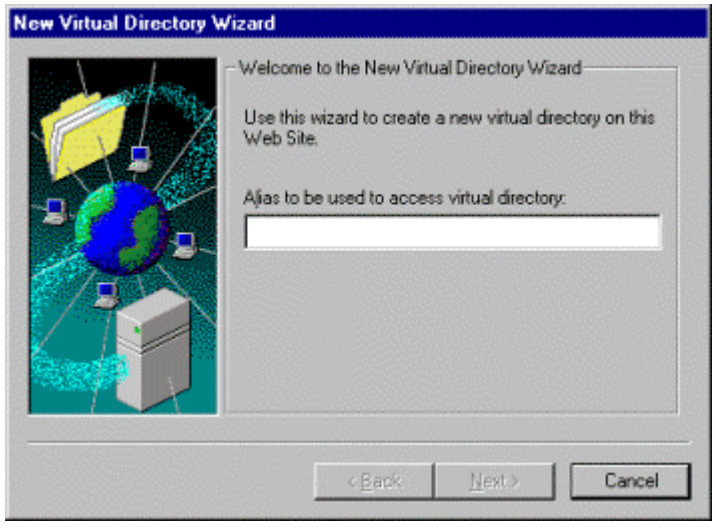
Diretórios Físicos e Virtuais

Um diretório físico é simplesmente o local onde está situada uma aplicação (conjunto de arquivos ASP e páginas HTML) no servidor, como por exemplo c:\inetpub\apps\ . Um diretório virtual nada mais é do que um atalho onde o IIS aponta para um diretório físico, o que não permite desse modo que se visualize todo o path dos arquivos acessados. Um exemplo de diretório virtual seria www.servidor.unicamp.br/dirvirtual .

É através do Management Console que podemos criar diretórios virtuais e ajustar as propriedades de cada um deles.

Criando diretórios virtuais

Para criar um novo diretório virtual, clique com o botão direito do mouse no Web Site onde está localizado este diretório e selecione a opção **New → Virtual Directory**. A tela a seguir será mostrada:



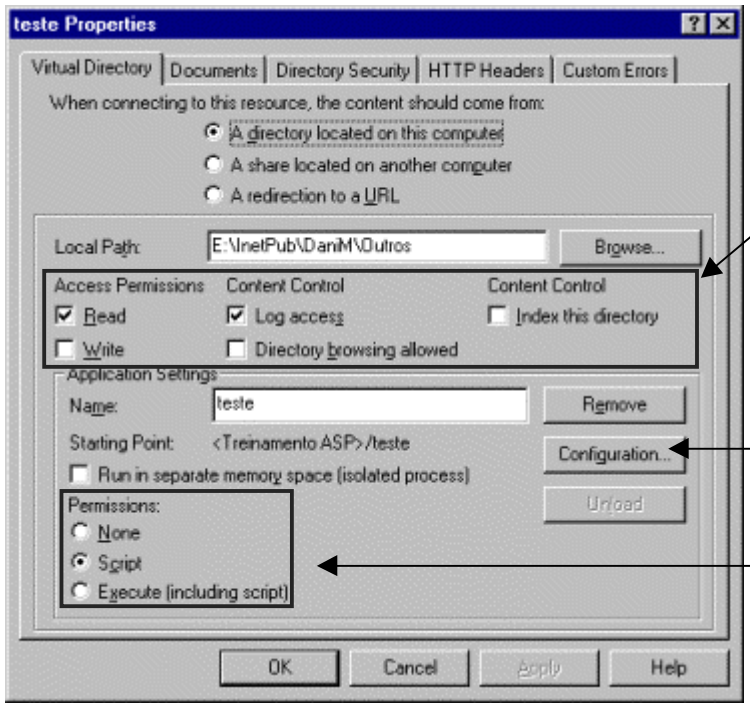
Especifique um alias para o diretório físico.

Nesta tela, você deve especificar o diretório físico onde estão suas aplicações e páginas.
Clique no botão Browse ..., para facilitar.

Nesta próxima página, especifique as permissões de acesso de seus usuários.
Para que um usuário tenha permissão de "rodar" um script, é preciso que seja dado a permissão **Allow Script Access**.

Configurando as características das aplicações

As propriedades de um diretório virtual é um pouco diferente das propriedades dos Sites, pois você não precisa configurar especificações do serviço HTTP. Para visualizar as propriedades dos diretórios virtual, selecione-o e clique com o botão direito do mouse, escolha a opção Properties.

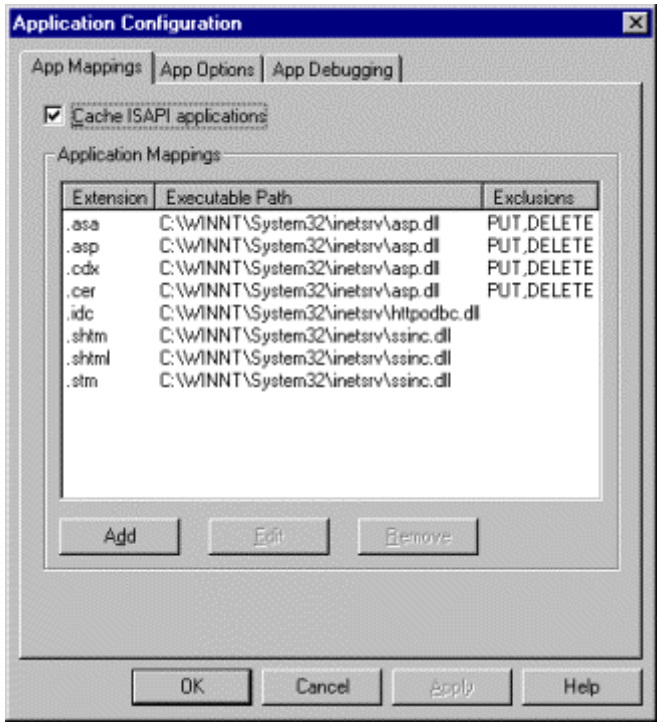


Permissões de acesso
neste item estão definidas as permissões de acesso que os usuários possuem para a aplicação localizada no diretório e:\InetPub\DaniM\Outros

Clicando neste botão, você poderá configurar as propriedades da aplicação.

Permissões de Script
Neste item, você deve configurar as permissões para execução de scripts. Para que um ASP possa ser "rodado" é preciso, pelo menos, permissão de Scritp.

Clicando no botão Configuration ..., a seguinte tela será mostrada:



Utilize esta interface para configurar os arquivos que a aplicação estará utilizando, dependendo da sua extensão. Por exemplo, se sua aplicação estiver utilizando arquivos com extensão .ASP, estes arquivos devem ser processados pela DLL asp.dll, localizada em c:\WINNT\system32\inetsrv\asp.dll. Observe a figura ao lado onde cada extensão tem sua DLL correspondente.

Requisitos de segurança serão abordados em outro capítulo. Por enquanto, vimos apenas como configurar o IIS para reconhecer Sites e aplicações.

Resumo

Neste capítulo aprendemos que ...

- Para instalar o IIS 4.0 é preciso do Windows NT Server 4.0, Internet Explorer 4.0 ou superior e Option Pack 4.0.
- O Management Console é a interface utilizada para a administração do IIS.
- É possível criar vários Web Sites para o mesmo endereço de IP e administrá-los de forma diferente.
- Com os diretórios virtuais é possível criar um atalho específico para sua aplicação dentro de um determinado Web Site.
- Para que seja possível executar scripts (CGIs, ASP) a partir de um endereço, é preciso configurar as permissões de acesso para, pelo menos, **Allow Script Access**.

CAPÍTULO III – FORMS E ASP

A relação entre form HTML e ASP é muito importante porque a partir de formulário podemos disparar ações, e é nesta ação que iremos "chamar" uma página ASP. Com isso podemos consistir os campos, passar parâmetros de uma página para outra.

A sintaxe para utilizar um Form no HTML é a seguinte:

```
<FORM ACTION = nome_arquivo.ASP NAME= nome_do_form METHOD = método>
    Objetos da Página
</FORM>
```

Os parâmetros da Tag Form:

ACTION: neste item, você deve especificar o diretório e nome do arquivo ASP a ser disparado.

NAME: especifique um nome para seu formulário. Item não obrigatório.

METHOD: define como seus dados serão enviados para o servidor. Existem vários métodos, mas o que iremos utilizar com mais frequência serão dois: GET e POST:

Get

Utilizando este método de envio dos dados, os dados que estão sendo enviados serão mostrados pelo browser.

```
<FORM ACTION = "nome_arq.asp" METHOD=GET>
    <INPUT TYPE=SUBMIT>
</FORM>
```

Post

Utilizando este método, os dados serão enviados para o ASP determinado no parâmetro ACTION.

```
<FORM ACTION = "nome_arq.asp" METHOD=POST>
    <INPUT TYPE=SUBMIT>
</FORM>
```

Uma mesma página HTML ou ASP pode conter vários FORMS disparando diferentes páginas ASP. Observação: a página que faz a chamada para uma outra página ASP não necessariamente precisa ser uma página ASP, pode ser um HTML. Exemplo:

Esta primeira página pode ser HTML:

```
<html>

<head>
<title>Untitled Normal Page</title>
</head>

<body bgcolor="#FFFFFF">

<form ACTION = "horas.asp" method="POST">
   Clique neste botão para saber as Horas:<P>
   <p><input type="submit" name="Horas" value="Horas"></p>
</form>
<BR>
<BR>
<form ACTION = "data.asp" method="POST">
   Clique neste botão para saber a Data:
   <p><input type="submit" name="Data" value="Data"></p>
</form>

</body>
</html>
```

Observe que no exemplo acima estamos utilizando dois formulários e cada um deles faz uma chamada para um ASP diferente. Vamos detalhar o código HTML:

```
<form ACTION = "horas.asp" method="POST">
   Clique neste botão para saber as Horas:<P>
   <p><input type="submit" name="Horas" value="Horas"></p>
</form>
```

Neste form, incluímos um botão do tipo Submit que deve disparar a ação do Form. A ação do Form (ACTION) está definido para chamar um ASP chamado horas.asp que se encontra no mesmo diretório do HTML. Quando o arquivo estiver em um diretório diferente do diretório da página que fez a chamada, é necessário que ele seja indicado.

O código abaixo faz a chamada a outro ASP, para isso definimos um outro formulário:

```
<form ACTION = "data.asp" method="POST">
   Clique neste botão para saber a Data:
   <p><input type="submit" name="Data" value="Data"></p>
</form>
```

Vamos ao ASP das Horas:

```
<html>

<head>
<title>Horas ... </title>
</head>

<body bgcolor="#FFFFFF">
São exatamente <%=Time %>.
</body>

</html>
```

Neste caso, a página obrigatoriamente deve ser nomeada com a extensão ASP pois contém código VBScript a ser interpretado pelo servidor. Dúvida: qual é o código a ser interpretado pelo servidor?

<%=Time %>

O ASP da Data:

```
<html>

<head>
<title>Data de Hoje ... </title>
</head>

<body bgcolor="#FFFFFF">
Hoje é <% = Date %>.
</body>
</html>
```

É o mesmo caso que o arquivo das Horas, também precisa ser nomeado com a extensão ASP, pois a linha **<% = Date %>** necessariamente será interpretada pelo servidor. O restante do código é HTML.

Resumo

Neste capítulo aprendemos que ...

- Podemos definir vários formulário em um arquivo HTML para disparar vários ASP.
- Um arquivo comum HTML pode disparar um ASP, porém um arquivo com códigos que devem ser interpretados pelo servidor, deve ser nomeado com a extensão ASP.
- Para disparar um ASP a partir de um formulário é preciso definir o parâmetro ACTION.
- Existem dois métodos diferentes para disparar um ASP: GET (mostra os dados no browser) e POST (não mostra os dados no browser).

CAPÍTULO IV – O QUE É VBSCRIPT?

Script

Antes de falarmos sobre o VBScript, é interessante esclarecermos o que é um script. As linguagens Scripts foram criadas para permitir a criação de aplicações para a Internet rapidamente. Se você comparar estas linguagens com as linguagens que costumamos utilizar, irá perceber que as primeiras são muitos mais fáceis e simples de aprender e desenvolver.
Os scripts nos permitem criar aplicações que desempenham funções como:

Alterar a linha de status do browser;
Definir um timeout para determinada procedure;
Rodar telas de alerta, confirmação e input;
"Forçar" a navegação para outras páginas;
Alterar cor de fundo, barra de títulos, etc.;
Criar uma nova página com textos;
Executar funções do browser como Back, Forward, Home, etc.;
Executar procedimentos quando funções do browser forem executadas.

Visual Basic Script Language (VBScript)

O **Visual Basic Script Language** é uma das muitas possibilidades de linguagem Script que rodam num servidor e, para o IIS, ela é a linguagem default (padrão). Desenvolver aplicações utilizando esta linguagem não é um bicho de sete cabeças. Vejamos algumas características da linguagem:

- É similar ao VBA (Visual Basic Application) linguagem criada para fornecer aos aplicativos outras funcionalidades e Visual Basic.
- Permite a manipulação de strings, datas, numéricos
- Permite a utilização de todos os comandos do Visual Basic, porém não permite a manipulação de banco de dados e acesso aos periféricos.
- A manipulação de banco de dados é feita através do objeto ADO, o VBScript apenas cria instâncias deste objeto, a partir daí, você pode utilizar os métodos deste objeto para manipular os dados.
- Tem um mecanismo de comunicação com servidores de objetos COM, como o Microsoft Exchange Server, Microsoft Index Server, Database Servers.

Baseada nas funcionalidades de programação do Visual Basic, é um linguagem leve que nativamente é executada pelo Internet Explorer (3.0 ou superor) e que pode ser executada por outros browsers a partir de pug-in.

Mas, como criar uma página ASP utilizando o VBScript?

Antes de mais nada, temos que ter em mente que alguma coisa tem que interpretar os códigos do VBScript. O responsável por esta interpretação é o IIS (Internet Information Server), por isso entendemos que o código deve ser interpretado no Servidor e não no cliente que só terá o browser.
Para que o código seja interpretado pelo Servidor, é preciso seguir algumas instruções:

1. **Utilizando a tag <% %>**
   Esta Tag deve ser utilizada quando você pretende executar qualquer código VBScript no servidor. Entre <% e %> é possível ser escrito qualquer código em VBScript.

Quando seu script possuir HTML e código VBScript o servidor saberá qual informação deverá ser retornada para o cliente (browser). Por exemplo:

```
<% if Hour(Now) < 12 then %>
        Bom dia!
<% else %>
        Boa tarde!
<% end if %>
```

No exemplo acima, as linhas que estiverem entre a tag <% %> serão interpretadas pelo Servidor IIS e dependendo da condição satisfeita, se a hora for menor que 12 será mostrado um HTML com “Bom dia!”, se não o HTML retornado pelo Servidor conterá “Boa Tarde!”.

Para retornar no HTML valores de variáveis diretamente do script, utilize a seguinte sintaxe:

```
A data de hoje é <% = Date %>.
```

Com a linha acima, o HTML a ser retornado será (imaginemos que o dia de hoje seja 20/08/1999):

A data de hoje é 20/08/1999.

2. **Utilizando a tag <SCRIPT>**
   Com esta tag também é possível criar scripts que “rodam” no servidor utilizando a sintaxe:

   <SCRIPT LANGUAGE = VBScript RUNAT=Server>

   </SCRIPT>

   É preciso especificar a linguagem script a ser utilizada e onde esta será interpretada. A partir disto, é só fazer a chamada de alguma parte do HTML. No caso acima, o script que estiver entre a tag <SCRIPT> </SCRIPT> deve ser escrito na linguagem VBScript e será interpretada por um servidor IIS. Esta tag é muito utilizada para criar funções e subrotinas. Por exemplo:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
        Function RetornaData()
                RetornaData = Date
        End Function
</SCRIPT>
```

Esta função retornará a data do sistema no Servidor. Para fazer a chamada desta função, será necessário utilizar as tags <% %>. Por exemplo:

A data de hoje é <% = RetornaData() %>.

Outra alternativa disponível no IIS para retornar valores direto para o HTML, sem utilizar a chamada acima, é usando o **Response.Write** (mais adiante iremos estudá-lo detalhamente):

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
        Function RetornaData()
                Response.Write "A data de hoje é " & Date & "."
                End Function
</SCRIPT>
```

O HTML retornado é (supondo que a data seja 20/08/1999):

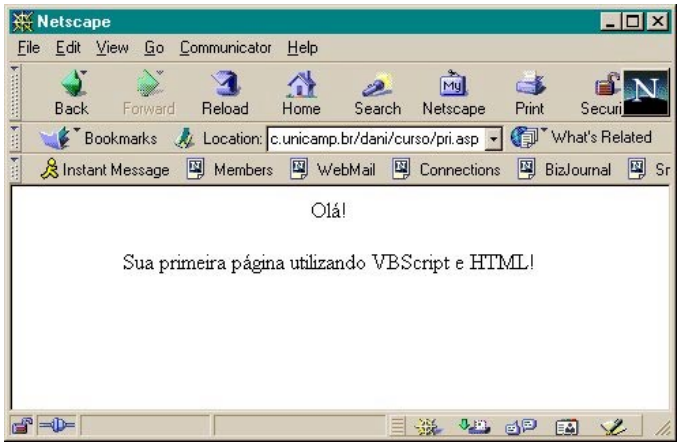
A data de hoje é 20/08/1999.

A pergunta imediata que você faria: como é possível retornar um HTML sendo que em momento algum foi feita a chamada para a função? Bem, quando é feito um pedido ao servidor pela execução de um ASP, o servidor executa todas as **funções** declaradas no ASP, então podemos concluir que esta função também foi executada. Como o Response.Write retorna diretamente para o HTML o conteúdo do texto, será mostrado para o usuário o texto definido.

O próximo passo para a criação de uma página ASP é introduzir o código HTML. Por exemplo:

```
<HTML>
<BODY>

<% if Hour(Now) < 12 then %>
        <CENTER> Bom dia! </CENTER><BR>
<% else %>
        <CENTER> Olá! </CENTER> <BR>
<% end if %>

<CENTER> Sua primeira página utilizando VBScript e HTML! </CENTER>

</BODY>
</HTML>
```

Este código inteiro constitui uma página ASP, na qual você mesclou código em HTML e VBScript. O Servidor saberá que o código entre <% %> será interpretado e não deve ser mostrado para o cliente, apenas seu resultado. Neste caso, a pagina a ser retornada para o browser será:



Lembrando que os arquivos com código VBScript devem ser nomeados com a extensão ASP, por exemplo: primeiro_programa.asp. Desta forma, o IIS entenderá que o pedido foi feito a partir de uma página ASP.

Se for utilizada a Tag <SCRITP> </SCRIPT>, o código VBScript e HTML ficará:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
        Function RetornaData()
                RetornaData = Date
        End Function
</SCRIPT>

<HTML>
<BODY>
<CENTER><B>A data de hoje é <% = RetornaData() %>.</B></CENTER>
</BODY>
</HTML>
```

O HTML retornado pelo servidor será:

Então, vimos como fazer para um script - utilizando o VBScript – "rodar" no servidor. Mas, e se você quiser fazer consistências de tela para o usuário e que estas estejam definidas na página corrente, ou seja, não haverá um pedido para o servidor, o código deverá ser executado no browser.

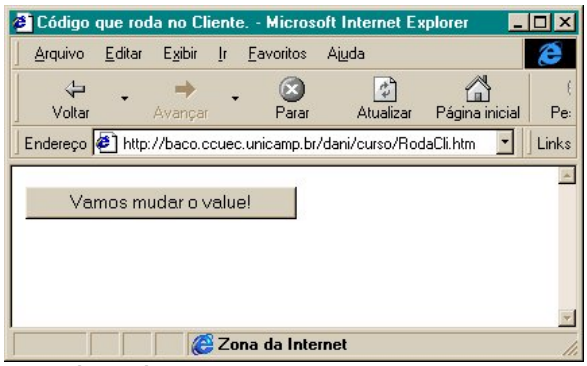
Com o VBScript é possível criar rotinas que sejam executadas pelo cliente. Porém, dois aspectos devem ser observados quando decidimos programar subrotinas e funções para rodar no browser:

- Em nosso caso, será necessário que o seu browser seja o Internet Explorer 4.0 ou superior.
- A programação deve ser baseada nos eventos dos objetos do formulário e do próprio formulário. Por exemplo: ONCLICK, CHANGE, GOTFOCUS.

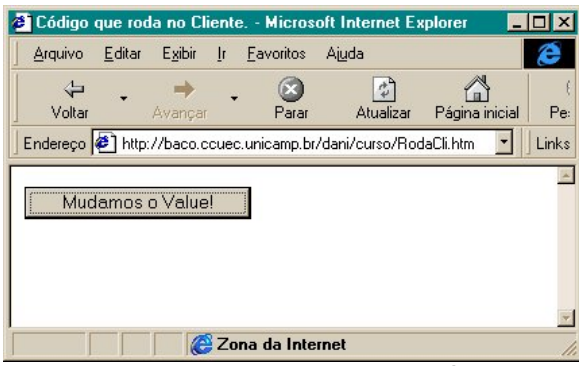
Exemplo de Código:

```
<html>

<head>
<title>Código que roda no Cliente.</title>
</head>

<script language="VBScript">
SUB BOTAO1_ONCLICK()
  RodarCli.BOTAO1.Value = "Mudamos o Value!"
END SUB
</script>

<body bgcolor="#FFFFFF">

<form NAME=RodarCli method="POST">
     <p><input type="button" name="BOTAO1" value="Vamos mudar o value!"></p>
</form>

</body>
</html>
```

O Resultado deste HTML:



Esta é a página aberta antes de clicar o botão "Vamos mudar o value!"



Depois de clicar o botão da página sua descrição é alterada para "Mudamos o Value"

O HTML:

```
<form NAME=RodarCli method="POST">
   <p><input type="button" name="BOTAO1" value="Vamos mudar o value!"></p>
</form>
```

Nesta parte, estamos montando o botão. Para que a subrotina seja executada, o botão deve ser criado em um form (formulários, mais adiante estaremos estudando de forma mais detalhada), deve ser definido um nome para este objeto e seu tipo deve ser button. Depois de definir o objeto que deve disparar a ação, você deve programar a subrotina:

```
<script language="VBScript">
SUB BOTAO1_ONCLICK()
  RodarCli.BOTAO1.Value = "Mudamos o Value!"
END SUB
</script>
```

Esta sub-rotina está sendo executada pelo browser, observe que não configuramos a Tag <SCRIPT> com o parâmetro RUNAT.

O evento que deve disparar a ação será o evento ONCLICK, ou seja, quando clicarmos no botão o código escrito será disparado.

Para fazer a referência ao objeto do formulário, você deve seguir a sintaxe abaixo:

NomeFormulário.NomedoObjeto.PropriedadedoObjeto

Por exemplo:

RodarCli.BOTAO1.Value

Resumo

Neste capítulo aprendemos que ...

- VBScript é a linguagem script default (padrão) do IIS (Internet Information Server).
- Quando o código deve "rodar" no servidor, é necessário que este esteja entre a Tag <% %> ou <SCRIPT RUNAT = Server> </SCRIPT>.
- Um script também pode rodar no cliente, basta apenas utilizar a tag <SCRIPT> </SCRIPT> indicando a linguagem a ser utilizada para que o browser possa interpretá-la.

Dicas

- É preciso prestar bastante atenção quando scripts são definidos para rodarem nos browsers, pois alguns scripts apenas são interpretados por determinado browser e versão. Por exemplo:

  <script language="javascript1.1"> - apenas será executado em Netscape 3.0+
  <script language="jscript"> - apenas será executado em Microsoft Internet Explorer 3.0+
  <script language="VBScript"> - apenas será executado em Microsoft Internet Explorer 4.0+

CAPÍTULO V – VARIÁVEIS

As variáveis são definidas quando precisamos armazenar temporariamente valores para que a aplicação tenha acesso às informações e consiga manipulá-las.

Tipo Suportado pelo VBScript

Enquanto o Visual Basic suporta muitos tipos de dados (variáveis), o VBScript apenas suporta o tipo **Variant**. Este tipo é único porque suporta todos os tipos suportados pelo Visual Basic: integer, double, string, date and currency. O tipo **Variant** assume o tipo de variável dependendo da atribuição feita a ele. Por exemplo, declaramos um variável chamada teste:

```
Dim teste        ' Esta variável é do tipo Variant, pois não tem a especificação do seu tipo:

Teste = "Acabamos de declarar a variável"        ' Teste na linha ao lado é do tipo String. E será tipo String até que haja outra atribuição de valores diferente de string.

Teste = 25       ' Teste nesta linha é do tipo numérico. E será tipo numérico até que haja outra atribuição de valores diferente de numérico.

Teste = "A variável recebeu 25"        ' Teste na linha acima volta a assumir o tipo String, pois o valor da atribuição foi texto.
```

SubTipos (Subtypes)

O tipo Variant consiste em pequenas unidades chamadas **Subtipos** (Subtypes). Estes subtipos identificam como um valor é armazenado pela variável definida como Variant. A tabela abaixo descreve os subtipos utilizados no VBScript 2.0:

SubTipo	Descrição
Empty	Valor é 0 (zero) para variáveis numéricas e comprimento zero (" ") para variáveis string.
Null	Variant possui um valor que não é válido.
Boolean	Verdadeiro ou Falso.
Byte	Inteiro com valores entre 0 a 255.
Integer	Inteiro com valores entre –32,768 até 32,767.
Currency	Tipo moeda. Valores entre 922,337,203,685,477.5808 até 922,337,203,685,477.5807.
Long	Inteiro com valores entre –2,147,483,648 até 2,147,483,647.
Single	Ponto Flutuante com precisão simples. Valores entre: -3.402823E38 até -1.401298E-45 para números negativos; 1.401298E-45 até 3.402823E38 para números positivos.
Double	Ponto Flutuante com dupla precisão. Valores entre: -1.79769313486232E308 até -4.94065645841247E-324 para valores negativos; 4.94065645841247E-324 até 1.79769313486232E308 para valores positivos.
Date (Time)	Contem um número que representa uma data entre 01/01/100 até 31/12/9999.
String	Contem um comprimento variável de texto que pode chegar até 2 bilhões de caracteres.
Object	Contem um objeto

Error	Contem um número de erro gerado por um objeto.

Exemplo do comportamento de uma variável do tipo **Variant**:

```
<% Dim teste
   Dim Mensagem

      teste = 25

      if teste = 30 then
         Mensagem = "O valor de Teste é igual a 30"
      Else
         Mensagem = "O valor de Teste é igual a 25"
      End if
%>
```

‘ Declaramos duas variáveis do tipo Variant, onde Teste assume um valor inteiro e Mensagem assume o valor string.

Funções de Conversão de Tipos de Dados

Antes de descrever as funções, precisamos saber para que serve uma função de conversão de tipo. Estas funções são utilizadas para “forçar” uma variável **Variant** assumir um subtipo específico.

A partir do momento que as páginas ASP começaram a ser distribuídas pela Internet, ou seja, pessoas do mundo inteiro acessando sua página, as configurações de datas, moeda passaram a ser um fator preocupante. Pois estas configurações são recuperadas da opção Configurações Regionais do Painel de Controle da máquina onde o VBScript está sendo executado. Como o VBScript, na maioria das vezes, é executado em um servidor, as configurações recuperadas são as configurações feitas no próprio servidor.

As funções de conversão de dados foram criadas para evitar erros em tempo de execução ou mesmo de lógica por sua aplicação não reconhecer o tipo de dados pelas configurações. Estas funções conseguem reconhecer as configurações regionais e interagir com vários formatos diferentes de dados sem mudá-los, evitando que erros ocorram.

A tabela abaixo descreve as funções que permitem a conversão de dados:

Função	Valor Retornado	Descrição
Cbool	Boolean	Verdadeiro/Falso
Cbyte	Byte	Valor numérico de 0 até 255
Ccur	Currency	Valores entre 922,337,203,685,477.5808 até 922,337,203,685,477.5807.
Cdate	Date	Qualquer data válida
Cdbl	Double	-1.79769313486232E308 até -4.94065645841247E-324 para valores negativos; 4.94065645841247E-324 até 1.79769313486232E308 para valores positivos.
Cint	Integer	–32,768 até 32,767. Frações são arredondadas.
CLng	Long	–2,147,483,648 até 2,147,483,647. Frações são arredondadas.
CSng	Single	-3.402823E38 até -1.401298E-45 para números negativos; 1.401298E-45 até 3.402823E38 para números positivos
CStr	String	Retorna texto, exceto Null

Identificando o Tipo de Dados

Como o VBScript consegue identificar uma série de subtipos de variáveis, será necessário, às vezes, que você saiba qual o subtipo com o qual você está trabalhando no seu script. Para isso, usaremos a função VarType que identifica o subtipo por um valor numérico. A tabela abaixo descreve os valores retornados pela função:

Subtipo	Valor Retornado
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date (Time)	7
String	8
Automation Object	9
Error	10
Boolean	11
Variant	12
Non-Automation Object	13
Byte	17
Array	8192

Exemplo

```
<%    Dim teste, Mensagem, ResultMedia

     Function Media(Valor)
          Dim Retorna, cont
          Retorna = 1
          For cont = 1 to Valor
              Retorna = Retorna * Cont
          Next
          Media = Retorna
    End Function

    teste = 5
    if teste = 5 then
      Mensagem = "O valor da variável Teste é igual a 5"
    End if
    ResultMedia = Media(teste)
%>
<HTML><BODY><center><b>
<FONT COLOR="#008000">O tipo que variável <i>Teste</i> assumiu foi: <% = VarType(teste) %></FONT><BR>

<FONT COLOR="#0000C0">O tipo que variável <i>Mensagem</i> assumiu foi: <% = VarType(Mensagem)
%></FONT><BR>

<FONT COLOR="#400040">O tipo que variável <i>ResultMedia</i> assumiu foi: <% = VarType(ResultMedia)
%></FONT><BR>

<FONT COLOR="#0000FF">As variáveis Retorna e Cont não podem ser referenciadas, pois existem apenas
quando a Função Media está sendo executada.</FONT>
</b></center></BODY></HTML>
```

Para verificar o subtipo de dados que a variável ***teste*** assumiu, utilize o código script:
<% = VarType(teste) %> que já retornará o valor numérico correspondente ao subtipo assumido. No caso da variável teste, o valor retornado é ***2***, que indica o subtipo ***Integer***.

Não podemos referenciar as variáveis ***cont*** e ***Retorna,*** pois estas variáveis existem apenas para a função, para o restante do script são inválidas.

O resultado retornado pelo servidor:

Declaração de Variáveis

Quando declaramos variáveis, precisamos ter em mente a sua utilização: se será uma variável que deverá ser utilizada por toda uma aplicação ou apenas por um módulo, uma função, subrotina. Este "período de vida" da variável é flexível, ou seja, você pode definir onde, quando e por quanto tempo esta variável deverá existir.

Escopo das variáveis

As variáveis no VBScript podem existir em dois níveis ou escopos: script ou procedimento. O termo escopo está relacionado ao espaço de tempo no qual uma memória pode ser referenciada na memória.
O espaço de tempo, que se refere ao "tempo de vida" da variável, depende em qual nível esta variável foi declarada:

Script

Quando uma variável é definida fora de qualquer função e procedimento, é caracterizada uma variável válida para todo o script.

Procedimento

Quando você verificar que uma variável precisa ser apenas válida para uma função ou procedimento, declare-a dentro deste procedimento ou função. Desta maneira, assim que a função ou procedimento for executado, a variável deixa de existir na memória.

Exemplo (o que é importante até este ponto que estudamos, está em **negrito**)

```
<%    Dim teste
      Dim Mensagem
      Dim ResultMedia

    Function Media(Valor)
        Dim Retorna
        Dim cont
        Retorna = 1
        For cont = 1 to Valor
            Retorna = Retorna + Cont
        Next
        Retorna = Retorna / cont
        Media = Retorna
    End Function

    teste = 5
    if teste = 30 then
      Mensagem = "O valor da variável Teste é igual a 30"
    Else
      Mensagem = "O valor da variável Teste é igual a 5"
    End if
    ResultMedia = Media(teste)%>

<HTML><BODY>
<CENTER><B><% = mensagem %><BR><BR>
A média ponderada dos valores da variável Teste é igual a: <% = ResultMedia %>
</B></CENTER>
</BODY></HTML>
```

O resultado deste ASP será a página:

Explicação do código

<% Dim teste
Dim Mensagem

Dim ResultMedia

No exemplo que criamos, as variáveis acima foram declaradas fora de qualquer função ou subrotina, desta forma, podem ser utilizadas por todo o script escrito para esta página o que significa que em qualquer ponto do script podem ser referenciadas.

Function Media(Valor)
Dim Retorna
Dim cont
Retorna = 1

Diferente das variáveis ***teste***, ***Mensagem*** e ***ResultMedia***, as variáveis ***Retorna*** e ***cont*** podem apenas ser referenciadas dentro da função. Para o restante do scritp, as variáveis Retorna e cont não são válidas.

Até este ponto, vimos os tipos de variáveis, funções de conversão e seu escopo ("tempo de vida"). Mas, como declará-las em um script?

Para declarar variáveis em seu script, você deve utilizar as cláusulas: **Dim**, **Public**, **Static**.

Dim

Variáveis declaradas com o Dim em um script estão disponíveis para todas os procedimentos inclusos neste script, porém as variáveis declaradas em um procedimento apenas poderão ser utilizadas neste procedimento.

Exemplo

```
Dim nomevar [([dimensão])]

    Para declarar diversas variáveis, utilize vírgulas para separá-las. Por exemplo:

Dim nomevar [([dimensão])], nomevar2
```

Onde ***nomevar*** e ***nomevar2*** são nomes de variáveis e ***dimensão*** representa o número da dimensão do vetor.

Com a cláusula Dim, é também possível declarar vetores. Um vetor é tratado como uma variável, porém é utilizado para armazenar uma coleção de informações similares. Esta coleção é acessada por seu índice, começando pelo 0 (zero). Por exemplo, o próximo código cria um vetor com 5 posições. Como os vetores começam com a posição 0 (zero), 5 posições da variável TipoProduto são alocadas. A segunda e terceira linha do código preenchem a primeira e segunda posições do vetor com os valores "Material de Construção" e "Material Escolar":

```
Dim TipoProduto(4)      'Como o vetor começa com 0 (zero), a primeira posição é 0 e a quinta posição é 4
TipoProduto(0) = "Material de Construção"
TipoProduto(1) = "Material Escolar"
```

ReDim

A função ReDim é utilizada para manipular vetores dinâmicos. Para o VBScript, os vetores podem ser definidos em dois tipos: Estáticos ou Dinâmicos. Um vetor estático tem seu número de índices definidos, como o vetor que definimos no exemplo acima que possui 5 posições. Um vetor

dinâmico tem o número de índices variado e este número é definido quando a aplicação é executada. Você pode criar um vetor dinâmico, apenas declarando-o com os parênteses vazios.

Os vetores dinâmicos são diferentes dos estáticos. Os dinâmicos são utilizados quando a alocação de memória precisar ser dinâmica, ou seja, o vetor pode assumir vários tamanhos. Para que isso seja possível, utilize o comando ReDim. Este comando é utilizado para redefinir o tamanho do vetor. Este tamanho pode aumentar ou diminuir. Se você precisar aumentar o tamanho de seu vetor e precisa manter os elementos já inclusos nas posições existentes, utilize o comando Preserve. Por exemplo, o código a seguir cria um vetor dinâmico chamado ProdutosSelecionados e o redimenciona com 3 posições. A última linha expande o número de elementos do vetor para 6, como estamos utilizando a cláusula Preserve, o conteúdo já incluso no vetor permanecerá.

```
Dim ProdutosSelecionados()
ReDim ProdutosSelecionados(3)
ProdutosSelecionados(0) = “Caderno”
Redim Preserve ProdutosSelecionados(6)
```

Public

Quando declaramos uma variável como pública, esta variável pode ser referenciada por todo o script da página. Você também pode declarar vetores estáticos ou dinâmicos com a cláusula **Public**.

Sintaxe

Public nomevar[([dimensão])]

Onde **nomevar** é o nome da variável e **dimensão** é o número de elementos do vetor.

Private

Variáveis declaradas com a cláusula **Private** são o oposto das variáveis declaradas com a cláusula Public. São válidas apenas dentro do script no qual foram declaradas. Podemos também declarar vetores dinâmicos e estáticos com o **Private**.

Sintaxe

Private nomevar[([dimensão])]

Onde **nomevar** é o nome da variável e **dimensão** é o número de elementos do vetor.

Declaração	Escopo
Dim	Script todo ou Procedimento
ReDim	Script todo ou Procedimento
Public	Script todo
Private	Script todo

Regras para nomear as variáveis

Para nomear suas variáveis, é preciso seguir algumas regras:

- Deve começar com um caracter alfabético;
- Não pode exceder o tamanho de 255 caracteres;
- Deve ser único no escopo no qual a variável é declarada.

Option Explicit

Por default (padrão), a declaração das variáveis não é obrigatória. Você pode apenas fazer referências a elas, sem declará-las explicitamente. A declaração explicita se refere ao processo de criação ou instanciação de uma variável

Exemplo

```
<%
    teste = 5
    if teste = 30 then
      Mensagem = "O valor da variável Teste é igual a 30"
    Else
      Mensagem = "O valor da variável Teste é igual a 5"
    End if
%>

<HTML><BODY>
<CENTER><B><% = Mensagem %><BR><BR>
</B></CENTER>
</BODY></HTML>
```

Onde ***teste*** e ***Mensagem*** são variáveis declaradas implicitamente, o VBScript se encarrega de entendê-las como variáveis.

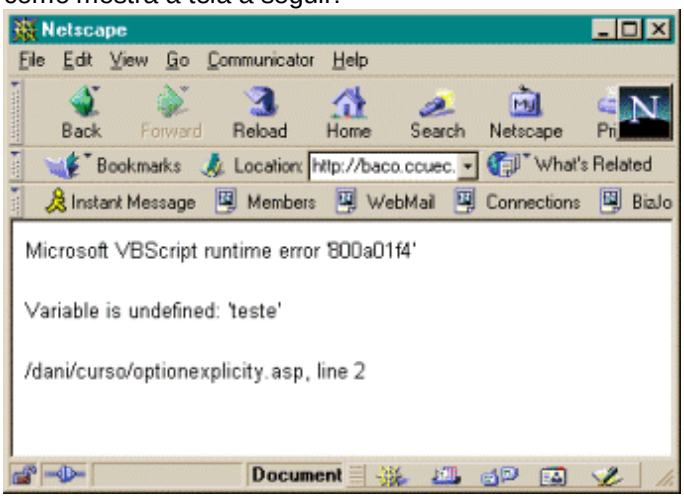
Esta declaração implícita não é considerada uma boa prática pelo programadores, pois estamos sujeitos a erros de digitação, e para uma este tipo de declaração, declarar uma variável chamada teste e outra tsete são consideradas variáveis diferentes. Para evitar este tipo de erro, a declaração do Option Explicit está disponível para que obrigatoriamente todas as variáveis sejam declaradas explicitamente.

Se você optar por utilizá-lo, a sua declaração deve ser feita antes de qualquer outra declaração.

Exemplo (utilizando o exemplo acima)

```
<% Option Explicit

    teste = 5
    if teste = 30 then
      Mensagem = "O valor da variável Teste é igual a 30"
    Else
      Mensagem = "O valor da variável Teste é igual a 5"
    End if
%>
<HTML><BODY>
<CENTER><B><% = Mensagem %><BR><BR>
</B></CENTER>
</BODY></HTML>
```

Observe neste exemplo que declaramos a clausula ***Option Explicit*** porém, não declaramos explicitamente a variável ***teste***. Ao executarmos este ASP no servidor, será retornado um erro como mostra a tela a seguir:



A mensagem do erro retornado ***Variable is undefined: 'teste'*** significa que a variável não foi declarada. Se voltarmos ao script, perceberemos que realmente a variável não foi declarada. Vamos estudar o script:

<% Option Explicit

teste = 5
.....
%>

Como utilizamos a cláusula ***Option Explicit***, necessariamente precisamos declarar a variável ***teste*** explicitamente, para isso, devemos escrever o script da seguinte maneira:

<% Option Explicit

Dim teste

teste = 5

```
    if teste = 30 then
      Mensagem = "O valor da variável Teste é igual a 30"
    Else
      Mensagem = "O valor da variável Teste é igual a 5"
    End if
%>
```

Se o ASP for executado novamente, não ocasionará o erro, pois a variável foi declarada.

Resumo

Neste capítulo aprendemos que ...

- Existe apenas um tipo de dados suportado pelo VBScritp: **Variant**.
- Embora haja apenas um tipo de dados, existem subtipos que identificaram o valor que é armazenado na variável, atribuindo-lhe o subtipo específico de acordo com o valor. Se o valor for numérico, o subtipo será integer, por exemplo.
- Com a função VarType, é possível identificar o subtipo assumido pela variável.
- Para o ASP, as variável possuem dois escopos: **Script** e **Procedimento**. Quando declaramos uma variável em nível Script, é possível referenciá-la em todo o script. Quando declaramos em um Procedimento, a variável é apenas válida para o procedimento em questão.
- Utilizando a cláusula Option Explicit, fica obrigatória a declaração explicita da variável.

Dicas

- Utilize a declaração Option Explicit para evitar erros de digitação do nome de uma variável existente ou para evitar confusão no código script onde o escopo das variáveis não é bem definido.
- Quando você utiliza a cláusula Dim em um procedimento, geralmente a declaração é feita no começo do procedimento.
- Antes de atribuir valores a um vetor dinâmico, é preciso redimenciona-lo usando o comando ReDim
- Quando uma variável é declarada, seu valor padrão depende do subtipo de dados que assumir. Por exemplo, uma variável numérica é inicializada com 0 (zero), uma variável string é inicializada com comprimento igual a zero (“ “).

CAPÍTULO VI – OPERADORES

Operadores

O VBScript possui vários operadores de programação que ajudam a manipular as variáveis em suas páginas ASP. Muitas linguagens dividem os modelos de objetos em grupos lógicos agrupados de acordo com suas funcionalidades. Isto dá maior flexibilidade para manipular objetos e variáveis dos formulários e é fundamental para o controle do processo de uma aplicação.

Operadores Aritméticos

Os operadores aritméticos nos permite fazer uma série de cálculos matemáticos. Geralmente, possuem a seguinte sintaxe:

Resultado = valor1 operador valor2

Onde **Resultado** é a variável que receberá o valor do cálculo, **valor1** e **valor2** são expressões numéricas e o **operador** representa a operação matemática a ser realizada.

A tabela a seguir lista todos os operadores reconhecidos pelo VBScript:

Operador	Símbolo	Descrição
Adição	+	Efetua a soma entre os valores.
Subtração	-	Efetua a subtração entre dois números ou atribui um valor negativo a número.
Multiplicação	*	Efetua a multiplicação entre os valores.
Divisão	/	Efetua a divisão entre dois números.
Divisão Inteira	\	Efetua a divisão entre dois números e retorna um valor inteiro como resultado.
Exponenciação	^	Efetua a operação de potenciação.

Adição

O operador de adição (+) é utilizado para efetuar somas entre valores numéricos, mas também pode ser utilizado para concatenar strings.

Sintaxe

Resultado = valor1 + valor2

Onde **resultado** recebe o valor da soma das expressões numéricas **valor1** e **valor2**. Lembre-se, existe apenas um tipo de variável no VBScript: **Variant**. Somar dois valores pode resultar em um resultado não esperado, por causa da capacidade do tipo Variant mudar automaticamente o subtipo baseado no valor atribuído à variável.

A tabela abaixo, nos mostra o compotamento do operador +, dependendo do tipo de dados:

Se os valores a serem somados forem:	O comportamento do operador será:
Numéricos	Adição.
Strings	Concatenação
String e Numérico	Adição

Vamos considerar uma situação onde a adição entre dois números é pedida. Siga o exemplo:

```
<% Dim valor1, valor2
    Dim resultado

     valor1 = 1
     valor2 = "3"

     Resultado = valor1 + valor2        'Retorna o resultado igual a 4
     Resultado = valor2 + valor2        'Retorna o resultado igual a 33
     Resultado = valor2 + valor1        'Retorna o resultado igual a 4
     Resultado = valor1 + valor1        'Retorna o resultado igual a 2
%>
```

Você pode perceber por este exemplo que sem um entendimento apropriado sobre o tipo de dados Variant e o comportamento do operador de adição, o resultado pode não ser o esperado. No exemplo acima, as variáveis são explicitamente declaradas e valor1 recebe o 1 (numérico), assumindo o subtipo Integer, e a variável valor2 recebe "3" (string), assumindo o subtipo String. Todas as variáveis declaradas são do tipo Variant. Entretanto, o VBScript atribui um subtipo dependendo do valor atribuído a variável.

Nas linhas abaixo, a variável resultado receberá o valor 4. Pois, pela regra, quando somamos um valor numérico e uma string, a soma numérica entre eles é efetuada.
Resultado = valor1 + valor2
Resultado = valor2 + valor1

Já nesta linha, o resultado é diferente. Pois os valores participantes da adição são valores numéricos, então será efetuada a concatenação das strings. A variável resultado conterá o valor (string) "33".
Resultado = valor2 + valor2

Na próxima linha, estamos somando dois valores numéricos, pois o subtipo da variável valor1 é Integer. O valor retornado para a variável resultado será 2.
Resultado = valor1 + valor1

Mas, e se você precisasse que o resultado da linha ***Resultado = valor2 + valor2*** fosse numérico? Para isso, utilize as funções de conversão de tipo. Esta linha poderia ser reescrita da seguinte maneira:

Resultado = CInt(valor2) + CInt(valor2)

A função CInt é utilizada para converter uma variável para Inteiro. E a variável resultado receberia o valor numérico 6.

O mesmo vale para a linha ***Resultado = valor1 + valor1***. Se o resultado a ser retornado precisasse ser uma concatenação de strings e não uma soma numérica, a linha seria reescrita da seguinte maneira:

Resultado = CStr(valor1) + CStr(valor1)

A função CStr é utilizada para converter uma variável para String. E a variável resultado receberia a string 22.

Subtração

O operador de subtração (-) é utilizado para efetuar a diferença entre valores numéricos ou para converter um valor numérico positivo em um valor numérico negativo. Para cada uma destas funcionalidades, é preciso uma sintaxe diferente, vejamos quais são:

Resultado = valor1 – valor2

Onde **resultado** recebe o valor da operação entre as variáveis numéricas **valor1** e **valor2**.

Resultado = -(valor1)

Onde **valor1** é uma valor numérico.

Multiplicação

O operador de multiplicação (*) é utilizado para retornar o resultado da multiplicação entre valores numéricos.

Sintaxe

Resultado = valor1 * valor2

Onde **resultado** recebe o valor da operação entre as variáveis numéricas **valor1** e **valor2**. Se ambas as variáveis estiverem com o valor igual a NULL, o valor retornado da operação também será NULL. Se estiverem vazias, o resultado será convertido para 0 (zero).

Divisão

O operador de divisão (/) é utilizado para dividir uma expressão numérica em outra expressão numérica.

Sintaxe

Resultado = valor1 / valor2

Onde **Resultado** recebe o valor da operação entre as variáveis numéricas **valor1** e **valor2**. As mesmas regras da multiplicação valem para a divisão. Se ambas as variáveis estiverem com o valor igual a NULL, o valor retornado da operação também será NULL. Se estiverem vazias, o resultado será convertido para 0 (zero).

Fique atento para a divisão por 0 (zero). Será o retornado o seguinte erro:

Microsoft VBScript runtime error '800a000b'

Division by zero

Divisão Inteira

O operador da divisão inteira (\) é similar ao da divisão, mas foi especificamente, criado para retornar resultados inteiros apenas.

Sintaxe

Resultado = valor1 \ valor2

Onde **Resultado** recebe o valor da operação entre as variáveis numéricas **valor1** e **valor2**. Estas variáveis são arredondadas e seu subtipo configurados para Byte, Integer ou Long antes da operação. Se ambas as variáveis estiverem com o valor igual a NULL, o valor retornado da operação também será NULL. Se estiverem vazias, o resultado será convertido para 0 (zero).

Exponenciação

O operador de exponenciação (^) é utilizado para "elevar" um número a outro.

Sintaxe

resultado = número1^número2

Onde **resultado** recebe o valor retornado da operação efetuada pelas expressões numéricas **número1** e **número2**.

Operadores de Comparação

Os operadores de Comparação analisam o relacionamento entre expressões. Lembre-se que uma expressão pode ser uma combinação de palavras-chaves, operadores, variáveis ou constantes que formam uma string, número ou objeto.

Sintaxe geral

Resultado = Expressão1 (operador) Expressão2

Onde **Expressão1** e **Expressão2** contem uma expressão qualquer e o **operador** representa o símbolo de relacionamento entre as expressões (o tipo de comparação).

A tabela abaixo nos mostra os Operadores de comparação utilizados no VBScript:

Operador	Símbolo
Igualdade	=
Desigualdade	<>
Maior que	>
Menor que	<
Maior que ou Igual a	>=
Menor que ou Igual a	<=
Equivalência de Objetos	IS

Operadores de Comparação dá a você a habilidade para determinar a relação entre expressões. Tradicionalmente, os desenvolvedores esperam que as comparações sejam avaliadas como True (verdadeiro) ou False (falso). Porém, é importante considerar uma terceira possibilidade: NULL. Se a expressão na comparação for NULL, a comparação retornará NULL. È importante manter isto em mente, quando desenvolvemos aplicações lógicas. Os desenvolvedores devem desenvolver uma aplicação que possa responder as três possibilidades de comparação: true, false, NULL.

A tabela abaixo nos mostra uma série de exemplos de comparações:

Operador	Retorna True se ...	Retorna False se ...	Retorna NULL se ...
<	Expr1 < Expr2	Expr1 >= Expr2	Expr1 ou Expr2 = Null
<=	Expr1 <= Expr2	Expr1 > Expr2	Expr1 ou Expr2 = Null
>	Expr1 > Expr2	Expr1 <= Expr2	Expr1 ou Expr2 = Null
>=	Expr1 >= Expr2	Expr1 < Expr2	Expr1 ou Expr2 = Null
=	Expr1 = Expr2	Expr1 <> Expr2	Expr1 ou Expr2 = Null
<>	Expr1 <> Expr2	Expr1 = Expr2	Expr1 ou Expr2 = Null

Os operadores de comparação atuam como os operadores aritméticos, pois alteram seu comportamento dependendo do tipo de dados das variáveis envolvidas. Na tabela baixo temos uma descrição do comportamento dos operadores de comparação:

Expressões ...	Comportamento
Numéricas	Comparação Numérica
String	Comparação String
Uma numérica e outra String	A expressão numérica é menor que a expressão string.
Uma é vazia e outra é numérica	Comparação numérica sendo que a expressão vazia é tratada como zero.
Uma é vazia e outra é string	Comparação string, sendo que a expressão vazia é tratada com comprimento zero (“ “).
Ambas as expressões são vazias	As expressões são iguais.

Outro tipo de comparação é a comparação entre objetos. O operador IS é utilizado para fazer esta comparação. A sintaxe a ser utilização está descrita abaixo:

Resultado = objeto1 IS objeto2

Onde **Objeto1** e **Objeto2** representam objetos e o uso da palavra-chave **IS** determina um relacionamento entre os objetos.

Quando comparamos objetos, não existe a possibilidade de NULL. O resultado sempre será True ou False. Se os objetos forem iguais, o resultado será igual a True. Se forem diferentes, o resultado retornado será False.

Exemplo

```
<%
        Dim Tobjeto, TSObjeto, Resultado

        Set Pobjeto = Tobjeto                   ‘Cria o objeto
        Set Sobjeto = Tobjeto                   ‘Cria o objeto
        Set TerObjeto = TSObjeto                ‘Cria o Objeto

        Resultado = Pobjeto is Sobjeto          ‘Compara
        Resultado = Pobjeto is TerObjeto        ‘ os Objetos
%>
```

Na primeira linha de comparação (***Resultado = Pobjeto is Sobjeto***), o valor retornado é True, pois os dois objeto (Pobjeto e Sobjeto) foram criados a partir de um mesmo objeto.

Já na segunda linha de comparação (***Resultado = Pobjeto is TerObjeto***), o valor retornado é False, pois os objetos têm origens diferentes, cada um deles foi criado a partir de objetos diferentes.

Operadores Lógicos

Os operadores lógicos ajudam a expandir as funcionalidades dos operadores de comparação. O operador lógico foca o método booleano para determinar se um processo retorna True ou False.

Os operadores lógicos mais utilizados estão listados na tabela abaixo:

Operador	Símbolo
Conjunção	AND
Negação	NOT
Desconjunção	OR

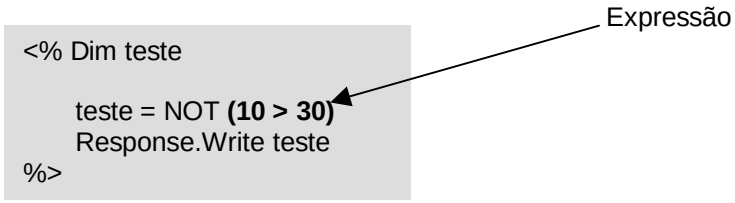
Operador NOT

Este operador é utilizado para realizar negação lógica em uma expressão, ou seja, para negar ou inverter um resultado.

Sintaxe

Resultado = NOT (Expressão)

Onde **Expressão** é qualquer expressão. Por exemplo:




```
<% Dim teste

     teste = NOT (10 > 30)
     Response.Write teste
%>
```

Resultados retornado pelo operador NOT:

Expressão	Resultado
True	False
False	True
NULL	NULL

Lembre-se que uma variável que contém o valor NULL, representa uma variável com valor inválido. Consequentemente, qualquer operador reproduzirá um resultado NULL. Tenha em mente que um variável com uma string vazia (“ “), não é igual a NULL. Por exemplo:

```
<%
   Variavel = NULL
   Resultado = IsNull(Variavel)        ‘ Retornará True, pois o conteúdo de
                                       variavel é igual a NULL.
%>
```

```
<%
     Variavel = “”
     Resultado = IsNull(Variavel)      ‘ Retornará False, pois o conteúdo de
                                       variável é uma string vazia.
%>
```

* A função IsNull verifica se o conteúdo de uma variável é igual a NULL.

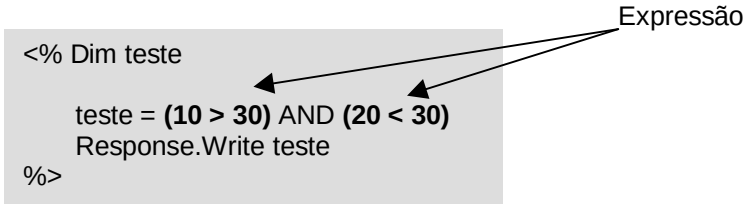
Operador AND

É utilizado para avaliar um conjunto de expressões como se fosse apenas uma comparação. O operador AND apenas retornará True, se e somente se todas as comparações forem verdadeiras.

Sintaxe

Resultado = Expressão1 AND Expressão2

Onde Expressão1 e Expressão2 são quaisquer expressão. Por exemplo:



Neste caso, teste receberá o valor False. Pois embora 20 seja menor que 30, 10 não é maior que 30. Então, como o AND apenas retorna True se todas as expressões forem verdadeiras, a variável teste receberá false.

A tabela abaixo nos lista o comportamento do operador AND:

Expressão1	Expressão2	Resultado
True	True	True
True	False	False
True	NULL	NULL
False	True	False
False	False	False
False	NULL	False
NULL	True	NULL
NULL	False	False
NULL	NULL	NULL

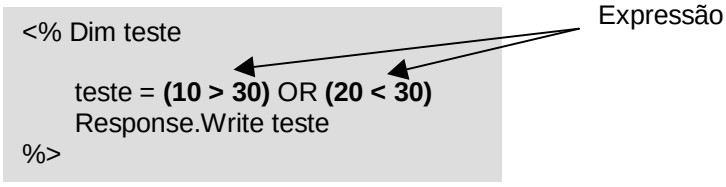
Operador OR

Avalia se alguma expressão de uma série de expressões é verdadeira.

Sintaxe

Resultado = Expressão1 OR Expressão2

Onde **Expressão1** e **Expressão2** são quaisquer expressão. Por exemplo:



Neste caso, a variável teste receberá True. Pois uma das expressões acima é verdadeira: 20 < 30.

A tabela abaixo descreve o comportamento do operador OR, dependendo do resultado da expressão:

Expressão1	Expressão2	Resultados
True	True	True
True	False	True
True	NULL	True
False	True	True
False	False	False
False	NULL	NULL
NULL	True	True
NULL	False	NULL
NULL	NULL	NULL

Operadores de Concatenação

Existem dois operadores no VBScript utilizados para “somar” (concatenar) strings. Veja a tabela abaixo:

Operador	Símbolo	Descrição
Concatenação de String	&	Usado para concatenar Strings.
Operador de Adição e String	+	Usado para concatenar Strings e Adição Numérica.

Operador &

Utilizado para somar duas cadeias de caracteres (strings).

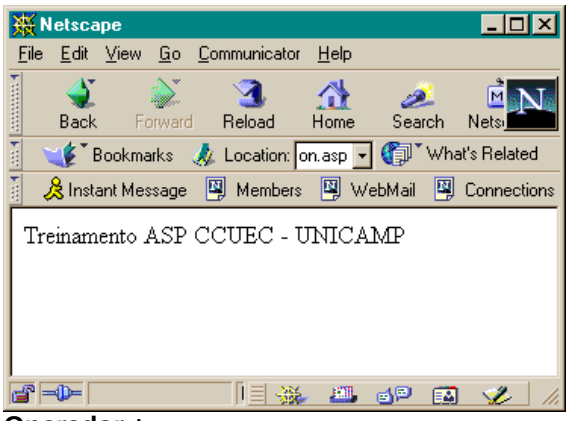
Sintaxe

resultado = string1 & string2

Onde **string1** e **string2** são expressões strings válidas para o VBScript.

```
<%
texto = "Treinamento ASP " & "CCUEC – UNICAMP"
Response.Write texto
%>
```

O resultado retornado:



Operador +

O operador de Adição também pode ser utilizado para concatenar strings. Porém, não é recomendado que você utilize este operador para este tipo de função, pois sua função principal seria a soma de valores numéricos. A ambiguidade ocorre quando o operador + é utilizado em diferentes subtipos, no caso, string e valores numéricos.

Sintaxe

Resultado = string1 + string2

Onde **string1** e **string2** são expressões do subtipo string válidas para o Vbscript. Por exemplo:

```
<%
texto = "Treinamento ASP " + "CCUEC – UNICAMP"
Response.Write texto
%>
```

O resultado retornado é o mesmo que foi retornado quando utilizamos o operador &.

Ordem de Avaliação das Expressões

A avaliação das expressões é baseada nas categorias dos operadores. Operadores aritméticos são os primeiros a serem processados, seguidos pelos operadores de comparação e por fim, pelos operadores lógicos.

Resumo

Neste capítulo aprendemos que ...

- Os operadores aritméticos aceitos pelo VBScript são: + (Adição), - (Subtração), * (Multiplicação), / (divisão), \ (divisão inteira) e ^ (Exponenciação).
- Os operadores de comparação são: = (igualdade), <> (desigualdade), > (maior que), < (menor que), >= (maior ou igual a), <= (menor ou igual a) e IS para comparação de objetos.
- Os operadores lógicos utilizam o método booleano para determinar se um processo retorna True ou False. Os símbolos dos operadores lógicos são: AND, OR, NOT.

Dicas

- Para prevenir uma concatenação de strings indesejável quando estiver utilizando o operador de adição, utilize as funções de conversão para certificar o tipo correto de dados.
- As concatenações são avaliadas depois dos operadores aritméticos e antes dos operadores de comparação.

CAPÍTULO VII – COMANDOS BÁSICOS DO VBSCRIPT

Neste capítulo, estaremos estudando os comandos para controle do fluxo de uma aplicação. Veremos as estruturas lógicas existentes no VBScript que devem permitir este controle: estruturas lógicas de decisão e repetição. Estruturas lógicas de decisão executam uma determinada condição e, dependendo do teste, retornam o HTML apropriado como resultado. A estrutura de repetição por sua vez, é um processo cíclico, ou seja, repete um determinado conjunto de instruções até que uma condição seja satisfeita.

Estruturas de Decisão

Utilizadas para tomar decisões em sua aplicação. Por exemplo, se a hora for menor que 12:00, retorne um HTML com "Bom dia", se não, retorne um HTML com "Ola!". Neste exemplo, você apenas mandou retornar um HTML com um texto para seu usuário porém, você poderia determinar que dependendo do horário, deve ser retornado determinada página, desviar para outra URL, etc. Estruturas como esta são utilizadas para determinar o fluxo da sua aplicação, com elas você pode definir qual página deve ser aberta, determinar o texto a ser escrito para o cliente, estados dos objetos na tela: selecionados, preenchidos, etc.

O primeiro comando que veremos será o **IF**. Este comando é utilizado para testar uma condição como verdadeira ou falsa. E dependendo dos resultados, determinado bloco de instruções será executado.

Sintaxe

```
If <condição> Then
    Bloco de Instrução (que pode ser apenas uma ou várias)
[Else
    Bloco de Instrução ]
End If
```

Onde ***<condição>*** é a expressão a ser testada. Se for verdadeira, o bloco de instrução que estiver definido depois do ***Then*** será executado. O ***Else*** representa o *"se não"*, ou seja, se a ***<condição>*** não for satisfeita (não for verdadeira), o bloco de instruções definido depois do ***Else*** será executado. ***End if*** termina o bloco **IF**.

Observe que a **<condição>** deve ser uma expressão que pode ser avaliada como true/false (verdadeiro/falso). Por exemplo:

```
<% Dim teste
    Dim Mensagem
        teste = 25
        if teste = 30 then
            Mensagem = "O valor de Teste é igual a 30"
        Else
           Mensagem = "O valor de Teste é igual a 25"
        End if
%>
```

Até este ponto, vimos como testar apenas um condição. Porém, podemos utilizar o comando **IF** quando existem várias condições a ser testadas. Veja a sintaxe:

```
If <condição> Then
      Bloco de Instrução (que pode ser apenas uma ou várias)
[ElseIf <condição>Then
   [Bloco de Instrução]]
[Else
   [Bloco de Instrução]]
End If
```

* O que estiver entre [] é opcional e entre <> é obrigatório.

Se você observar, a estrutura é basicamente a mesma utilizada para um **IF** simples. A diferença está no ***Elseif***. Quando você utiliza este tipo de sintaxe é preciso especificar um outra condição a ser testada. Por exemplo:

```
<% Dim teste
     Dim Mensagem
         teste = 25
         If teste = 30 then
             Mensagem = “O valor de Teste é igual a 30”
         Elseif teste = 26 then
            Mensagem = “O valor de Teste é igual a 26”
         Else
            Mensagem = “O valor de Teste é igual a 25”
         End if
%>
```

E como inserir este código em uma página ASP?

Lembre-se que você deve escrever os códigos que devem ser interpretados pelo servidor entre <% %>, partindo deste ponto, vamos a um exemplo:

```
<html>

<head>
<title>Horas ... </title>
</head>

<body bgcolor="#FFFFFF">
<% if Time < #12:00# then %>
  Bom Dia!!! <BR>
  São exatamente <%=Time %>.
<% else %>
  São exatamente <%=Time %>.
<% end if %>
</body>

</html>
```

O código criado:

```
<% if Time < #12:00# then %>
  Bom Dia!!! <BR>
  São exatamente <%=Time %>.
<% else %>
  São exatamente <%=Time %>.
<% end if %>
```

O teste que o IF faz da condição é se esta é verdadeira. Então, se a hora do servidor for menor que 12:00, ou seja, ainda é manhã, será mostrado para o usuário o texto (supondo que seja 10:05 da manhã):

Bom dia!
São exatamente 10:05.

Caso contrário, será mostrado (supondo que seja 13:00):

São exatamente 13:00.

O próximo comando é o **SELECT**. Este comando foi desenvolvido para melhorar a eficiência do IF ... Then ... ElseIf ... Sua função é selecionar um bloco de instrução a partir do teste de uma condição e compará-lo com uma série de valores.

Sintaxe

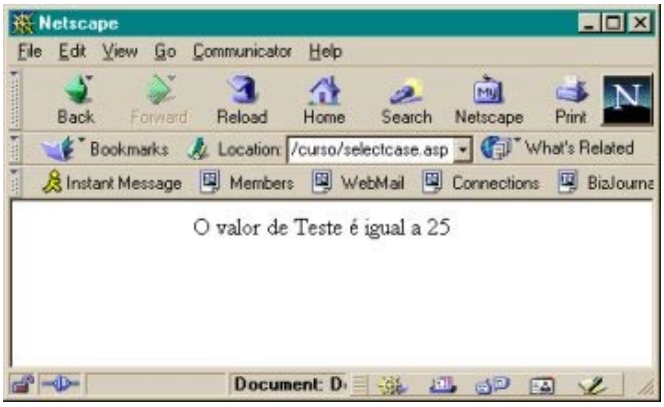
```
Select Case <expressão a ser testada - critério>
   [Case ValorX-Condição
      [Bloco de Instruções]]
   [Case Else ValoresN-CondiçãoN
      [Bloco de Instruções]]
End Select
```

Onde ***<Critério>*** é uma variável do tipo numérica ou texto (string), ***Condição*** e ***CondiçãoN*** são as possibilidades de valores que esta variável pode assumir.
A lógica do **SELECT** é similar a do **IF** ... Then ... Else. Ambas as estruturas permitem o caso de todas as condições falharem e o fluxo do bloco ser desviado para um caso de *"se não"*. A cláusula Else do SELECT é utilizada quando nenhuma condição é satisfeita pelo critério. Podemos reescrever o exemplo utilizado com o IF ... Then ... ElseIf, utilizando o o **SELECT**:

* O que estiver entre [] é opcional e entre <> é obrigatório.

```
<% Dim teste
     Dim Mensagem
        teste = 25
        Select Case teste
           Case 30
                Mensagem = "O valor de Teste é igual a 30"
           Case 26
                Mensagem = "O valor de Teste é igual a 26"
           Case Else
                Mensagem = "O valor de Teste é igual a " & teste
        End Select%>

<HTML>
<BODY>
<center><% = Mensagem %></center>
</BODY>
</HTML>
```

O resultado do exemplo:



O código VBScript (ASP)

Este arquivo deve ser nomeado com a extensão ASP, pois contem códigos que apenas um servidor IIS pode interpretar.

<% Dim teste
Dim Mensagem

Depois da declaração das variáveis ***teste*** e ***Mensagem***, atribuímos um valor qualquer para a variável ***teste*** que servirá como critério do **SELECT**. Como estamos atribuindo um valor numérico para ***teste***, não é preciso utilizar caracteres especiais para indicar o valor a ser atribuído.

teste = 25

O próximo bloco de instruções é o bloco do **SELECT**. Definimos que o critério é a variável teste, ou seja, a variável teste será testada nas condições que definimos nas cláusulas **CASE**. Em nosso caso, a "pergunta" que o SELECT fará será a seguinte:

Teste é igual a 30?
Se sim: Mensagem recebe o texto: "O valor de Teste é igual a 30"
Teste é igual a 26?
Se sim: Mensagem recebe o texto: "O valor de Teste é igual a 26"
Se não (teste não assumiu nenhum dos valores acima (30 ou 26) ...
Mensagem recebe o texto: "O valor de Teste é igual a 25"

Por que no CASE Else a variável mensagem recebe o texto: **"O valor de Teste é igual a 25"**?
Antes da comparação da variável Teste, atribuímos para esta o valor 25. Como no código concatenamos com o texto: **"O valor de Teste é igual a "** o valor da variável teste, como resultado é mostrado seu conteúdo. Então:

Código VBScript	Resultado
"O valor de Teste é igual a " & teste	→ "O valor de Teste é igual a 25"

```
Select Case teste
   Case 30
        Mensagem = "O valor de Teste é igual a 30"
   Case 26
        Mensagem = "O valor de Teste é igual a 26"
   Case Else
        Mensagem = "O valor de Teste é igual a " & teste
End Select%>
```

Neste ponto da aplicação, fechamos o bloco do SELECT e terminamos a parte dos códigos que são interpretados pelo servidor. Colocamos algum código HTML e para mostrar a mensagem, novamente temos que 'mandar" o código para o servidor: ***<% = Mensagem %>***.

```
<HTML>
<BODY>
<center><% = Mensagem %></center>
</BODY>
</HTML>
```

Estruturas de Repetição

São estruturas que repetirão um bloco de instruções até que a condição especificada seja satisfeita ou por um número de vezes especificado. No VBScript, trabalharemos com 3 destas estruturas:

Do ... Loop
For ... Next
While ... Wend

A primeira que veremos será a **Do ... Loop**. É utilizado para executar um conjunto de instruções até que a condição determinada seja verdadeira. O Do ... Loop oferece dois operadores condicionais para avaliar uma condição: While e Until.

Sintaxe:

```
Do {Until | While } <condição>
   <bloco de instruções>
   Exit Do
   <bloco de instruções>
Loop
```

Ou

```
Do
   <bloco de instruções>
   Exit Do
   <bloco de instruções>
Loop {Until | While } <condição>
```

Onde ***<condição>*** deve ser uma expressão que pode ser avaliada como true/false (verdadeiro/falso), ***<bloco de instruções>*** são os comandos que devem ser executados enquanto a condição não for satisfeita.

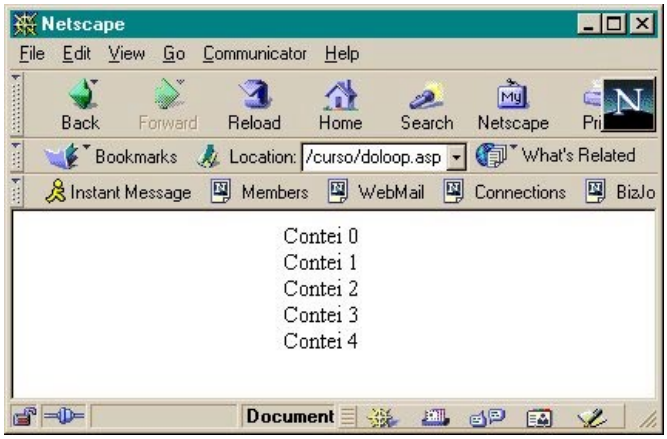
A diferença entre as sintaxes acima é o local da condição. Na primeira sintaxe, a condição está localizada na mesma linha de execução do Do. Na segunda, a condição está localizada no final do laço ao lado do Loop. Esta mudança de localização dá aos programadores flexibilidade para testar

a validade da condição primeiro e então processar o bloco de instruções, ou processar o bloco de instruções pelo menos uma vez, e depois avaliar a condição.

Exemplo

```
<html><body>
<%
  Dim Mensagem
  Dim Cont

  Mensagem = “”
  cont = 0

  Do Until cont=5
        Mensagem = “Contei “ & cont %>
        <center><% = Mensagem %></center>
        <% cont = cont + 1
  Loop
%>
</body></html>
```

O resultado retornado pelo servidor:



O código escrito (ASP)

Lembre-se que todo o código ASP deve ser escrito entre <% %>, pois será interpretado pelo servidor que retornará como resultado o HTML correspondente.

<%
Dim Mensagem
Dim Cont

Como vimos no capítulo anterior, a cláusula DIM é utilizada para declarar variáveis locais, ou seja, as variáveis ***Mensagem*** e ***Cont*** existirão apenas para este código script.

Mensagem = “”
cont = 0

Nas linhas acima, estamos "limpando" as variáveis, ou seja, atribuindo-lhes valores nulos para tirar qualquer conteúdo que não nos seja útil. No caso de cont atribuímos o valor 0 (zero), e para Mensagem, atribuímos "" que caracteriza o comprimento zero de string.

O próximo bloco de instruções é o bloco definido pelo **Do ... Loop**. Este bloco será executado 5 vezes – isto é definido pela condição: cont = 5. E mostrará uma mensagem todas as vezes que as instruções dentro do comando de repetição forem executadas.

Do Until cont=5

A linha a seguir define a mensagem a ser mostrada para o usuário. É uma atribuição de um texto para uma variável, desta forma, o que for atribuído deve estar entre " " (aspas). Ainda nesta linha, você deve observar que existe uma concatenação de strings (***"Contei " & cont***), para isso, deve-se utilizar o operador &.

Mensagem = "Contei " & cont %>

Neste ponto do programa, encerramos o trecho que deve ser interpretado pelo Servidor (%>).

A linha a seguir contem código HTML ***( )*** e código VBScript a ser interpretado pelo servidor ***(<% = Mensagem %>)*** que mostrará o conteúdo da mensagem.

*** <% = Mensagem %> ***

As próximas linhas de instrução fazem parte do **Do ... Loop**. Onde o ***cont*** é incrementado para que a repetição tenha um limite (cont = 5) e ***Loop*** encerra todo o bloco do **Do ... Loop**.

```
        <% cont = cont + 1
    Loop
%>
```

While ... Wend

Outra estrutura similar ao Do ... Loop é o While ... Wend. Esta estrutura repete um bloco de instruções até que um condição seja satisfeita (true).

Sintaxe

```
While <condição>
        <bloco de instruções>
Wend
```

Onde <**condição**> é uma expressão que pode ser avaliada como verdadeira ou falsa (true/false) e <**bloco de instruções**> representa o conjunto de comandos que devem ser executados até que a condição seja falsa.

Quando a condição descrita no While for verdadeira todos os comandos serão processados até chegar no Wend. Neste ponto, o fluxo do programa passa para o While que testa a condição novamente e este ciclo é repetido até que a condição seja falsa.

Exemplo

O exemplo a seguir incrementa a variável contador até que seu valor chegue a 10.

```
<% Dim contador
   contador = 0
   While contador < 10
          contador = contador + 1
   Wend
%>

<HTML><BODY>
<CENTER><B>Contei até <% = contador %></B></CENTER>
</BODY></HTML>
```

Resultado retornado:

Código

```
While contador < 10                ' Avalia a condição
        contador = contador + 1    ' Incrementa a variável contador
Wend                               ' Finaliza o laço quando a variável contador for igual a 9.
```

For ... Next

Repete um conjunto de instruções por um número especificado de vezes. Este comando é usado quando sabemos o número de ciclos ou iterações que são necessárias.

Sintaxe

```
For contador = inicio To fim [Step passo]
   [bloco de instruções]
   [Exit For]
   [bloco de instruções]
Next
```

Onde **contador** é uma variável numérica usada no laço como o contador. Esta variável não pode ser um elemento de um vetor, nem um elemento de um Tipo definido pelo usuário. **Inicio** é o valor inicial do contador e **fim** é o valor final que o contador assumirá. **Passo** define o valor do incremento ou decremento do **FOR**, se não for definido, o default (padrão) é 1 (um).

O valor do parâmetro **passo** pode ser tanto positivo (incremento) ou negativo (decremento). O valor deste parâmetro determina o processo da seguinte maneira:

Permanece no laço se ...
Positivo ou 0 (zero) → contador <= fim
Negativo → contador >= fim

Exemplo

Se o parâmetro **passo** for positivo:

```
<% Dim contador

    For contador = 65 to 122 step 2
        Response.Write chr(contador)
    Next
%>
```

O resultado será igual a: ACEGIKMOQSUWY[]_acegikmoqsuwy

For contador = 65 to 122 step 2

No exemplo, a variável contador é inicializada com o valor 65, sendo incrementada até 122. O incremento é feito de 2 em 2, ou seja, a primeira vez, a variável contador está com 65, a segunda vez, o contador passa a ter 67, pois o incremento é de 2.
Se o parâmetro **passo** for negativo:

```
<% For contador = 122 to 65 step -1
        Response.Write chr(contador)
    Next
%>
```

O resultado será igual a: zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA

For contador = 122 to 65 step -1

Neste exemplo, o parâmetro passo foi determinado com –1, ou seja, o contador começa com 122 e a cada clico, é decrementado deste valor 1 até chegar ao valor 65.

Observe que o operador **Next** no VBScript defere do operador **Next** do VB. No VBScript, não é preciso especificar o contador na declaração do **Next**, como acontece no VB. O operador **Next** automaticamente incrementa o contador designado na declaração do **FOR**. Se você adicionar o nome da variável contador na cláusula **Next**, um erro será gerado pelo VBScript:

Expected end of Statement

Em FOR encadeados, não é permitido a utilização do mesmo nome de variável. Um erro de runtime será gerado. Por exemplo, o código abaixo utiliza dois FOR encadeados, com o mesmo nome de variável, o VBScript irá gerar o seguinte erro:

Invalid 'For' loop control variable.

```
<% For contador = 0 to 5
        For contador = 1 to 7
            Soma = Soma + contador
        Next
  Next %>
```

Resumo

Neste capítulo aprendemos que ...

- As estruturas de decisão são utilizadas para determinar o fluxo da aplicação. As estruturas mais utilizadas são:

```
If <condição> then
        <bloco de instruções>
else
        <bloco de instrução>
end if

Select <expressão a ser testada – expressão>
        [Case ValorX-Condição
                [bloco de instrução]]
        [Case Else ValoresN-CondiçãoN
                [Bloco de instrução]]
End Select
```

- As estruturas de repetição que são utilizadas para executar um bloco de instrução até que uma condição seja satisfeita. As estruturas mais utilizadas são:

```
Do Until <condição>
        <bloco de instruções>
Loop

While <condição>
        <bloco de instruções>
Wend

For contador = inicio To fim [Step passo]
      [bloco de instruções]
      [Exit For]
      [bloco de instruções]
Next
```

Dicas

- A declaração do Else é sempre interessante. Porque, às vezes, nos esquecemos de algum teste da condição. Desta forma, evitamos erros de lógica,
- No caso do **FOR**, modificar o valor do contador dentro do laço, pode dificultar a leitura e a interpretação do código por outro programador.
- Se a condição a ser avaliada for igual a Null, o VBScript retornará a condição como false.

CAPÍTULO VIII - OS OBJETOS DO ASP

O ASP possui alguns objetos básicos que operam as funcionalidades de uma aplicação. Cada objeto possui seus próprios métodos e eventos, por isso estudaremos cada um deles separadamente, observando suas funções mais importantes e úteis.

O objeto Application

Podemos entender como sendo *aplicação* todo o conjunto de páginas ASP e HTML que formam o programa como um todo. Uma aplicação começa quando uma página ASP é requisitada pela primeira vez ao servidor e termina quando o servidor é desligado ou quando é desativada (tirada do uso, sendo apagada, por exemplo).

Desta forma, o objeto Application irá referir-se e estar disponível para toda a aplicação, ou seja, será "enxergado" por toda página ASP e por qualquer usuário da aplicação.

Qual a utilidade de um objeto Application?

Devido a múltiplos acessos a uma aplicação, é possível, através do objeto Application, controlar possíveis erros causados por tentativas de alterações simultâneas de variáveis, como também disponibilizar determinadas informações que serão acessíveis em qualquer parte do programa, e/ou por qualquer usuário.

Declarando variáveis Application

Variáveis Application poderão ser acessadas por qualquer parte do programa, por qualquer usuário.

Sintaxe: Application("nomevar") = conteúdo

Para visualizarmos melhor o efeito de uma variável deste tipo, vejamos o exemplo:

```
<%
Application("DicaDeHoje") = "Que tal estudar ASP?"
%>
```

Suponha a existência da linha de código acima em algum arquivo ASP de uma aplicação. Com ela, armazenamos a string "Que tal estudar ASP?" numa variável Application que chamamos de DicaDeHoje. Se nesta mesma aplicação possuirmos outra página ASP com o código:

```
<HTML>
<BODY>
<CENTER>
  A nossa dica do dia hoje é <br>
<b> <%= Application("DicaDeHoje") %> <b>
</CENTER>
</BODY>
</HTML>
```

Qualquer usuário que acessar a página acima obterá o seguinte resultado:

Importante também lembrar que é possível acessar a variável DicaDeHoje em qualquer outra parte da aplicação.

Controle de acesso a variáveis Application

Para prevenir-se de eventuais problemas com alterações simultâneas de valores de variáveis do tipo Application, mantendo deste modo os dados sempre consistentes, é necessário lançar mão de dois métodos deste objeto: *lock* e *unlock*.

Método Lock

O objetivo deste método é "trancar" o acesso às variáveis Application, deixando-as disponíveis para somente um único usuário.

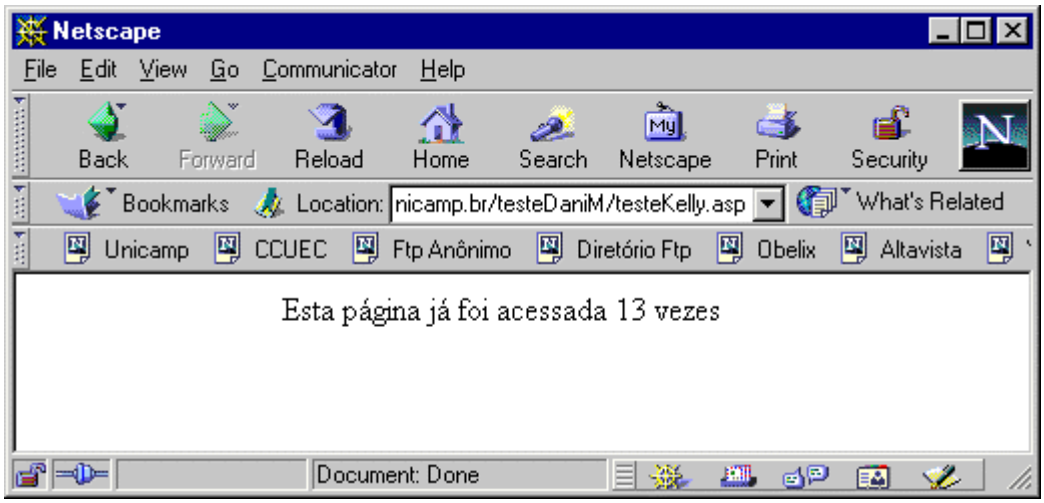
Método UnLock

O método UnLock "destranca" as variáveis Application que foram trancadas por um método Lock. Quando este método for executado, todas as variáveis Application estarão novamente disponíveis.

Observe o exemplo:

```
<HTML>
<BODY>
<%
Application.Lock
Application("visitantes") = Application("visitantes") + 1
Application.UnLock
%>
<CENTER> Esta página já foi acessada
<%=Application("visitantes")%> vezes </CENTER>
</HTML>
</BODY>
```

No caso acima, o método Lock foi utilizado para que o número de visitantes seja sempre exibido de forma correta, não ocorrendo nenhum problema no caso de haverem duas tentativas simultâneas de modificação no valor desta variável. Depois de alterar o valor de *visitantes*, a aplicação é destrancada. Veja o resultado deste código:



O objeto Server

Este objeto é capaz de interagir com serviço HTTP, criando uma interface programável de seus métodos e propriedades. Outra função do objeto Server é instanciar componentes ASP no servidor, o que torna a tecnologia muito mais poderosa e ampla.

O que é um componente?

Componentes são bibliotecas de objetos que possuem finalidades diversas. Podemos encontrar inúmeros tipos de componentes. Há, por exemplo, componentes que operam serviços de e-mail (como enviar mensagens e arquivos atachados), que criam interface para upload de arquivos, que manipulam bancos de dados, etc. Existem componentes que já vêm incorporados ao pacote ASP e existem também diversos componentes disponíveis na Internet para download.
Para instalar um novo componente (que geralmente é uma dll), basta registrá-lo no *Registry* do servidor. Depois, para utilizá-lo, basta instanciá-lo em qualquer programa ASP.

Método CreateObject

Este método cria uma instância de um componente no servidor. A capacidade de utilizar componentes amplia muito as potencialidades de uma aplicação ASP.

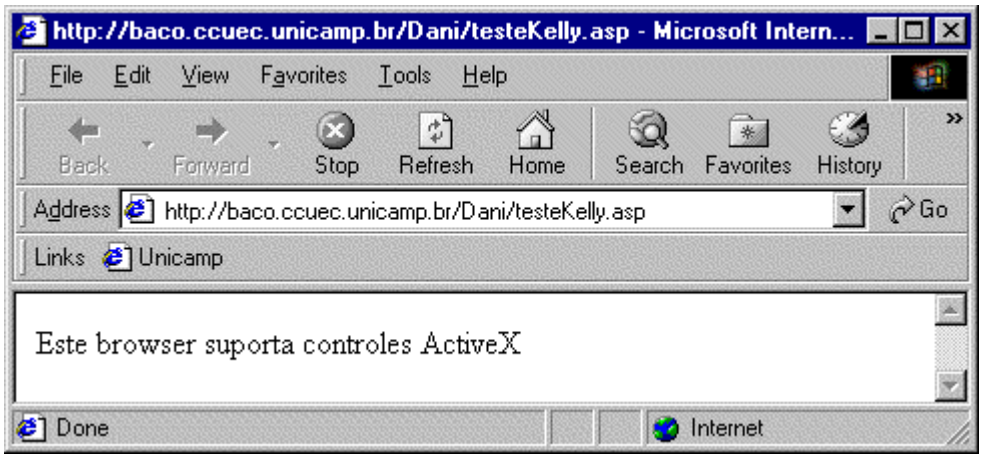
Sintaxe: Server.CreateObject ("nomecomp.nomeclasse")

Onde *nomecomp* é o nome da biblioteca do componente e *nomeclasse* é a classe desta biblioteca da qual se deseja criar o objeto.

Como exemplo, criaremos uma instância da classe BrowserType, da bibliioteca MSWC (Microsoft Web Components):

```
<HTML>
<BODY>
<%
Set brtp = Server.CreateObject("MSWC.BrowserType")
  If brtp.ActiveXControls then %>
     Este browser suporta controles ActiveX
  <% Else %>
      Este browser não suporta controles ActiveX
  <% End if %>
</BODY>
</HTML>
```

Um possível resultado obtido será:



No exemplo acima, criamos o objeto *brtp*, pertencente à classe BrowserType da biblioteca MSWC e testamos sua propriedade *ActiveXControls*. Observe que o comando **Set** foi utilizado. Desta forma, o método CreateObject retornará à variável *brtp* um ponteiro que aponta para o objeto instanciado. Este é apenas um exemplo de um objeto de inúmeros outros existentes. Ao longo deste curso abordaremos outros componentes, e certamente ao pesquisar na Web, você encontrará cada vez mais novos componentes sendo criados.

Métodos Encode

A tecnologia Web é basicamente baseada na transferência de textos simples (ASCII) pelo protocolo TCP/IP e na interpretação destes textos pelos browsers. Devido a este fato, é possível ocorrerem erros de interpretações de caracteres peculiares (como letras acentuadas ou espaços em branco, por exemplo). Para resolver estes problemas, a tecnologia ASP dispões de dois métodos: *HTMLEncode* e *URLEncode*.

Método HTMLEncode

Este método faz com que os caracteres sejam exibidos em tela exatamente como foram especificados. Para isso o ASP cria, para a string fornecida, códigos especiais para caracteres não-ANSI. Este método é importante para se assegurar de que a informação será exibida numa página HTML exatamente como se deseja.

Sintaxe: Server.HTMLEncode ("string")
Onde *string* é a cadeia de caracteres que se deseja formatar.

Método URLEncode

Semelhante ao método HTMLEncode, este método formata a string especificada para que não ocorram erros de interpretação de caracteres. Porém, utilizamos o método URLEncode quando a string especificada trata-se de uma URL.
Sintaxe: Server.URLEncode ("string")

O objeto Session

Entende-se por sessão o tempo que um usuário utiliza uma aplicação. Cada vez que uma aplicação é acessada por um usuário, uma sessão no servidor é aberta para ele. Quando a aplicação termina, a sessão é finalizada. Dessa forma, informações disponibilizadas no escopo da sessão estarão disponíveis durante toda a aplicação para um determinado usuário.
Este objeto é baseado em cookies, portanto será somente acessível aos browsers que os suportam e que estejam habilitados a os aceitarem.

Declarando variáveis do tipo Session

A declaração de variáveis do tipo Session possibilita que as mesmas estejam disponíveis durante toda a sessão de um usuário. São úteis, por exemplo, quando desejamos identificar características de usuário em qualquer parte de um programa.
Sintaxe: Session("nomevar") = conteúdo
Observe o exemplo:

```
<HTML>
<BODY>

<% Set brtp = Server.CreateObject("MSWC.BrowserType")
Session ("vbsc") = brtp.vbscript %>

</HTML>
</BODY>
```

Com o código acima, teremos armazenado na variável *vbsc* um valor booleano (true ou false) que indica se o browser suporta ou não a tecnologia VBScript sendo executada no cliente. Como esta variável foi definida como tipo Session, poderemos utilizá-la em outra parte da aplicação, como no exemplo abaixo:

```
<%
If Session("vbsc") then
  Call VersaoVbSc ()
Else
  Call VersaoNoVbSc ()
End if
%>
```

O Objeto Response

O objeto Response controla os dados que serão enviados para o cliente. Estes dados podem ser simplesmente HTML, cookies, valores de variáveis, etc.

Coleção Cookies

É através desta coleção que podemos enviar cookies para um cliente. Com cookies, podemos armazenar temporariamente valores em arquivos-texto no browser cliente.

Sintaxe: Response.Cookies ("nomecookie")("nomedachave").atributo = conteúdo

Devemos entender como *nomecookie* como sendo o nome do arquivo texto a ser armazenado no cliente e *nomedachave* como sendo o nome do campo a ser armazenado. Existem ainda atributos de cookies que estudaremos posteriormente.

Veja o exemplo:

```
<%
Response.Cookies ("MeuTeste")("Valor1") = "Este é o primeiro valor"
Response.Cookies ("MeuTeste")("Valor2") = "Segundo valor"
Response.Cookies ("MeuTeste")("Valor3") = "Valor 3"
%>
<HTML>
<BODY>
  Você acabou de receber um Cookie!
</BODY>
</HTML>
```

Neste exemplo, gravamos no cliente um cookie com o nome *MeuTeste* que contém três valores separados em três campos que se chamam: *Valor1*, *Valor2* e *Valor3*. Quando desejarmos recuperar as informações deste cookie, utilizaremos estes nomes. Veremos como recuperar valores de um cookie ao estudar o objeto Request.

Note também que as linhas de código que gravam o cookie vêm antes de qualquer tag HTML. Isto ocorre porque não há como gravar informações em um cookie depois de qualquer código HTML ser enviado ao cliente. Se houvesse qualquer tag antes da gravação do cookie, seria emitida uma mensagem de erro. Portanto, lembre-se: **não é possível gravar um cookie depois que alguma diretiva HTML já foi enviada ao cliente.**

Atributo Expires

Podemos especificar qual a "data de validade" de um cookie através do atributo Expires. Desta forma, um cookie existirá no cliente até a data estipulada neste atributo. Se não estipularmos nenhuma data para o atributo Expires, o cookie perderá sua validade assim que a sessão do usuário se encerrar. Observe, no código abaixo, a utilização deste atributo no cookie *MeuTeste*, onde o mesmo irá expirar em 11 de setembro de 1999

```
<%
Response.Cookies ("MeuTeste")("Valor1") = "Este é o primeiro valor"
Response.Cookies ("MeuTeste")("Valor2") = "Segundo valor"
Response.Cookies ("MeuTeste")("Valor3") = "Valor 3"
Response.Cookies ("MeuTeste").Expires = "11/09/99"
%>
```

Método Write

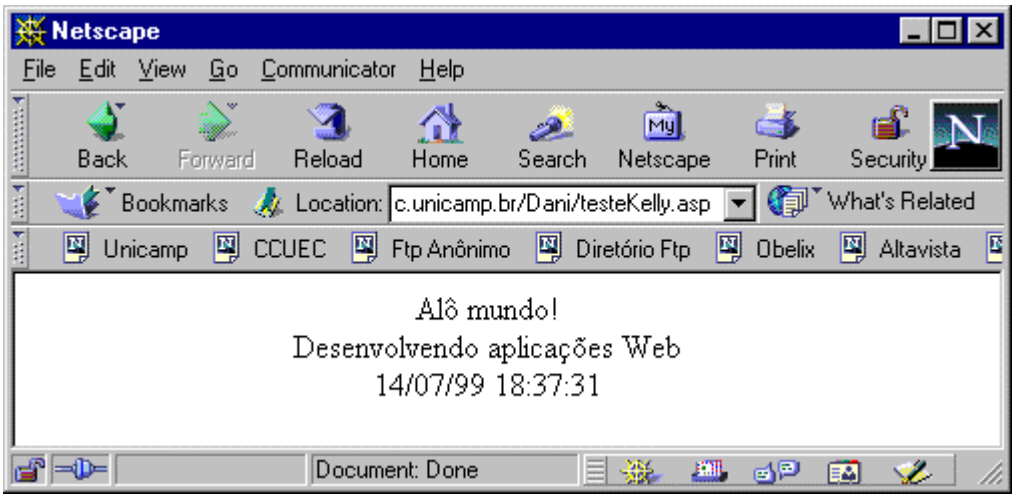
Este método é utilizado para enviar texto à página HTML.

Sintaxe: Response.Write conteúdo

Podemos utilizar como conteúdo valores de variáveis, funções ou mesmo textos simples. Veja no exemplo abaixo diferentes possibilidades de se utilizar este método:

```
<HTML>
<BODY>
<% Session("curso") = "Desenvolvendo aplicações Web" %>
<CENTER>
<% Response.Write "Alô mundo!" %> <BR>
<% Response.Write Session("curso") %> <BR>
<% Response.Write now %> <BR>
</CENTER>
</BODY>
</HTML>
```

Observe no resultado obtido que podemos então escrever textos (*Alò mundo!*), variáveis (*Session("curso")*) ou funções (*now*) com o método Write:



Método Redirect

Este método é utilizado para redirecionar o browser para outra URL.

Sintaxe: Response.Redirect URL

É importante ressaltar que o método Redirect deve ser colocado **antes** de qualquer tag HTML ser enviada ao cliente. Caso contrário, ocorrerá um erro.

Veja um exemplo deste método:

```
<%
If Session("idioma") = "ingles" then
   Response.Redirect "http://www.empresa.com.br/ingles"
Else if Session("idioma") = "alemao"
   Response.Redirect " http://www.empresa.com.br/alemao"
End if
End if
%>
<HTML>
<BODY>
 Olá! Obrigado pela visita!
<% var = "teste" %>
</BODY>
</HTML>
```

No exemplo acima, suponha que a variável *Session("idioma")* tivesse o valor *"ingles"*. Desta forma, o browser seria redirecionado para o endereço *http://www.empresa.com.br/ingles* . É importante notar que neste caso **nenhum** código seguinte será executado (no exemplo, não seria atribuído nenhum valor à variável *var*, pois esta linha de código não seria executada) e **nenhuma** tag HTML deste arquivo será enviada ao browser. O contrário ocorrerá somente se a variável *Session("idioma")* tiver valor diferente de *"ingles"* ou *"alemao"*.

O Objeto Request

Este objeto é capaz de receber informações do cliente, como, por exemplo, ler cookies e receber dados digitados em formulários HTML A seguir, estudaremos suas coleções e métodos mais importantes.

Coleção Cookies

Esta coleção é responsável por ler as informações armazenadas previamente nos cookies existentes nos clientes.

Sintaxe: Resquest.Cookies ("nomecookie")("nomedachave")

Onde *nomecookie* é o nome do cookie armazenado e *nomedachave* é o nome do campo do cookie que se deseja obter. Observe o exemplo:

```
<%
Request.Write (Request.Cookies("MeuTeste")("Valor1"))
%>
```

Com o código acima, obtemos, do cookie *MeuTeste* o valor do campo *Valor1* e o escrevemos na tela através do Response.Write.

Coleção Form

A coleção Form permite que se obtenha dados digitados em formulários HTML enviados pelo método HTTP*Post*.

Sintaxe: Request.Form ("nomecampo")

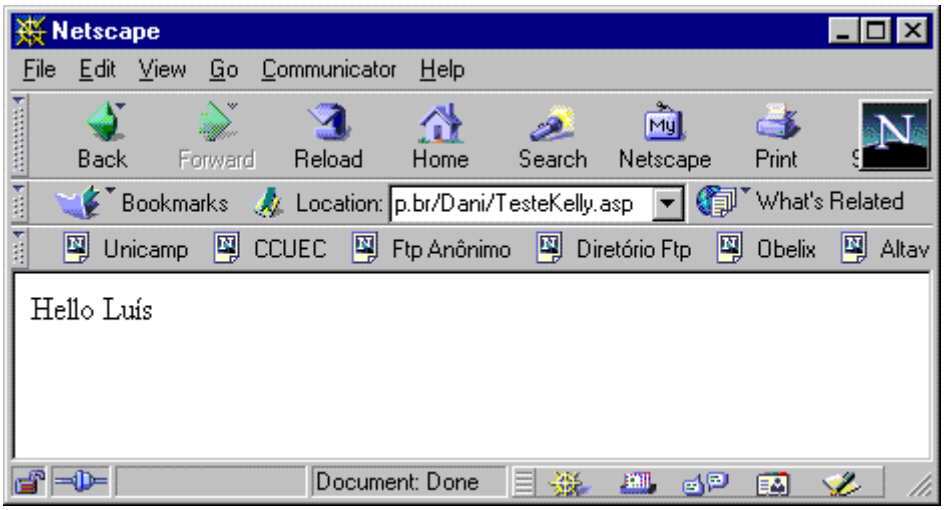
Onde *nomecampo* é o nome dado à propriedade *NAME* do campo do formulário HTML. Considere o seguinte formulário HTML:

```
<HTML>
<BODY>
<FORM ACTION="Request.asp" METHOD = Post>
Nome:<INPUT TYPE = Text NAME = "txtNome">
<BR>
Idioma:<BR>
<INPUT TYPE=Radio NAME="opIdioma" VALUE="Ingles">Inglês<BR>
<INPUT TYPE=Radio NAME="opIdioma" VALUE="Alemao">Alemão<BR>
<INPUT TYPE=Radio NAME="opIdioma" VALUE="Portugues">Português <BR>
<INPUT TYPE=Submit VALUE="Enviar">
</FORM>
</BODY>
</HTML>
```

Quando o botão Submit for clicado, o arquivo *Request.asp* será chamado e executado. Este arquivo poderia ser o seguinte:

```
<%
nome = Request.Form("txtNome")
idioma = Request.Form("opIdioma")

If idioma = "Ingles" then
  Response.Write "Hello " & nome
Else if idioma = "Alemao" then
  Response.Write "Hallo " & nome
Else
  Response.Write "Olá " & nome
End if
End if
%>
```

No código acima, armazenamos nas variáveis *nome* e *idioma* os valores colocados nos campos *txtNome* e *opIdioma* do formulário HTML. Com isto, podemos utilizar estes valores no programa normalmente. Um possível resultado obtido depois de se digitar no formulário o nome *Luís* e se escolher o idioma Português seria:



Coleção QueryString

Esta coleção é utilizada para se obter informações vindas da string de pesquisa HTTP. Esta string encontra-se depois do ponto de interrogação (?) na linha da URL. (Por exemplo, http://www.empresa.com.br?cliente=especial).

Sintaxe: Request.QueryString("nomevar")

CAPÍTULO IX – ODBC

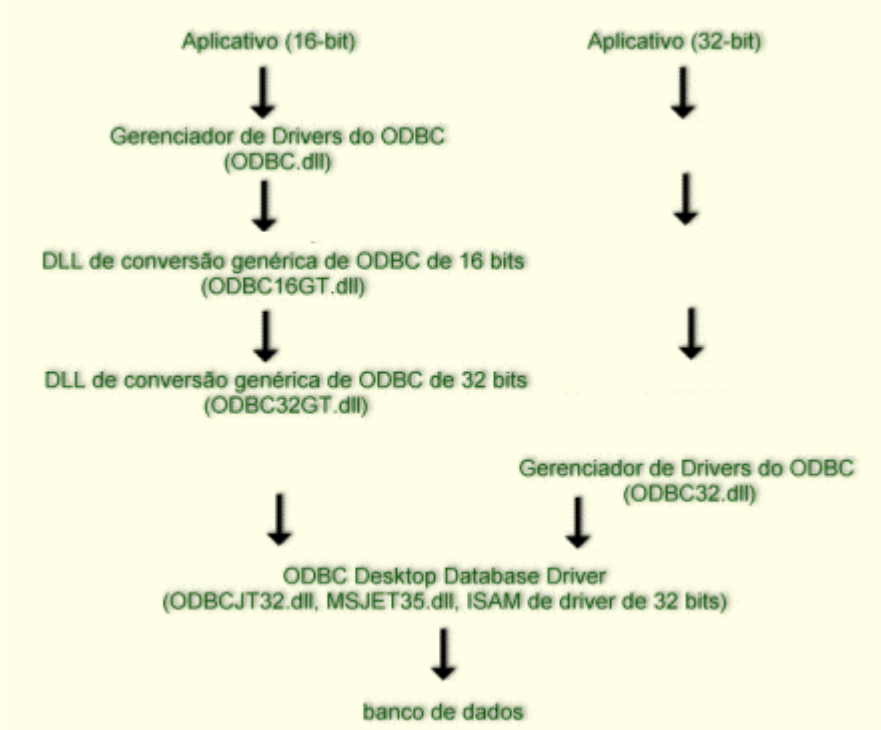
ODBC 3.0

O ODBC (Open DataBase Connectivity) 3.0 é o meio mais conhecido para acessar um banco de dados em ambiente Windows. Utilizando o ODBC, desenvolvedores não precisam se preocupar com as particularidades dos bancos de dados que irão acessar e trabalhar.

O ODBC é uma API para acessar, manipular e criar bancos de dados. Como um desenvolvedor de aplicações Web, você não precisa saber exatamente qual é a API para o ODBC. Entretanto, ter um conhecimento sobre ele será bastante útil.

Quando você acessa um banco de dados através do ODBC, este banco necessariamente tem que estar registrado como uma origem de dados ODBC. Registrando o banco como uma origem de dados, a aplicação apenas precisa saber o nome desta origem de dados. A localização do banco não faz diferença, nem mesmo o tipo de banco de dados.

A arquitetura de aplicativo/driver no Windows NT 3.51 ou posterior, é a seguinte:

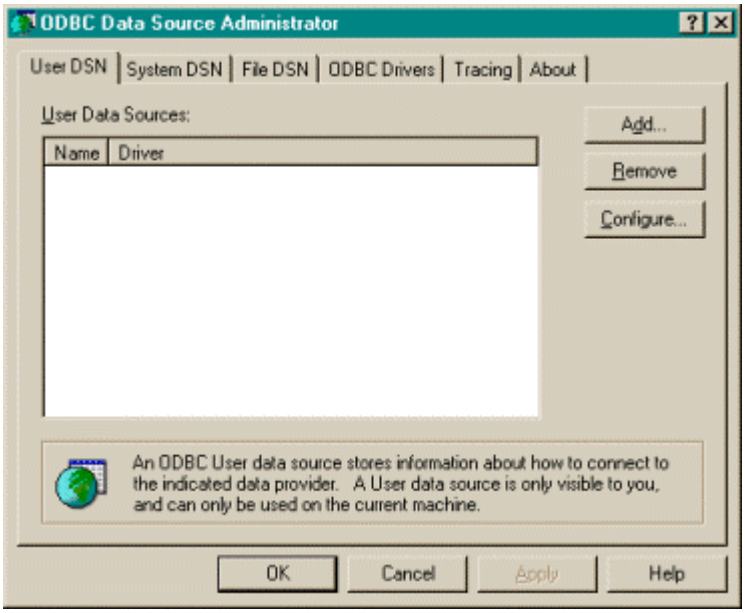


Abrindo o ODBC Desktop

Para executá-lo, siga os passos:

1. A partir do menu Start (Iniciar) do Windows, selecione Settings (Configurações)
2. Escolha a opção Control Panel, na janela que será aberta dê um duplo clique no ícone do ODBC:

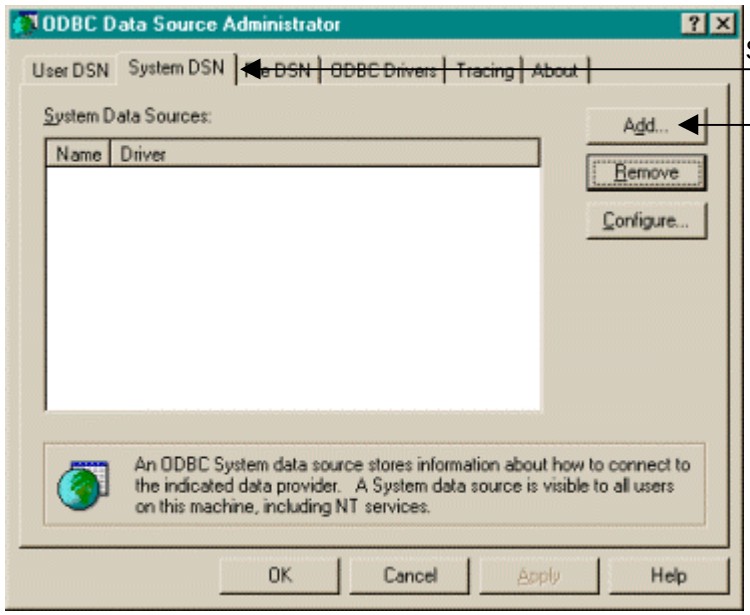
A tela abaixo nos mostra o painel de controle do ODBC.



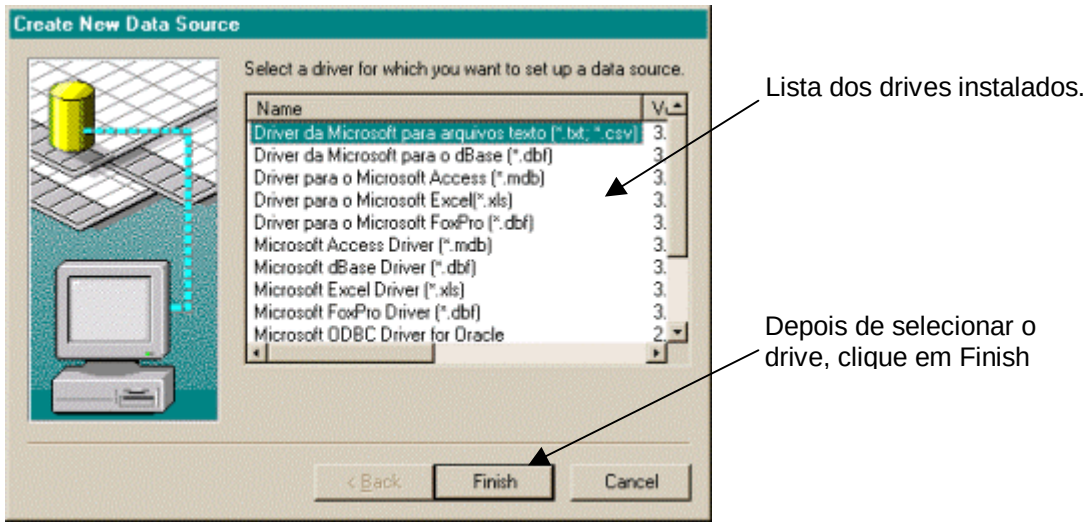
Existem três tipos de origem de dados que você pode criar: System, User e File. Quando você define uma origem de dados na opção System DSN o banco de dados será aberto para qualquer usuário do sistema, quando definido em User DSN o banco será aberto para um usuário específico e File DSN é uma descrição do banco.

Geralmente, para aplicações Web é criado uma origem de dados na opção System DSN.

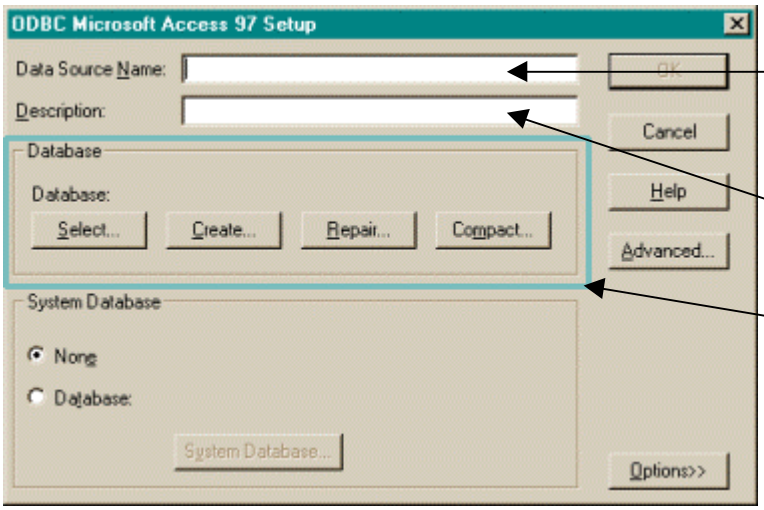
Como criar um System DSN



Depois de pressionado o botão Add, será mostrada a tela abaixo, onde você deve especificar o drive ODBC a ser utilizado para acessar seus dados:



Depois de escolher o drive, é preciso selecionar o arquivo que contém os dados. A próxima tela pedirá para você indicar o local físico (diretório e nome) do arquivo e um nome para a origem de dados que está sendo criada:



Nome da origem de dados (não precisa ser o nome do banco de dados).

Descrição da origem de dados.

Na parte sublinhada estão disponíveis as opções para seleção, criação, reparo e compactação do arquivo de banco de dados. Para selecionar um arquivo clique no botão Select.

Seguindo estes passos você cria uma origem de dados para seu banco de dados. Quando precisarmos abrir um banco de dados, será preciso apenas fazer a referência a origem de dados.

Resumo

Neste capítulo aprendemos que ...

- O ODBC é uma API utilizada para criar, manipular e acessar bancos de dados.
- Utilizando esses drives, não é necessário ter conhecimento das particularidades de cada banco de dados.
- Existem três tipos de origem de dados: System DSN, User DSN e File DSN. O System DSN é utilizado quando vários usuários de um sistema podem acessar a base. O User DSN é utilizado quando um usuário específico pode ter acesso, e o File DSN é uma descrição do banco.
- Geralmente, a opção escolhida para criar uma origem de dados é o System DSN, pois uma aplicação na maioria das vezes, é acessada por vários usuários.

Dicas

- Os drivers ODBC são fornecidos com ODBC 3.0, mas funcionam com ODBC 2.5. Eles foram desenvolvidos para ser usados no Microsoft Windows 95 ou posterior ou Windows NT 3.51 ou posterior. O Windows 95 ou posterior, aceita somente aplicativos de 32 bits. O Windows NT 3.51 ou posterior, aceita aplicativos de 16 e de 32 bits.
- Você pode obter mais informações sobre a versão do ODBC a ser usada com os drivers ODBC, consultando o ODBC 2.0 Programmer's Reference e o SDK Guide e notas de versão do ODBC até versão 2.5. O ODBC 3.0 Programmer's Reference não deve ser usado como material de referência para problemas com esses drivers.
- Você pode utilizar um componente ASP que implementado em seu script faz mudanças no Registry do servidor e criar o DSN necessário. Esses componentes são de terceiros e podem ser encontrados em sites especializados em ASP.

O que acontece se você não tiver um DSN?
Se você deve conhecer o nome do arquivo (baseado em banco de dados como Access, Paradox, FoxPro, etc.) ou o nome do Data Source (no caso do SQLserver, por exemplo). Abaixo está uma maneira de abrir um Data Source sem um DSN. Note que você deve conhecer o caminho completo do arquivo no servidor, isto é, apenas o nome do banco não é suficiente. É preciso ter todo o caminho, por exemplo: "e:\Inetpub\banco\inscricoes.mdb".
Existe uma função no ASP chamada Server.MapPath que com o nome de arquivo como parâmetro, retorna o diretório completo do arquivo no servidor, mas fique atento pois não é muito seguro utilizar este recurso.

```
<%
        Set conn = Server.CreateObject("ADODB.Connection")
        Caminho = "DBQ=" & Server.MapPath("nome_banco.mdb")
        conn.Open "DRIVER={Microsoft Access Driver (*.mdb)}; " & Caminho
        Set rs = conn.Execute("Select * from Cliente")
%>
```

Abaixo, estão os tipos mais comuns de nomes para drives ODBC:

Para Access --> driver = {Microsoft Access Driver (*.mdb)};
Para SQL -----> driver = SQL Server;

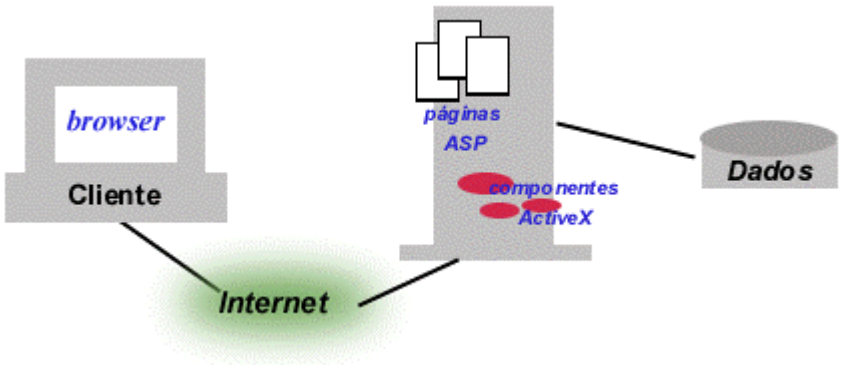
CAPÍTULO X – ACESSANDO BANCO DE DADOS

Desenvolvendo Aplicações Web

Em uma aplicação cliente/servidor, a aplicação cliente mantém uma conexão com a aplicação servidor, consultando-o periodicamente para verificar se a conexão está aberta. Se o servidor estiver fora do ar, a aplicação cliente não conseguirá a conexão e então tomará as medidas apropriadas (se esta aplicação estiver preparada para isso) como, por exemplo, enviar uma mensagem de erro para o usuário.

Porém, desenvolver aplicações baseadas na Web pode ser muito diferente de outros tipos de programação. Diferente de típicas aplicações cliente/servidor, essas aplicações compreendem páginas Web como resposta ao cliente. Estas são estáticas, ou seja, depois do cliente (browser) solicitar a página, a mesma é processada no servidor e então, o documento solicitado é retornado. O processo da página termina quando o servidor Web envia a página. Como um desenvolvedor de aplicações Web, você precisa também se preocupar com a conexão de seu cliente e a Internet.

Neste capítulo aprenderemos como recuperar e alterar informações em um banco de dados através de aplicações Web, utilizando para isto a tecnologia **ADO** (ActiveX Data Objects).



ADO (ActiveX Data Objects)

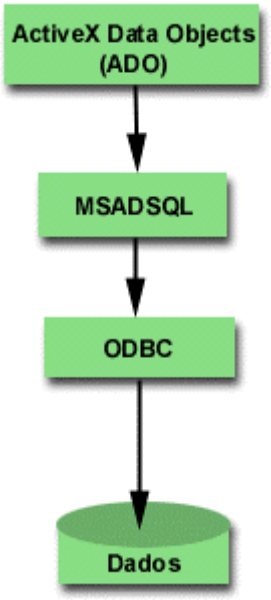
É uma coleção de objetos utilizados para recuperação, alteração, inclusão e exclusão de registros em bancos de dados ODBC e OLE DB. O ADO é instalado com Microsoft Internet Information Server (IIS), versão 3.0.

O ADO é escrito em páginas ASP e executado no servidor Web, e retorna as informações dos bancos de dados em formato HTML. A página retornada pode ser visualizada por qualquer browser em qualquer plataforma, pois o código é todo executado pelo Servidor.

Esta coleção de objetos constitui uma camada entre a sua página na Web e o seu banco de dados. Para acessar o banco de dados, você escreverá códigos que configurarão propriedades e métodos dos objetos do ADO.

A comunicação do ADO com o banco de dados é feita através de OLE-DB (baseado na tecnologia COM) ou banco de dados que possuem drives ODBC (nesta opção a comunicação é feita através da biblioteca MSDASQL.dll). Neste curso abordaremos a comunicação ADO → ODBC.

O esquema a seguir ilustra a comunicação do ADO com um banco de dados ODBC.

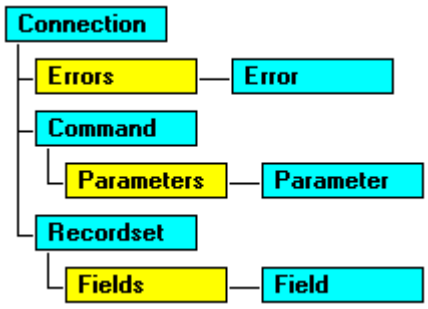


Instalando o ADO

Como o ADO faz parte do Microsoft Data Access Components, este pacote é automaticamente instalado e registrado pelo IIS.

Objetos do ADO

Com o ADO, você pode criar os objetos independentemente, o que permite a escrita de um código mais simples e a criação apenas dos objetos que você precisa.



Primeiramente, você irá trabalhar com os objetos **Connection**, **Command** e **Recordset**. Os objetos **Connection** e **Command** possuem o método **Execute** que executa um comando e retorna um recordset. Você também pode utilizar a função CreateObject para instanciar objetos **Recordset** explicitamente, e então utilizar o método Open do objeto instanciado para criar um recordset.

Connection

O objeto **Connection** representa uma conexão aberta com a origem de dados. Você utiliza o **Connection** para manter aberta em memória uma conexão com o banco de dados para que outros objetos possam acessá-lo, como por exemplo, o objeto **Recordset**.

Este objeto possui uma coleção que armazena qualquer erro que ocorra quando o acesso é feito chamada **Errors**.

Abrindo uma conexão através do objeto Connection

Para estabelecer uma conexão com a fonte de dados, primeiramente devemos criar um objeto **Connection**. Para isto, utilizamos o método **CreateObject** (objeto Server). Para conectar com a origem de dados, você utiliza o método **Open**.

Sintaxe do método Open

Connection.Open DataSourceName, User, Senha

Onde **DataSourceName** é o nome do DSN criado no ODBC, **User** é o nome do usuário que tem permissão de acesso e sua **Senha**.

Exemplo

O exemplo abaixo cria um objeto **Connection** chamado *MinhaConexao* e, através dele, abre uma conexão com a origem de dados chamada *Externo* (DSN definido no ODBC).

```
<%
   Set MinhaConexao = Server.CreateObject ("ADODB.Connection")
   MinhaConexao.Open "Externo", " ", " "
%>
```

Fechando uma conexão

Quando você termina de trabalhar com seu banco de dados, você deve utilizar o método **Close** do objeto **Connection** para liberar qualquer associação com o recurso do sistema. Usando o método **Close**, você não libera memória. Você pode modificar as propriedades do objeto, e então utilizar o **Open** para abri-lo novamente. Para remover o objeto da memória, você deve atribuir ao objeto o valor **Nothing**.
Exemplo:

```
<%
   MinhaConexao.close
   Set MinhaConexao = Nothing
%>
```

Se você não fechar o objeto **Connection**, ele será fechado automaticamente quando o arquivo .asp terminar de ser executado pelo Servidor. Se houver a necessidade da conexão ficar aberta durante toda uma sessão ou aplicação, você pode armazenar o objeto **Connection** como uma variável dos objetos **Session** ou **Application**.

Quando devo abrir ou fechar um objeto Connection?

É muito importante pensar quando deve-se abrir uma conexão com o banco de dados, pois a origem de dados possui um limite de conexões. Quanto mais conexões existirem simultaneamente, é mais provável que outra aplicação que tente abrir a mesma origem de dados tenha o acesso negado, justamente pelo limite de conexões. Por isso, é sempre importante verificar a necessidade de uma conexão permanecer durante toda a aplicação ou sessão.

Além do mais, o pedido de abertura de uma conexão com a origem de dados é um serviço que exige múltiplos ciclos da CPU. Abrir e fechar conexões repetidamente pode causar grande demanda no servidor. Se uma conexão será utilizada novamente em um espaço de tempo relativamente curto, é uma boa idéia não fechá-la.

Recordset

O objeto Recordset representa um conjunto de registros recuperados de uma consulta a um banco de dados. Utilizaremos este objeto para realizar todas as ações num banco, como recuperar, gravar, alterar ou excluir registros.

Criando um RecordSet

Para criar um novo RecordSet, utilizaremos o método *CreateObject* (objeto Server). Observe o exemplo:

```
<%
  Set MeuRecordSet = Server.CreateObject ("ADODB.RecordSet")
%>
```

No exemplo acima, criamos um objeto RecordSet chamado *MeuRecordSet*. Agora, é necessário "preenchê-lo" com dados e "abrí-lo". Para isso, utilizaremos seu método **Open**. Veja a sintaxe:

NomeRecordSet.Open *Origem, Conexão, TipoCursor, TipoTrava*

Onde:
NomeRecordSet: Nome dado ao objeto RecordSet criado com o método CreateObject.
Origem: O comando SQL o qual se vai "carregar" o RecordSet.
Conexão: O nome da conexão que será utilizada (objeto Connection criado)
TipoCursor: Um RecordSet contém um cursor que aponta para o registro atual. Quando se chama um *MoveNext* ou qualquer outro método de navegação entre registros, o cursor é movido. Existem quatro tipos de cursores. São eles:

Forward Only (0): É o cursor mais rápido. É usado somente para mover o cursor para frente e é somente leitura. Útil quando se deseja ler o registro uma única vez e exibir os dados.

Keyset (1): Relativamente rápido. Associa um valor-chave a cada um dos registros para uma navegação rápida. Move-se para frente e para trás e atualiza os dados. O usuário não conseguirá ver os registros adicionados por outros usuários. Útil quando se quer atualizar dados.

Dynamic (2): É o mais flexível dos cursores e o mais lento. Quando utilizado, é possível modificar registros e ver quaisquer modificações e inserções realizadas por outros usuários. Útil quando se deseja uma visão atual dos dados.

Static (3): Permite que se mova para frente e para trás e que se realize escrita e leitura de registros, mas não visualiza alterações, adições ou exclusões feitas por outros usuários. Útil quando é necessário manipular registros.

TipoTrava: Ao selecionar um registro (ou um conjunto deles) num banco de dados, o mesmo tem que ser travado para evitar inconsistências geradas a partir de tentativas de modificações simultâneas. Deste modo, devemos especificar qual tipo de trava utilizada para isso:

Read-Only (0): Os dados não podem ser alterados.

Pessimistic (1): O provedor faz o que é necessário para assegurar a edição com sucesso de um registro., trancando os registros no banco imediatamente antes da edição dos mesmos

Otimistic (2):O provedor só tranca os registros quando o método Update é chamado.

Batch Otimistic (3): Requerido para edição de blocos de registros.

Agora veja um exemplo:

```
<%
  Set Conex = Server.CreateObject (“ADODB.Connection”)
  Conex.Open “BancoDados”, “ ”, “ ”

  Set RS = Server.CreateObject ("ADODB.RecordSet")
  RS.Open "SELECT * FROM TabClientes", Conex, 3, 3
%>
```

Com o código acima, criamos a conexão *Conex* com o DSN *BancoDados*. Instanciamos um RecordSet *RS* e nele colocamos todos os registros da tabela *TabClientes* através de uma instrução SQL. O RecordSet criado utiliza como tipo de cursor o *Static* e como tipo de trava de registros o *Batch Optimistic*.

Propriedades BOF e EOF

As propriedades BOF (Beginning Of File) e EOF (End Of File) permitem que seja testada se a posição do registro atual encontra-se no começo do RecordSet (BOF) ou no fim do mesmo (EOF). Veja um exemplo:

```
(...)
sql = "SELECT * FROM Produtos WHERE CodProd = ' " & CodDigitado & " ' "
RS.Open sql, conexao, 3, 3
If RS.EOF then
   Response.Write "Código inexistente no cadastro!"
End if
%>
```

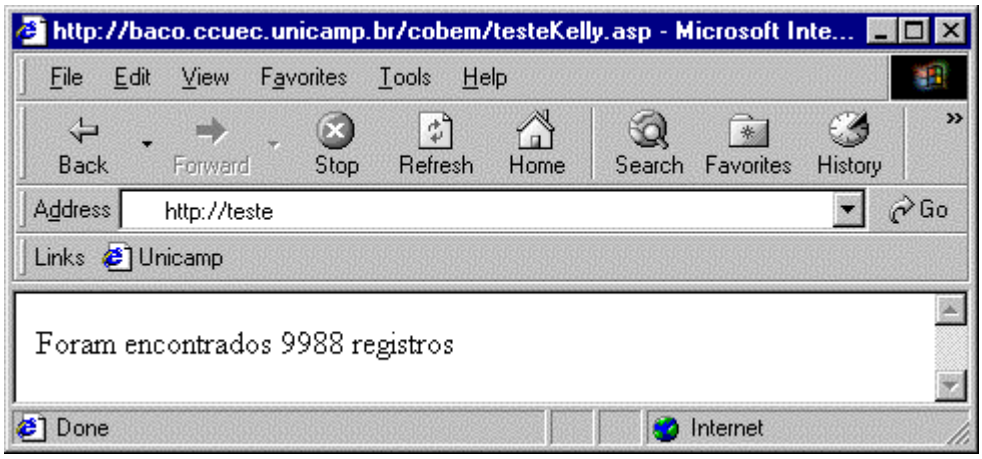
No exemplo acima, a instrução SQL cria um RecordSet *RS* com registros da tabela *Produtos* cujo campo *CodProd* seja igual à variável *CodDigitado*. Caso esta pesquisa não retorne resultado algum (ou seja, caso a cláusula WHERE não for satisfeita ou a tabela estiver vazia) o registro atual estará posicionado no fim do RecordSet. Desta forma, foi utilizada a propriedade EOF para emitir uma mensagem de parâmetro não encontrado.

Propriedade RecordCount

Esta propriedade contém o número de registros que um RecordSet possui. Observe um exemplo:

```
(...)
sql = "SELECT * FROM Produtos WHERE CodProd = ' " & CodDigitado & " ' "
RS.Open sql, conexao, 3, 3
Response.Write  "Foram encontrados " & RS.RecordCount & " registros"
%>
```

Um possível resultado obtido para este exemplo seria:



Navegação entre registros de um RecordSet

MoveFirst
Este método move o ponteiro do registro atual para o primeiro registro do RecordSet. Lembre-se que nesta posição, a propriedade *BOF* do RecordSet valerá *True*. O seguinte exemplo move o ponteiro para o primeiro registro do RecordSet RS:

```
RS.MoveFirst
```

MoveLast
Este método move o ponteiro do registro atual para o fim do RecordSet. Nesta posição, a propriedade *EOF* do RecordSet valerá *True*. O exemplo abaixo move o ponteiro para o final do RecordSet RS:

```
RS.MoveLast
```

MoveNext
O método *MoveNext* move o ponteiro do registro atual para o próximo registro do RecordSet. Lembre-se: se o ponteiro estiver no fim do RecordSet (EOF), ao utilizar este método um erro ocorrerá. Veja o seguinte exemplo, que desloca de uma posição o ponteiro do RecordSet RS

```
RS.MoveNext
```

MovePrevious
MovePrevious move o ponteiro para o registro anterior ao atual. Atenção: se o ponteiro localizar-se no primeiro registro (BOF) o uso deste método acarretará num erro. O seguinte exemplo move o ponteiro para o primeiro registro do RecordSet RS:

```
RS.MoveFirst
```

Manipulando Registros

Para manipular (adicionar, excluir, editar) registros em uma tabela, devemos fazer uso de RecordSets. Para isso, podemos utilizar dois modos: através de *métodos do RecordSet* ou através de *Instruções SQL*.

Métodos do objeto RecordSet para manipulação de registros

Update

É através do métodoUpdate que concretizamos todas as alterações realizadas com os métodos de manipulação de RecordSets.
Sintaxe: NomedoRecordSet.Update
Onde *NomedoRecordSet* é o nome dado ao objeto do tipo RecordSet que está sendo manipulado.

AddNew

Este método permite que se adicione um novo registro ao RecordSet.
Sintaxe: NomedoRecordSet.AddNew
NomedoRecordSet("campo1tabela") = valor1
NomedoRecordSet("campo2tabela") = valor2
NomedoRecordSet.Update

Onde *NomedoRecordSet* é o nome dado ao objeto do tipo RecordSet que está sendo manipulado, *campo1tabela* e *campo2tabela* são os nomes dos campos da tabela do banco de dados e *valor1* e *valor2* são os valores a que se deseja atribuir a estes campos. Para visualizarmos melhor estes conceitos, vejamos um exemplo:

```
(...)
varnome = Request.Form("txtNome")
vartelefone = Request.Form("txtTelefone")
RS.MoveLast
RS.AddNew
  RS("nome") = varnome
  RS("telefone") = vartelefone
RS.Update
%>
```

Com o código acima, gravamos no banco de dados (que está associado à conexão do RecordSet RS) um novo registro (de campos são *nome* e *telefone)* com os valores armazenados nas variáveis *varnome* e *vartelefone* (que vieram de um formulário HTML).

Delete

O método Delete exclui o registro atual.
Sintaxe: NomedoRecordSet.Delete
Onde *NomedoRecordSet* é o nome dado ao objeto do tipo RecordSet que está sendo manipulado. Observe o exemplo:

```
(...)
RGDigitado = Request.Form("txtRG")
sql = "SELECT * FROM Clientes WHERE RG = ' " & RGDigitado & " ' "
RS.Open sql, conexao, 3, 3
If Not RS.EOF then
  RS.Delete
  Response.Write "Registro excluído!"
Else
  Response.Write "RG inexistente!"
End if
%>
```

No exemplo citado, o registro que contém o campo *RG* procurado é excluído. Caso não exista o registro, é exibida uma mensagem de RG inexistente.

Manipulando registros através de instruções SQL

Para manipular registros através de instruções SQL, devemos utilizar o método **Execute** do objeto **Connection**.
Sintaxe: Set NomeRecordSet = NomeConexao.Execute (sql)
Onde *NomedoRecordSet* é o nome dado ao objeto do tipo RecordSet que será manipulado, *NomeConexao* é o nome dado ao objeto do tipo Connection existente na aplicação e *sql* é a instrução SQL a ser executada no RecordSet.
Veja um código exemplo de inserção de registro utilizando uma instrução SQL:

```
<%
 Set Conexao = Server.CreateObject("ADODB.Connection")
 Conexao.Open "BancoDados", "", ""
varcliente = Request.Form("txtCliente")
varmail = Request.Form("txtMail")
sql = "INSERT INTO Clientes (Cliente, EMail) VALUES (' "& varcliente &" ' , ' "& varmail &" ') "
set RS = Conexao.execute(sql)
%>
```

No exemplo acima adicionamos na tabela *Clientes* um registro (com campos *Cliente* e *EMail*) com os valores das variáveis *varcliente* e *valmail*, utilizando para isso o método Execute do objeto Connection criado, passando como parâmetro para este método a instrução SQL.

Veja agora um exemplo de exclusão de registros utilizando instruções SQL:

```
<%
 Set Conexao = Server.CreateObject("ADODB.Connection")
 Conexao.Open "BancoDados", "", ""
varcod = Request.Form("txtcod")
sql = "DELETE Clientes.* FROM Clientes WHERE ((Clientes.codigo)= ' " & varcod &" ') "
set RS = Conexao.execute(sql)
%>
```

No exemplo citado, são excluídos da tabela *Clientes* os registros cujo campo *codigo* seja igual ao valor da variável *varcod*.

A atualização de registros também é feita a partir do mesmo procedimento, alterando somente a instrução SQL. Veja:

```
<%
 Set Conexao = Server.CreateObject("ADODB.Connection")
 Conexao.Open "BancoDados", "", ""
varcod = Request.Form("txtcod")
sql = "UPDATE Clientes SET Clientes.fone = ' " varfone & " ' WHERE Clientes.cod = ' " & varcod & " ' "
set RS = Conexao.execute(sql)
%>
```

Com o código acima, atualizamos o campo *fone* do registro cujo campo *cod* possui valor igual à variável v*arcod*.

CAPÍTULO XI – SEGUNRANÇA

O **Microsoft Internet Information Server** (IIS) está integrado ao sistema operacional Microsoft Windows NT Server o que significa que o IIS possui as mesma características de segurança do NT.

Segurança Integrada

A arquitetura de segurança do Windows NT é utilizada por muitos componentes de sistemas com uma camada de autenticação para controlar o acesso a todos os recursos do sistema. O IIS está integrado ao modelo de segurança do NT e aos serviços do sistema operacional como file system e diretórios. Como o IIS utiliza o banco de contas de usuários do NT, os administradores não precisam criar contas separadas em cada servidor Web, e em uma Intranet, os usuários apenas logam em suas contas apenas uma vez. O IIS automaticamente utiliza as mesmas permissões de arquivos e grupos.

Gerenciando

As permissões para controle de acesso aos arquivos e diretórios podem ser configuradas graficamente pelo Windows NT, pois o IIS utiliza o Server Access Control Lists (ACLs): uma lista contendo todas as permissões de cada usuário.

As permissões configuradas para um servidor Web não são diferentes das permissões criadas para outros arquivos no servidor (permissões NTFS). Desta forma, estes arquivos podem ser acessados através de outros protocolos, como o FTP, NFS sem duplicar suas permissões.

Administradores não precisam manter bancos de contas de usuários duplicados. Todos os serviços de um servidor Intranet podem ser gerenciados a partir de uma ferramenta gráfica. O IIS e o Windows NT Server 4.0 formam uma plataforma que permite aos administradores criar novos usuários para acessar recursos do sistema, como páginas em HTML, arquivos compartilhados, impressoras, bancos de dados corporativos e outras aplicações através de ferramentas gráficas.

A integração Windows NT e IIS também permite a auditoria de sistema para maior monitoramento da segurança dos recursos utilizados. Por exemplo, tentativas de acesso de um usuário a um recurso não autorizado pode ser armazenado pelo Windows NT Event Log e visualizado pelas ferramentas administrativas do NT.

Controle de Acesso

Um dos itens mais importantes de segurança do IIS é o controle de acesso aos arquivos e aplicações no servidor. O IIS possui os seguintes itens para controle de acesso e segurança:

- Suporte para autenticação Windows NT Challenge/Response
- Acesso por IP das máquinas
- Habilidade para implementar restrições de acesso ao servidor e diretório virtual.
- Suporte para Windows NT File System (NTFS).
- Certificados digitais para clientes e servidores
- Filtros de Segurança

Autenticação e Autorização de Usuários

Como vimos, a segurança do IIS é integrada ao Windows NT File System. Tenha sempre em mente que para acessar qualquer recurso do NT é preciso uma senha e um usuário. Este recurso permite aos administradores gerenciar as contas, incluindo a sua auditoria e o armazenamento de

toda a atividade desempenhada em arquivos de logs, configuração de restrições e data de expiração das senhas.

Acesso Anônimo

No setup, o IIS cria uma conta anônima para conexões Web não autenticadas. Quando a segurança não é requisitada, o pedido é processado pelo servidor no contexto de segurança de uma conexão anônima. Esta conta pode ter acesso apenas a arquivos e aplicações que possui permissão para acessar.

Senha e User Name

As aplicações e arquivos que possuem o acesso restrito, necessitam que os usuários se identifiquem através de uma conta e senha para permitir a visualização dos dados. O IIS pode ser configurado para pedir a autenticação HTTP básica. Um prompt é mostrado para que o usuário possa digitar o nome de sua conta e sua senha, depois disso uma comparação é feita com o banco de contas existentes no Windows NT. Entretanto, estes dados são passados pela rede sem encriptação, podendo ser interceptados por um sniffer facilmente.

Segurança Windows Chanllege/Response

O IIS também suporta este tipo de autenticação do NT, que utilização a técnica de encriptação para autenticar os usuários. Na verdade, a senha nunca é passada pela rede. Uma vez que toda conexão é mapeada diretamente para a conta do usuário no Windows NT, os usuários da Internet apenas precisam efetuar o logon uma vez que será válido para todos os servidores e serviços no domínio do Windows NT.

Certificados Digitais

Um certificado verifica a identificação de um usuário do mesmo modo como fazemos a identificação de uma licença para um drive ou cartões de corporação. A rigorosidade desta identificação depende do nível de segurança requerido para a informação ou aplicação a ser acessada. O usuário digita a senha quando assina o certificado, e esta senha é requisitada toda vez que o usuário ativa a certificação. Porém, apenas a certificação não garante que o usuário que está tentando realizar o acesso é realmente a pessoa que tem o acesso à aplicação ou às informações. Importante lembrar que a senha deve ser secreta, ou seja, somente a pessoa que tem a permissão de acesso à aplicação deve sabê-la e ter a consciência de não revela-la à mais ninguém.

Uma autenticação baseada em certificados digitais requer um protocolo que esteja habilitado a “entender” este certificação tanto no cliente como no servidor. O certificado do servidor aparece para o cliente, desta maneira o cliente pode validar a identidade do servidor.

Quando o SSL é executado, é necessário que um servidor tenha este certificado. Opcionalmente, um servidor pode solicitar ao cliente este certificado. Um certificado para o servidor, contem o nome do Website, e o browser verificará se o endereço URL é o nome especificado no certificado.

Controle de Acesso utilizando filtros

O IIS possui um conjunto de APIs que podem ser utilizadas pelo programadores para criar filtros que autenticam usuários baseados em regras customizadas. Este recurso oferece aos administradores maior flexibilidade para o controle de acesso utilizando qualquer esquema de autenticação ou diretórios externos.

Recursos para o Controle de Acesso

Uma vez que os usuários estão autenticados, o IIS verifica se estes têm permissão para acesso aos arquivos e aplicações requisitadas.

Endereços IP

Para a Internet, cada servidor e cliente tem uma endereço Internet específico chamado “IP address”. O IIS pode ser configurado para garantir ou negar o acesso a uma aplicação para um específico endereço IP. Configurando o IIS desta forma, os administradores podem permitir apenas que um determinado domínio acesse determinada aplicação.

Permissão Windows NT File System (NTFS)

O objeto do NTFS é oferecer segurança para os servidores Web nos contextos Internet e Intranet. Permite que os administradores configurem permissões de usuários a arquivos individuais, não apenas para pastas ou diretórios. Utilizando este recurso nos arquivos e diretório disponibilizados pelo IIS, os administradores ajudam a verificação do usuário

Uma vez que a restrição do endereço IP do usuário é satisfeita, a conta do usuário ou a senha são validadas, e a permissão do diretório virtual é contemplada, o IIS tentará acessar o recurso solicitado pelo cliente (baseado na URL) utilizando o contexto de segurança da autenticação do usuário. Todos estes itens de validação garantem a verificação do acesso baseado nas permissões do NTFS, oferecendo aos administradores um controle maior dos recursos e informações disponibilizadas.

O Windows NT identifica cada usuário por uma identificação única de segurança global (SID) e não pelo nome da conta. É este SID que o NT utiliza para verificar as permissões dos usuários. Os nomes das contas foram criados para a interface com o usuário ficar mais amigável.

Quando uma conta é excluída, todas as entradas no ACL e associações de grupos também são excluídas. Os SIDs e a sincronização garantem que uma conta criada com o mesmo nome de uma que já tenha sido excluída, não receba as configurações de permissão da conta excluída.

Permissões no IIS

O IIS permite que o administrador configure permissões de apenas leitura e execução nos diretórios virtuais. Para cada pedido, o IIS examina a URL e o tipo de pedido feito e então, verifica se as permissões configuradas no diretório virtual são contempladas. Esta verificação assegura que os pedidos para leitura de arquivos configurados com apenas execução não sejam lidos e arquivos apenas de leitura, não sejam executados.

Auditoria

Uma das maneiras de determinar se um usuário está tentando acessar
O IIS suporta duas maneiras de logs. A primeira é o log padrão onde todos os pedidos de arquivos e objetos e erros gerados são armazenados. A segunda maneira, seria a utilização da ferramenta de administração do NT – **Event Viewer**.

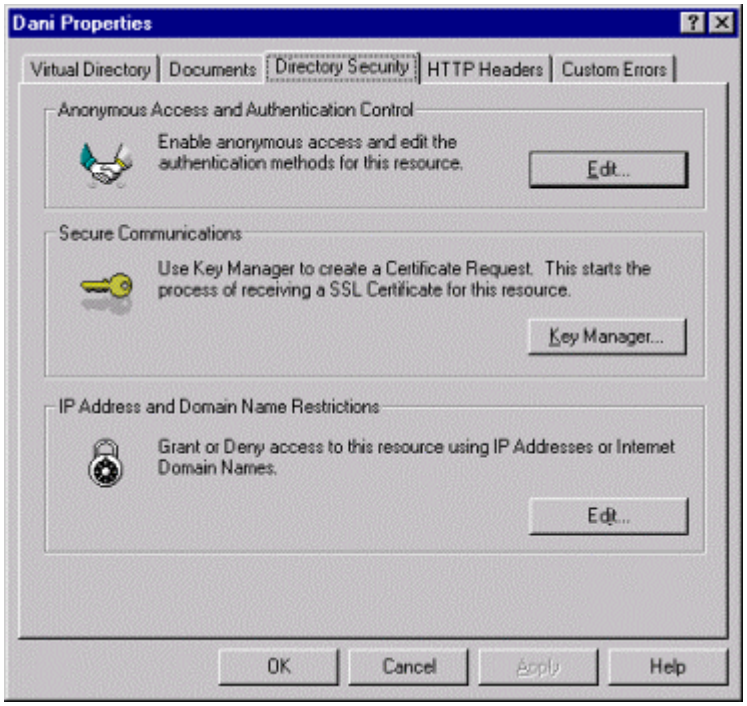
Com isso, um administrador pode obter as seguintes informações:

- Acesso a todos os arquivos do servidor.
- Tentativas de logon inválidas.
- Todos os logons efetuados.

Configurando os requisitos de segurança no IIS

Os requisitos de segurança no IIS devem ser configurados para cada diretório virtual. Este item está localizado dentro de suas propriedades. Para habilitar as propriedades de um diretório virtual, clique-o com o botão direito do mouse e selecione o item Properties.

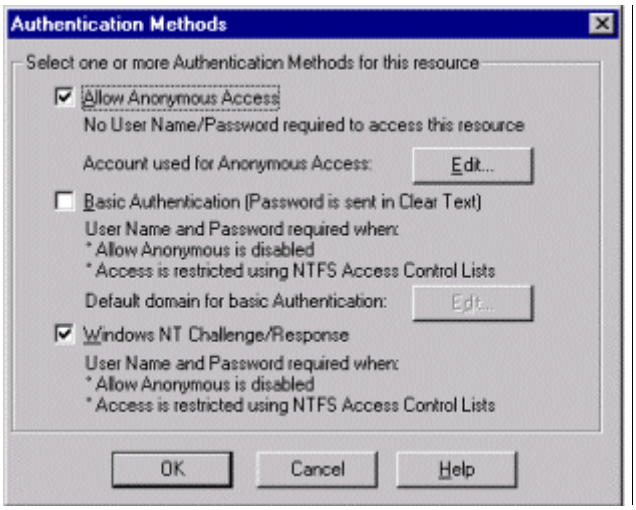
Será então aberta a janela de propriedades. Para visualizar os itens de segurança selecione a pasta **Directory Security**:



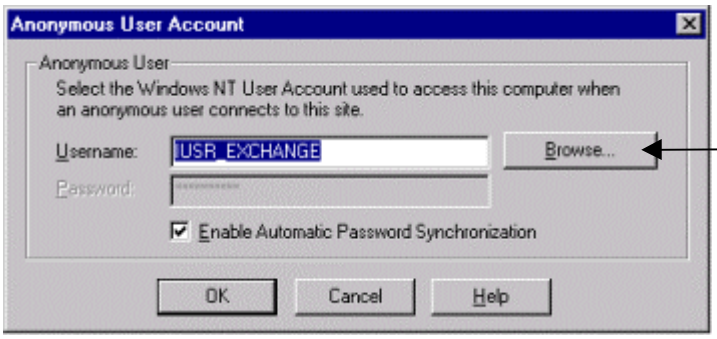
A tela acima será mostrada. Vamos descrever cada item desta página:

Anonymous Access and Authentication Control

Ao configurar este item, você poderá permitir o acesso anônimo, o acesso solicitando ao usuário sua identificação no Windows NT por encriptação ou sem encriptação.



- Primeiro item: **Allow Anonymous Access.** Pressionando o botão Edit ..., você pode especificar a conta que deve fazer o acesso anônimo:



Se você quiser modificar a conta, clique em Browse ... e selecione a nova conta.

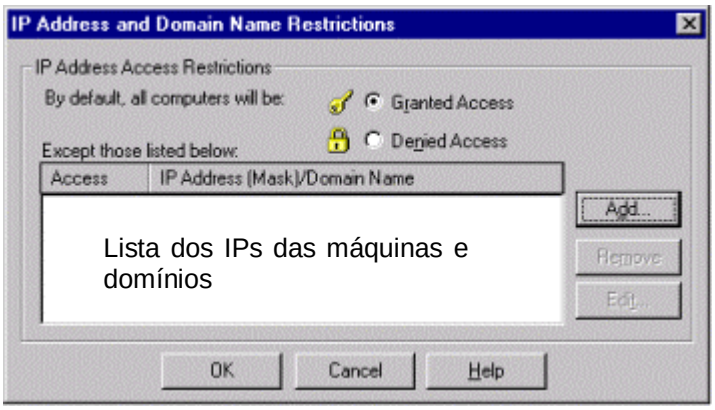
- Segundo item: **Basic Authentication**. Neste item será solicitado ao usuário sua identificação. É importante lembrar que a senha é passada pela rede sem encriptação.

Ao escolher o item **Basic Authentication**, uma mensagem de aviso é mostrada, alertando que é um tanto inseguro a utilização deste tipo de autenticação.
Para continuar, pressione **Yes**.

Nesta próxima tela, você deve especificar o domínio onde estão cadastrados os usuários que terão permissão de acesso:

Para selecionar um domínio, clique em Browse ...

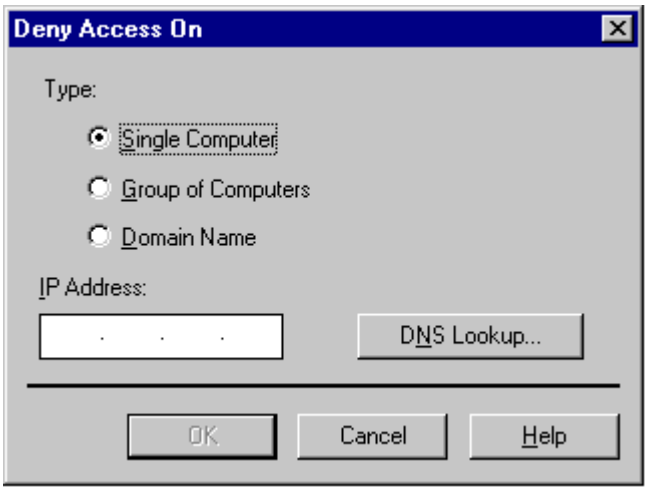
Terceiro item: **IP Address and Domain Name Restrictions**. Neste item, você estará configurando os IPs e domínios que terão acesso ao diretório.



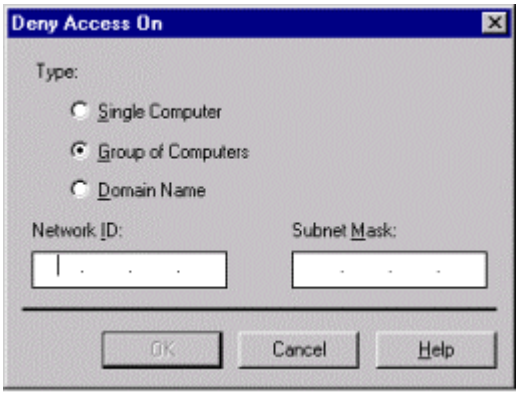
- Granted Access:
  Garante permissão para todas as máquinas e domínios, menos para as listadas no item **Except those listed below.**

- Denied Access:
  Desabilita o acesso a todos, menos para as máquinas e domínios listados no item **Except those listed below**.

Para adicionar IPs de máquinas e domínios, clique em Add ... A tela a seguir será mostrada:



- Single Computer
  Se este item for especificado, lhe será pedido o IP da máquina.



Se você escolher o item **Group of Computers**, será preciso especificar a identificação da rede a Ter o acesso e a máscara.

Se você escolher conceder permissão de acesso à um domínio, a mensagem de aviso abaixo será mostrada:

O aviso diz que restringindo o acesso pelo Domínio, é necessário utilizar o DNS reverso para cada conexão realizada. E este tipo de operação exige recursos da máquina.

Depois da mensagem acima, será mostrada a tela onde você deve especificar o nome do domínio a ter o acesso negado ou concedido.

Resumo

Neste capítulo aprendemos que ...

- A segurança do IIS é integrada à do Windows NT.
- Os itens mais importantes de segurança são:
  - Suporte para autenticação Windows NT Challenge/Response
  - Acesso por IP das máquinas
  - Habilidade para implementar restrições de acesso ao servidor e diretório virtual.
  - Suporte para Windows NT File System (NTFS).
  - Certificados digitais para clientes e servidores
  - Filtros de Segurança
- Os requisitos de segurança estão localizados nas propriedades do Diretório Virtual e do Web Site, na pasta Directory Security.
- Existem vários tipos de identificação do usuário: anônimo, com senha sem encriptação e com senha e encriptação.
- Podemos restringir que apenas um domínio ou IPs de máquinas possam acessar determinada aplicação.