



Curso de Introdução a Plataforma .Net Framework SDK



Aprenda de uma forma simples , objetiva e direta os principais conceitos da **Plataforma .Net Framework SDK**.

O curso apresenta o **ASP.Net** a primeira ferramenta RAD produzida para o desenvolvimento de aplicações baseadas na Web. Além de apresentar as linguagens C# e Visual Basic.Net.



Instrutores:

Breno Gontijo – brenogt@tremdoido.com.br

Cláudio Júnior – claudiojunior@globo.com

Índice Analítico

Capítulo 1 – Introdução a Plataforma .Net Framework SDK	1
Introdução	1
Suítes	1
Termos da Plataforma	2
 Capítulo 2 – ASP.Net	 4
Introdução	4
Do HTML ao ASP.Net	4
Como o ASP.Net Funciona	6
Criando Páginas ASP.Net	7
Mantendo o Estado de uma Página ASP.Net	11
ViewState	11
Importando Espaços de Nomes	13
Objeto SESSION	14
Vantagens Adicionais	15
 Capítulo 3 – Variáveis	 16
Variáveis na Linguagem C#	18
Inteiros	18
Lógicos	19
Caracteres	19
Ponto Flutuante	20
Decimal	20
String	21
Object	22
Conversão de Valores	22
Checked e Unchecked	24
Operadores	25
Enum	25
Struct	26
New	27
Arrays	29
Exemplos e Exercícios	30
Variáveis na Linguagem Visual Basic.Net	33
Inteiros	33
Lógicos	34
Caracteres	34
Ponto Flutuante	35
Date	35
String	36
Object	37
Conversão de Valores	37

Operadores	41
Array	42
Atribuindo Valores aos Arrays	42
Arrays Multidimensionais	43
As funções REDIM e ERASE	43
Exemplos e Exercícios	44

Capítulo 4 – Estruturas de Controle46

Estruturas de Decisão	46
Instrução if .. then .. else ..elseif	46
Instrução switch e case	48
Estruturas de Repetição	49
Instrução For	49
Instrução While	50
Instrução Do While	50
Instrução Do Loop	51
Exemplos e Exercícios	52

Capítulo 5 – Formatação57

Formatação de Números	57
Formatação de Números na Linguagem C#	57
Formatação de Números na Linguagem VB.Net	60
Exemplos	64
Formatação de Datas e Horas	65
Formatação de datas e Horas na Linguagem C#	65
Exemplos	69
Formatação de datas e Horas na Linguagem VB.Net	70
Exemplos	76

Capítulo 6 – WEB FORMS78

Principais Eventos do Web Forms	79
Page_Load	79
OnClick	80
OnSelectedIndexChanged	81
Principais Componentes do ASP.Net	83
TextBox	85
Label	86
Button	86
DropDownList	87
ListBox	89
CheckBox	89
CheckBoxList	90
RadioButton	91
RadioButtonList	91
HyperLink	92
Panel	93
Exemplos e Exercícios	94

Capítulo 7 – Controle de Validação	98
RequiredFieldValidator	99
CompareValidator	100
RangeValidator	101
CustomValidator	103
RegularExpressionValidator	103
ValidationSummary	105
Exemplos e Exercícios	106
 Capítulo 8 - ADO.NET	 109
Vantagens do ADO.NET	109
OleDb	110
SQL	111
DataSet	111
Conectando com um Banco de Dados	112
Lendo um Banco de Dados	113
Incluir Dados	115
Excluir Dados	116
Atualizar Dados	117
DataGrid	118
Exemplos e Exercícios	121
 Capítulo 9 – Componentes	 126
Vantagens	126
Problemas	126
Namespaces	127
Criando um Componente	128
Usando o Componente	129
Exemplos e Exercícios	130
 Capítulo 10 - Web Services	 133
SOAP	134
Criando um WebService	134
Usando o WebService	135

Capítulo 1 - Introdução a Plataforma .Net Framework

INTRODUÇÃO

A Plataforma Microsoft.NET oferece uma alternativa de ambiente para produzir e executar aplicações web, rodando-as em PCs, micros de mão e outros dispositivos, como telefones celulares. O plano da Microsoft é tornar a infra-estrutura dessa plataforma amplamente disponível. Tanto que ela já pode ser baixada em seu site e deverá fazer parte das próximas versões do Windows.

A Plataforma .NET é também a principal arma com a qual a Microsoft tenta marcar posição no concorridíssimo mercado dos Serviços Web (Web Services) - nome dado a programas ou componentes que devem ser utilizados na Internet.

Estes serviços on-line são a pedra de toque da Internet, tal como os estrategistas das grandes empresas a imaginam num futuro próximo.

Por meio de serviços web, empresas trocarão informações e farão negócios. Aplicações que rodam num local poderão usar módulos localizados num servidor remoto, consolidando um modelo de computação distribuída. Residentes em servidores web, esses serviços podem fornecer produtos finais - por exemplo, documentos e informações - ou desempenhar tarefas específicas, como realizar cálculos e autenticar transações. Espera-se, assim, que os sites operem de forma integrada, gerando benefícios para empresas e indivíduos. Na essência, essa visão dos serviços web é hoje compartilhada por grandes nomes como IBM, Sun e Oracle, e todos têm iniciativas nessa área.

Uma característica central da Plataforma .NET é aderir aos padrões da Internet sem abrir mão de procedimentos já consagrados no Windows. Para isso conta com o Visual Studio.NET, suíte de programação definida pela Microsoft como "especialmente voltada para a rápida construção e integração de Web Services".

O produto incorpora as linguagens Visual Basic, Visual C++ e Visual C# ("CSharp"), todas com sobrenome .NET. Linguagens tradicionais, as duas primeiras sofreram ajustes para a nova plataforma, enquanto o C# começa do zero.

SUÍTES

A suíte, que já está no mercado brasileiro, é oferecida em três pacotes diferentes: Enterprise Architect, Enterprise Developer e Professional. O primeiro é o mais completo e inclui, além das três linguagens, ferramentas para depuração e modelagem, desenvolvimento em grupos e todos os servidores do Windows. O Enterprise Developer, mais simples, não tem, por exemplo, os recursos de modelagem. Mais voltada para o programador individual, a edição Professional não traz servidores nem itens de trabalho em grupo.

TERMOS DA PLATAFORMA

CLR - Sigla de Common Language Runtime. Base comum a todas as linguagens .NET, o CLR é o ambiente que gerencia a execução de código escrito em qualquer linguagem. Faz parte do Framework.

FRAMEWORK - É o modelo da plataforma .NET para construir, instalar e rodar qualquer aplicação, no desktop ou na Internet. Para executar um programa .NET, é preciso ter o Framework instalado.

IDE COMPARTILHADO - Ambiente integrado de programação (Integrated Development Environment) do Visual Studio.NET. Diferentes linguagens usam o mesmo editor de código e depurador e compilam executáveis na linguagem MSIL. Além das linguagens da Microsoft, já há mais de 20 outras (Perl, Cobol, Pascal, etc) que podem usar esse ambiente.

MSIL - Microsoft Intermediate Language. Quando se compila uma aplicação .NET, ela é convertida para uma linguagem intermediária, a MSIL, um conjunto de instruções independentes de CPU. Na hora de executar o programa, um novo compilador, chamado Just-in-time (JIT) Compiler, o converte para o código nativo, ou seja, específico para o processador da máquina.

MANAGED CODE - Código administrado, ou seja, código escrito para rodar com o runtime do VS.NET. No VS.NET, somente o C++ produz programas que não dependem do runtime, o chamado *Unmanaged code*.

SOAP - Sigla de Simple Object Access Protocol, ou protocolo simples de acesso a objetos. O SOAP é um padrão aberto, baseado em XML, criado pela Microsoft, Ariba e IBM para padronizar a transferência de dados entre aplicações. Pode ser usado em combinação com vários outros protocolos comuns da Internet, como HTTP e SMTP.

UDDI - Iniciais de Universal Description, Discovery and Integration, é uma espécie de páginas amarelas para web services. Na UDDI, empresas expõem seus serviços para que outras possam utilizá-los.

WEB SERVICES - programa completo ou componente de software residente num servidor web.

XML - Sigla de Extensible Markup Language, o XML é uma linguagem baseada em tags semelhante ao HTML. Sua principal característica é a extensibilidade. Quem emite um documento XML pode criar tags personalizadas, que são explicadas num documento anexo, que tem extensão XSD.

XSD - Sigla de XML Schema Definition. Arquivo associado a um documento XML que descreve e valida aos dados no documento. Assim como as linguagens de programação, os XSDs aceitam dados de diferentes tipos, como números, data e moeda.

XML WEB SERVICES - Blocos fundamentais para a criação de sistemas de computação distribuída na Internet. Um serviço web é uma porção de código localizada num servidor web e que pode ser utilizada por uma aplicação qualquer. O web service pode produzir documentos ou procedimentos. Uma das características centrais dos web services é serem baseados em padrões abertos.

WSDL - Web Service Description Language. Submetida à W3C - o órgão padronizador da Internet - . A linguagem WSDL define regras baseadas em XML para descrever serviços web.

Capítulo 2 - ASP.NET

Introdução

A Microsoft sabendo da dificuldade que os desenvolvedores têm para construir sites complexos para Web, resolveu criar uma nova plataforma de desenvolvimento. É aí que aparece o ASP.Net, uma solução poderosa, produtiva, e fácil de usar que vai muito além de páginas HTML convencionais.

O ASP.Net é uma nova versão do ASP (Active Server Pages), é uma plataforma unificada de desenvolvimento da Web, que fornece os serviços necessários para os desenvolvedores construírem conjuntos de aplicações da Web através de um novo modelo de programação, a arquitetura de três camadas.

O ASP.Net vai revolucionar o desenvolvimento para a Web, pois é a primeira ferramenta RAD (Rapid Application Design) para o desenvolvimento de aplicativos específicos para a Web, por isso podemos afirmar que todo o ciclo de desenvolvimento será o mesmo de outras tecnologias como – Visual Basic tradicional e Delphi.

Os aplicativos ASP.Net rodam sob o Servidor Web da Microsoft IIS (Information Internet Server) com algumas rotinas suplementares instaladas.

Apesar do ASP.Net rodar somente no Servidor Web **IIS** da Microsoft, as páginas são acessíveis em qualquer navegador, mesmo que não tenha sido desenvolvido pela Microsoft, e em qualquer Sistema Operacional, pois aplicação dependerá apenas do Browser, ou seja, do navegador.

Se repararmos o processo de evolução para a criação de aplicativos para a Web, vamos observar que o grande intuito desta nova tecnologia é resolver os problemas encontrados no ciclo de desenvolvimento das ferramentas atuais.

Para quem deseja aplicar esta nova tecnologia, é importante ressaltar que não é necessário entender de outras tecnologias como – JavaScript, HTML, VbScript, ou mesmo o ASP tradicional, pois o ASP.Net tem como base a inspiração em ferramentas RAD como – VB Tradicional e o Delphi, por isso podemos afirmar que o enfoque é o desenvolvimento de software. Isso tudo representa uma nova maneira de analisar o desenvolvimento de software para a Web.

Do HTML ao ASP.NET

Originalmente, os sites para a Web eram muito simples. Havia páginas somente em HTML, por isso dizemos que eram aplicações estáticas. O que funcionava muito bem para alguns sites como – Jornais, Exposição de Catálogos e Materiais Didáticos, dentre outros.

O HTML evoluiu, já era possível criar uma interatividade com o usuário, mas um conteúdo dinâmico ainda estava faltando. Então o processamento de servidor foi introduzido, surgindo daí o modelo cliente/ servidor.

A Internet trabalha no **modelo cliente/servidor**. Isto é, dois computadores trabalham juntos, enviando as informações de um lado para outro, para realizar uma tarefa. Neste cenário as informações estão contidas no servidor, o cliente apenas faz a solicitação daquilo que lhe é conveniente.

Quando o cliente envia uma solicitação de informações ao computador servidor, este então processa o pedido e retorna as informações de solicitação em HTML.

Esse paradigma é o **modelo de solicitação/resposta**, e é parte integrante do modelo cliente/ servidor.

Este modelo permite que o servidor retorne conteúdo dinâmico ao cliente, como dados em banco de dados e o resultado de cálculos.

Este fluxo de trabalho é o seguinte:

1. O Cliente (Navegador Web) localiza o servidor da Web por URL (como www.globo.com).
2. O Cliente solicita uma página (como [cadastro.asp](#))
3. O Servidor examina a solicitação e processa a saída codificada em HTML.
4. O Cliente recebe o documento e o exibe.

Neste cenário, o servidor não tem a mínima idéia do que o cliente esta fazendo, a menos que este faça outra solicitação.

O ASP.Net trabalha em outro modo de comunicação entre clientes e servidores Web, que é conhecido como **modelo baseado em eventos**. Isso quer dizer que o servidor espera algo acontecer no cliente. Quando ocorre, o Servidor toma medidas para realizar alguma funcionalidade.

Então podemos afirmar que o Servidor vai responder a suas ações. Se o cliente clica em uma imagem ou seleciona um CheckBox, o Servidor exerce uma ação.

O ASP.Net funciona assim – detecta ações e as responde. Para isso o ASP.Net conta com um processamento inteligente do lado cliente para simular um modelo baseado em eventos.

Você deve estar questionando então, como o ASP.Net sabe o que está acontecendo no cliente já que um script do lado cliente não pode interagir com o código do lado servidor, e sabendo também que a única maneira de o cliente comunicar-se como servidor é durante uma solicitação.

Bem, o ASP.Net inteligentemente limita esse problema. Ao utilizar o script do lado cliente, o ASP.Net fornece informações sobre o que o cliente está fazendo as solicitações, é como se o ASP.Net coloca-se alguns espiões no cliente, de modo que o servidor seja informado sobre tudo que está acontecendo do lado do cliente.

Os espiões do ASP.Net são os scripts do lado do cliente, que não podem interagir com o lado servidor, mas podem transmitir mensagens por meio de postagens para o servidor.

Portanto, o ASP.Net faz essa união com o servidor e o cliente.

Este conceito que o ASP.Net implementa facilita a vida o programador, que não precisa mais se concentrar em direcionar solicitações e respostas, o programador fica livre para se concentrar na construção da lógica.

Você pode reagir às ocorrências do usuário imediatamente em vez de esperar que os formulários sejam submetidos. E pode conhecer a estrutura da interface com o usuário e como lidar com elas antecipadamente.

O ASP.Net realmente facilita a vida dos desenvolvedores.

Como o ASP.Net Funciona

Assim como o VB e o Delphi, o desenvolvimento é feito sobre formulários (páginas **.aspx** + fonte em linguagem de alto nível).

Um formulário está associado a um arquivo com a extensão **.aspx** e também a um fonte de linguagem de alto-nível, como – VB.Net, C#, J#, entre outras.

Sobre os formulários são adicionados componentes que têm propriedades, métodos e eventos. Por isso podemos afirmar que esta tecnologia é orientada a objetos.

Este modelo é de fácil entendimento e muito mais produtivo.

O ASP.Net é também mais rápido que os scripts como o JavaScript, e tecnologias como o antigo ASP.

Isso ocorre porque seus programas são compilados, o que dá mais velocidade e segurança de código-fonte.

Entretanto devemos observar que, em vez de compilar em algo que o computador entenda, o desenvolvedor o compila em uma linguagem intermediária, chamada Microsoft Intermediate Language (**MSIL**).

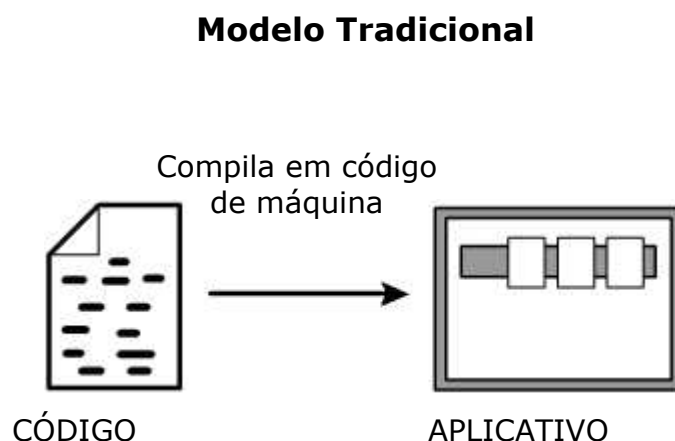
As páginas ASP.Net são compiladas em MSIL. Ao compilar em MSIL, suas aplicações criam algo chamado **metadados**. Que são as informações descritas sobre o seu aplicativo.

Então, quando você quiser executar seu programa, a CLR (Common Language Runtime) assume e compila o código mais uma vez na linguagem nativa do computador.

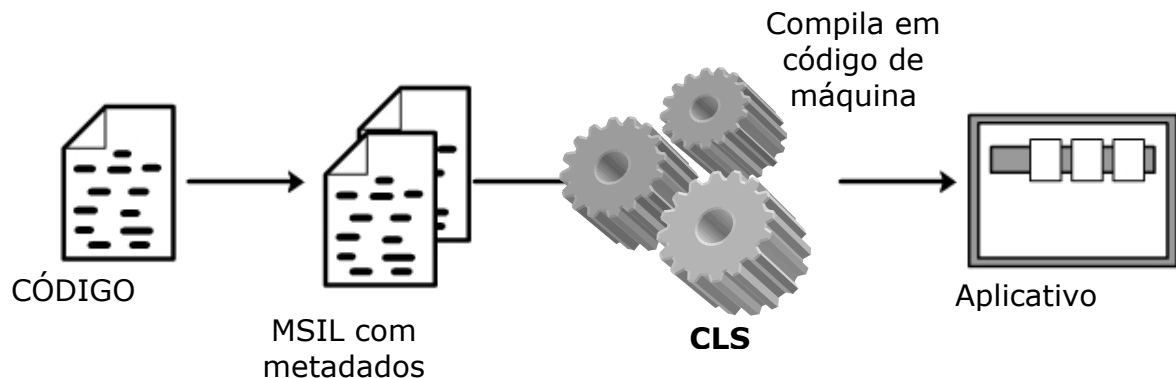
Dessa maneira, a MSIL pode se comunicar com qualquer tipo de computador. A CLR pode falar muitas linguagens de computador diferentes, como – C#, VB.Net, J#, Cobol.Net, Pascal.Net, entre outras, além de fazer toda a compilação.

Acompanhe na figura 2.1 o esboço da estrutura tradicional de um aplicativo versus a .Net Framework.

Figura 2.1 – Modelo Tradicional versus Modelo .Net Framework.



Modelo .Net Framework



A CLR utiliza os metadados para descobrir como executar o aplicativo.

Com metadados não há nenhuma necessidade de requerer informações sobre os aplicativos a serem armazenados em um registro, como era realizado no modelo tradicional, porque todas as informações necessárias são armazenadas com os arquivos do aplicativo; assim, qualquer alteração que você fizer será colocada em efeito automaticamente. Imagine instalar um novo aplicativo somente copiando alguns arquivos!

O código que trabalha com CLR é chamado **código gerenciado**. Esse nome é dado ao CLR pelo fato dele gerenciar sua execução e fornecer benefícios (como gerenciamento de recursos, tratamento de erros, recursos de segurança, controle de versão, etc) sem o programador ter que construí-lo manualmente. O código que é executado fora da CLR é conhecido como código não-gerenciado.

Criando Páginas ASP.Net

As páginas ASP.Net são simplesmente texto puro, como arquivos em HTML. Para que as páginas possam ser executadas e mostradas, você precisa ter instalado o .Net Framework e um servidor Web(IIS).

As páginas ASP.Net têm extensão **.aspx**, então quaisquer arquivos que você quer que o servidor interprete como um aplicativo ASP.Net deve terminar em .aspx, como – index.aspx.

Mas antes de apresentar um aplicativo na maneira ASP.Net, vamos mostrar a criação e execução de um programa em C# na maneira convencional.

Maneira Convencional

Vamos apresentar um exemplo de um programa não interativo, que apenas escreve as palavras – MEU PRIMEIRO EXEMPLO!, na tela do monitor.

Existem no mercado muitas ferramentas de desenvolvimento para os aplicativos .Net, como – Microsoft Visual Studio.Net e Macromedia Dreamweaver MX. Neste curso utilizaremos o famoso Bloco de Notas para desenvolver nossos aplicativos. Teremos com certeza mais trabalho em desenvolver nossos aplicativos com o bloco de notas, mas teremos a vantagem de não ficaremos presos a uma ferramenta.

Siga os seguintes passos para a criação e execução deste exemplo:

1. Abra o bloco de notas e digite as linhas abaixo:

```
using System;

class Primeiro {

    public static void Main( ) {

        // Parte renderizável
        Console.WriteLine(VarMensagem);
    }

    // Parte programática
    static String VarMensagem = "Meu Primeiro Exemplo!";
}
```

2. Crie um diretório (digamos com o nome: CursoNet) e nela grave o arquivo digitado acima com o nome: **Primeiro.cs**.
3. Abra a janela de comando (janela do DOS) e através o comando **cd** dirija-se para o diretório criado anteriormente onde você salvou o arquivo.
4. Para compilar o programa, digite:

csc Primeiro.cs

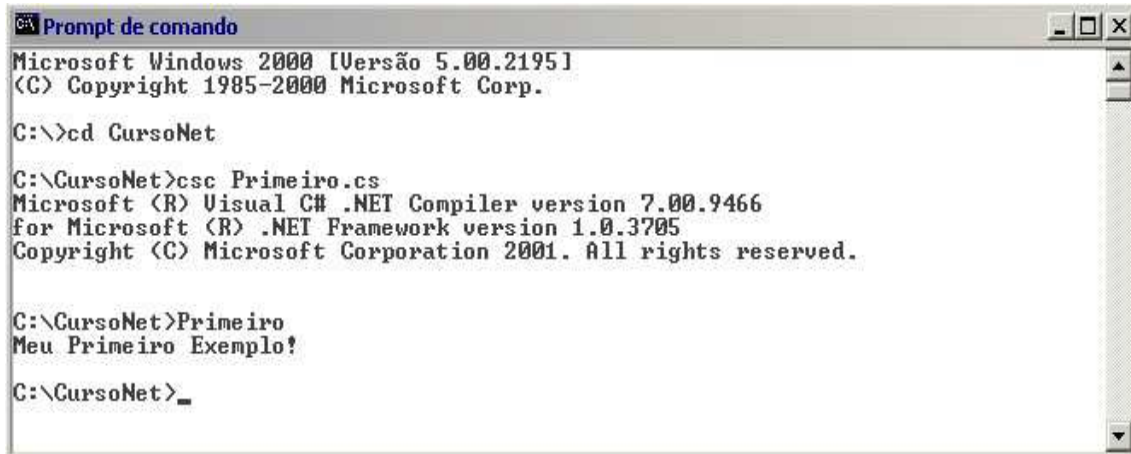
Será criado um arquivo: **Primeiro.exe**

5. Para disparar o programa basta digitar:

Primeiro

6. O programa começa a ser executado, o momento é chamado de **init**.
7. É feito o **render** (desenho na tela do monitor) do que o programa previa que deveria ser renderizado.

A figura 2.2 apresenta os passos seguidos.

Figura 2.2 – acompanhe a compilação e execução do programa.

```
Prompt de comando
Microsoft Windows 2000 [Versão 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd CursoNet

C:\CursoNet>csc Primeiro.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

C:\CursoNet>Primeiro
Meu Primeiro Exemplo!

C:\CursoNet>
```

Maneira ASP.Net

Se você já instalou o .Net Framework SDK 1.0 em seu computador, então você está pronto para criar seu primeiro programa na tecnologia ASP.Net.

Este nosso primeiro exemplo tem o mesmo objetivo do programa desenvolvido para a maneira convencional, o programa exemplo vai renderizar na tela do monitor de um cliente a mensagem - Meu Primeiro Exemplo! – que vem de um computador servidor.

Para o exemplo usaremos apenas um único computador. Ou seja, o computador servidor e o computador cliente serão o mesmo. Para que o cliente então possa acessar o servidor (virtualmente falando), usaremos o endereço: <http://127.0.0.1> no browser, ou também o endereço <http://localhost>.

Basta acompanhar os passos abaixo para que o exemplo funcione:

1. Abra o bloco de notas e digite as linhas de código do programa.

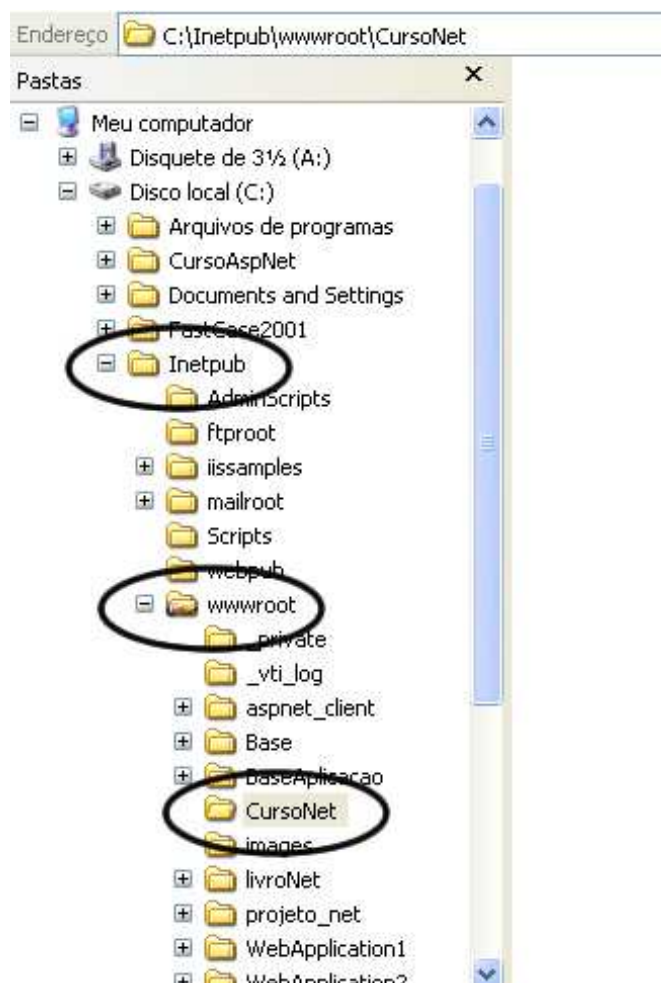
```
<html>
<!--Parte RENDERIZAVEL -->
<form runat="server">

  <asp:Label id="saida"
    Font-Name="Arial"
    Font-Italic="True"
    runat="server"/>

</form>

<!--Parte PROGRAMATICA -->
<script language="C#" runat="server">
  private String VarMensagem = "Meu Primeiro Exemplo!";
  public void Page_Load(Object sender, EventArgs ea) {
    saida.Text = VarMensagem;
  }
</script>
</html>
```

2. Vamos criar um diretório com o nome CursoNet, sob o path de diretórios já existentes(são diretórios ligadas ao IIS), onde o padrão é: c:**Inetpub\Wwwroot**, e salve o arquivo digitado acima com o nome de: primeiro.aspx.



3. Abrir o browser e colocar o seguinte endereço :

<http://127.0.0.1/cursonet/primeiro.aspx>

ou

<http://localhost/cursonet/primeiro.aspx>

Este passo acontece no cliente!

4. O programa é compilado automaticamente por um compilador Just In Time e depois executado. Então o programa é disparado automaticamente e começa a ser executado. Este momento é chamado de **init**.

Este passo acontece no servidor!

5. É realizado o **render** (desenho na tela do monitor do cliente) do que o programa previa que deveria ser renderizado.

Este passo acontece no servidor e no cliente!

Mantendo o Estado de uma Página ASP.Net

A manutenção de estado dos componentes quando existe umPostBack (um post para a própria página) é automática.

Isso é possível porque quando uma determinada página **.aspx** é chamada pela primeira vez é criado na memória do servidor um pool de algumas instancias da classe que tem o **nome da página** e que é herdeira de uma classe **Page**.

Novas solicitações daquela mesma página, seja ela pelo mesmo cliente ou por outro, não criarão, ao menos que o arquivo sofra alguma mudança, novas instancias desse objeto.

O que é criado junto com a primeira solicitação daquela página é um objeto da classe **ViewState**, que por sua vez cria uma referencia àquele dado objeto herdeiro de **Page**.

Quando uma página é reenviada para o servidor começa um novo ciclo chamado de **ROUNDTRIP** que consegue localizar aquela sua instancia de **ViewState** e remonta um objeto tipo **Page** idêntico ao anterior.

Este esquema desenvolvido faz parecer como se cliente tivesse um objeto tipo **Page** permanentemente dele, o tempo todo conectado, o que não é verdade.

Isso faz com que os componentes utilizados na aplicação consigam manter automaticamente os seus valores.

A informação de estado em si pode ser armazenada em diversos locais. O mais comum é na memória do servidor, como o Asp tradicional faz hoje, mas podemos também armazenar informações de estado em um servidor de estado, no SQL Server ou até mesmo escrevendo código para exercer tal funcionalidade.

ViewState

O **ViewState** descreve o aspecto de um objeto em um dado momento.

Dizemos que um aplicativo que monitora essas informações mantém (ou monitora) o estado.

Os formulários Web em HTML não têm esse poder, isso porque a Web é uma mídia sem informações de estado.

O ASP.Net monitora automaticamente o **ViewState** para você. Isso pode ser feito porque o ASP.Net gera saída de campos de formulários ocultos em HTML sempre que você instruir um formulário a executar no servidor.

Para que isto ocorra é preciso que o desenvolvedor instrua o formulário e seus componentes a executar no servidor, portanto devemos incluir a clausula **runat** em cada um destes componentes:

Acompanhe a sintaxe:

```
runat="server"
```

Por exemplo, veja a linha a seguir escrita no servidor:

```
<form runat="server">
```

Isso envia o código a seguir em HTML para o navegador:

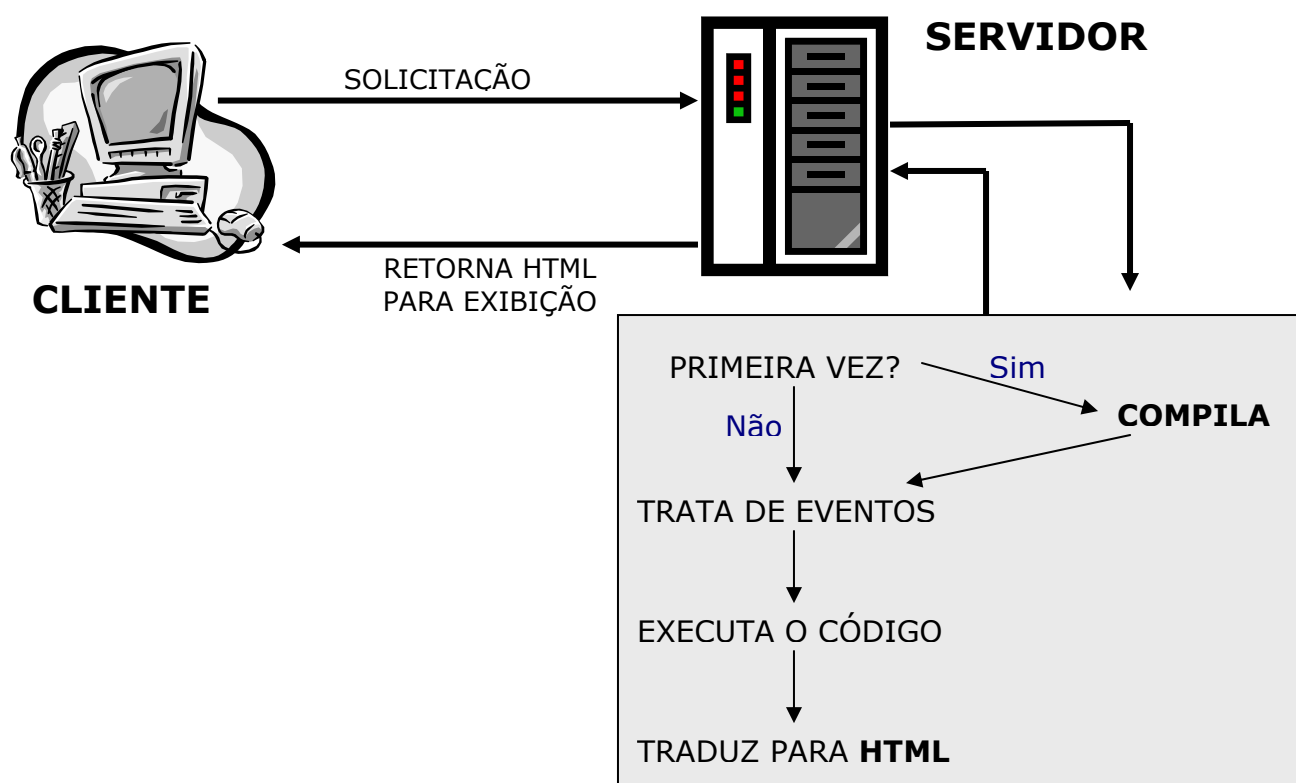
```
<form name="ctr2" method="post" action="primeiro.aspx" id="ctr2">
```

```
<input type="hidden" name="_VIEWSTATE" value="YJ5373JdyAdfh7JDhs" />
```

O campo oculto do formulário com o nome - **_VIEWSTATE**, possui um valor em uma string de caracteres, que é a forma que o ASP.Net utiliza para reconhecer cada controle. Quando o formulário é enviado, o ASP.Net recupera esse valor automaticamente e utiliza para preencher as informações do formulário novamente. Este esquema o ASP.Net utiliza para escrever lembretes para si próprio nas páginas que são enviadas para o cliente.

Na figura 2.3 podemos acompanhar o fluxograma do ASP.Net, desde o instante da solicitação ao servidor até à exibição do código para o cliente.

Figura 2.3 - Fluxograma do ASP.Net, da solicitação à exibição.



Quase todo o HTML da página é gerado em tempo de execução pelos componentes. Por isso o componente gera HTML conforme o seu estado e o navegador usado naqueles instantes. Se necessário o componente gera JavaScript. É uma maneira de os componentes funcionarem independentes do navegador.

Em termos de desenvolvimento de ASP.Net, a CLR dá ao programador muito menos com que se preocupar. Ele gerencia a memória e oferece aos aplicativos ASP.Net muito mais desempenho e estabilidade. Com o melhor isolamento de falhas não é mais necessário a pratica de reinicializar o servidor de Web IIS ou qualquer outro servidor Web, se a aplicação cair. Por isso podemos dizer que o ASP.Net é muito mais estável e eficiente que os métodos anteriores de desenvolvimento para a Web.

Importando Espaços de Nome

Cada espaço de nome na .Net FrameWork é essencialmente uma coleção de bibliotecas. O ASP.Net possui suas próprias bibliotecas, mas, às vezes, esse conjunto não é suficiente. Portanto você pode desenvolver suas próprias bibliotecas, o que já é de costume na programação. Você pode acessar suas bibliotecas adicionais utilizando a palavra-chave Import.

Acompanhe um exemplo:

```
<%@ Import Namespace = "System.Globalization" %>  
  
<%@ Import Namespace = "System.Data.SqlClient" %>  
  
<%@ Import Namespace = "Minha_Biblioteca" %>
```

Estas linhas importam todas as classes dos namespaces descritos acima. O espaço de nome System.Data.SqlClient importa por exemplo, os objetos SqlConnection, SqlCommand, SqlDataReader e SqlDataAdapter.

Este recurso facilita bastante para o desenvolvedor chamar seus objetos e métodos no aplicativo. É apenas uma maneira mais fácil e rápida de fazer isso.

Existe ainda uma outra maneira de se fazer isso, utilizando a seguinte sintaxe:

```
Dim Conn As System.Data.SqlClient.SqlConenction  
  
Dim Cultura As System.Globalization.CultureInfo
```

DICA:

Importar um espaço de nome, porém, não importa os espaços de nome abaixo dele. Somente as classes que pertencem a essa interface são importadas.

Objeto SESSION

Na primeira solicitação de um determinado cliente ao aplicativo ASP.Net, é criado um objeto do tipo Dictionary chamado Session. Nele podemos guardar qualquer valor com strings e números que poderão ser lidos em qualquer página da aplicação.

Em cada objeto tipo ViewState do cliente existe um pointer para ele.

Para fixar melhor então, podemos dizer que um cliente vai ter um objeto Session e vários objetos do tipo ViewState.

Quando uma página é chamada pela primeira vez é disparado o evento OnStarSession. Na pasta da aplicação podemos adicionar um arquivo chamado global.asax em que é possível programar o método Session_OnStart que será executado quando ocorrer o evento. Podemos então criar variáveis de sessão no objeto Session, que poderão ser lidas em todas as páginas. Este recurso é muito utilizado para contar o numero de usuários conectados no site por exemplo.

Portanto, para tornar uma variável acessível em outras páginas da aplicação, devemos desenvolver **variáveis de sessão** usando aquele objeto Session.

Acompanhe o exemplo abaixo:

```
<html>
<form runat="server">
<asp:Panel id="panel1"
    BackColor="#E0E0E0"
    HorizontalAlign="Center"
    Width="200"
    runat="server">

<br/>
<asp:TextBox id="entrada"
    BackColor="yellow"
    Width="150"
    runat="server"/>

<p>
<asp:Button id="bot01"
    Text="- Enviar -"
    OnClick="MetodoEnviar"
    ToolTip="Escreva e Clique"
    runat="server"/> </p>

<p>
<asp:Label id="saida"
    Font-Name="Arial"
    runat="server"/> </p>
</asp:Panel>
</form>
<script language="VB" runat="server">
    public Sub MetodoEnviar(Obj As Object, ea As EventArgs)
        Dim VarTemp As String
        VarTemp = entrada.Text
        Session.Add("VarSessao", VarTemp)
        saida.Text = Convert.ToString(Session.Item("VarSessao"))
    End Sub
</script>
</html>
```

Vantagens Adicionais

Além de aumentar a produtividade no desenvolvimento, o ASP.Net traz diversos novos recursos que os desenvolvedores irão apreciar como novidades para manter sessões, configurar o Servidor através de arquivos xml, gerenciar idiomas, criar WebServices e manter a segurança do código fonte através de compilação.

Vamos ver com mais detalhes:

- ✓ Flexibilidade para manter estado do aplicativo, pois evita o uso de variáveis de sessões, mantendo automaticamente os seus valores.
- ✓ A configuração do aplicativo é feita através de arquivos XML. Isto significa que ela é fácil de alterar e não exige a interferência do administrador do servidor.
- ✓ Facilidade em utilizar diversas culturas diferentes. Trocar a cultura usada em seu aplicativo implica em mudar uma única linha no arquivo XML de configuração. Pode também ser tratada no código, em tempo de execução.
- ✓ Possui um mesmo modelo de projeto, podendo conter páginas aspx e também WebServices.
- ✓ Os aplicativos em ASP.Net são compilados ganhando em velocidade de execução e possui a impossibilidade de pessoas não autorizadas enxergarem o seu código-fonte.

O ASP.Net é um grande passo à frente no desenvolvimento de aplicativos que rodam em servidores Web.

Os ganhos de produtividade e a facilidade de desenvolvimento são inúmeras vezes maiores que algumas tecnologias utilizadas, como – ASP, PHP, CGI, JavaScript, entre outras.

Capítulo 3 – Variáveis

Podemos dizer que uma variável é um espaço alocado na memória RAM para ser utilizada por um ou mais processos que necessitam de armazenar ou manipular alguma informação. Ou ainda, variáveis representam locais de armazenamento.

Essas variáveis são reconhecidas por um nome que lhe é atribuído.

As variáveis devem ser declaradas, assim terá um tipo associado a ela que determinará que valores podem ser armazenados e manipulados.

O conteúdo de uma variável pode mudar no decorrer do programa.

Você pode dar o nome que quiser a uma variável, mas existem algumas restrições como:

- O nome deve conter letras, números, ou caracteres de sublinhado(_);
- Este nome deve conter no máximo 255 caracteres.

É importante ressaltar que você adote um padrão para a declaração destas variáveis, para facilitar possíveis reparos no sistema desenvolvido.

Como definido não podemos atribuir qualquer valor a uma variável, sem antes defini-la.

Veja abaixo a sintaxe para declaração de variáveis em C# e VB.NET:

Para C#: **TIPO** *NomeDaVariavel*

Exemplo: **string** *Recebe_Indice*;

Para VB.Net: **Dim** *NomeDaVariavel* **AS TIPO**

Exemplo: **Dim** *Recebe_Índice* **AS Integer**;

Se estas variáveis forem declaradas dentro de um procedimento ou bloco, poderão ser acessíveis apenas dentro deste procedimento ou bloco.

Na figura 3.1, o programa feito em C# tem um procedimento chamado de *TestaVariavel()*, onde é declarado uma variável com o nome de **VarEscreve**.

Esta variável estará acessível somente no procedimento em que foi declarada, neste exemplo, no procedimento *TesteVariavel()*. Portanto se tentar solicitar esta variável fora do procedimento acontecerá um erro.

Figura 3.1 – programa em C#.

```
using System;

public class BlocoTeste
{
    public void TestaVariavel( )
    {
        string VarEscreve; // será acessível apenas neste procedimento
        VarEscreve = "Teste de variável";
        Console.WriteLine(VarEscreve);
    }

    public static void Main( ) {
        BlocoTeste objBloco = new BlocoTeste( );
        objBloco.TestaVariavel( );
    }
}
```

Na figura 3.2, um programa em Visual Basic.Net , tem um exemplo de uma variável declarada com o nome **VarEscreve** em um bloco de execução.

Esta variável será acessível apenas neste bloco de execução, qualquer chamada a esta variável fora, vai ocorrer um erro.

Figura 3.2 - programa em Visual Basic.net

```
Imports System

Public Class BlocoTeste
    Public Shared Sub Main()
        Dim K AS Integer
        For K = 1 To 10
            Dim VarEscreve E AS Integer ' será acessível apenas neste bloco
            If (k Mod 2) = 0 Then
                VarEscreve = k
                Console.WriteLine(VarEscreve)
            End If
        Next K
    End Sub
End Class
```

Dica: utilize nomes que sejam adequadamente descritivos. Não utilize nomes de variáveis temporárias, nem reutilize nomes - isso vai tornar o código confuso!

3.1 Variáveis na linguagem C#.

Na linguagem C# as variáveis estão agrupadas em algumas categorias como:

- Static: Existe apenas uma única cópia desta variável para todas as instancias de uma classe. Uma variável **static** começa a existir quando um programa começa a executar, e deixa de existir quando o programa terminar.
- Instance: existe uma cópia para cada instancia de uma classe. Uma variável **Instance** começa a existir quando uma instancia daquele tipo é criado, e deixa de existir quando não houver nenhuma referência àquela instancia ou quando o método Finalize é executado.
- Array: é uma matriz que é criada em tempo de execução.

Tipos

Podemos armazenar muitos tipos de informações diferentes dentro de uma variável, como números, datas, e strings.

Nesta linguagem, seus tipos podem ter dois tipos:

- Por Valor: os valores são gerenciados diretamente na memória.
- Por Referencia: os valores são passados por meio de um ponteiro.

Inteiros (por valor)

Este tipo é um numero inteiro sem fração ou parte decimal. Por exemplo, o número 1058 é um numero inteiro.

O tipo inteiro é subdividido em alguns subtipos. Veja a *figura 3.1.1* que mostra estes tipos de inteiro.

Figura 3.1.1 – tipos de inteiro.

Tipo	Descrição
byte	Inteiro de 1 bit sem sinal (0 a 255)
sbyte	Inteiro com sinal de 8 bits (-127 a 128)
int	Inteiro de 32 bits com sinal (-2.147.483.648 a 2.147.483.147)
uint	Inteiro de 32 bits sem sinal (0 a 4.294.967.295)
long	Inteiro com sinal de 64 bits (-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807)
ulong	Inteiro sem sinal de 64 bits (0 a 18.446.744.073.709.551.615)
short	Inteiro com sinal de 16 bits (-32.768 a 32.767)
ushort	Inteiro sem sinal de 16 bits (0 a 65.535)

Exemplos:

```
byte VarIdade = 25;  
int VarValor = 1.542;  
long VarValor = 5.684.548.520;  
short VarInd = 25.620;
```

Lógicos (por valor)

Esses tipos são usados para comparação. Portanto esses tipos só podem assumir os valores verdadeiro ou falso.

Tipo	Descrição
bool	true ou false

Exemplos:

```
bool VarDesativado = false;  
bool VarCondicao = true;
```

Caracteres (por valor)

Este tipo é armazenado em 16 bits, representa um caractere de uma string. Essas variáveis são armazenadas no padrão Unicode.

Tipo	Descrição
char	Um único caractere Unicode de 16 bits.

Exemplos:

```
Char VarChar = 'a';  
    ↳ Representa um caractere literal.  
Char VarChar = '\x0025';  
    ↳ Representa um caractere hexadecimal.  
Char VarChar = '\u0025';  
    ↳ Representa um caractere Unicode.  
Char VarChar = (char)25;  
    ↳ Retorna o caractere correspondente na Tabela ASCII.
```

Figura 3.1.2 - neste exemplo é retornado o valor correspondente ao caractere 25 da tabela ASCII.

```
using System;  
public class CharTeste  
{  
    public static void Main()  
    {  
        char VarTeste = (char)25;  
        Console.WriteLine(VarTeste);  
    }  
}
```

Ponto Flutuante (por valor)

As variáveis com este tipo possuem números com *ponto flutuante* utilizando o padrão IEEE de 4 a 8 bytes.

Tipo	Descrição
double	Ponto flutuante binário de 8 bytes ($\cong 1.5 \times 10^{-45}$ a $\cong 1.7 \times 10^{308}$) Este tipo tem uma precisão de 15 casas decimais.
float	Ponto flutuante binário de 4 bytes ($\cong 1.5 \times 10^{-45}$ a $\cong 3.4 \times 10^{38}$) Este tipo tem uma precisão de 7 casas decimais.

As operações com este tipo não geram erros, mas há alguns ressaltos que devemos saber:

- Se o valor retornado para a variável conter um número muito pequeno, este valor torna-se zero.
- Uma divisão por zero não causa um erro, mas é produzido um valor infinito.
- Operações de ponto flutuante devolvem **NaN** (Not a Number) para sinalizar que aquele resultado da operação é indefinido. Portanto se um dos operadores for NaN o resultado retornado será NaN.

Exemplos:

`float` VarIndice = 3.1356F;

`double` VarCorrecao = 5.12D;

Decimal (por valor)

O tipo decimal representa com mais precisão números não inteiros e com um valor muito alto.

Tipo	Descrição
decimal	Ponto flutuante decimal de 128 bytes ($\cong 1.5 \times 10^{-45}$ a $\cong 1.7 \times 10^{308}$) Este tipo tem uma precisão de 28 casas decimais.

Exemplos:

`decimal` VarIndice = 560.5m;

`decimal` VarCorrecao = 545.2m;

String (por referência)

Este tipo pode conter até 1 gigabyte de caractere e é alocado dinamicamente, por isso dizemos que este tipo é por referência.

Tipo	Descrição
string	Unicode com até 1 gigabyte de caractere.

Exemplos:

```
string VarNome = "Claudio Junior";  
string VarEmail = "claudiojunior@estadao.com.br";  
string VarConcatenar = VarNome + VarEmail;
```

As strings são compostas por caracteres e podem conter literais caracteres. Veja a representação dos literais strings:

Tipo	Descrição
\'	Apóstrofo
\"	Aspas
\\	Barra invertida
\0	Zero binário ou nulo
\a	Alerta
\b	Retrocesso
\f	Avanço de página
\n	Salto de linha
\r	Retorno de carro
\t	Tabulação horizontal
\v	Tabulação vertical
\uNNNN	NNNN é o código em hexadecimal

Uma representação importante das string é a arroba (@), utilizada para especificar nomes de arquivos e pastas.

Exemplos:

```
string VarLinha = "Pula Linha \n";  
string VarCaminho = @"c:\temp\oledb.txt";
```

➡ Neste caso com o uso da arroba (@) a representação dos caracteres de barra invertida (\) não é feita dentro da string.

Object (por referência)

Este tipo é muito utilizado por programadores, é um termo geral para uma variável que não é especificado como outro tipo de dados, por isso torna-se um tipo universal.

Em certas situações torna-se indispensável atribuir e instanciar um objeto.

Tipo	Descrição
object	Atribuir a um objeto.

Exemplos:

Object VarNome = "Rubens Barrichello";

↳ Este objeto recebe o valor de uma string.

Object VarIdade = 29;

↳ Este objeto recebe o valor de um numero;

Object VarAtivar = true;

↳ Este objeto recebe um valor lógico.

Conversão de Valores

Converter um tipo de dado em número ou em literal é comum em situações de programação. Quando esta aplicação for destinada a Web com intervenções de internautas, esse recurso é utilizado com mais frequência ainda. É comum o internauta testar a aplicação para ter certeza que o desenvolvedor se preocupou com as devidas conversões de valores.

Devemos considerar alguns aspectos para a conversão de números:

- ↳ Como existem diversos tipos de números, inteiros, ponto flutuante ou decimal, os valores são convertidos sempre para o tipo de maior faixa de valores. Por exemplo, o tipo long é convertido para o ponto flutuante, mais é importante ressaltar que o contrario causa um erro.
- ↳ Os tipos de menor faixa são convertidos para os de maior faixa. Por exemplo, o tipo int pode ser convertido para: long, float, double ou decimal.
- ↳ A conversão dos tipos de ponto flutuante(float, double) para decimal causa erro.
- ↳ A conversão entre os tipos com sinal e sem sinal de valores inteiros com o mesmo tamanho causa erro. Por exemplo, entre o tipo int e unit.

Caso precise forçar uma conversão mesmo que haja perda de informações, a linguagem disponibiliza um operador com este recurso.

Exemplos:

```
int VarValor = (int)8544555L;
```

↳ Neste caso a variável inteira vai receber o quanto poder suportar do tipo long.

```
long VarValor = (long)29.145;
```

↳ Neste caso a variável inteira do tipo long suportará o quanto for possível do número não inteiro.

Em alguns casos os tipos de conversão não precisam ser especificados pelo desenvolvedor, essa conversão é feita automaticamente.

Figura 3.1.3 - os tipos de conversão automática:

Tipo	Converte em
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	long, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double

Exemplos:

```
int VarInteiro = 32450;
```

```
long VarLong = VarInteiro;
```

```
float VarFloat = VarLong;
```

```
double VarDouble = VarFloat;
```

```
decimal VarDecimal = VarLong;
```

```
byte VarByte = (byte)VarInteiro;
```

```
int VarInteiro = (int)31.245F;
```

Checked e Unchecked

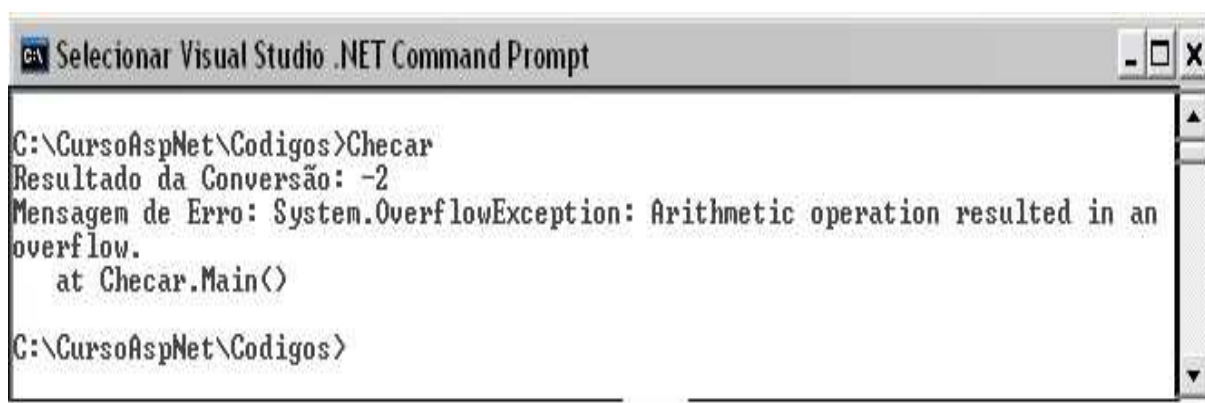
Toda conversão pode causar erro, mas existe uma forma de forçar a conversão sem que cause um erro. Para isso são utilizados o checked e o unchecked.

Figura 3.1.4 – Exemplo do checked e do unchecked.

```
using System;

public class Checar
{
    public static void Main()
    {
        int X = 2147483647;
        int Y = 2;
        int Produto = 0;
        unchecked
        {
            Produto = X * Y;
            Console.WriteLine("Resultado da Conversão: "+Produto); // retorna -2
        }
        checked
        {
            try
            {
                Produto = X * Y;
                Console.WriteLine("Resultado da Conversão: "+Produto);
                // Causa erro na compilação
            }
            catch(OverflowException e)
            {
                Console.WriteLine("Mensagem de Erro: "+e.ToString());
            }
        }
    }
}
```

Veja a saída deste programa:



```
Selecionar Visual Studio .NET Command Prompt

C:\CursoAspNet\Codigos>Checar
Resultado da Conversão: -2
Mensagem de Erro: System.OverflowException: Arithmetic operation resulted in an overflow.
   at Checar.Main()
C:\CursoAspNet\Codigos>
```

Operadores

Operadores são símbolos utilizados para efetuar alguma ação.

Veja a tabela de operadores que o C# suporta.

Tipo	Operador
aritméticos	+ - * / %
lógicos	& ^ ! ~ &&
concatenação	+
Incremento e decremento	++ --
deslocamento	<< >>
relacional	< > <= >=
Igualdade	== !=
atribuição	= *= /= %= += <<= >>= &= ^=
condicional	? :
criação de objetos	new
primários	typeof sizeof is checked unchecked

Tipos definidos pelo programador

Uma grande maioria das linguagens de programação permite ao desenvolvedor definir seus próprios tipos, que podem ser usados em qualquer situação que normalmente um outro tipo poderia ser utilizado.

Enum (por valor)

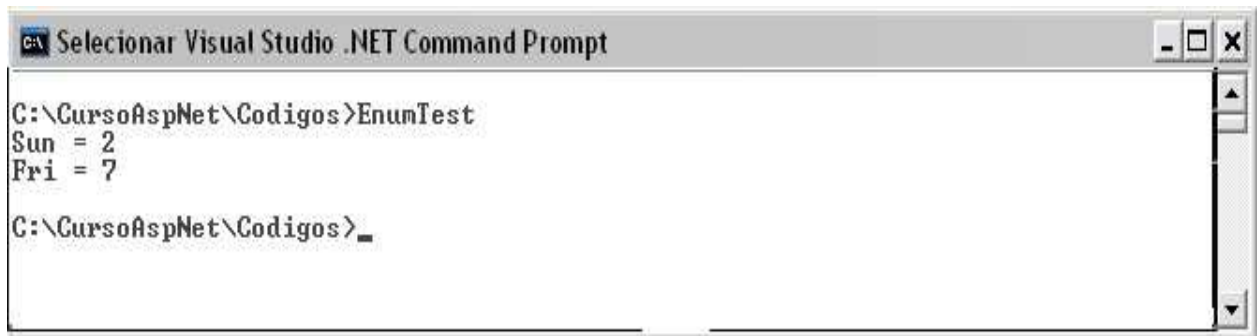
O tipo **enum** é usado para declarar uma enumeração, um tipo distinto que consiste em um jogo de constantes nomeadas chamadas a lista de enumerados.

Figura 3.1.5 – exemplo de um tipo **enum**, definido pelo desenvolvedor.

```
using System;
public class EnumTest
{
    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

    public static void Main()
    {
        int x = (int) Days.Sun;
        int y = (int) Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
    }
}
```

Veja a saída do programa da figura 3.1.5.



Struct (por valor)

Permite declarar tipos que contem diversos valores identificados pelo nome.

Um tipo de struct é um tipo de valor que pode conter construtores, constantes, campos, métodos, propriedades, dentre outros. A declaração de um struct leva a forma seguinte:

Atributo **struct** *IdentificadorDoTipo*

Figura 3.1.6 – exemplo de um tipo struct.

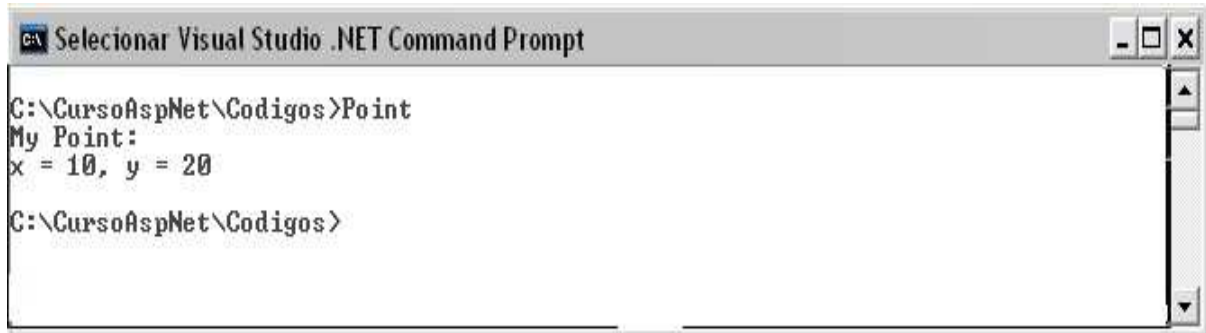
```
using System;
public struct Point
{
    public int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
class MainClass
{
    public static void Main()
    {
        // Declare an object:
        Point myPoint;

        // Initialize:
        myPoint.x = 10;
        myPoint.y = 20;

        // Display results:
        Console.WriteLine("My Point:");
        Console.WriteLine("x = {0}, y = {1}", myPoint.x, myPoint.y);
    }
}
```

Veja a saída do programa da figura 3.1.6.



```
C:\CursoAspNet\Codigos>Point
My Point:
x = 10, y = 20
C:\CursoAspNet\Codigos>
```

New

Independente do tipo todas as variáveis podem ser inicializadas com o operador **new**. Caso sejam do tipo class, struct ou array, este operador é obrigatório.

Exemplos do operador **new**:

↳ Para tipos primitivos:

```
Tipo NomeVariavel = new Tipo( );
```

```
Int Valor = new int( );
```

↳ Para classes:

```
NomeClasse NomeInstancia = new NomeClasse( );
```

```
ClasseAluno ObjAluno = new ClasseAluno( );
```

↳ Para structs:

```
NomeStruct InstanciaTipo = new NomeStruct( );
```

```
RegistroAluno RegAluno = new RegistroAluno( );
```

No exemplo a seguir temos a criação de um tipo definido pelo desenvolvedor, o tipo struct, e a criação de um objeto de classe.

```
using System;
class NewTest
{
    struct MyStruct
    {
        public int x;
        public int y;
        public MyStruct (int x, int y)
        {
            this.x = x;
            this.y = y;
        }
    }
    class MyClass
    {
        public string name;
        public int id;

        public MyClass ()
        {
        }

        public MyClass (int id, string name)
        {
            this.id = id;
            this.name = name;
        }
    }
    public static void Main()
    {
        // Criação de objetos usando o construtor sem valores.
        MyStruct Location1 = new MyStruct();
        MyClass Employee1 = new MyClass();

        // Valores de saída:
        Console.WriteLine("Valores Default:");
        Console.WriteLine("  Membros do Struct: {0}, {1}",
            Location1.x, Location1.y);
        Console.WriteLine("  Membros da Classe: {0}, {1}",
            Employee1.name, Employee1.id);

        // Criando objetos usando construtores parametrizados:
        MyStruct Location2 = new MyStruct(10, 20);
        MyClass Employee2 = new MyClass(1234, "John Martin Smith");

        // Valores de saída:
        Console.WriteLine("Valores Inicializados pelo construtor:");
        Console.WriteLine("  Struct members: {0}, {1}",
            Location2.x, Location2.y);
        Console.WriteLine("  Class members: {0}, {1}",
            Employee2.name, Employee2.id);
    }
}
```


Arrays

Um tipo array é uma matriz de valores do mesmo tipo, que é criada em tempo de execução, podendo ser acessada por meio de um índice.

A declaração do array sempre faz o uso de um colchete (`[]`) depois do tipo da variável. O uso da instrução **new** sempre deve ser utilizado, pois é obrigatório.

O tipo array pode ter diversas dimensões, o tamanho desta é definido pelo desenvolvedor, mas devemos saber que o primeiro índice é sempre zero.

No tipo array devemos sempre inicializar seus elementos, pois é obrigatório também.

Veja abaixo a forma de sintaxe para a declaração de arrays.

```
TIPO [ ] NomeDoArray = new TIPO [ tamanho do array ];  
  
float [ ] ValorIndice = new float [ 10 ];  
  
string [ ] ElementoVetor = new string [ 10 ];
```

Veja abaixo a forma de sintaxe para a declaração de um array de duas ou mais dimensões.

```
TIPO [ , ] NomeDoArray = new TIPO [ tamanho do array, tamanho do array ];  
  
float [ , ] ValorIndice = new float [ 10 , 10 ];  
  
string [ , , ] ElementoVetor = new string [ 10 , 10 , 10 ];
```

Veja abaixo a forma de sintaxe para a declaração de uma matriz de arrays.

```
TIPO [ ] [ ] NomeDoArray = new TIPO [ tamanho do array ] [ tamanho do array ];  
  
float [ ] [ ] ValorIndice = new float [ 10 ] [ 10 ];  
  
string [ ] [ ] [ ] ElementoVetor = new string [ 10 ] [ 10 ] [ 10 ];
```

Veja abaixo a forma de sintaxe para a inicialização de arrays.

```
TIPO [ ] NomeDoArray = new TIPO [ tamanho do array ] { valores };  
  
float [ ] ValorIndice = new float [ 5 ] { 1.25, 2, 3.23, 1.32, 5 };  
  
string [ , ] ElementoVetor = new string[3, 3] {{"ab", "ac", "bc"}, {"ab", "ac", "bc"}};  
  
int [ ] [ ] MatrizDeInteiro = new int [ 2 ] [ ];  
MatrizDeInteiro[ 0 ] = new int [ 5 ] {1,3,5,7,9};  
MatrizDeInteiro[ 1 ] = new int [ 4 ] {2,4,6,8};
```

Exemplos e Exercícios:

Exemplo 01 – comparação.

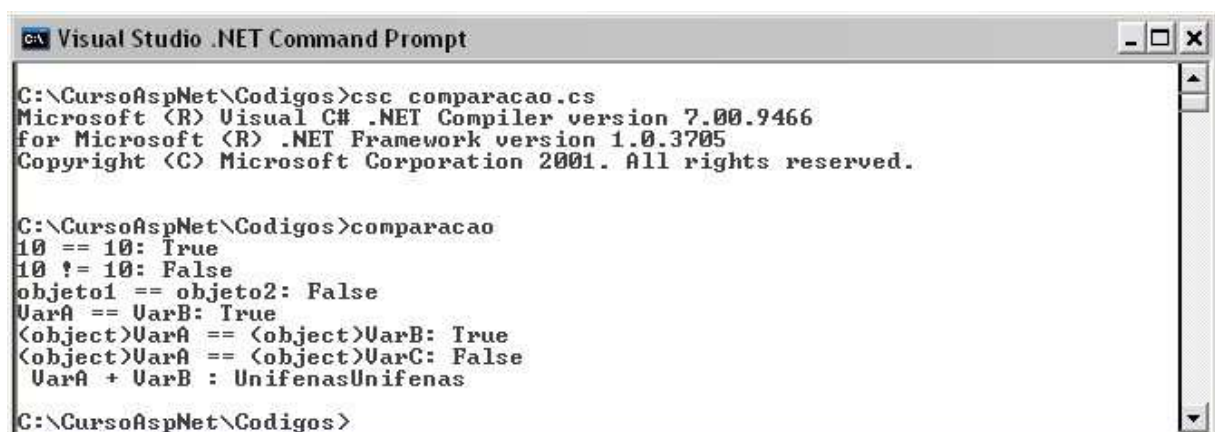
```
using System;
public class Comparacao
{
    public static void Main( )
    {
        // Exemplos com números.
        Console.Write("10 == 10: ");
        Console.WriteLine(10 == 10);
        Console.Write("10 != 10: ");
        Console.WriteLine(10 != 10);

        // Exemplos com objetos.
        object objeto1 = 10;
        object objeto2 = 10;
        Console.Write("objeto1 == objeto2: ");
        Console.WriteLine(objeto1 == objeto2);

        // Exemplos com strings.
        string VarA = "Unifenas";
        string VarB = "Unifenas";
        string VarC = String.Copy(VarA);
        Console.Write("VarA == VarB: ");
        Console.WriteLine(VarA == VarB);
        Console.Write("(object)VarA == (object)VarB: ");
        Console.WriteLine((object)VarA == (object)VarB);
        Console.Write("(object)VarA == (object)VarC: ");
        Console.WriteLine((object)VarA == (object)VarC);

        Console.Write(" VarA + VarB : "); // Concatenando strings
        Console.WriteLine(VarA + VarB);
    }
}
```

Veja a saída do programa acima:



```
C:\CursoAspNet\Codigos>csc comparacao.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

C:\CursoAspNet\Codigos>comparacao
10 == 10: True
10 != 10: False
objeto1 == objeto2: False
VarA == VarB: True
(object)VarA == (object)VarB: True
(object)VarA == (object)VarC: False
 VarA + VarB : UnifenasUnifenas

C:\CursoAspNet\Codigos>
```

Exemplo 2 – Operações matemáticas.

```
using System;
public class OperacoesMat
{
    public static void Main( )
    {
        int Valor1; // forma normal
        int Valor2 = new int ( ); // forma alternativa
        Valor1=Valor2=10;

        Console.WriteLine(" Valor1 = Valor2 = 10: ");
        Console.WriteLine(" Valor1: "+Valor1);
        Console.WriteLine(" ++Valor1: "+ ++Valor1);
        Console.WriteLine(" Valor2- - : "+Valor2--);
        Console.WriteLine(" Valor1 + 15 : "+(Valor1 + 15));
        Console.WriteLine(" Valor2 - 5 : "+(Valor2 - 5));
    }
}
```

Veja a saída do programa acima:



```
C:\> Selecionar Visual Studio .NET Command Prompt

D:\Claudio\cursoAspNet\codigos>csc operacaoMat.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

D:\Claudio\cursoAspNet\codigos>operacaoMat
Valor1 = Valor2 = 10:
Valor1: 10
++Valor1: 11
Valor2- - : 10
Valor1 + 15 : 26
Valor2 - 5 : 4
```

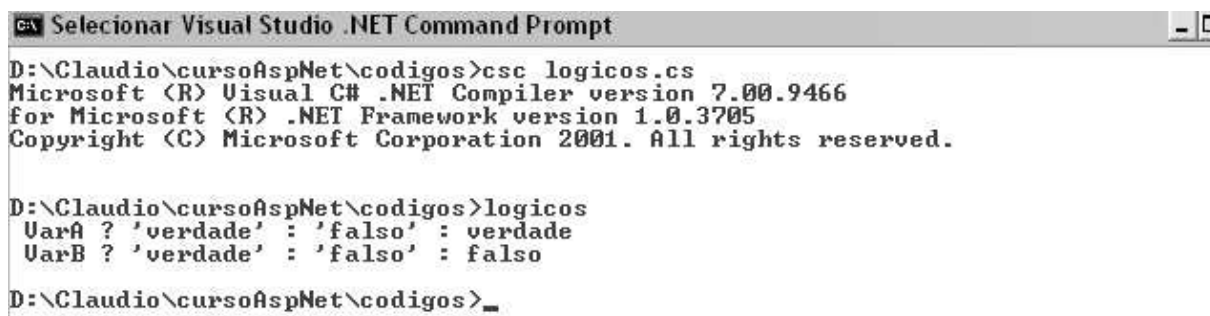
Exemplo 3 – operadores lógicos:

```
using System;
public class Logicos
{
    public static void Main( )
    {
        bool VarA = true;
        bool VarB = !true;

        Console.Write(" VarA ? 'verdade' : 'falso' : ");
        Console.WriteLine(VarA ? "verdade" : "falso");

        Console.Write(" VarB ? 'verdade' : 'falso' : ");
        Console.WriteLine(VarB ? "verdade" : "falso");
    }
}
```

Veja a saída do programa acima:



```
C:\> Selecionar Visual Studio .NET Command Prompt

D:\Claudio\cursoAspNet\codigos>csc logicos.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

D:\Claudio\cursoAspNet\codigos>logicos
VarA ? 'verdade' : 'falso' : verdade
VarB ? 'verdade' : 'falso' : falso

D:\Claudio\cursoAspNet\codigos>
```

3.2 Variáveis na linguagem Visual Basic.Net

A linguagem Visual Basic.net também faz uso de variáveis como toda linguagem de programação.

O uso de variáveis nos aplicativos desenvolvidos é muito comum, principalmente para armazenamentos temporários de informações. Para a Web a utilização de variáveis é inevitável.

Tipos

Toda variável deve ter um tipo que define qual o tipo de informações ou dados lhe convém. Para isso, toda variável tem um conjunto de regras que determina o seu uso.

Nesta linguagem os tipos são divididos em cinco categorias diferentes: inteiros, números de ponto flutuante, booleanas, datas e strings.

Inteiros (por valor)

Um número inteiro sem fração ou parte decimal é dito como inteiro.

Existem alguns subtipos de inteiros, de modo que você possa assegurar um tamanho de memória adequado para um determinado dado.

A figura 3.2.1 mostra os tipos de inteiros que a linguagem suporta.

Figura 3.2.1 – Subtipos de inteiros.

Tipo	Descrição
Byte	Inteiro de 1 byte sem sinal (0 a 255) (também conhecido como System.Int)
Short	Inteiro de 2 bytes com sinal (-32.678 a 32.677) (também conhecido como System.Int16)
Integer	Inteiro de 4 bytes com sinal (-2.147.483.648 a 2.147.483.647) (também conhecido como System.Int32)
Long	Inteiro de 8 bytes com sinal (- 9.223.372.036.854.775.808 a 9.223.372.036.854.775.807) (também conhecido como System.Int64)

Exemplos:

Dim VarIdade As Byte = 25

Dim VarValor As Int = 1542145120

Dim VarValor As Long = 5684548520

Dim VarInd As Short = 25620

Lógicos (por valor)

O tipo lógico ou booleano é utilizado para valores do tipo verdadeiro/falso.

Na linguagem Visual basic.Net este tipo só pode ser verdadeiro ou falso, diferentes de outras linguagem onde poderia receber sim/não ou ainda 1/0.

A figura 3.2.2 mostra o tipo booleano que a linguagem suporta.

Figura 3.2.2 – Tipos lógicos ou booleanos.

Tipo	Descrição
Boolean	Pode ser somente true ou false . (também conhecido como System.Boolean)

Exemplos:

```
Dim VarAtivar As Boolean = true
```

```
Dim VarAtivar As Boolean = false
```

```
Dim VarAtivar As Boolean = Not true
```

Caracteres (por valor)

Este tipo é armazenado em 16 bits, representa um caractere de uma string. Essas variáveis são armazenadas no padrão Unicode.

A figura 3.2.3 mostra o tipo char.

Figura 3.2.3 – tipo Char.

Tipo	Descrição
char	Um único caractere Unicode. (também conhecido como System.Char)

Exemplos:

Option Strict On

' ...

```
Dim CharVar As Char
```

```
CharVar = "Z" ' Não pode converte de string para caractere com Option Strict On.
```

```
CharVar = "Z"C ' Sucesso em atribuir um único caractere a variável CharVar.
```

Ponto Flutuante (por valor)

Os números com parte fracionária ou não inteiros são definidos como ponto flutuante.

Nesse tipo também há subtipos, para que o desenvolvedor possa adequar melhor o tipo de informação com o espaço reservado na memória.

A figura 3.2.4 mostra os subtipos do ponto flutuante.

Figura 3.2.4 – Subtipos do ponto flutuante.

Tipo	Descrição
Single	Um número de 4 bytes com ponto de fração decimal. ((-3.4028235E+38 a -1.401298E-45 para valores negativos) e (1.401298E-45 a 3.4028235E+38 para valores positivos)). (também conhecido como System.Single)
Double	Um numero de 8 bytes com ponto de fração decimal. ((-1.79769313486231570E+308 a -4.94065645841246544E-324 para números negativos) e (4.94065645841246544E-324 a 1.79769313486231570E+308 para valores positivos)). (também conhecido como System.Double)
Decimal	Um numero de 12 bytes com ponto de fração decimal. (também conhecido como System.Decimal)

Exemplos:

```
Dim VarSingle As Single = 312. 12
Dim VarDouble As Double = 5400. 45
Dim BigDec1 As Decimal = 9223372036854775807      ' No overflow.
Dim BigDec2 As Decimal = 9223372036854775808      ' Overflow.
Dim BigDec3 As Decimal = 9223372036854775808D     ' No overflow.
```

Date (por referência)

Esse tipo armazena valores de datas e horas. O tipo de dados real é chamado *Date Time*. As datas podem ser armazenadas em muitas formas diferentes, como por exemplo, “13/1/2002”, “segunda-feira, 13 de janeiro de 2002 6:02:58PM”, dentre outras.

Para o Visual Basic.net tudo isso é data e é fácil de fazer a conversão de uma para outra.

Você deve estar perguntando – porque não armazenar essas datas em strings. Bem você pode se assim preferir, mais perde algumas funções que o Visual Basic.Net traz para este tipo de dado como - somar dias, horas e minutos.

Vejamos abaixo a figura que mostra alguns detalhes do tipo Date.

Figura 3.2.5 – Tipo Date

Tipo	Descrição
Date	São armazenados em números inteiros longos com 8 bytes que representam uma data na faixa de 1/1/1 até 31/12/9999. Com relação às horas valem de 0:00:00 até 23:59:59.

Exemplos:

```
Dim VarHoje As Date = Now
Dim VarData As Date = Today
Dim VarHoras As Date = TimeOfDay
Dim VarSegundos As Date = Time
```

Strings (por referência)

Este tipo pode conter até 1 gigabyte de caractere e é alocado dinamicamente, por isso dizemos que este tipo é por referência.

As strings são grupos de caracteres como – “Cláudio Junior”, “dizer-lhe”, “#2!455^%” ou ainda “1234”. As Strings em Visual Basic.Net são incluídas entre aspas duplas (“ Visual Basic.Net ”).

Veja na figura 3.2.6 o tipo string.

Figura 3.2.6 – Tipo String

Tipo	Descrição
string	Aproximadamente 2 bilhões de caracteres.

Exemplos:

```
Dim VarNome As String = "Claudio"
Dim VarEmail As String = "claudiojunior@estadao.com.br"
Dim VarSobrenome As String = "Junior"
Dim VarConcatenar As String = VarNome & VarSobrenome
Dim VarConcatenar As String = VarNome + ( VarSobrenome )
```


Object (por referência)

Este tipo é muito utilizado por programadores, é um termo geral para uma variável que não é especificado como outro tipo de dados, por isso torna-se um tipo universal.

Em certas situações torna-se indispensável atribuir e instanciar um objeto.

Figura 3.2.7 – Tipo Object.

Tipo	Descrição
object	Atribuir a um objeto.

Exemplos:

`Dim VarObjeto As Object = "Rubens Barrichello"`

`Dim VarObjeto As Object = 234.214D`

`Dim VarObjeto As Object = Not false`

`Dim VarObjeto As Object = TimeOfDay`

Conversão de Valores

Converter um tipo de dado em número ou em literal é comum em situações de programação. Quando esta aplicação for destinada a Web com intervenções de internautas, esse recurso é utilizado com mais frequência ainda.

A conversão pode ser implícita(conversões implícitas) ou explícita(conversões especificadas – explícitas) de uma variável.

Devemos observar que uma variável é convertida num outro tipo de maior faixa.

Em alguns casos os tipos de conversão não precisam ser especificados pelo desenvolvedor, essa conversão é feita automaticamente.

Figura 3.2.8 - os tipos de conversão automática:

Tipo	Converte em
Byte	Byte, Short, Integer, Long, Decimal, Single, Double.
Short	Short, Integer, Long, Decimal, Single, Double.
Integer	Integer, Long, Decimal, Single, Double.
Long	Long, Decimal, Single, Double.
Decimal	Decimal, Single, Double.
Single	Single, Double.
Double	Double
Char	String
Qualquer Tipo	Object

Figura 3.2.9 - veja as funções para os tipos de conversões explícitas.

Função	Converte em
Asc	Retorna o numero correspondente a tabela ASCII
Cbool	Boolean
Cbyte	Byte
CChar	Char
Cdate	Date
CDbl	Double
Cdec	Decimal
Chr	Char
Cint	Integer
CLng	Long
Cobj	Object
Cshort	Short
CSng	Single
CStr	String
Ctype	Converte para o tipo especificado
Str	String
Val	Converte para o tipo numérico especificado.

Exemplos:

```
Dim MyInt As Integer
MyInt = Asc("A")      ' MyInt recebe 65.
MyInt = Asc("a")      ' MyInt recebe 97.
MyInt = Asc("Apple")  ' MyInt recebe 65.
```

```
Dim A, B, C As Integer
Dim Check As Boolean
A = 5
B = 5
Check = CBool(A = B)  ' Check recebe True.
C = 0
Check = CBool(C)      ' Check recebe False.
```

```
Dim MyDouble As Double
Dim MyByte As Byte
MyDouble = 125.5678
MyByte = CByte(MyDouble) ' MyByte recebe 126.
```

```
Dim MyString As String
Dim MyChar As Char
MyString = "BCD" ' CChar converte somente o primeiro caracter do string.
MyChar = CChar(MyString) ' MyChar recebe "B".
```

```
Dim MyDateString, MyTimeString As String
Dim MyDate, MyTime As Date
MyDateString = "February 12, 1969"
MyTimeString = "4:35:47 PM"
' ...
MyDate = CDate(MyDateString) ' Converte para Data.
MyTime = CDate(MyTimeString) ' Converte para Data.
```

```
Dim MyDec As Decimal
Dim MyDouble As Double
MyDec = 234.456784D
MyDouble = CDBl(MyDec * 8.2D * 0.01D) ' Converte o resultado para Double.
```

```
Dim MyDouble As Double
Dim MyDecimal As Decimal
MyDouble = 10000000.0587
MyDecimal = CDec(MyDouble) ' Converte para Decimal.
```

```
Dim MyDouble As Double
Dim MyInt As Integer
MyDouble = 2345.5678
MyInt = CInt(MyDouble) ' MyInt recebe 2346.
```

```
Dim MyDbl1, MyDbl2 As Double
Dim MyLong1, MyLong2 As Long
MyDbl1 = 25427.45
MyDbl2 = 25427.55
MyLong1 = CLng(MyDbl1) ' MyLong1 recebe 25427.
MyLong2 = CLng(MyDbl2) ' MyLong2 recebe 25428.
```

```
Dim MyDouble As Double
Dim MyObject As Object
MyDouble = 2.7182818284
MyObject = CObj(MyDouble) ' Valor de MyDouble é apontado para por MyObject.
```

```
Dim MyByte as Byte
Dim MyShort as Short
MyByte = 100
MyShort = CShort(MyByte) ' Converte para Short.
```

```
Dim MyDouble1, MyDouble2 As Double
Dim MySingle1, MySingle2 As Single
MyDouble1 = 75.3421105
MyDouble2 = 75.3421567
MySingle1 = CSng(MyDouble1) ' MySingle1 recebe 75.34211.
MySingle2 = CSng(MyDouble2) ' MySingle2 recebe 75.34216.
```

```
Dim MyDouble As Double
Dim MyString As String
MyDouble = 437.324
MyString = CStr(MyDouble) ' MyString recebe "437.324".

Dim MyDate As Date
MyDate = #2/12/69 00:00:01#
MyString = CStr(MyDate) ' MyString recebe "2/12/1969 12:00:01 AM".
```

```
Dim MyNumber As Long
Dim MyNewType As Single
MyNumber = 1000
MyNewType = CType(MyNumber, Single) ' MyNewType recebe 1000.0.
```

```
Dim MyString As String
MyString = Str(459) ' Retorna " 459".
MyString = Str(-459.65) ' Retorna "-459.65".
MyString = Str(459.001) ' Retorna " 459.001".
```

```
Dim ValResult As Double
ValResult = Val("2457") ' ValResult recebe 2457.
ValResult = Val(" 2 45 7") ' ValResult recebe 2457.
ValResult = Val("24 and 57") ' ValResult recebe 24.
```

Operadores

Os operadores são símbolos utilizados para executar alguma ação.

Você já deve conhecer alguns dos operadores, pois toda linguagem faz-se uso de operadores.

Acompanhe na figura abaixo os tipos de operadores e suas respectivas funções.

Figura 3.2.10 – Operadores da linguagem Visual Basic.net

Operador	Função
^	Exponencial
+, -	Adição e subtração
*, /	Multiplicação e Divisão
\	Divisão ($10 \div 3 = 3.333333$)
Mod	Módulo ($6 \bmod 4 = 2$)
&, + (string)	Concatenação
=, <, >, <=, >=	Igual a, não-igual a, maior que, menor que
<=, >=	Menor ou igual a, Maior ou igual a
=, ^=, *=, /=, \=, +=, -=, &=	Atribuição
NOT, AND, OR, XOR	Lógico: Negação, E, Ou, Ou exclusivo
BitNot, BitAnd, BitOr, BitVor	Lógico Binário: Negação, E, Ou, Ou exclusivo
TypeOf ... Is, Is, Like	Relacional

Exemplos

```
Sub ControlProcessor(ByVal MyControl As Control)
    If TypeOf MyControl Is ComboBox Then
        Console.WriteLine ("é do tipo " & TypeName(MyControl))
    ElseIf TypeOf MyControl Is CheckBox Then
        Console.WriteLine ("É do tipo " & TypeName(MyControl))
    ElseIf TypeOf MyControl Is TextBox Then
        Console.WriteLine ("É do tipo " & TypeName(MyControl))
    End If
End Sub
```

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim myCheck As Integer
myCheck = (A And B) ' Retorna 8.
myCheck = (A And C) ' Retorna 2.
myCheck = (B And C) ' Retorna 0.
Dim myTest As Boolean
myTest = A > B And B > C ' Retorna True.
myTest = B > A And B > C ' Retorna False.
```

Array

Um tipo array é uma matriz de valores do mesmo tipo, ou seja, um conjunto de elementos do mesmo tipo dentro de uma única variável, que é criada em tempo de execução, podendo ser acessada por meio de um índice.

A declaração do array sempre faz o uso de um parêntese () depois do nome da variável.

O tipo array pode ter diversas dimensões, o tamanho desta é definido pelo desenvolvedor, mas devemos saber que o primeiro índice é sempre zero. Portanto um array declarado com 10 posições, na realidade será de 11 posições – de 0 a 10.

Veja abaixo a forma de sintaxe para a declaração de arrays.

```
Dim NomeDoArray ( tamanho do vetor ) As Tipo

Dim VarValores ( 10 ) As Integer

Dim VarValores ( ) As Integer = { valores }

Dim Nomematriz ( tamanho do vetor, tamanho do vetor ) As Tipo

Dim VarMatriz ( 5, 5 ) As String
```

Atribuindo Valores aos Arrays

Para atribuir um valor aos elementos de um **array**, você deve montar um looping ou atribuir um valor específico a cada elemento do array.

Veamos abaixo a atribuição nos dois casos comentados.

```
Imports System
Module Atribuir
    Sub Main ( )
        Dim i As Integer
        Dim VarValores(5) As Integer
        For i=0 to 5
            VarValores(i) = i * 2
        Next i
    End Sub
End Module
```

```
Imports System
Module Atribuir
    Sub Main ( )
        Dim VarCores ( ) As String = {"azul", "preto", "branco", "verde"}
        Dim VarNomes ( 3 ) As String
        VarNomes (0) = "Adriana"
        VarNomes (1) = "Beatriz"
        VarNomes (2) = "Juliana"
        VarNomes (3) = "Maria"
    End Sub
End Module
```

Arrays Multidimensionais

O array também pode ser declarado com várias dimensões. Muito utilizado quando precisamos de uma matriz.

Vejamos abaixo a sintaxe deste array.

```
Dim NomeDaMatriz ( tamanho do vetor, tamanho do vetor ) As Tipo
```

```
Dim VarMatriz ( 15 , 15 ) As Integer
```

Vejamos abaixo um exemplo de atribuição de valores para um array multidimensional.

```
Imports System
Module Atribuir
  Sub Main ( )
    Dim VarMatriz ( , ) As Integer = {{1, 2, 3}, {4, 5, 6}}
    Dim I As Integer
    Dim J As Integer
    For I=0 to 2
      For J=0 to 2
        Console.WriteLine("VarMatriz("& I &", "& J &"): "& VarMatriz(I, J))
      Next J
    Next I
  End Sub
End Module
```

As Funções REDIM e ERASE

Depois de declarado um array com um número específico de elementos você fica limitado a este tamanho. Entretanto o Visual Basic.Net traz uma função chamada **REDIM** para redimensionar o tamanho do array.

Devemos observar algumas minuciosidades deste processo:

- ↳ Quando o vetor é redimensionado para outro tamanho, seus valores são destruídos. Porém se for utilizada a palavra-chave *preserve* os valores são salvos e atribuídos ao novo array.
- ↳ Se o novo valor especificado para o redimensionamento for *maior*, os índices extras são inicializados com o valor padrão.
- ↳ Se o novo valor especificado para o redimensionamento for *menor*, os índices que não fazem parte da nova faixa do vetor são destruídos.

Uma outra função bastante útil é a **ERASE**.

Esta função configura cada elemento do array para Nothing, ou seja, atribui um valor NULO as variáveis do vetor.

Exemplos e Exercícios

Exemplo 1 – Trabalhando com arrays no Visual Basic.Net

```
Imports System
Module CapArray
    Sub Main ( )
        Dim I As Integer
        Dim J As Integer
        Dim VarCores( ) As String = {"azul", "amarelo", "preto", "verde"}
        Console.WriteLine(" ----- Inicialização do array VarCores ----- ")
        For I=0 to 3
            Console.WriteLine("VarCores("& I &"): "& VarCores(I))
        Next I
        Dim VarValores(5) As Integer
        Console.WriteLine()
        Console.WriteLine(" ----- Inicialização do array VarValores ----- ")
        For I=0 to 5
            VarValores(I) = I
            Console.WriteLine("VarValores("& I &"): "& VarValores(I))
        Next I
        Dim VarMatriz(3, 2) As Integer
        Console.WriteLine()
        Console.WriteLine(" ----- Inicialização do array VarMatriz ----- ")
        For I=0 to 3
            For J=0 to 2
                VarMatriz(I, J) = I + J
                Console.WriteLine("VarMatriz("& I & ", "& J &"): "& VarMatriz(I, J))
            Next J
        Next I
        Dim VarMatrizB( , ) As Integer = {{1, 2, 3}, {4, 5, 6}}
        Console.WriteLine()
        Console.WriteLine(" ----- Inicialização do array VarMatrizB ----- ")
        For I=0 to 1
            For J=0 to 2
                Console.WriteLine("VarMatrizB("& I & ", "& J &"): "& VarMatrizB(I, J))
            Next J
        Next I
        Redim VarValores(8)
        Console.WriteLine()
        Console.WriteLine(" ---- Redimensionamento do array VarValores ---- ")
        J=0
        For Each I in VarValores
            Console.WriteLine("VarValores("& J &"): "& VarValores(I))
            J = J+1
        Next
        ReDim Preserve VarMatriz(3, 5) 'Apenas o ultimo valor pode ser redimensionado
        Console.WriteLine()
        Console.WriteLine(" ---- Redimensionamento do array VarMatriz ---- ")
        For I=0 to 3
            For J=0 to 5
                Console.WriteLine("VarMatriz("& I & ", "& J &"): "& VarMatriz(I, J))
            Next J
        Next I
        Erase VarCores
        Redim Preserve VarCores(3)
        Console.WriteLine()
        Console.WriteLine(" ---- Redimensionamento do array VarCores ---- ")
        For I=0 to 3
            Console.WriteLine("VarCores("& I &"): "& VarCores(I))
        Next I
    End Sub
End Module
```



```
Visual Studio .NET Command Prompt

C:\CursoAspNet\Codigos>vbc CapArray.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>CapArray
----- Inicialização do array VarCores -----
VarCores(0): azul
VarCores(1): amarelo
VarCores(2): preto
VarCores(3): verde

----- Inicialização do array VarValores -----
VarValores(0): 0
VarValores(1): 1
VarValores(2): 2
VarValores(3): 3
VarValores(4): 4
VarValores(5): 5

----- Inicialização do array VarMatriz -----
VarMatriz(0, 0): 0
VarMatriz(0, 1): 1
VarMatriz(0, 2): 2
VarMatriz(1, 0): 1
VarMatriz(1, 1): 2
VarMatriz(1, 2): 3
VarMatriz(2, 0): 2
VarMatriz(2, 1): 3
VarMatriz(2, 2): 4
VarMatriz(3, 0): 3
VarMatriz(3, 1): 4
VarMatriz(3, 2): 5

----- Inicialização do array VarMatrizB -----
VarMatrizB(0, 0): 1
VarMatrizB(0, 1): 2
VarMatrizB(0, 2): 3
VarMatrizB(1, 0): 4
VarMatrizB(1, 1): 5
VarMatrizB(1, 2): 6

----- Redimensionamento do array VarValores -----
VarValores(0): 0
VarValores(1): 0
VarValores(2): 0
VarValores(3): 0
VarValores(4): 0
VarValores(5): 0
VarValores(6): 0
VarValores(7): 0
VarValores(8): 0

----- Redimensionamento do array VarMatriz -----
VarMatriz(0, 0): 0
VarMatriz(0, 1): 1
VarMatriz(0, 2): 2
VarMatriz(0, 3): 0
VarMatriz(0, 4): 0
VarMatriz(0, 5): 0
VarMatriz(1, 0): 1
VarMatriz(1, 1): 2
VarMatriz(1, 2): 3
VarMatriz(1, 3): 0
VarMatriz(1, 4): 0
VarMatriz(1, 5): 0
VarMatriz(2, 0): 2
VarMatriz(2, 1): 3
VarMatriz(2, 2): 4
VarMatriz(2, 3): 0
VarMatriz(2, 4): 0
VarMatriz(2, 5): 0
VarMatriz(3, 0): 3
VarMatriz(3, 1): 4
VarMatriz(3, 2): 5
VarMatriz(3, 3): 0
VarMatriz(3, 4): 0
VarMatriz(3, 5): 0

----- Redimensionamento do array VarCores -----
VarCores(0):
VarCores(1):
VarCores(2):
VarCores(3):

C:\CursoAspNet\Codigos>_
```

Capítulo 4 – Estruturas de Controle

Para o aplicativo executar a lógica desejada dependendo das informações que lhe são atribuídas, existem as estruturas de controle. Assim o aplicativo executará as instruções de acordo com uma certa decisão, tendo ou não a participação do internauta.

Estas estruturas de controle existem em qualquer linguagem de programação.

4.1 Estruturas de decisão

As estruturas de decisão ou também chamada de lógica condicional, permite especificar qual código deve ser executado dependendo das condições satisfeitas.

Este recurso torna-se necessário para qualquer lógica implementada.

4.1.1 Instrução **if .. then ... else ... elseif**

É a forma mais simples de se aplicar à lógica condicional.

A instrução **if** é simples. Se a condição for satisfeita então o código é executado.

Vamos ver a forma de sintaxe para esta instrução (**if**).

Para C#	If (condição) { Instruções; }
exemplo C#:	if (VarIdade >= 18) { Console.WriteLine("Usuário de maior idade"); }
Para VB	if (condição) Then Instruções end if
exemplo VB:	if VarIdade >= 18 Then Console.WriteLine("Usuário de maior idade") end if

A instrução **else** é opcional e indica a instrução que o aplicativo deve executar se a condição imposta pelo **if** não for satisfatória.

Temos também a instrução **elseif** que permite testar outras condições.

Vamos ver a forma de sintaxe para estas instruções (**else**, **elseif**).

Para C#

```
If ( condição ) {
    Instruções;
}
else if ( condição ) {
    Instruções;
}
else {
    Instruções;
}
```

exemplo C#:

```
If ( Numero < 10 ) {
    Console.WriteLine("O Número possui 1 dígito");
}
else if ( Numero < 100 ) {
    Console.WriteLine("O Número possui 2 dígitos");
}
else {
    Console.WriteLine("O Nº possui mais de 2 dígitos");
}
```

outra forma para c#: **if** (condição ? InstrucaoVerdade : InstrucaoFalsa);

exemplo c#: Console.Write (idade >= 18 ? "Maior idade" : "Menor idade");

Para VB

```
If ( condição ) then
    Instruções
elseif ( condição ) then
    Instruções
else
    Instruções
end if
```

exemplo VB:

```
If ( Number < 10 ) then
    Console.WriteLine("O Número possui 1 digito")
elseif ( Number < 100 ) then
    Console.WriteLine("O Número possui 2 dígitos")
else
    Console.WriteLine("O Nº possui mais de 2 dígitos")
end if
```

outra forma para vb: **iif** (condição , InstrucaoVerdade , InstrucaoFalsa)

exemplo vb: Console.Write(**iif** (idade >= 18 , "Maior idade" , "Menor idade"))

4.1.2 Instrução switch(c#) e case(vb)

A instrução switch(c#) e case(vb) são utilizadas em situações em que houver a necessidade de diversos testes condicionais. São basicamente uma combinação de instrução **if** com orações de **elseif**.

Estas instruções examinam uma condição e especifica qual bloco de código deve ser executado.

Vamos examinar a sintaxe da instrução **switch(c#)**.

```
switch ( condição )  
{  
    case condição 1 : Instruções; break;  
  
    case condição 2 : Instruções; break;  
  
    case condição N : Instruções; break;  
  
    default: instruções default;  
}
```

Exemplo:

```
switch ( Estado )  
{  
    case "PR" : Console.WriteLine("Viagem para Curitiba"); break;  
  
    case "MG" : Console.WriteLine("Viagem para Belo Horizonte"); break;  
  
    case "RJ" : Console.WriteLine("Viagem para o Rio de Janeiro"); break;  
  
    default: Console.WriteLine("Viagem para São Paulo"); break;  
}
```

Vamos examinar a sintaxe para a instrução **Select case(vb)**.

```
select case condição  
  
    case condição 1 : Instruções  
  
    case condição 2 : Instruções  
  
    case condição N : Instruções  
  
    case else: instruções default  
  
end select
```

Exemplo:

Select case Estado

```
    case "PR" : Console.WriteLine("Viagem para Curitiba")  
  
    case "MG" : Console.WriteLine("Viagem para Belo Horizonte")  
  
    case "RJ" : Console.WriteLine("Viagem para o Rio de Janeiro")  
  
    case select: Console.WriteLine("Viagem para São Paulo")  
  
end select
```

4.2 Estruturas de Repetição

As estruturas de repetição são utilizadas em situações onde é preciso executar um conjunto de instruções determinadas vezes, até que a condição imposta para o fim da repetição for satisfeita.

4.2.1 Instrução For

Esta estrutura de repetição é utilizada quando o desenvolvedor sabe quantas vezes o código deve ser executado. Por isso devemos definir as faixas inicial e final.

Esse loop incrementa um contador para dizer quando encerrar a execução desta repetição.

Vamos acompanhar abaixo a sintaxe desta instrução (**for**).

```
Para C#:      for ( variável; condição; incremento )
              {
                Instruções;
              }

Exemplo c#:   for ( int i=1; i <= 10; i++ )
              {
                Console.WriteLine("Valor de i: " + i );
              }

Para vb:      for variavel to condicao
              Console.WriteLine("Valor de i: " & i );
              Next

Exemplo vb:   for i=1 to 10
              Console.WriteLine("Valor de i: " & i );
              Next
```

Uma variação desta instrução é o **foreach...in(c#)** ou **for...each(vb)**.

A diferença desta instrução é que o desenvolvedor não precisa especificar as faixas inicial e final. É utilizado para percorrer uma coleção ou um array por exemplo.

Vamos acompanhar a sintaxe da instrução.

```
Para C#:      foreach ( tipo variável in coleção/array )
              {
                instruções;
              }

exemplo:      foreach ( int i in VetorInteiro )
              {
                Console.WriteLine("Valor do vetor: {0}", i );
              }

Para vb:      For Each variável in coleção/array
              Instruções;
              Next

exemplo:      For Each Numero in VetorInteiro
              Console.WriteLine("Valor do vetor:" & Numero )
              Next
```

4.2.2 Instrução While

Esta instrução executa um loop enquanto uma determinada condição de execução for satisfatória. **While** significa **enquanto**, portanto enquanto a condição for verdade serão executadas as instruções.

Acompanhe a sintaxe abaixo:

```
Para c#:  
    while ( condição )  
    {  
        Instrucoes;  
    }  
  
exemplo c#:  
    while ( Numero < 10 )  
    {  
        Console.WriteLine( Numero );  
    }  
  
Para vb:  
    while ( condição )  
        Instrucoes  
    end while  
  
exemplo vb:  
    while Numero < 10  
        Console.WriteLine( Numero )  
    end while
```

4.2.3 Instrução Do While

O loop **do while** tem a mesma função da instrução **While**, a única diferença é que a condição é testada no final da instrução. Com isso podemos assegurar em alguns casos na linguagem C# que o loop seja realizado pelo menos uma única vez.

Acompanhe a sintaxe abaixo:

```
Para c#:  
    do  
    {  
        Instrucoes;  
    }  
    while ( condição );  
  
exemplo c#:  
    do  
    {  
        Console.WriteLine( Numero );  
    }  
    while ( Numero < 10 );  
  
Para vb:  
    do while ( condição )  
        Instrucoes  
    loop  
  
exemplo vb:  
    do while Numero < 10  
        Console.WriteLine( Numero )  
    loop
```

4.2.4 Instrução Do Loop (apenas Visual Basic.Net)

O objetivo desta instrução é fazer o papel do looping **While** e **Do While**, mas com a diferença de executar pelo menos uma vez a repetição.

Acompanhe a sintaxe abaixo:

```
Para vb:      Do
               Instruções
               Loop Until condição

Exemplo vb:   Do
               Console.WriteLine( Numero )
               Loop Until numero < 10
```

Exemplos e Exercícios:

Exemplo 1 – Estruturas de controle na linguagem C# .

```
using System;
public class ExemploIf
{
    public static void Main()
    {
        int VarNumero = 0;
        Console.WriteLine(" ----- Instrução IF ----- ");
        Console.Write("Forneça um número: ");
        string Leia = Console.ReadLine( );
        VarNumero = int.Parse( Leia );
        if (VarNumero < 10) {
            Console.WriteLine(" O Numero possui 1 digito ");
        } else if (VarNumero < 100) {
            Console.WriteLine(" O Numero possui 2 dígitos ");
        }
        else { Console.WriteLine(" o Numero possui 3 ou mais números ");
        }

        Console.WriteLine( );
        Console.WriteLine(" ----- Instrução SWITCH ----- ");

        Console.Write("forneça a sigla de um Estado da região Sudeste: ");
        Leia = Console.ReadLine();
        Leia = Leia.ToUpper(); // Transforma o string com letras maiúsculas.
        switch ( Leia )
        {
            case "MG": Console.WriteLine("A capital se chama: Belo Horizonte"); break;
            case "SP": Console.WriteLine("A capital se chama: São Paulo"); break;
            case "RJ": Console.WriteLine("A capital se chama: Rio de Janeiro"); break;
            case "ES": Console.WriteLine("A capital se chama: Vitória"); break;
            default: Console.WriteLine("Você não digitou uma sigla correta"); break;
        }

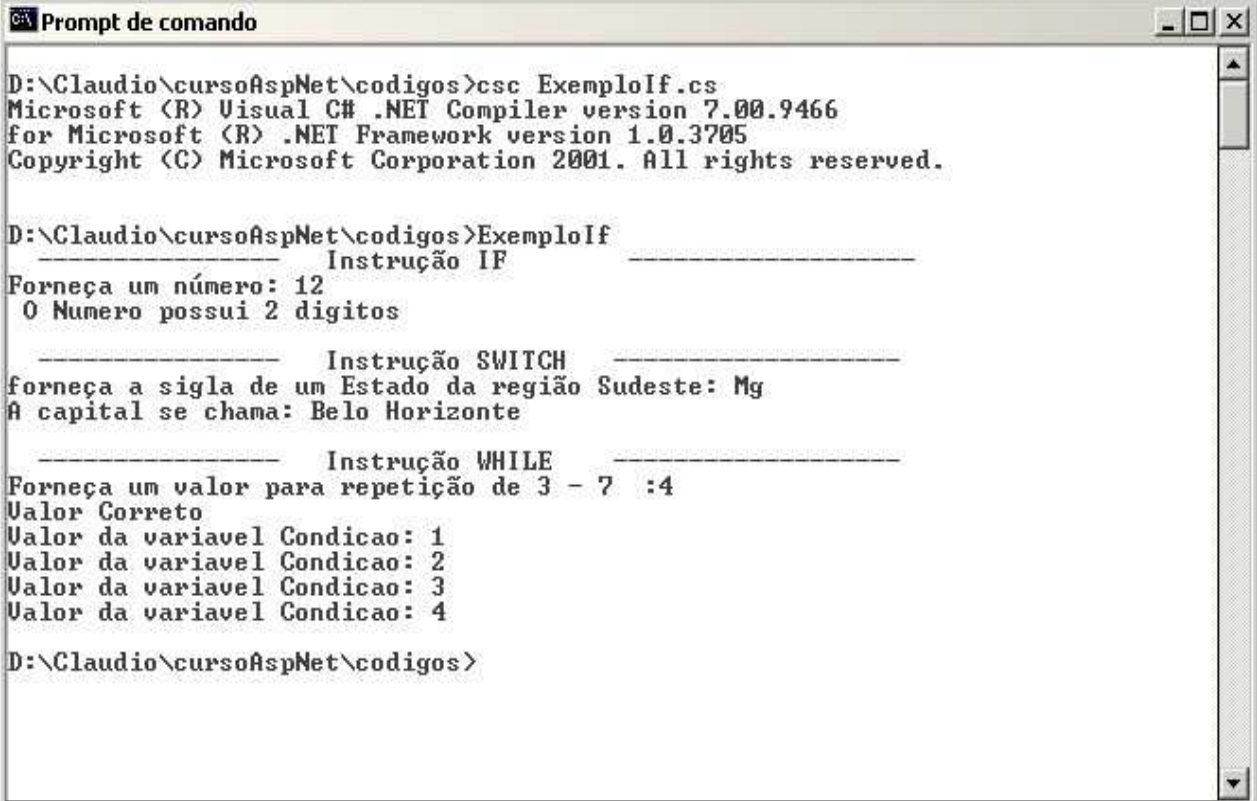
        Console.WriteLine( );
        Console.WriteLine(" ----- Instrução WHILE ----- ");
        int Condicao = 1;
        Console.Write("Forneça um valor para repetição de 3 - 7 :");
        Leia = Console.ReadLine( );
        VarNumero = int.Parse( Leia );
        int Final =1;

        if (VarNumero > 2) {
            if (VarNumero < 8) {
                Final = VarNumero;
            }
            else { Final = 2;
            }
        }

        Console.WriteLine( Final==2 ? "Valor Incorreto" : "Valor Correto" );

        while ( Condicao <= Final )
        {
            Console.WriteLine("Valor da variavel Condicao: {0}", Condicao);
            Condicao++;
        }
    }
}
```


Acompanhe a saída do programa acima:



```
D:\Claudio\cursoAspNet\codigos>csc ExemploIf.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

D:\Claudio\cursoAspNet\codigos>ExemploIf
_____ Instrução IF _____
Forneça um número: 12
O Numero possui 2 digitos

_____ Instrução SWITCH _____
forneça a sigla de um Estado da região Sudeste: Mg
A capital se chama: Belo Horizonte

_____ Instrução WHILE _____
Forneça um valor para repetição de 3 - 7 :4
Valor Correto
Valor da variavel Condicao: 1
Valor da variavel Condicao: 2
Valor da variavel Condicao: 3
Valor da variavel Condicao: 4

D:\Claudio\cursoAspNet\codigos>
```

Exemplo 2 – Estruturas de controle na linguagem Visual Basic.Net

```
Imports System
Module ExemploVb
    Sub Main()
        Dim VarNumero As Integer = 0

        Console.WriteLine(" ----- Instrução IF ----- ")

        Console.WriteLine("Forneça um número: ")
        Dim Leia As String
        Leia = Console.ReadLine()
        VarNumero = Integer.Parse(Leia)

        If (VarNumero < 10) Then
            Console.WriteLine(" O Numero possui 1 digito ")
        ElseIf (VarNumero < 100) Then
            Console.WriteLine(" O Numero possui 2 dígitos ")
        Else : Console.WriteLine(" o Numero possui 3 ou mais números ")
        End If

        Console.WriteLine()
        Console.WriteLine(" ----- Instrução SELECT CASE ----- ")

        Console.WriteLine("forneça a sigla de um Estado da região Sudeste: ")
        Leia = Console.ReadLine()
        Leia = Leia.ToUpper()

        Select Case Leia
            Case "MG" : Console.WriteLine("A capital se chama: Belo Horizonte")
            Case "SP" : Console.WriteLine("A capital se chama: São Paulo")
            Case "RJ" : Console.WriteLine("A capital se chama: Rio de Janeiro")
            Case "ES" : Console.WriteLine("A capital se chama: Vitória")
            Case Else : Console.WriteLine("Você não digitou uma sigla correta")
        End Select

        Console.WriteLine()
        Console.WriteLine(" ----- Instrução WHILE ----- ")
        Dim Condicao As Integer = 1
        Console.WriteLine("Forneça um valor para repetição de 3 - 7 :")
        Leia = Console.ReadLine()
        VarNumero = Integer.Parse(Leia) 'Transforma string em inteiro
        Dim Final As Integer = 1

        If VarNumero > 2 And VarNumero < 8 Then
            Final = VarNumero
        Else : Final = 2
        End If

        if Final = 2 then
            Console.WriteLine("Valor Incorreto")
        Else : Console.WriteLine("Valor Correto")
        End if

        While Condicao <= Final
            Console.WriteLine("Valor da variavel Condicao: " & Condicao)
            Condicao = Condicao + 1
        End While

        End Sub
    End Module
```

Acompanhe a saída do programa acima:

```
C:\CursoAspNet\Codigos>vbc ExemploVb.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>ExemploVb
----- Instrução IF -----
Forneça um número: 12
O Numero possui 2 digitos

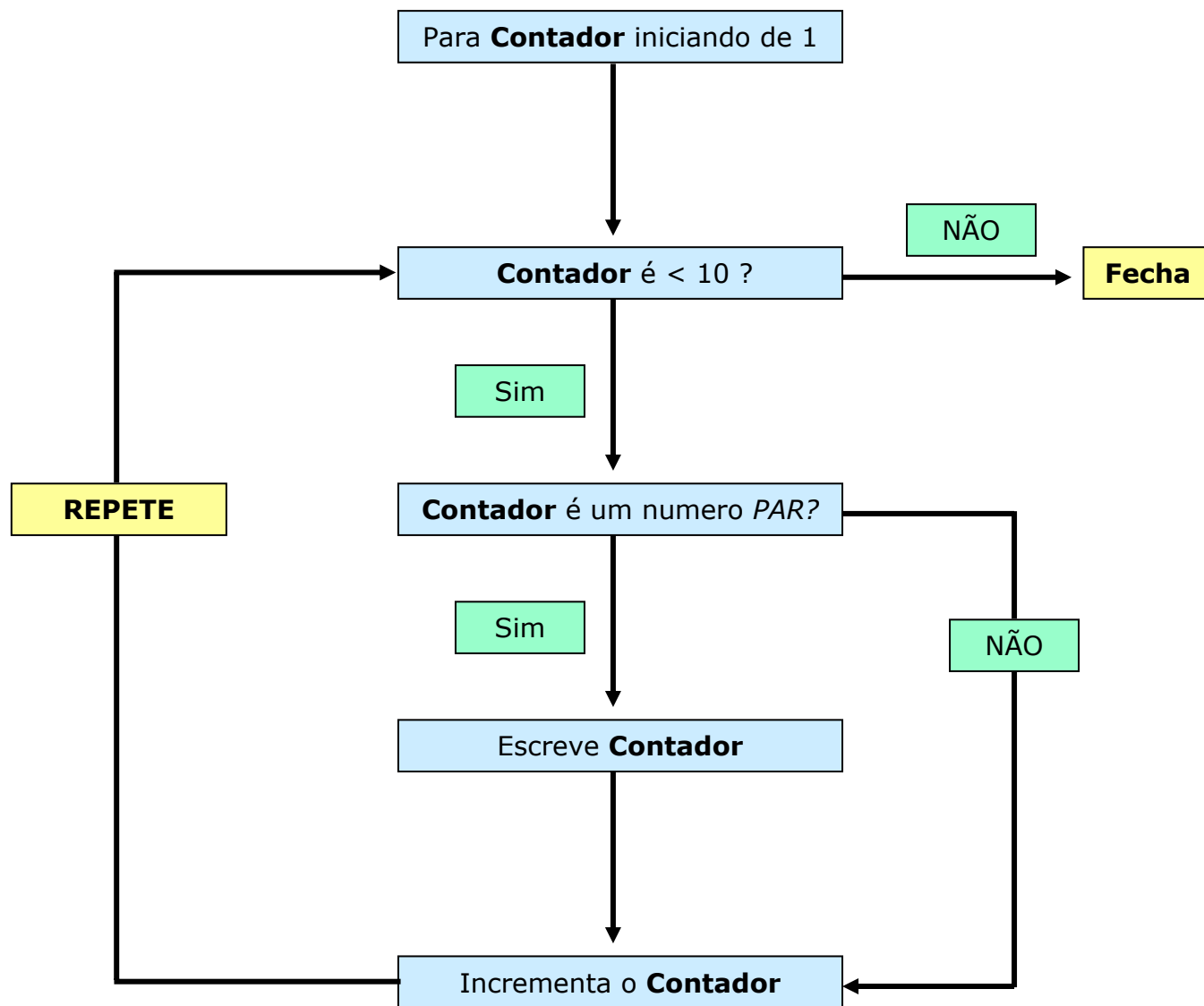
----- Instrução SELECT CASE -----
forneça a sigla de um Estado da região Sudeste: mG
A capital se chama: Belo Horizonte

----- Instrução WHILE -----
Forneça um valor para repetição de 3 - 7 :4
Valor Correto
Valor da variavel Condicao: 1
Valor da variavel Condicao: 2
Valor da variavel Condicao: 3
Valor da variavel Condicao: 4
C:\CursoAspNet\Codigos>
```

Exercício

Implemente nas linguagens Visual Basic.Net e C# o esboço de um looping mostrado abaixo na figura 4.3.

Figura 4.3 – Gráfico de um looping.



Capítulo 5 - Formatação

A formatação é muito utilizada pelos desenvolvedores em suas aplicações. Este processo busca uma forma de apresentação mais inteligível e funcional, em função das configurações de cada país, principalmente na formatação de valores monetários e datas. Para isso a plataforma .Net traz muitas funções para as linguagens C# e VB.net.

Formatação de Números

Para a formatação de números você pode utilizar funções prontas das linguagens da plataforma .Net, ou criar seu próprio formato personalizado.

Assim os desenvolvedores conseguirão trabalhar com valores monetários específicos de cada região.

5.1 Formatação de Números na Linguagem C#

É essencial a formatação de números nas aplicações desenvolvidas.

A linguagem C# trabalha bem com estas formatações. Para isso destacaremos a função ToString:

- **ToString()**: retorna a formatação padrão dos tipos numéricos. Este método é disponível para os tipos numéricos. Aceita uma string de formatação para os tipos como – valores monetários, decimal, ponto fixo, dentre outros, e uma referência para uma cultura.

Acompanhe abaixo a sintaxe do método *ToString()*:

```
VariavelNumerica.ToString("StringFormatação", ReferenciaCultural);
```

```
VarValor.ToString("c", null);
```

Figura 5.1.1 – Formatação de Números.

```
using System;
using System.Globalization;

public class FormatC
{
    public static void Main()
    {
        double VarValor = 15680.1248D;
        string VarMostra;

        CultureInfo us = new CultureInfo("en-US"); //Formatacao USA
        VarMostra = VarValor.ToString("n",us);
        Console.WriteLine("ToString('n',us) = "+VarMostra);
        VarMostra = VarValor.ToString("n6",us);
        Console.WriteLine("ToString('n6',us) = "+VarMostra);
        VarMostra = VarValor.ToString("c",us);
        Console.WriteLine("ToString('c',us) = "+VarMostra);
        Console.WriteLine();

        CultureInfo br = new CultureInfo("pt-BR"); //Formatacao Brasil
        VarMostra = VarValor.ToString("n",br);
        Console.WriteLine("ToString('n',br) = "+VarMostra);
        VarMostra = VarValor.ToString("e",br);
        Console.WriteLine("ToString('e',br) = "+VarMostra);
        VarMostra = VarValor.ToString("c",br);
        Console.WriteLine("ToString('c',br) = "+VarMostra);
        Console.WriteLine();

        CultureInfo fr = new CultureInfo("fr-FR"); //Formatacao França
        VarMostra = VarValor.ToString("n",fr);
        Console.WriteLine("ToString('n',fr) = "+VarMostra);
        VarMostra = VarValor.ToString("f3",fr);
        Console.WriteLine("ToString('f3',fr) = "+VarMostra);
        VarMostra = VarValor.ToString("c4",fr);
        Console.WriteLine("ToString('c4',fr) = "+VarMostra);
        Console.WriteLine();

        CultureInfo es = new CultureInfo("es-ES"); //Formatacao Espanha
        VarMostra = VarValor.ToString("n3",es);
        Console.WriteLine("ToString('n3',es) = "+VarMostra);
        VarMostra = VarValor.ToString("g4",es);
        Console.WriteLine("ToString('g4',es) = "+VarMostra);
        VarMostra = VarValor.ToString("c2",es);
        Console.WriteLine("ToString('c2',es) = "+VarMostra);
    }
}
```

Acompanhe a saída do exemplo anterior.

```

C:\CursoAspNet\Codigos>csc FormatC.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

C:\CursoAspNet\Codigos>FormatC
ToString('n',us) = 15,680.12
ToString('n6',us) = 15,680.124800
ToString('c',us) = $15,680.12

ToString('n',br) = 15.680,12
ToString('e',br) = 1,568012e+004
ToString('c',br) = R$ 15.680,12

ToString('n',fr) = 15 680,12
ToString('f3',fr) = 15680,125
ToString('c4',fr) = 15 680,1248 ?

ToString('n3',es) = 15.680,125
ToString('g4',es) = 1,568e+04
ToString('c2',es) = 15.680,12 ?

C:\CursoAspNet\Codigos>

```

A figura 5.1.1 mostra-nos os caracteres de formatação que são utilizados na função ToString().

Figura 5.1.1 - Caracteres de Formatação.

Caractere	Descrição
C ou c	Representam valores monetários
D ou d	Formatação Decimal. Apenas para números Inteiros.
E ou e	Formato de Notação Científica
F ou f	Representam o formato de Ponto Fixo
G ou g	Formato Geral dos números
N ou n	Representação Numérica
P ou p	Formato de porcentagem
R ou r	Assegura que um número convertido para string, terá o mesmo formato se convertido de volta para número.
X ou x	Formatação para Hexadecimal

5.2 Formatação de Números na Linguagem Visual Basic.Net

Toda tecnologia que se preze deve trabalhar com eficiência na formatação de informações.

O Visual Basic.Net traz três importantes funções para manipulação de números além do método `ToString()` utilizado também pela linguagem C#. Sendo assim o desenvolvedor poderá criar seus formatos personalizados, na maneira em que lhe convier necessário.

Vejamos na figura 5.2.1 as três funções de formatação.

Figura 5.2.1 – descrição das funções de formatação numérica.

Função	Descrição
FormatNumber (expressão, <i>casas decimais</i>)	Formata o número de acordo com o numero de casas decimais definido.
FormatCurrency (expressão, <i>casas decimais</i>)	Formata o numero de acordo com o numero de casas decimais definido e mais a inserção do símbolo da moeda.
FormatPercent (expressão, <i>casas decimais</i>)	Formata o numero de acordo com o numero de casas decimais definido e multiplica o resultado por 100.

Vamos ver com detalhes a sintaxe das funções de formatação:

FormatNumber (**campo1**, *campo2*, *campo3*, *campo4*, *campo5*)

FormatCurrency (**campo1**, *campo2*, *campo3*, *campo4*, *campo5*)

FormatPercent (**campo1**, *campo2*, *campo3*, *campo4*, *campo5*)

campo1 – de uso obrigatório. É a expressão da função.

campo2 – de uso opcional. Valor numérico que indica quantas casas serão exibidas à direita do número. O valor padrão é -1 que indicam que as colocações regionais do computador são usadas.

campo3 – de uso opcional. Faz uso da constante `TriState`, que indica se um zero principal é exibido para valores fracionários. Veja as configurações para a constante `TriState` mais em diante.

campo4 – de uso opcional. Faz uso da constante `TriState`, que indica se deve colocar valores negativos dentro de parênteses.

campo5 – de uso opcional. Faz uso da constante `TriState`, que indica se deve agrupar os números usando a delimitação dos grupos especificados nas colocações de locais.

Configurações para a constante **TriState**:

Constante	Valor	Descrição
TriState.True	-1	Condição verdadeira (True)
TriState.False	0	Condição Falsa (False)
TriState.UseDefault	-2	Configurações padrão do computador.

Vejamos um exemplo para estas funções de formatação:

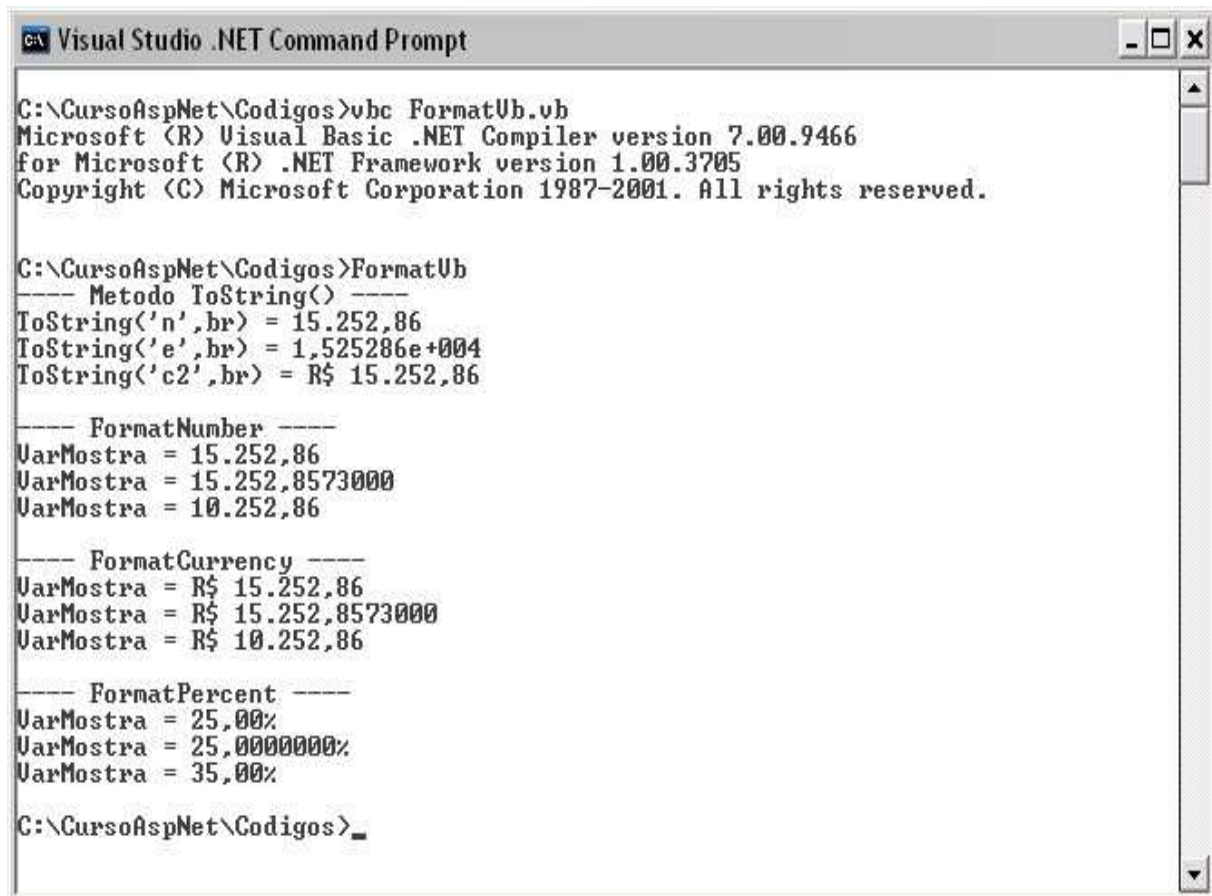
```
Imports System
Imports System.Globalization
Imports Microsoft.VisualBasic.Strings

Module FormatVB
    Sub Main()
        Dim VarValor As Double = 15252.8573
        Dim VarMostra As String
        Dim br As New CultureInfo("pt-BR")

        Console.WriteLine("---- Método ToString() ----")
        VarMostra = VarValor.ToString("n",br)
        Console.WriteLine("ToString('n',br) = "& VarMostra)
        VarMostra = VarValor.ToString("e",br)
        Console.WriteLine("ToString('e',br) = "& VarMostra)
        VarMostra = VarValor.ToString("c2",br)
        Console.WriteLine("ToString('c2',br) = "& VarMostra)
        Console.WriteLine()
        Console.WriteLine("---- FormatNumber ----")
        VarMostra = FormatNumber(VarValor, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatNumber(VarValor, 7, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatNumber(VarValor - 5000, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        Console.WriteLine()
        Console.WriteLine("---- FormatCurrency ----")
        VarMostra = FormatCurrency(VarValor, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatCurrency(VarValor, 7, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatCurrency(VarValor - 5000, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        Console.WriteLine()
        VarValor = 0.25
        Console.WriteLine("---- FormatPercent ----")
        VarMostra = FormatPercent(VarValor, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatPercent(VarValor, 7, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)
        VarMostra = FormatPercent(VarValor + 0.10, 2, -1, -1, -2)
        Console.WriteLine("VarMostra = " & VarMostra)

    End Sub
End Module
```

Veja a saída do programa acima:



```
C:\CursoAspNet\Codigos>vbc FormatUb.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>FormatUb
----- Metodo ToString() -----
ToString('n',br) = 15.252,86
ToString('e',br) = 1,525286e+004
ToString('c2',br) = R$ 15.252,86

----- FormatNumber -----
VarMostra = 15.252,86
VarMostra = 15.252,85730000
VarMostra = 10.252,86

----- FormatCurrency -----
VarMostra = R$ 15.252,86
VarMostra = R$ 15.252,85730000
VarMostra = R$ 10.252,86

----- FormatPercent -----
VarMostra = 25,00%
VarMostra = 25,00000000%
VarMostra = 35,00%

C:\CursoAspNet\Codigos>
```

Alem das funções discutidas acima, vamos apresentar formatos genéricos e personalizados de formatação que a linguagem Visual Basic.Net suporta.

A seguir vamos conferir a sintaxe do método Format():

Format (*Expressão*, *caracter de formatação* ou *caracteres especiais de formatação*)

Format (*VariavelValor*, "C4")

Format (*VariavelEspecial*, "##000000.00#-000#")

Veja na figura 5.2.2 os caracteres de formatação permitidos e na figura 5.2.3 os caracteres utilizados na formatação personalizada dos números.

Figura 5.2.2 – Caracteres de Formatação.

Caractere	Descrição
C ou c	Representam valores monetários
D ou d	Formatação Decimal. Apenas para números Inteiros.
E ou e	Formato de Notação Científica
F ou f	Representam o formato de Ponto Fixo
G ou g	Formato Geral dos números
N ou n	Representação Numérica
P ou p	Formato de porcentagem
R ou r	Assegura que um número convertido para string, terá o mesmo formato se convertido de volta para número.
X ou x	Formatação para Hexadecimal
Yes / No	Se o número for zero retorna No, senão retorna Yes.
True / False	Se o número for zero retorna False, senão retorna True.
On - Off	Se o número for zero retorna Off, senão retorna On.

Figura 5.2.3 – Caracteres especiais para Formatação Personalizada.

Caractere	Descrição
None	Exibe o numero sem formatação.
(0)	Exibe o digito ou zero. Se na expressão contiver o dígito onde tiver 0, então é exibido o digito, ao contrario será exibe o numero zero.
(#)	Exibe o digito ou nada. Se na expressão contiver o dígito onde tiver #, então é exibido o digito, ao contrario não exibe nada.
(.)	Usado como separador de decimal.
(%)	Usado para exibir um valor percentual, sempre lembrando que este valor será multiplicado por 100.
(,)	Usado para separar centenas e milhares.
(E- E+ e- e+)	Usado para exibir formato científico.

Exemplos e Exercícios.

Exemplo 01 – Formatos genéricos e personalizados.

```
Imports System
Imports Microsoft.VisualBasic.Strings

Module FormatVB

    Sub Main()
        Dim VarValor As Double = 15252.8573
        Dim VarMostra As String

        Console.WriteLine("---- Format Padrão ----")
        VarMostra = Format(VarValor, "g")
        Console.WriteLine(" Format(VarValor, 'g'): "& VarMostra)
        VarMostra = Format(VarValor, "c")
        Console.WriteLine(" Format(VarValor, 'c'): "& VarMostra)
        VarMostra = Format(VarValor, "e")
        Console.WriteLine(" Format(VarValor, 'e'): "& VarMostra)
        VarMostra = Format(VarValor, "n")
        Console.WriteLine(" Format(VarValor, 'n'): "& VarMostra)
        Console.WriteLine( )

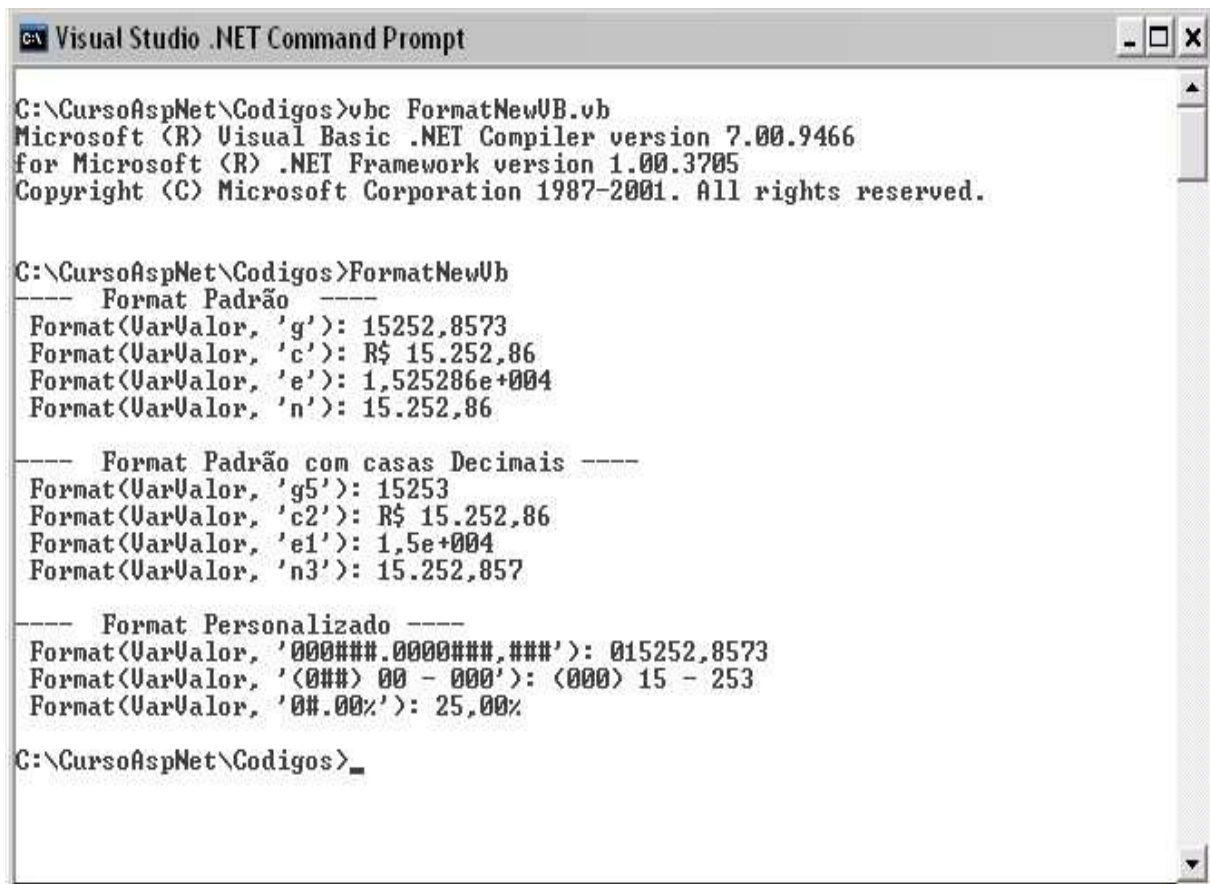
        Console.WriteLine("---- Format Padrão com casas Decimais ----")
        VarMostra = Format(VarValor, "g5")
        Console.WriteLine(" Format(VarValor, 'g5'): "& VarMostra)
        VarMostra = Format(VarValor, "c2")
        Console.WriteLine(" Format(VarValor, 'c2'): "& VarMostra)
        VarMostra = Format(VarValor, "e1")
        Console.WriteLine(" Format(VarValor, 'e1'): "& VarMostra)
        VarMostra = Format(VarValor, "n3")
        Console.WriteLine(" Format(VarValor, 'n3'): "& VarMostra)
        Console.WriteLine( )

        Console.WriteLine("---- Format Personalizado ----")
        VarMostra = Format(VarValor, "000###.0000###,###")
        Console.WriteLine(" Format(VarValor, '000###.0000###,###'): "& VarMostra)
        VarMostra = Format(VarValor, "(0##) 00 - 000")
        Console.WriteLine(" Format(VarValor, '(0##) 00 - 000'): "& VarMostra)
        VarValor = 0.25
        VarMostra = Format(VarValor, "0#.00%")
        Console.WriteLine(" Format(VarValor, '0#.00%'): "& VarMostra)

    End Sub

End Module
```

Acompanhe a saída do programa acima:



```
C:\CursoAspNet\Codigos>vbc FormatNewUB.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>FormatNewUb
----- Format Padrão -----
Format<VarValor, 'g'>: 15252,8573
Format<VarValor, 'c'>: R$ 15.252,86
Format<VarValor, 'e'>: 1,525286e+004
Format<VarValor, 'n'>: 15.252,86

----- Format Padrão com casas Decimais -----
Format<VarValor, 'g5'>: 15253
Format<VarValor, 'c2'>: R$ 15.252,86
Format<VarValor, 'e1'>: 1,5e+004
Format<VarValor, 'n3'>: 15.252,857

----- Format Personalizado -----
Format<VarValor, '000###.0000###,###'>: 015252,8573
Format<VarValor, '(0##) 00 - 000'>: (000) 15 - 253
Format<VarValor, '0#.00%'>: 25,00%

C:\CursoAspNet\Codigos>
```

Formatação de Datas e Horas

Para um aplicativo que é utilizado em vários países, devemos ter o cuidado para o tratamento de datas e horas.

É necessária a formatação para as aplicações que necessitam das informações de datas e horas para executar determinadas funções.

5.3 Formatação de Datas e Horas na Linguagem C#

A linguagem C# sabendo da importância do tratamento de datas e horas, dispõe de duas classes para isso: `DateTime` e `TimeSpan`.

Vejamos abaixo a sintaxe do método `ToString()` para a formatação de datas.

```
VariavelDateTime.ToString("StringFormatação", ReferenciaCultural);
```

```
VarData.ToString("dddd", null);
```

Como visto acima, o método ToString() possui como parâmetros: uma string de formatação e uma expressão para a referência Cultural.

Acompanhe na figura 5.3.1 os caracteres de formatação correspondente para a manipulação de datas e horas.

Figura 5.3.1 – Caracteres de Formatação.

Caractere	Descrição	Exemplo
d	Exibe o dia do mês sem o zero	1, 2, 6, 12, 31
dd	Exibe o dia do mês com o zero	01, 02, 06, 12, 31
ddd	Exibe o nome abreviado do dia	Seg, Ter, Qua
dddd	Exibe o nome completo do dia	Segunda-feira
M	Exibe o mês sem o zero	1, 2, 6, 12
MM	Exibe o mês com o zero	01, 02, 06, 12
MMM	Exibe o nome abreviado do mês	Jan, Mar, Dez
MMMM	Exibe o nome completo do mês	Janeiro, Dezembro
y	Exibe os dois últimos dígitos do ano sem o zero	1, 2, 6, 99
yy	Exibe os dois últimos dígitos do ano com o zero	01, 02, 06, 99
yyyy	Exibe os quatro dígitos do ano	2001, 2002, 1999
h	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 12 horas.	1, 2, 6, 12
hh	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 12 horas.	01, 02, 06, 12
H	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 24 horas.	1, 2, 9, 13, 15
HH	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 24 horas.	01, 02, 09, 13, 15
m	Exibe os minutos sem o zero para minutos de 1 a 9.	1, 2, 6, 12, 58
mm	Exibe os minutos com o zero para minutos de 1 a 9.	01, 02, 06, 12, 58
s	Exibe os segundos sem o zero para seg. de 1 a 9.	1, 2, 3, 16, 59
ss	Exibe os segundos com o zero para seg. de 1 a 9.	01, 02, 03, 16, 59
:	Separador de tempo	13: 49: 52
/	Separador de data	13/ 01/ 2002

Veja o exemplo a seguir:

```
using System;
using System.Globalization;

public class FormDataC
{
    public static void Main(string[] args)
    {
        string VarMostra;
        CultureInfo us = new CultureInfo("en-US");
        CultureInfo br = new CultureInfo("pt-BR");
        CultureInfo fr = new CultureInfo("fr-FR");

        DateTime VarData = new DateTime(2001,02,19,13,25,20);

        Console.WriteLine("---- Formatos de Datas USA ---");
        VarMostra = VarData.ToString("dd",us);
        Console.WriteLine("ToString('dd') = "+ VarMostra);
        VarMostra = VarData.ToString("dddd",us);
        Console.WriteLine("ToString('dddd') = "+ VarMostra);
        VarMostra = VarData.ToString("MMM",us);
        Console.WriteLine("ToString('MMM') = "+ VarMostra);
        VarMostra = VarData.ToString("yyyy",us);
        Console.WriteLine("ToString('yyyy') = "+ VarMostra);
        Console.WriteLine();

        Console.WriteLine("---- Formatos de Datas Brasil ----");
        VarMostra = VarData.ToString("M",br);
        Console.WriteLine("ToString('M') = "+ VarMostra);
        VarMostra = VarData.ToString("dddd",br);
        Console.WriteLine("ToString('dddd') = "+ VarMostra);
        VarMostra = VarData.ToString("MM",br);
        Console.WriteLine("ToString('MM') = "+ VarMostra);
        VarMostra = VarData.ToString("yy",br);
        Console.WriteLine("ToString('yy') = "+ VarMostra);
        Console.WriteLine();

        Console.WriteLine("---- Formatos de Datas Francês ----");
        VarMostra = VarData.ToString("M",fr);
        Console.WriteLine("ToString('M') = "+ VarMostra);
        VarMostra = VarData.ToString("dddd",fr);
        Console.WriteLine("ToString('dddd') = "+ VarMostra);
        VarMostra = VarData.ToString("MMMM",fr);
        Console.WriteLine("ToString('MMMM') = "+ VarMostra);
        VarMostra = VarData.ToString("y",fr);
        Console.WriteLine("ToString('y') = "+ VarMostra);
        Console.WriteLine();
    }
}
```

A classe `DateTime` possui métodos e propriedades para auxiliar na manipulação de datas e horas.

Na figura 5.3.2 são descritos os principais métodos.

Figura 5.3.2 – Métodos da classe `DateTime`.

Método	Descrição
Add	Adiciona um valor a um <code>TimeSpan</code> .
AddDays	Soma um numero de dias a uma data.
AddHours	Soma um numero de horas a uma hora.
AddMonths	Soma um numero de meses a uma data.
AddYears	Soma um numero de anos a uma data.
Subtract	Subtrai um valor de uma data e hora.
ToString	Converte uma data e hora em uma string.
Compare	Compara as datas. Se as datas forem iguais o método retorna 0. Se a primeira data for maior retorna 1, senão retorna -1.
DaysInMonth	Exibe o numero de dias que o mês possui. Para isso o método exige de parâmetro o ano e o mês.
Parse	Cria uma string com a data e a hora na formatação especificada.

Na figura 5.3.3 são apresentadas as propriedades da Classe `DateTime`.

Figura 5.3.3 – Propriedades da classe `DateTime`.

Propriedade	Descrição
Now	Exibe a data e a hora atual do servidor.
Today	Exibe a data atual do Servidor.
Date	Exibe a data com o valor setado para meia-noite.
Day	Exibe o dia de uma data.
DayOfWeek	Exibe o dia da semana, sendo: 0-Domingo, 1-Segunda, ...
DayOfYear	Exibe o dia do ano, sendo de 1 até 366.
Hour	Exibe a hora.
Minute	Exibe os minutos.
Month	Exibe o mês de uma data.
Second	Exibe os segundos.
Year	Exibe o ano de uma data.

Exemplos e Exercícios

Exemplo 01 – Métodos e Propriedades da Classe DateTime.

```
using System;

public class FormDataMetodos
{
    public static void Main(string[] args)
    {
        string VarMostra;
        int VarDia;
        DateTime VarData;

        Console.WriteLine("Método DaysInMonth");
        for(int I=1; I<=12; I++)
        {
            VarDia = DateTime.DaysInMonth(2002,I);
            Console.WriteLine("Nº dias do mês "+I+" é igual a: "+VarDia);
        }

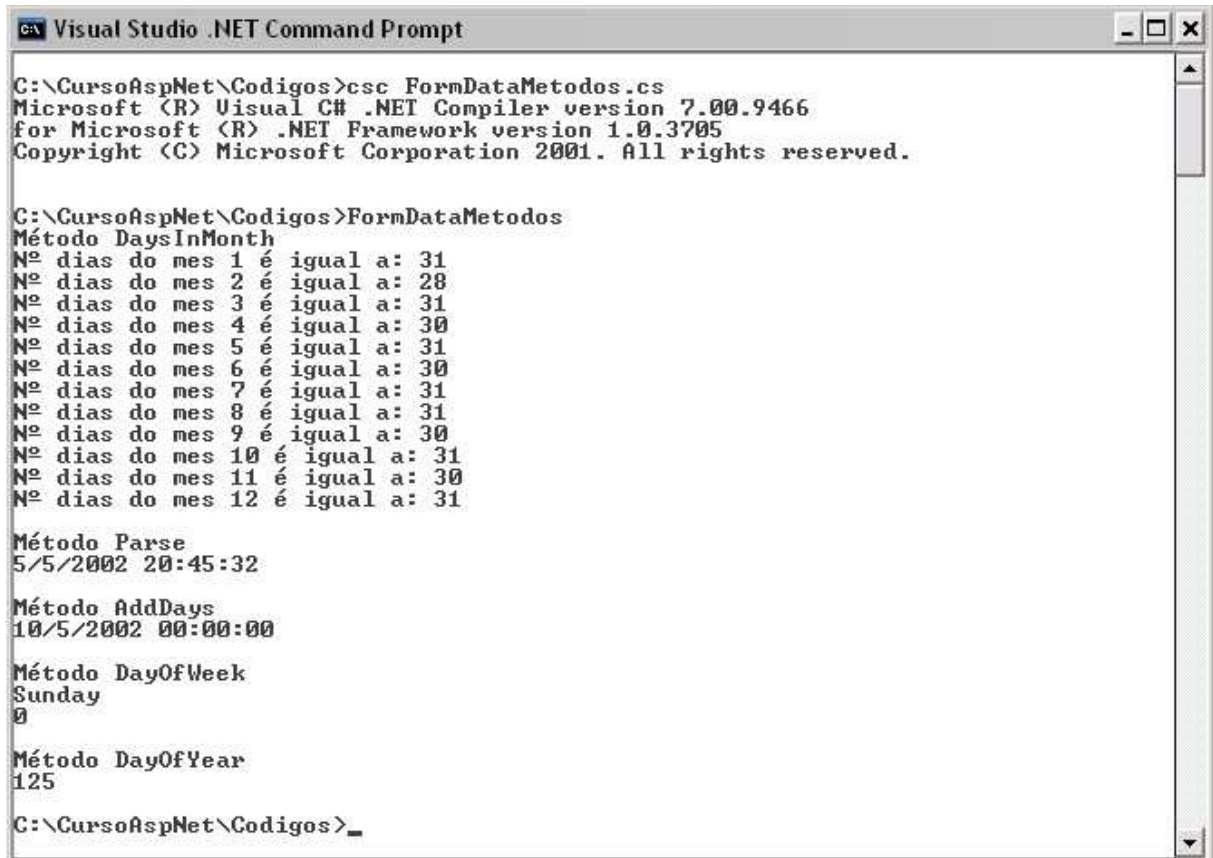
        Console.WriteLine("");
        Console.WriteLine("Método Parse");
        VarMostra = DateTime.Parse(DateTime.Now.ToString()).ToString();
        Console.WriteLine(VarMostra);

        Console.WriteLine("");
        Console.WriteLine("Método AddDays");
        VarData = DateTime.Today;
        Console.WriteLine(VarData.AddDays(5));

        Console.WriteLine("");
        Console.WriteLine("Método DayOfWeek");
        Console.WriteLine(VarData.DayOfWeek);
        int diaSemana = Convert.ToInt32(VarData.DayOfWeek);
        Console.WriteLine(diaSemana);

        Console.WriteLine("");
        Console.WriteLine("Método DayOfYear");
        Console.WriteLine(VarData.DayOfYear);
    }
}
```

Acompanhe a saída do programa acima.



```
C:\CursoAspNet\Codigos>csc FormDataMetodos.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

C:\CursoAspNet\Codigos>FormDataMetodos
Método DaysInMonth
Nº dias do mes 1 é igual a: 31
Nº dias do mes 2 é igual a: 28
Nº dias do mes 3 é igual a: 31
Nº dias do mes 4 é igual a: 30
Nº dias do mes 5 é igual a: 31
Nº dias do mes 6 é igual a: 30
Nº dias do mes 7 é igual a: 31
Nº dias do mes 8 é igual a: 31
Nº dias do mes 9 é igual a: 30
Nº dias do mes 10 é igual a: 31
Nº dias do mes 11 é igual a: 30
Nº dias do mes 12 é igual a: 31

Método Parse
5/5/2002 20:45:32

Método AddDays
10/5/2002 00:00:00

Método DayOfWeek
Sunday
0

Método DayOfYear
125

C:\CursoAspNet\Codigos>
```

5.4 Formatação de Datas e Horas na Linguagem Visual Basic.Net

O Visual Basic.Net dispõe de funções próprias para a formatação, seja utilizando os formatos existentes ou os formatos criados e personalizados pelo desenvolvedor.

Vejamos a sintaxe de um método comum, o ToString().

```
VariavelDateTime.ToString("StringFormatação", ReferenciaCultural)
```

```
VarData.ToString("dddd", null)
```

Como visto acima, o método ToString() possui como parâmetros: uma string de formatação e uma expressão para a referencia Cultural.

Acompanhe na figura 5.4.1 os caracteres de formatação correspondente para a manipulação de datas e horas.

Figura 5.4.1 – Caracteres de Formatação.

Caractere	Descrição	Exemplo
d	Exibe o dia do mês sem o zero	1, 2, 6, 12, 31
dd	Exibe o dia do mês com o zero	01, 02, 06, 12, 31
ddd	Exibe o nome abreviado do dia	Seg, Ter Qua
dddd	Exibe o nome completo do dia	Segunda-feira
M	Exibe o mês sem o zero	1, 2, 6, 12
MM	Exibe o mês com o zero	01, 02, 06, 12
MMM	Exibe o nome abreviado do mês	Jan, Mar, Dez
MMMM	Exibe o nome completo do mês	Janeiro, Dezembro
y	Exibe os dois últimos dígitos do ano sem o zero	1, 2, 6, 99
yy	Exibe os dois últimos dígitos do ano com o zero	01, 02, 06, 99
yyyy	Exibe os quatro dígitos do ano	2001, 2002, 1999
h	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 12 horas.	1, 2, 6, 12
hh	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 12 horas.	01, 02, 06, 12
H	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 24 horas.	1, 2, 9, 13, 15
HH	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 24 horas.	01, 02, 09, 13, 15
m	Exibe os minutos sem o zero para minutos de 1 a 9.	1, 2, 6, 12, 58
mm	Exibe os minutos com o zero para minutos de 1 a 9.	01, 02, 06, 12, 58
s	Exibe os segundos sem o zero para seg. de 1 a 9.	1, 2, 3, 16, 59
ss	Exibe os segundos com o zero para seg. de 1 a 9.	01, 02, 03, 16, 59
:	Separador de tempo	13: 49: 52
/	Separador de data	13/ 01/ 2002

Veja o exemplo a seguir:

```
Imports System
Imports System.Globalization
Module FormDataC
    Sub Main( )
        Dim VarMostra As String
        Dim br As New CultureInfo("pt-BR")
        Dim fr As New CultureInfo("fr-FR")
        Dim us As New CultureInfo("en-US")
        Dim VarData As new DateTime(2001,02,19,13,25,20)
        Console.WriteLine("---- Formatos de Datas USA ---")
        VarMostra = VarData.ToString("dd",us)
        Console.WriteLine("ToString('dd') = "& VarMostra)
        VarMostra = VarData.ToString("dddd",us)
        Console.WriteLine("ToString('dddd') = "& VarMostra)
        VarMostra = VarData.ToString("MMM",us)
        Console.WriteLine("ToString('MMM') = "& VarMostra)
        VarMostra = VarData.ToString("yyyy",us)
        Console.WriteLine("ToString('yyyy') = "& VarMostra)
        Console.WriteLine()

        Console.WriteLine("---- Formatos de Datas Brasil ----")
        VarMostra = VarData.ToString("M",br)
        Console.WriteLine("ToString('M') = "& VarMostra)
        VarMostra = VarData.ToString("dddd",br)
        Console.WriteLine("ToString('dddd') = "& VarMostra)
        VarMostra = VarData.ToString("MM",br)
        Console.WriteLine("ToString('MM') = "& VarMostra)
        VarMostra = VarData.ToString("yy",br)
        Console.WriteLine("ToString('yy') = "& VarMostra)
        Console.WriteLine()

        Console.WriteLine("---- Formatos de Datas Francês ----")
        VarMostra = VarData.ToString("M",fr)
        Console.WriteLine("ToString('M') = "& VarMostra)
        VarMostra = VarData.ToString("dddd",fr)
        Console.WriteLine("ToString('dddd') = "& VarMostra)
        VarMostra = VarData.ToString("MMMM",fr)
        Console.WriteLine("ToString('MMMM') = "& VarMostra)
        VarMostra = VarData.ToString("y",fr)
        Console.WriteLine("ToString('y') = "& VarMostra)
        Console.WriteLine()
    End Sub
End Module
```

Acompanhe a saída do programa acima:

```

C:\CursoAspNet\Codigos>vb FormDataUb.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>FormDataUb
----- Formatos de Datas USA -----
ToString('dd') = 19
ToString('dddd') = Monday
ToString('MMM') = Feb
ToString('yyyy') = 2001

----- Formatos de Datas Brasil -----
ToString('M') = 19 de fevereiro
ToString('dddd') = segunda-feira
ToString('MM') = 02
ToString('yy') = 01

----- Formatos de Datas Francês -----
ToString('M') = 19 février
ToString('dddd') = lundi
ToString('MMMM') = février
ToString('y') = février 2001

C:\CursoAspNet\Codigos>_
  
```

O Visual Basic.Net possui outra função para a manipulação de datas.

Veja a sintaxe da função `FormatDateTime` que pode retornar uma data, uma hora ou ambas. Esta função requer dois parâmetros – a expressão, e o formato.

FormatDateTime(expressão, formato)

FormatDateTime(VarData, DateFormat.GeneralDate)

FormatDateTime(VarData, DateFormat.ShortDate)

Veja no quadro abaixo os tipos de formatos para a função `FormatDateTime()`.

Formato	Descrição
<code>DateFormat.GeneralDate</code>	Exibe a data e a hora. Este é o padrão.
<code>DateFormat.LongDate</code>	Exibe a data completa.
<code>DateFormat.ShortDate</code>	Exibe a data no formato curto.
<code>DateFormat.LongTime</code>	Exibe a hora completa.
<code>DateFormat.ShortTime</code>	Exibe a hora no formato 24 horas. Exemplo: hh:mm.

Além da função `FormatDateTime` o Visual Basic.Net possui a função `Format()` que tem o objetivo de manipular datas para um formato personalizado.

Acompanhe a sintaxe da função abaixo.

Format (expressão , **stringFormatacao**)

Format (VarValor , **"ddd"**)

Veja na figura 5.4.2 os Strings de formatação possíveis para a função Format().

Figura 5.4.2 – Caracteres de Formatação.

Caractere	Descrição	Exemplo
d	Exibe o dia do mês sem o zero	1, 2, 6, 12, 31
dd	Exibe o dia do mês com o zero	01, 02, 06, 12, 31
ddd	Exibe o nome abreviado do dia	Seg, Ter, Qua
dddd	Exibe o nome completo do dia	Segunda-feira
M	Exibe o mês sem o zero	1, 2, 6, 12
MM	Exibe o mês com o zero	01, 02, 06, 12
MMM	Exibe o nome abreviado do mês	Jan, Mar, Dez
MMMM	Exibe o nome completo do mês	Janeiro, Dezembro
y	Exibe os dois últimos dígitos do ano sem o zero	1, 2, 6, 99
yy	Exibe os dois últimos dígitos do ano com o zero	01, 02, 06, 99
yyyy	Exibe os quatro dígitos do ano	2001, 2002, 1999
h	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 12 horas.	1, 2, 6, 12
hh	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 12 horas.	01, 02, 06, 12
H	Exibe as horas sem o zero para horas de 1 a 9. Apresenta formato de 24 horas.	1, 2, 9, 13, 15
HH	Exibe as horas com o zero para horas de 1 a 9. Apresenta formato de 24 horas.	01, 02, 09, 13, 15
m	Exibe os minutos sem o zero para minutos de 1 a 9.	1, 2, 6, 12, 58
mm	Exibe os minutos com o zero para minutos de 1 a 9.	01, 02, 06, 12, 58
s	Exibe os segundos sem o zero para seg. de 1 a 9.	1, 2, 3, 16, 59
ss	Exibe os segundos com o zero para seg. de 1 a 9.	01, 02, 03, 16, 59
:	Separador de tempo	13: 49: 52
/	Separador de data	13/ 01/ 2002

A linguagem Visual Basic.net traz ainda duas classes – DateTime e TimeSpan, que são recheadas de métodos e propriedades para facilitarem ainda mais o serviço de manipulação de datas e horas do desenvolvedor, como – adicionar dias, meses, horas, minutos, dentre outros.

Veja logo a seguir na figura 5.4.3 as principais propriedades desta classe.

Figura 5.4.3 – Propriedades das Classes DateTime.

Propriedade	Descrição
Now	Exibe a data e a hora atual do servidor.
Today	Exibe a data atual do Servidor.
Date	Exibe a data com o valor setado para meia-noite.
Day	Exibe o dia de uma data.
DayOfWeek	Exibe o dia da semana, sendo: 0-Domingo, 1-Segunda, ...
DayOfYear	Exibe o dia do ano, sendo de 1 até 366.
Hour	Exibe a hora.
Minute	Exibe os minutos.
Month	Exibe o mês de uma data.
Second	Exibe os segundos.
Year	Exibe o ano de uma data.

Acompanhe na figura 5.4.4 os principais métodos das classes DateTime.

Figura 5.4.4 – Métodos das Classes DateTime.

Método	Descrição
Add	Adiciona um valor a um TimeSpan.
AddDays	Soma um numero de dias a uma data.
AddHours	Soma um numero de horas a uma hora.
AddMonths	Soma um numero de meses a uma data.
AddYears	Soma um numero de anos a uma data.
Subtract	Subtrai um valor de uma data e hora.
ToString	Converte uma data e hora em uma string.
Compare	Compara as datas. Se as datas forem iguais o método retorna 0. Se a primeira data for maior retorna 1, senão retorna -1.
DaysInMonth	Exibe o numero de dias que o mês possui. Para isso o método exige de parâmetro o ano e o mês.
Parse	Cria uma string com a data e a hora na formatação especificada.

Exemplos e Exercícios

Exemplo 01 – Função Format().

```
Imports System
Imports Microsoft.VisualBasic
Imports Microsoft.VisualBasic.Strings

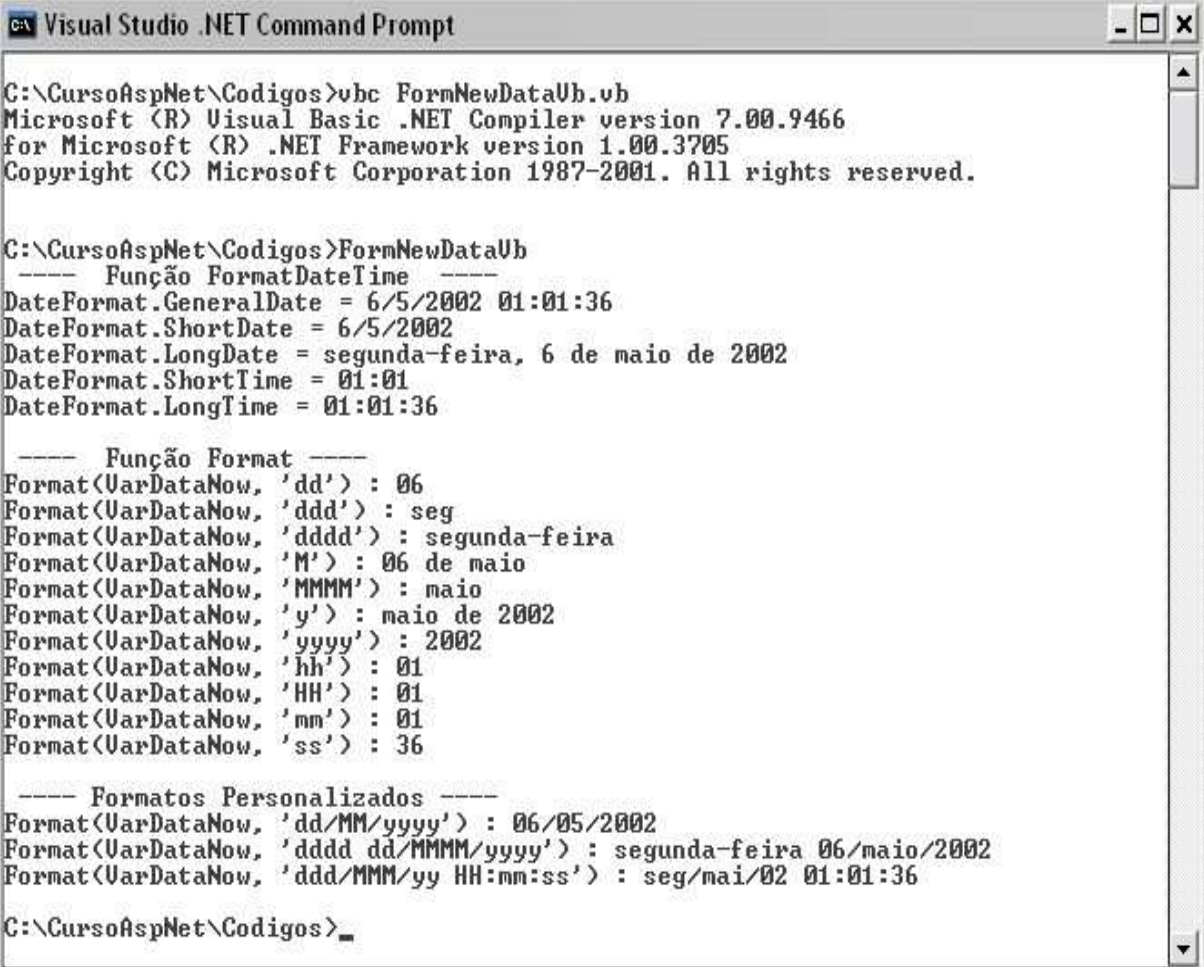
Module FormNewDataVB
    Sub Main( )
        Dim VarDataNow As DateTime = DateTime.Now
        Dim VarMostra As String

        Console.WriteLine(" ---- Função FormatDateTime ---- ")
        VarMostra = FormatDateTime(VarDataNow, DateFormat.GeneralDate)
        Console.WriteLine("DateFormat.GeneralDate = "& VarMostra)
        VarMostra = FormatDateTime(VarDataNow, DateFormat.ShortDate)
        Console.WriteLine("DateFormat.ShortDate = "& VarMostra)
        VarMostra = FormatDateTime(VarDataNow, DateFormat.LongDate)
        Console.WriteLine("DateFormat.LongDate = "& VarMostra)
        VarMostra = FormatDateTime(VarDataNow, DateFormat.ShortTime)
        Console.WriteLine("DateFormat.ShortTime = "& VarMostra)
        VarMostra = FormatDateTime(VarDataNow, DateFormat.LongTime)
        Console.WriteLine("DateFormat.LongTime = "& VarMostra)
        Console.WriteLine( )

        Console.WriteLine(" ---- Função Format ---- ")
        VarMostra = Format(VarDataNow, "dd")
        Console.WriteLine("Format(VarDataNow, 'dd') : "& VarMostra)
        VarMostra = Format(VarDataNow, "ddd")
        Console.WriteLine("Format(VarDataNow, 'ddd') : "& VarMostra)
        VarMostra = Format(VarDataNow, "dddd")
        Console.WriteLine("Format(VarDataNow, 'dddd') : "& VarMostra)
        VarMostra = Format(VarDataNow, "M")
        Console.WriteLine("Format(VarDataNow, 'M') : "& VarMostra)
        VarMostra = Format(VarDataNow, "MMMM")
        Console.WriteLine("Format(VarDataNow, 'MMMM') : "& VarMostra)
        VarMostra = Format(VarDataNow, "y")
        Console.WriteLine("Format(VarDataNow, 'y') : "& VarMostra)
        VarMostra = Format(VarDataNow, "yyyy")
        Console.WriteLine("Format(VarDataNow, 'yyyy') : "& VarMostra)
        VarMostra = Format(VarDataNow, "hh")
        Console.WriteLine("Format(VarDataNow, 'hh') : "& VarMostra)
        VarMostra = Format(VarDataNow, "HH")
        Console.WriteLine("Format(VarDataNow, 'HH') : "& VarMostra)
        VarMostra = Format(VarDataNow, "mm")
        Console.WriteLine("Format(VarDataNow, 'mm') : "& VarMostra)
        VarMostra = Format(VarDataNow, "ss")
        Console.WriteLine("Format(VarDataNow, 'ss') : "& VarMostra)
        Console.WriteLine( )

        Console.WriteLine(" ---- Formatos Personalizados ---- ")
        VarMostra = Format(VarDataNow, "dd/MM/yyyy")
        Console.WriteLine("Format(VarDataNow, 'dd/MM/yyyy') : "& VarMostra)
        VarMostra = Format(VarDataNow, "dddd dd/MMMM/yyyy")
        Console.WriteLine("Format(VarDataNow, 'dddd dd/MMMM/yyyy') : "& VarMostra)
        VarMostra = Format(VarDataNow, "ddd/MMM/yy HH:mm:ss")
        Console.WriteLine("Format(VarDataNow, 'ddd/MMM/yy HH:mm:ss') : "& VarMostra)
    End Sub
End Module
```


Veja a saída do programa - exemplo 01.



```
Visual Studio .NET Command Prompt

C:\CursoAspNet\Codigos>vbc FormNewDataUb.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

C:\CursoAspNet\Codigos>FormNewDataUb
----- Função FormatDateTime -----
DateFormat.GeneralDate = 6/5/2002 01:01:36
DateFormat.ShortDate = 6/5/2002
DateFormat.LongDate = segunda-feira, 6 de maio de 2002
DateFormat.ShortTime = 01:01
DateFormat.LongTime = 01:01:36

----- Função Format -----
Format(VarDataNow, 'dd') : 06
Format(VarDataNow, 'ddd') : seg
Format(VarDataNow, 'dddd') : segunda-feira
Format(VarDataNow, 'M') : 06 de maio
Format(VarDataNow, 'MMMM') : maio
Format(VarDataNow, 'y') : maio de 2002
Format(VarDataNow, 'yyyy') : 2002
Format(VarDataNow, 'hh') : 01
Format(VarDataNow, 'HH') : 01
Format(VarDataNow, 'mm') : 01
Format(VarDataNow, 'ss') : 36

----- Formatos Personalizados -----
Format(VarDataNow, 'dd/MM/yyyy') : 06/05/2002
Format(VarDataNow, 'dddd dd/MMMM/yyyy') : segunda-feira 06/maio/2002
Format(VarDataNow, 'ddd/MMM/yy HH:mm:ss') : seg/mai/02 01:01:36

C:\CursoAspNet\Codigos>
```

Capítulo 6 – WEB FORMS

Um formulário WEB é a maneira que a tecnologia .Net dispõe para o usuário interagir com o aplicativo, exercendo alguma função ou obtendo informações para satisfazer suas necessidades.

Web Forms pode ser definido como uma característica do ASP.NET que você usa para criar a interface do usuário para suas aplicações de Web.

Com a utilização desta tecnologia podemos associar os eventos ao código, o que antes era permitido usando apenas JavaScript. Você deve estar se perguntando – Será que vou conseguir construir um aplicativo para Web como um aplicativo DeskTop desenvolvido como no Delphi por exemplo?

A resposta é sim, você poderá construir seus aplicativos de uma maneira mais rápida e eficaz, utilizando o paradigma que as ferramentas de desenvolvimento como o Delphi utiliza.

Um outro ponto a ressaltar era para a construção de uma aplicação Web utilizando ASP (Active Server Pages), onde o programador utilizava uma gama de linguagens, como: JavaScript, HTML, VBScript, Flash, Perl, dentre outras, em uma única página, tendo cada uma destas linguagens uma função específica neste conjunto para obter o resultado esperado. Com isso o desenvolvedor era obrigado a entender de tudo um pouco.

Então a Microsoft pensando em facilitar para o desenvolvedor, criou em um único ambiente tudo que é necessário para criar uma aplicação DeskTop ou Web.

Enfim, podemos programar para a Web com linguagens de alta performance como o C#, Visual Basic.Net, J#, C++, Cobol.Net, dentre outras.

A Microsoft também desenvolveu uma ferramenta RAD (Rapid Application Design) que permite ao desenvolvedor atribuir com facilidade eventos e propriedades as páginas, aos controles e aos componentes.

A figura 6.1 traz um quadro mostrando os recursos disponíveis no Web Forms.

Figura 6.1 - Recursos no Web Forms.

Característica	Web Forms
Plataforma	Requer apenas o browser. No servidor deve estar rodando o .Net.
Modelo de programação	Modelo desconectado e assíncrono em que os componentes são adicionados na aplicação front end.
Uso de recursos (CPU, memória, disco)	Usa os recursos do servidor.
Gráficos	GD+ pode ser usado no servidor, pois a capacidade do browser não satisfaz.
Interatividade	Roda desconectado, por isso toda solicitação é preciso ir até ao servidor.
Controle do fluxo de texto e formulários	São baseados em HTML, permitindo ricos recursos de formatação.

6.1 Principais Eventos do Web Forms

Evento pode ser dito como uma ocorrência significativa no aplicativo que deve ser tratada pelo código. Ou ainda, um evento é um conjunto de instruções que ocorrem quando uma determinada ação é realizada pelo usuário.

Esta tecnologia permite ao desenvolvedor associar os eventos ao código da aplicação. Para isso estudaremos três eventos importantes.

Evento - Page_Load

O objeto PAGE contém todas as propriedades e métodos para cada página asp.net que você constrói. Ao executar uma página asp.net, ela herda a partir da classe Page na .Net Framework.

O evento Page_Load é muito importante para os desenvolvedores. Ele ocorre quando os formulários são carregados na aplicação.

Este evento tem bastante utilidade nas páginas Asp.Net, como – redirecionar usuários, carregar dados do banco de dados, verificar identidade de um usuário, criar componentes dinamicamente, dentre outros.

Uma propriedade importante do objeto Page é o IsPostBack.

A propriedade IsPostBack recebe *false* se o objeto Page está sendo chamado pela primeira vez, e recebe *true* quando o objeto Page é chamado pela segunda vez em diante.

Vejamos abaixo um exemplo na linguagem C#.

```
<html>
<form runat="server">
<asp:Label id="saida"
    runat="server"/>
<p align="center">
<asp:Button id="bot01"
    Text=" - Verificar - "
    OnClick="bot01_Clique"
    runat="server"/> </p>
<asp:Label id="verifica"
    runat="server"/>
</form>

<script language="C#" runat="server">
    static int Contador=1;
    public void Page_Load(Object sender, EventArgs ea)
    {
        verifica.Text = "IsPostBack = " + this.IsPostBack.ToString( );
    }

    public void bot01_Clique(Object sender, EventArgs ea)
    {
        saida.Text = "Você Atualizou a página "+Contador+" Vezes.";
        Contador++;
    }
</script>
</html>
```

Vejamos o exemplo anterior na linguagem VB.Net.

```
<html>
<form runat="server">
<asp:Label id="saida"
    runat="server"/>
<p align="center">
<asp:Button id="bot01"
    Text=" - Verificar - "
    OnClick="bot01_Clique"
    runat="server"/>
</p>
<asp:Label id="verifica"
    runat="server"/>
</form>

<script language="VB" runat="server">
    shared Contador As Integer = 1

    Sub Page_Load(Obj as Object, ea As EventArgs)
        verifica.Text = "IsPostBack = "& Me.IsPostBack.ToString( )
    End Sub

    Sub bot01_Clique(Obj As Object, ea As EventArgs)
        saida.Text = "Você Atualizou a página "& Contador &" Vezes."
        Contador = Contador + 1
    End Sub
</script>
</html>
```

Evento - **OnClick**

É um evento muito utilizado nas páginas asp.net.

Este evento ocorre quando um estímulo de um clique é percebido por um componente.

O componente Button faz-se necessário deste método.

Nos exemplos acima podemos perceber o evento OnClick.

Todo componente Button tem a propriedade OnClick. Então na declaração deste componente você especifica qual o nome do método que o evento deve chamar.

Vejamos abaixo uma sintaxe resumida deste evento.

```
...
<asp: Button id = "Identidade"
    Text = "TextoNoBotao"
    OnClick = "NomeDoMetodo"
    Runat="server"/>
....
```

Evento - **OnSelectedIndexChanged**

Depois dos atributos especificados de um componente como o DropDownList e o CheckBoxList, conseguimos captar o que foi clicado, usando a propriedade Selected.

Vamos acompanhar a sintaxe deste evento.

```
...
<asp: DropDownList id = "Identidade"
    AutoPostBack = "True"
    OnSelectedIndexChanged = "NomeDoMetodo"
    Runat="server"/>
....
```

O evento OnSelectedIndexChanged ocorre quando um atributo de um componente do tipo DropDownList é selecionado.

Veja um exemplo deste evento.

```
<html>
<form runat="server">
<asp:Label id="saida"
    Text = "Selecione itens abaixo"
    runat="server"/>
<p align="center">
<asp:DropDownList id="Dd1"
    AutoPostBack="true"
    OnSelectedIndexChanged="MetodoDd1"
    runat="server">
<asp:ListItem Value="VB"> Visual Basic.Net </asp:ListItem>
<asp:ListItem Value="CS"> C # </asp:ListItem>
<asp:ListItem Value="JS"> J# </asp:ListItem>
<asp:ListItem Value="delphi"> Delphi </asp:ListItem>
<asp:ListItem Value="Java"> Java </asp:ListItem>
</asp:DropDownList>
</p>
<p align="center">
<asp:TextBox id="saida_dd1"
    BackColor="yellow"
    runat="server"/>
</p>
</form>

<script language="C#" runat="server">

    public void MetodoDd1(Object sender, EventArgs ea)
    {
        saida_dd1.Text = Dd1.SelectedItem.Text;
    }
</script>
</html>
```

Vejamos o mesmo exemplo anterior na linguagem VB.Net.

```
<html>
<form runat="server">
<asp:Label id="saida"
    Text = "Selecione itens abaixo"
    runat="server"/>
<p align="center">
<asp:DropDownList id="Dd1"
    AutoPostBack="true"
    OnSelectedIndexChanged="MetodoDd1"
    runat="server">
<asp:ListItem Value="VB"> Visual Basic.Net </asp:ListItem>
<asp:ListItem Value="CS"> C # </asp:ListItem>
<asp:ListItem Value="JS"> J# </asp:ListItem>
<asp:ListItem Value="delphi"> Delphi </asp:ListItem>
<asp:ListItem Value="Java"> Java </asp:ListItem>
</asp:DropDownList>
</p>
<p align="center">
<asp:TextBox id="saida_dd1"
    BackColor="yellow"
    runat="server"/>
</p>
</form>

<script language="VB" runat="server">

    Sub MetodoDd1(Obj As Object, ea As EventArgs)
        saida_dd1.Text = Dd1.SelectedItem.Text
    End Sub
</script>
</html>
```

Como visto anteriormente, apresentamos apenas três eventos, dos quais julgamos os essenciais há qualquer aplicação na plataforma .Net FrameWork.

Esta tecnologia possui muitos outros eventos, dos quais você poderá acompanhá-los na documentação da plataforma .Net FrameWork.

No próximo tópico apresentaremos os principais componentes que compõem o asp.net.

6.2 Principais Componentes do Asp.Net

A tecnologia .Net sabendo da dificuldade em agregar componentes numa aplicação para Web, revolucionou com o asp.net a maneira de se trabalhar com a camada de interface.

A plataforma dispõe de componentes prontos com os quais o desenvolvedor consegue manipular suas propriedades como: cor, tamanho, posição no formulário, dentre outros, e atribuir eventos para exercer uma funcionalidade a aplicação.

Veja abaixo a sintaxe para inserir um componente na aplicação asp.net.

```
<asp:NomeComponente Id="IdentidadeComponente" runat="server"/>
```

```
<asp:Button Id="MeuComponente" runat="server"/>
```

Existe um conjunto de propriedades comuns para os componentes e que podem ser definidas quando está criando o projeto (tempo desenvolvimento) ou durante a execução do projeto (tempo execução).

Vejam na figura 6.2.1 as propriedades genéricas aos componentes.

Figura 6.2.1 - Propriedades Genéricas.

Propriedade	Descrição	Conteúdo
BackColor	Cor de fundo	Cor(Custom, Web ou System)
BorderColor	Cor da borda	Cor(Custom, Web ou System)
Columns	Largura do controle em caracteres	Número
CssClass	Classe do css.	Nome do css
Font	Fonte da letra	
Bold	Negrito	True / false
Italic	Itálico	True / false
Name	Nome de fonte da letra	Nome da fonte
Names	Seqüência dos nomes de fonte da letras	Nomes das fontes
Size	Tamanho	Smaller, Larger, Small, ou números.
UnderLine	Sublinhado	True / false
ForeColor	Cor de fundo	Cor(Custom, Web ou System)
Text	Texto a ser escrito	Texto

Propriedade	Descrição	Conteúdo
AccessKey	Define a tecla de atalho	Letra simples (A, G, C)
AutoPostBack	Envia uma solicitação após uma alteração.	True / False
Enabled	Torna o controle ativado ou não.	True / False
MantainState	Mantem o estado do controle	True / False
TabIndex	Índice ou a ordem de tabulação.	Número
TextMode	Modo de texto	NotSet, SingleLine, MultiLine, Password
ToolTip	Texto de dicas	Texto
Visible	Torna o componente visível ou não.	True / False
DataSource	Determina a origem de dados.	Array, expressão, dataset
DataTextField	Campo de Texto do DataSource definido.	Texto
DataValueField	Campo de valor do DataSource definido.	Texto
Items	Lista de itens do controle	Texto
Height	Altura do componente	Número
Width	Largura do componente	Número
Wrap	Define se poderá ocorrer quebra de linha no texto.	True / False
Modifiers	Define o nível de visibilidade do componente	Public, Protected, Private, Internal.

Agora que você já sabe como inserir um componente, vamos mostrar o processo de funcionamento destes componentes.

Quando uma página asp.net é carregada, todas as rotinas de suporte são invocadas. O prefixo **<asp:** indica o namespace do componente, e a propriedade runat = "server" informa que serão rodados no servidor.

Acompanhem os passos que acontecem quando uma página asp.net é carregada.

- Os componentes são criados no servidor;
- As propriedades dos componentes são ajustadas a partir do _VIEWSTATE;
- Caso estes componentes tenham algum POST, são invocados os eventos respectivos;
- O componente cria a saída em HTML;
- A página é enviada para o usuário.

A partir de agora vamos apresentar os principais componentes das aplicações asp.net.

TextBox

O TextBox é um controle de contribuição que deixa o usuário entrar com um texto. Através de falta, o TextMode da caixa de texto é SingleLine, mas pode ser modificado para ser MultiLine ou Password.

A largura de exibição da caixa de texto é determinada por sua propriedade de colunas. Se for uma caixa de texto de multiline, sua altura de exibição é determinada pela propriedade de linhas.

Veja abaixo a sintaxe simplificada do componente TextBox.

```
<asp: TextBox id="Identidade"
    AutoPostBack="true/false"
    Columes="n"
    MaxLength="n"
    Rows="n"
    Text="Texto"
    TextMode="Single | MultiLine | Password"
    Wrap="true/false"
    OnTextChanged="Nome-do-Método"
    Runat="server"/>
```

Acompanhe o exemplo abaixo do componente TextBox.

```
<html>
<form runat="server">
<br> <br>
<asp:TextBox id="entrada"
    AutoPostBack="true"
    BackColor="yellow"
    MaxLength="12"
    TextMode="Password"
    OnTextChanged="MetodoText"
    runat="server"/>
<p> Pressione ENTER apos digitar </p>
<p>
<asp:Label id="saida"
    Font-Name="verdana"
    runat="server"/> </p>
</form>
<script Language="C#" runat="server">
public void MetodoText(Object sender, EventArgs ea)
{
    String Temp = entrada.Text;
    saida.Text = Temp;
}
</script>
</html>
```

Label

Use o Label para exibir texto em um local fixo da página.

Este controle exibe um Texto estático, onde o desenvolvedor poderá personalizar o texto exibido pela propriedade Text.

Acompanhe a sintaxe deste componente.

```
<asp:Label id="Identidade"  
    Text="Texto"  
    runat="server"/>
```

Button

Use o controle Button para criar um botão de clique na sua página de Web. Você pode criar um botão Submit ou um botão de comando.

Um botão Submit não tem um nome de comando (especificado pela propriedade CommandName) associado com o botão, este controle simplesmente posta a página Web de volta para o servidor. O default deste controle é Submit. Você pode prover para este controle o evento de Clique programado para que um controle as ações possa ser executado quando o botão Submit for clicado.

Um botão de comando tem uma identificação associada com o botão, fixado pela propriedade CommandName. Isto lhe permite criar múltiplos botões de controle em uma página Web , porque é possível determinar qual botão o usuário clicou. Você também pode usar a propriedade CommandArgument com um botão de comando para prover uma informação adicional sobre o comando a ser executado.

```
<asp:Button id="Identidade"  
    Text="Título"  
    OnClick="Metodo-Controle"  
    Runat="server"/>
```

Na página seguinte podemos acompanhar um exemplo deste componente.

```
<html>
<form runat="server">
<asp:Label id="saida"
    runat="server"/>
<p align="center">
<asp:Button id="bot01"
    Text=" - Contador - "
    OnClick="bot01_Clique"
    runat="server"/>
</p>
</form>

<script language="VB" runat="server">
    shared Contador As Integer = 1
    Sub bot01_Clique(Object As Object, ea As EventArgs)
        saida.Text = "Você Atualizou a página "& Contador & " Vezes."
        Contador = Contador + 1
    End Sub
</script>
</html>
```

DropDownList

Use o componente DropDownList para controlar uma única seleção para um controle de lista. Você pode controlar o aparecimento do controle de DropDownList fixando o BorderColor, BorderStyle, e propriedades de BorderWidth.

Para especificar os itens que você deseja que apareça no controle de DropDownList, coloque um objeto de ListItem, para cada entrada, entre a tag de abertura e a tag final do controle de DropDownList.

O controle de DropDownList também suporta a ligação de dados. Então, use o método Control.DataBind para ligar a fonte de dados ao controle de DropDownList.

Use a propriedade SelectedIndex para determinar o índice do item selecionado pelo usuário do controle de DropDownList.

Acompanhe a sintaxe deste controle.

```
<asp:DropDownList id="Identidade"
    AutoPostBack="true/false"
    OnSelectedIndexChanged="Metodo-a-disparar"
    runat="server"/>
    <asp:ListItem Value="Valor" Selected="true/false">
        Texto
    </asp:ListItem>
</asp:DropDownList>
```

Veja o exemplo a seguir deste controle.

```
<html>
<form runat="server">
<p align="center">
<asp:DropDownList id="Dd1"
    BackColor="yellow"
    AutoPostBack="true"
    OnSelectedIndexChanged="MetodoEscreve"
    runat="server">
<asp:ListItem Value="CSharp"> Curso Microsoft C# </asp:ListItem>
<asp:ListItem Value="VB"> Curso Microsoft Visual Basic.Net
</asp:ListItem>
<asp:ListItem Value="JSharp"> Curso Microsoft J# </asp:ListItem>
<asp:ListItem Value="Asp"> Curso Microsoft Asp.Net </asp:ListItem>
<asp:ListItem Value="Plataforma"> Curso Microsoft Plataforma .Net
</asp:ListItem>
<asp:ListItem Value="Delphi"> Curso Borland Delphi </asp:ListItem>
</asp:DropDownList>
<br><br>
<asp:TextBox id="saida"
    Width="400"
    Rows="6"
    HorizontalAlign="center"
    TextMode="MultiLine"
    BackColor="#EEEEEE"
    runat="server"/>
<br><br>
<asp:Button id="bot01"
    Text="Limpar Campo"
    OnClick="MetodoLimpar"
    runat="server"/>
</p>
</form>

<script language="C#" runat="server">
    static string temp = null;
    public void MetodoEscreve(Object sender, EventArgs ea)
    {
        for( int i = 0; i < Dd1.Items.Count; i++) {
            if (Dd1.Items[i].Selected) {
                temp = temp + Dd1.Items[i].Text;
                temp = temp + "\n";
            }
        }
        saida.Text = temp;
    }

    public void MetodoLimpar(Object sender, EventArgs ea)
    {
        temp=null;
        saida.Text=null;
    }
</script>
</html>
```

ListBox

O componente ListBox permite que os usuários selecionem um ou mais itens de uma lista preferida. Difere de um controle DropDownList que pode exibir múltiplos itens, mas permite a escolha de apenas um único item de uma lista.

Você pode fixar o controle para exibir um número específico de itens.

Como qualquer outro controle para a Web, você pode usar objetos de estilo para especificar o aparecimento do controle.

Enfim, use o componente ListBox para criar um controle de lista que permite única ou múltipla seleção de itens. Use a propriedade ROWS para especificar a altura do controle. Para habilitar a seleção de múltiplos itens, fixe a propriedade SelectionMode para Multiple.

Veja a sintaxe do controle ListBox abaixo:

```
<asp:ListBox id="Identidade"
    Rows="3"
    SelectionMode="Multiple"
    runat="server">
<asp:ListItem Value="Valor" Selected="true/false"> Item 1 </asp:ListItem>
<asp:ListItem Value="Valor" Selected="true/false"> Item 2 </asp:ListItem>
<asp:ListItem Value="Valor" Selected="true/false"> Item 3 </asp:ListItem>
<asp:ListItem Value="Valor" Selected="true/false"> Item 4 </asp:ListItem>
<asp:ListItem Value="Valor" Selected="true/false"> Item 5 </asp:ListItem>
</asp:ListBox>
```

CheckBox

O controle CheckBox cria uma caixa de cheque no Web Forms que permite o usuário trocar entre verdadeiro ou falso o estado. Você pode especificar para exibir a legenda no controle, fixando a propriedade Text. A propriedade TextAlign serve para especificar o lado onde a legenda deve aparecer.

Para determinar se o controle CheckBox é true, teste a propriedade Checked. O evento CheckChanged recebe um estímulo quando o estado do controle CheckBox muda, para postar até ao servidor. Você pode prover um estímulo para o evento CheckChanged executar uma tarefa específica quando o estado do CheckBox sofrer alguma mudança.

Através de falta, o controle CheckBox não posta a forma automática até ao servidor quando o usuário clicar. Para habilitar a postagem automática, fixe a propriedade AutoPostBack para true.

Vamos acompanhar a sintaxe do controle CheckBox.

```
<asp:CheckBox id="Identidade"
  AutoPostBack="true / false"
  Text="Titulo"
  TextAlign="Right / Left"
  Checked="True / false"
  OnCheckedChanged="Metodo-disparar"
  runat="server"/>
```

CheckBoxList

O controle CheckBoxList cria múltiplas seleções em um grupo de caixas de cheque que pode ser gerado usando ligação de dados dinamicamente.

Para especificar itens que você quer que apareça no controle CheckBoxList, coloque um elemento ListItem.

Utilize a propriedade DataBind para ligar a fonte de dados ao controle CheckBoxList. Use as propriedades DataTextField e DataValueField para especificar qual campo na fonte de dados deve ligar com o Texto e com o Valor de cada item da lista no controle.

Você pode especificar o modo que a lista é exibida usando as propriedades RepeatLayout e RepeatDirection.

Vejamos a sintaxe para o uso deste componente.

```
<asp:CheckBoxList id="CheckBoxList1"
  AutoPostBack="True|False"
  CellPadding="Pixels"
  DataSource='<% databindingexpression %>'
  DataTextField="DataSourceField"
  DataValueField="DataSourceField"
  RepeatColumns="ColumnCount"
  RepeatDirection="Vertical|Horizontal"
  RepeatLayout="Flow|Table"
  TextAlign="Right|Left"
  OnSelectedIndexChanged="Metodo-disparar"
  runat="server">

  <asp:ListItem value="value"
    selected="True|False">
    Text
  </asp:ListItem>

</asp:CheckBoxList>
```

RadioButton

O controle RadioButton cria um botão de rádio na página Web.

Especifique a propriedade Text do controle para que o texto seja exibido.

O texto pode aparecer na esquerda ou na direita do botão de rádio.

A propriedade TextAlign é utilizada para controlar o lado onde o texto deve aparecer.

Você pode também agrupar vários botões de rádio num mesmo conjunto, para isso devemos especificar um único nome na propriedade GroupName de cada RadioButton. Se agruparmos os controles em um conjunto, será permitida apenas uma única seleção exclusiva do grupo.

Determinamos se um elemento foi selecionado testando a propriedade Checked.

Vejam os exemplos abaixo a sintaxe deste controle.

```
<asp:RadioButton id="RadioButton1"
  AutoPostBack="True|False"
  Checked="True|False"
  GroupName="GroupName"
  Text="label"
  TextAlign="Right|Left"
  OnCheckedChanged="Metodo-disparar"
  runat="server"/>
```

RadioButtonList

O controle RadioButtonList promove uma única seleção do grupo de botões radio, que pode ser gerado dinamicamente por uma ligação de dados. Contem uma coleção de itens que correspondem aos itens individuais na lista.

Para determinar se um item foi selecionado, teste a propriedade Checked.

Veja a sintaxe para este controle.

```
<asp:RadioButtonList id="RadioButtonList1"
  AutoPostBack="True|False"
  CellPadding="Pixels"
  DataSource="<% databindingexpression %>"
  DataTextField="DataSourceField"
  DataValueField="DataSourceField"
  RepeatColumns="ColumnCount"
  RepeatDirection="Vertical|Horizontal"
  RepeatLayout="Flow|Table"
  TextAlign="Right|Left"
  OnSelectedIndexChanged="Metodo-disparar"
  runat="server">

  <asp:ListItem Text="label"
    Value="value"
    Selected="True|False" />

</asp:RadioButtonList>
```

Hyperlink

O controle HyperLink cria ligações(links) em uma página de Web que permite ao usuário chamar em sua aplicação uma outra página. A vantagem primária de usar controle de HyperLink é que você pode fixar propriedades de ligação em código de servidor. Por exemplo, você pode mudar o texto de ligação ou página de destino dinamicamente baseado em condições em sua página.

Outra vantagem de usar o controle de HyperLink é que você pode usar dados que ligam para especificar a URL designada para a ligação. Um exemplo típico é criar controles de HyperLink baseado em uma lista de produtos; os pontos de URL designados para uma página onde o usuário pode ler para mais detalhe sobre o produto.

Ao contrário da maioria dos controles de um servidor de Web, o controle de HyperLink não gera quaisquer eventos em código de servidor quando os usuários clicarem. Ao invés disto, o controle simplesmente redireciona os usuários.

Propriedades do controle HyperLink.

Propriedades	Descrição
ImageUrl	O URL de uma imagem a exibir para o link.
NavigateUrl	O URL a que esse link direciona os usuários.
Target	A janela de alvo para exibir o conteúdo vinculado.
Text	O texto a exibir para o link.

Acompanhe a sintaxe abaixo:

```
<asp:HyperLink id="HyperLink1"
  NavigateUrl="url"
  Text="HyperLinkText"
  ImageUrl="url"
  Target="window"
  runat="server"/>
```

OU

```
<asp:HyperLink id="HyperLink1"
  NavigateUrl="url"
  ImageUrl="url"
  Target="window"
  runat="server">
  Text
</asp:HyperLink>
```

Observe as configurações para a propriedade Target:

Target	Descrição
_blank	Carrega numa outra página.
_self	Carrega sobre si mesma.
_parent	Carrega página referenciada na página "pai" desta.
_top	Carrega uma página numa nova janela sem frames.

Panel

O controle Panel provê um controle de recipiente dentro de um Web Forms que você pode usar como um pai para texto estático e para outros controles. O controle de Panel é satisfatório para:

Agrupar comportamento: Você pode administrar um grupo de controles como – TextBox, CheckBox, DataGrid, em uma única unidade, adicionando os controles em um painel, e manipulando a partir daí o componente Panel.

Geração de Controle dinâmico: O controle de Panel provê um recipiente conveniente para controles que você cria em tempo de corrida.

Aparecimento: o controle Panel apóia o aparecimento em propriedades como BackColor e BorderWidth.

NOTA

✎ O controle de Painel não é exigido para agrupar controles como RadioButton e CheckBox.

```
<asp:Panel id="Panel1"  
  BackImageUrl="url"  
  HorizontalAlign="Center|Justify|Left|NotSet|Right"  
  Wrap="True|False"  
  runat="server">  
  
  (Other controls declared here)  
  
</asp:Panel>
```

Exemplos e Exercícios.

Exemplo 01 – uso do controle ListBox para ordenar uma lista.

```
<html>
<form id="form1" method="post" runat="server" >
<p> <font face="arial" size="4"> Digite seu Nome: </font>
<asp:TextBox id="nome" runat="server" >
</asp:TextBox> </p> <p>
<asp:Button id="botao01" runat="server" Text="Incluir nome na lista" width="175"
Height="25" OnClick="Incluir"> </asp:Button> </p> <p>
<asp:Button id="botao02" runat="server" Text="Excluir nome da lista" width="175"
Height="25" OnClick="Excluir"> </asp:Button> </p> <p>
<asp:Button id="botao03" runat="server" Text="Limpar nome(s) da Lista"
width="175" Height="25" OnClick="Limpar"> </asp:Button></p> <p>
<asp:Button id="botao04" runat="server" Text="Ordenar a Lista" width="175"
Height="25" OnClick="Ordenar"> </asp:Button> </p> <p>
<asp:ListBox id="RecebeNome" runat="server" width="175" Height="150">
</asp:ListBox>
<asp:Label id="saida" runat="server">Veja ao lado a lista ordenada: </asp:Label>
<asp:ListBox id="saiObj" runat="server" width="175" Height="150">
</asp:ListBox> </p>
</form>
<script language="C#" runat="server">
    public void Page_Load(object sender, EventArgs ea)
    {
        if (!IsPostBack)
        {
            nome.Text="foi atualizada";
        }
    }

    public void Incluir(object sender, EventArgs ea)
    {
        if (nome != null)
        {
            RecebeNome.Items.Add(nome.Text); //Adiciona um item na lista
            nome.Text=null; //limpa o campo nome do TextBox
        }
    }

    public void Excluir(object sender, EventArgs ea)
    {
        if (RecebeNome.SelectedIndex >= 0) {
            RecebeNome.Items.RemoveAt(RecebeNome.SelectedIndex); //Exclui o item
            //selecionado no ListBox
        }
    }

    public void Limpar(object sender, EventArgs ea)
    {
        RecebeNome.Items.Clear( ); //limpa o ListBox
    }
}
```

```
public void Ordenar(object sender, EventArgs ea)
{
    ArrayList objVetor = new ArrayList( );
    int contador;

    contador=RecebeNome.Items.Count;

    if (contador > 0)
    {
        for(int i=0; i<contador; i++)
        {
            objVetor.Add(RecebeNome.Items[i].Value);
        }

        int a=0;
        bool condicao = true;
        while ( contador > a )
        {
            for(int j=a, i=a+1; i < contador; i++)
            {
                if ( objVetor[j].ToString().CompareTo(objVetor[i].ToString()) > 0 )
                {
                    string temp;
                    temp=objVetor[i].ToString();
                    objVetor[i] = objVetor[j];
                    objVetor[j] = temp;
                    condicao = false;
                }
            }
            if (condicao != false)
            {
                a++;
            }
            condicao=true;
        }

        if (saiObj != null)
        {
            saiObj.Items.Clear();
        }

        for (int i=0; i < contador; i++)
        {
            saiObj.Items.Add(objVetor[i].ToString());
        }
    }
}
```

</script>

</html>

Exemplo 02 – Propriedade Visible dos controles.

```

<%@ Page Language="VB" %>
<html>
<head>
  <script runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
      If Check1.Checked Then
        Panel1.Visible = False
      Else
        Panel1.Visible = True
      End If
      Dim numlabels As Integer = Int32.Parse(DropDown1.SelectedItem.Value)
      Dim i As Integer
      For i = 1 To numlabels
        Dim l As New Label()
        l.Text = "Label" + i.ToString()
        l.ID = "Label" + i.ToString()
        Panel1.Controls.Add(l)
        Panel1.Controls.Add(New LiteralControl("<br>"))
      Next i
      Dim numtexts As Integer = Int32.Parse(DropDown2.SelectedItem.Value)
      For i = 1 To numtexts
        Dim t As New TextBox()
        t.Text = "TextBox" & i.ToString()
        t.ID = "TextBox" & i.ToString()
        Panel1.Controls.Add(t)
        Panel1.Controls.Add(New LiteralControl("<br>"))
      Next i
    End Sub
  </script>
</head>
<body>
  <h3>Panel Example</h3>
  <form runat="server">
    <asp:Panel id="Panel1" runat="server"
      BackColor="gainsboro"
      Height="200px"
      Width="300px"> Panel1: Controles Viviseis... <p> </asp:Panel>
    <p>
    Numero de Labels Visíveis:
    <asp:DropDownList id=DropDown1 runat="server">
      <asp:ListItem Value="0">0</asp:ListItem>
      <asp:ListItem Value="1">1</asp:ListItem>
      <asp:ListItem Value="2">2</asp:ListItem>
      <asp:ListItem Value="3">3</asp:ListItem>
      <asp:ListItem Value="4">4</asp:ListItem> </asp:DropDownList> <br>
    Numero de TextBoxes Visíveis:
    <asp:DropDownList id=DropDown2 runat="server">
      <asp:ListItem Value="0">0</asp:ListItem>
      <asp:ListItem Value="1">1</asp:ListItem>
      <asp:ListItem Value="2">2</asp:ListItem>
      <asp:ListItem Value="3">3</asp:ListItem>
      <asp:ListItem Value="4">4</asp:ListItem> </asp:DropDownList> <p>
    <asp:CheckBox id="Check1" Text="Propriedade VISIBLE do Panel"
  runat="server"/>
    <p>
    <asp:Button Text="Refresh Panel" runat="server"/>
  </form>
</body>
</html>

```

Exercício 01 - Implemente o exemplo 01 na linguagem VB.Net.

Exercício 02 – Implemente o exemplo 02 na linguagem C#.

Capítulo 7 – Controle de Validação

É essencialmente importante para a aplicação que o desenvolvedor tenha uma política de validação de dados eficiente. O asp.net traz alguns controles para facilitar a vida do programador. Você pode acrescentar validação de entrada em suas páginas Web Forms usando controles de validação prontos.

Controles de validação provêm um mecanismo fácil de usar para todos os tipos comuns de critérios de validação - por exemplo, testando datas válidas ou valores dentro de uma gama – entre outros modos para prover validação de escrita. Além disso, os controles de validação lhe permitem personalizar completamente como será exibida a informação de erro ao usuário.

O controle de validação pode ser usado com qualquer controle que é processado no arquivo da classe de uma página Web Form, incluindo HTML e controles Web Server.

O controle de validação é semelhante aos outros controles estudados anteriormente. Portanto como todo controle, também tem suas propriedades comuns, das quais veremos mais em diante.

A diferença entre estes controles é que os controles de validação não exibem nada, a não ser a mensagem de erro causa ocorra. Enfim, o usuário não pode interagir com eles. Portanto, a função do controle de validação é observar um controle de servidor e validar seu conteúdo.

Para especificar qual controle vai ser validado, usaremos a propriedade `ControlToValidate` comum a todos controles de validação.

É válido observar que as consistências são digitadas pelo usuário, e não em conteúdos válidos. Caso o browser não suporte tal consistência, ela é realizada no servidor, pois o próprio .Net FrameWork identifica o browser que fez a solicitação e designa se tem ou não o suporte para realizar tal tarefa.

Para validação existem tais componentes que a plataforma disponibiliza:

Componente	Descrição
RequiredFieldValidator	Verifica se um campo requerido está em branco.
CompareValidator	Compara o valor de um ou mais controles.
RangeValidator	Compara se a informação digitada esta dentro de uma faixa de valor determinada para a validação. Um valor entre 5 e 10 por exemplo.
CustomValidator	Verifica os valores digitados em relação a uma validação que você mesmo codifica.
RegularExpressionValidator	Verifica se o valor de um campo satisfaz uma expressão constante determinada.
ValidationSummary	Serve para agrupar em uma única lista na página todos os erros gerados.

Os controles de validação apenas validarão a entrada com um subconjunto de controles de servidor do asp.net. Na maioria das vezes estes controles serão mais que suficiente para validar toda a entrada do usuário.

Veja abaixo a relação dos controles que podem ter sua entrada validada pelo controles de validação.

Podem ser validados os seguintes controles:

- ↳ HtmlInputText
- ↳ HtmlTextArea
- ↳ HtmlSelect
- ↳ HtmlInputFile
- ↳ TextBox
- ↳ ListBox
- ↳ DropDownList
- ↳ RadioButtonList

Para que um controle de validação passe a funcionar em sua aplicação basta o desenvolvedor adicioná-lo como qualquer outro controle.

Acompanhe a sintaxe do controle de validação RequiredFieldValidator e um exemplo abaixo:

```
<asp:RequiredFieldValidator
  id="ProgrammaticID"
  ControlToValidate="ProgrammaticID of control to validate"
  InitialValue="value"
  ErrorMessage="Message to display in ValidationSummary control"
  Text="Message to display in control"
  ForeColor="value"
  BackColor="value" ...
  runat="server" >

</asp:RequiredFieldValidator>
```

```
<html>
  <form runat="server">
    Name:
    <asp:TextBox id="Text1"
      Text="Enter a value"
      BackColor="yellow"
      runat="server"/>

    <asp:RequiredFieldValidator id="RequiredFieldValidator1"
      ControlToValidate="Text1"
      Text="Campo Requerido"
      runat="server"/> <p>

    <asp:Button id="Button1"
      runat="server"
      Text="Validate"/>
  </form>
</html>
```

Acompanhe a sintaxe do controle de validação CompareValidator e um exemplo abaixo:

```
<asp:CompareValidator
  id="Identificação"
  ControlToValidate="controle determinado para a verificação"
  ValueToCompare="value"
  ControlToCompare="value"
  Type="DataType"
  Operator="Operator Value"
  ErrorMessage="mensagem de erro"
  Text="mensagem de erro para o controle de verificação"
  ForeColor="value"
  BackColor="value" ...
  runat="server" >
</asp:CompareValidator>
```

```
<html>
<form runat="server">
<p align="center">
  Digite um valor maior ou igual a 10:
  <asp:TextBox id="entrada"
    BackColor="yellow"
    runat="server"/> <br/> <br/>
  <asp:Button id="Bot01"
    BackColor="green"
    Text="- Validar -"
    OnClick="MetodoValidar"
    CausesValidation="False"
    runat="server"/> <br/> <br/>

  <asp:Label id="saida"
    runat="server"/>

  <asp:CompareValidator id="Comparar"
    ControlToValidate="entrada"
    ValueToCompare="10"
    Type="Integer"
    Operator="GreaterThanEqual"
    runat="server"/> <br/> </p>
</form>

<script language="C#" runat="server">
  public void MetodoValidar(Object sender, EventArgs ea) {
    Comparar.Validate( );
    if (Comparar.IsValid == true) {
      saida.Text = "Numero CORRETO ! ";
    }
    else {
      saida.Text="Numero INCORRETO ! ";
    }
  }
</script>
</html>
```


Vamos acompanhar agora, uma tabela mostrando os operadores que o controle CompareValidator utiliza para a propriedade Operator.

Tabela 7.1 – Operadores para a propriedade Operator.

Operador	Descrição
DataTypeCheck	Verifica se o tipo de informação é compatível com um certo tipo de dados(string, Integer e assim por diante)
Equal	Verifica se é igual
GreaterThan	Verifica se é maior que
GreaterThanEqual	Verifica se é maior ou igual
LessThan	Verifica se é menor que
LessThanEqual	Verifica se é menor ou igual
NotEqual	Verifica se é diferente

Vamos acompanhar agora, uma tabela mostrando os tipos de informação que o controle CompareValidator utiliza para a propriedade Type.

Tabela 7.2 – Tipos de informações para a propriedade Type.

Tipo	Descrição
Currency	Valores monetários.
Date	Valores de data.
Double	Valores de números fracionários.
Integer	Valores de números inteiros.
String	Valores de string

RangeValidator

O controle RangeValidator testa se um valor de um controle de entrada está dentro de uma faixa de valores especificada no componente.

O controle RangeValidator utiliza quatro propriedades chave para executar sua validação. A propriedade ControlToValidate contém o nome do controle de entrada para validar. As propriedades MinimumValue e MaximumValue especificam respectivamente o valor mínimo e o valor máximo para validar a entrada de informações em uma faixa de valores válidos.

A propriedade Type é usada para especificar os tipos de dados usados para realizar a comparação de valores. Estes valores são convertidos para o tipo determinado para que a validação possa ser realizada.

A tabela seguinte lista os diferentes tipos de dados que podem ser comparados.

Tabela 7.3 – Tipos de dados da propriedade Type do controle RangeValidator.

Tipo	Descrição
Currency	Valores monetários.
Date	Valores de data.
Double	Valores de números fracionários.
Integer	Valores de números inteiros.
String	Valores de string

Veja a sintaxe deste controle abaixo:

```
<asp:RangeValidator
    id="ControleValidacao"
    ControlToValidate="NomeControleEntradaDados"
    MinimumValue="valor"
    MaximumValue="valor"
    Type="DataType"
    ErrorMessage="Mensagem a ser exibida no caso de erro"
    Text="Mensagem de visualização do controle"
    ForeColor="valor"
    BackColor="valor" ...
    runat="server" >
</asp:RangeValidator>
```

```
<%@ Page Language="VB" %>
<html>
<head>
    <script runat="server">
        Sub ButtonClick(sender As Object, e As EventArgs)
            If Page.IsValid Then
                Label1.Text="Pagina é valida !!"
            Else
                Label1.Text="Pagina não é valida !!"
            End If
        End Sub
    </script>
</head>
<body>
    <form runat="server">
        <h3>RangeValidator Exemplo</h3> Entre com valores entre 1 e 10:    <br>
        <asp:TextBox id="TextBox1"
            runat="server"/> <br>
        <asp:RangeValidator id="Range1"
            ControlToValidate="TextBox1"
            MinimumValue="1"
            MaximumValue="10"
            Type="Integer"
            EnableClientScript="false"
            Text="The value must be from 1 to 10!"
            runat="server"/> <br><br>
        <asp:Label id="Label1"
            runat="server"/> <br><br>
        <asp:Button id="Button1"
            Text="Submit"
            OnClick="ButtonClick"
            runat="server"/>
    </form>
</body>
</html>
```

CustomValidator

Use o controle CustomValidator para prover uma função de validação definida pelo desenvolvedor para um controle de entrada. O controle CustomValidator é um controle separado dos demais controles de entrada válidos, e que permite controlar onde a mensagem de validação é exibida.

Em outras palavras, quando o usuário insere dados, o controle CustomValidator executa um método de validação personalizado, implementado pelo próprio desenvolvedor.

Os controles de validação sempre executam validação no servidor. Eles também têm implementação no lado cliente desde que tenha suporte a DHTML que é o caso dos browsers mais recentes (como Internet Explorer 4.0 e depois).

A validação no lado cliente aumenta o processo de validação conferindo a entrada de dados do usuário antes de fosse enviado ao servidor. Isto permite descobrir erros no cliente, antes de a aplicação ser submetida, evitando com isso a viagem de ida-e-volta de informação.

Acompanhe a sintaxe deste controle.

```
<asp:CustomValidator
  id="IdentidadeControle"
  ControlToValidate="Controle-a-Validar"
  ClientValidationFunction="ClientValidateID"
  OnServerValidate="ServerValidateID"
  ErrorMessage="Mensagem de Erro"
  Text="Mensagem de visualização do controle"
  ForeColor="valor"
  BackColor="valor" ...
  runat="server" >

</asp:CustomValidator>
```

RegularExpressionValidator

O controle RegularExpressionValidator confere se o valor do controle de entrada corresponde ao definido pela expressão regular. Este tipo de validação lhe permite conferir previsíveis sucessões de caráter, como esses números de previdência social, endereços de e-mail, números de telefone, e códigos postais.

A validação tem sucesso se o controle de contribuição estiver vazio. Se um valor é requerido para o controle de contribuição associado, use um controle de RequiredFieldValidator para requerer um campo.

As validações são executadas no lado servidor e no lado cliente a menos que o browser não apóie a validação no lado cliente ou a validação no lado cliente é explicitamente inválida (fixando a propriedade de EnableClientScript para falso).

Acompanhe a sintaxe e um exemplo deste controle abaixo:

```
<asp:RegularExpressionValidator
    id="IdentidadeControle"
    ControlToValidate="IdentidadeControle-a-Validar"
    ValidationExpression="Expressao-de-Validacao"
    ErrorMessage="Mensagem de erro da validação"
    Text="Mensagem de visualização no controle"
    ForeColor="valor"
    BackColor="valor" ...
    runat="server" >

</asp: RegularExpressionValidator>
```

```
<%@ Page Language="C#" %>
<html>
<script runat="server">
    void ValidateBtn_Click(Object sender, EventArgs e) {
        if (Page.IsValid) {
            lblOutput.Text = "Validação CORRETA !";
        }
        else {
            lblOutput.Text = "Validação INCORRETA !";
        }
    }
</script>
<body>
<h3>RegularExpressionValidator - EXEMPLO</h3>
<p>
<form runat="server">
    <asp:Label id="lblOutput"
        Text="Entre com 5 dígitos"
        Font-Name="Verdana"
        Font-Size="10pt"
        runat="server"/>
<br/> <br/>
<b>INFORMAÇÃO PESSOAL</b>
```

Código:

```
<asp:TextBox id="TextBox1"
    runat="server"/>

    <asp:RegularExpressionValidator id="RegularExpressionValidator1"
        ControlToValidate="TextBox1"
        ValidationExpression="\d{5}"
        Display="Static"
        EnableClientScript="false"
        ErrorMessage="O Código deve conter 5 Dígitos"
        runat="server"/>

    <p>
    <asp:Button text="- Validar Código -"
        OnClick="ValidateBtn_Click"
        runat="server"/>
    </p>
</form>
</body>
</html>
```

A tabela 7.4 mostra os elementos da linguagem das expressões regulares.

Tabela 7.4 – Elementos para a expressão regular.

Caractere(s)	Significado
Caracteres Regulares	Todos os caracteres, exceto ., \$, ^, {, [, (, ,), *, +, ? e \, são identificados por eles próprios.
.	Especifica qualquer caractere.
\$	Especifica padrões no final de uma string.
^	Especifica padrões no início de uma string.
{ }	Especifica uma certa quantidade de caracteres.
[]	Especifica um grupo caracteres.
()	Utilizado para agrupar strings.
	Significa or lógico.
*	Especifica zeros ou mais correspondências.
+	Especifica uma ou mais ocorrências.
?	Especifica zero ou mais ocorrências.
\	Um caractere de escape.

ValidationSummary

O controle ValidationSummary é usado para resumir em um único local as mensagens de erro de todos os controles de validação contidos em uma página de Web.

O resumo pode ser exibido como uma lista com marcadores, ou como um parágrafo único. A propriedade DisplayMode define o tipo de exibição para a mensagem de erro.

A mensagem de erro pode ser exibida na própria página de Web ou em uma caixa de mensagem, fixando as propriedades ShowSummary e ShowMessageBox respectivamente.

Acompanhe a sintaxe do controle.

```
<asp:ValidationSummary
  id="IdentidadeControle"
  DisplayMode="BulletList | List | SingleParagraph"
  EnableClientScript="true | false"
  ShowSummary="true | false"
  ShowMessageBox="true | false"
  HeaderText="Titulo da Mensagem"
  runat="server"/>
```

Exemplos e Exercícios

Exemplo 01 – Controles de validação em um cadastro.

```
<html>
<head>
<title>Projeto .Net FrameWork SDK 1.0</title>
</head>

<body bgcolor="#6699CC" text="#FFFFFF">
<br><br>
<font face="arial" size="3pt">
<h3 align="center">Projeto .Net FrameWork SDK 1.0a</h3>
</font>
<br>

<form runat="server">

<table width="75%" border="0" align="center">
  <tr>
    <td width="15%" align="right">
      <asp:RequiredFieldValidator id="rfnome"
        ControlToValidate = "txtNome"
        ForeColor="yellow"
        ErrorMessage = "Entre com dados no campo NOME."
        runat="server">
      *
    </asp:RequiredFieldValidator>

    Nome:</td>
    <td width="4%">&nbsp;</td>
    <td width="80%">
      <asp:TextBox id="txtNome"
        BackColor="yellow"
        Width="250"
        runat="server"/>
    </td>
  </tr>

  <tr>
    <td width="15%" align="right">
      <asp:RegularExpressionValidator id="rev1"
        ControlToValidate="txtEmail"
        ForeColor="yellow"
        ErrorMessage="Entre com um e-mail válido"
        ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
        runat="server">
      *
    </asp:RegularExpressionValidator>

    <asp:RequiredFieldValidator id="rfemail"
      ControlToValidate = "txtEmail"
      ForeColor="yellow"
      ErrorMessage = "Entre com dados no campo E-MAIL."
      runat="server">
      *
    </asp:RequiredFieldValidator>
```

```
e-mail:</td>
<td width="4%">&nbsp;</td>
<td width="80%">
  <asp:TextBox id="txtEmail"
    BackColor="yellow"
    Width="200"
    runat="server"/>
</td>
</tr>

<tr>
<td width="11%" align="right">curso:</td>
<td width="4%">&nbsp;</td>
<td width="85%">
  <asp:DropDownList id="txtCurso"
    BackColor="yellow"
    Width="200"
    AutoPostBack="true"
    runat="server">
    <asp:ListItem Value="101"> 101 - Ciência da Computação </asp:ListItem>
    <asp:ListItem Value="102"> 102 - Matemática Computacional </asp:ListItem>
    <asp:ListItem Value="201"> 201 - Fisioterapia </asp:ListItem>
    <asp:ListItem Value="202"> 202 - Odontologia </asp:ListItem>
    <asp:ListItem Value="301"> 301 - Administração </asp:ListItem>
  </asp:DropDownList>
</td>
</tr>

<tr>
<td width="15%" align="right">
  <asp:RequiredFieldValidator id="rfsenha"
    ControlToValidate = "txtSenha"
    ForeColor="yellow"
    ErrorMessage = "Entre com dados no campo SENHA."
    runat="server">
    *
  </asp:RequiredFieldValidator>
  senha:</td>
<td width="4%">&nbsp;</td>
<td width="80%">
  <asp:TextBox id="txtSenha"
    BackColor="yellow"
    Width="200"
    TextMode="password"
    runat="server"/>
</td>
</tr>
</table>
```

```
<br><br>
<table width="75%" border="0" align="center">
  <tr>
    <td width="33%">
      <div align="center">
        <asp:Button id="bot1"
          Text="Cadastrar"
          width="120"
          runat="server"/>
      </div>
    </td>

    <td width="33%">
      <div align="center">
        <asp:Button id="bot2"
          Text="Limpar Formulário"
          CausesValidation="False"
          runat="server"/>
      </div>
    </td>

    <td width="34%">
      <div align="center">
        <asp:Button id="bot3"
          Text="Voltar"
          width="120"
          CausesValidation="False"
          runat="server"/>
      </div>
    </td>
  </tr>
</table>
<br><br>
<p>

<asp:ValidationSummary id="vs1"
  HeaderText = "Algo errado! Veja abaixo:"
  DisplayMode="BulletList"
  ForeColor="white"
  runat="server"/>
</p>
</form>
</body>
</html>
```


Capítulo 8 - ADO.NET

Neste capítulo vamos apresentar o acesso ao banco de dados, um recurso indispensável em qualquer aplicação.

Com a necessidade de grandes empresas precisarem integrar todo o banco de dados corporativo com a Internet, tornou-se indispensável facilitar o acesso à base de dados com uma boa performance, para que os aplicativos consigam uma importância significativa.

A Plataforma .Net inova também nesta parte. O velho ADO necessitava de mudanças, ficou obsoleto principalmente com relação às aplicações baseadas na Web, então a Microsoft criou o ADO.NET e toda uma arquitetura baseada em XML. Com isso podemos afirmar que esta tecnologia trabalha com dados desconectados.

Vantagens do ADO.NET

Como dizemos anteriormente, o antigo ADO necessitava de mudanças. A Microsoft então criou o ADO.NET e com ele surgiram algumas vantagens, facilitando a programação e a comunicação com outros ambientes.

Vamos acompanhar abaixo algumas destas principais mudanças:

- ↗ **Escalabilidade** – pelo fato de o DataSet ser baseado em acesso a dados desconectados, por ter uma arquitetura baseada no XML, o tempo de manipulação dos dados no banco de dados torna-se mínimo. Portanto mesmo com um número simultâneo de acesso maior, a aplicação consegue garantir uma boa escalabilidade;
- ↗ **Performance** – no ADO.NET a transmissão de dados é feita em XML, com isso pode se comunicar com diversas plataformas e aplicativos;
- ↗ **Segurança** – um firewall não consegue bloquear um arquivo texto. Portanto como o ADO.NET é baseado em XML, as portas dos firewalls não são mais problemas.

O ADO.NET disponibiliza classes para a manipulação dos dados. Portanto depende da classe System.Data que contem os seguintes namespaces:

- ↗ System.Data
- ↗ System.Data.OleDb
- ↗ System.Data.SqlTypes
- ↗ System.Data.SqlClient

Para criar a conexão com o banco de dados o ADO.NET criou provedores de dados, dos quais se dividem em duas classes:

OleDb

A Plataforma.Net criou esta classe com vários provedores para o acesso de qualquer plataforma, como: SQL Server, Oracle e Access.

Este provedor pertence à classe System.Data.OleDb e seus principais objetos estão representados na tabela 7.1:

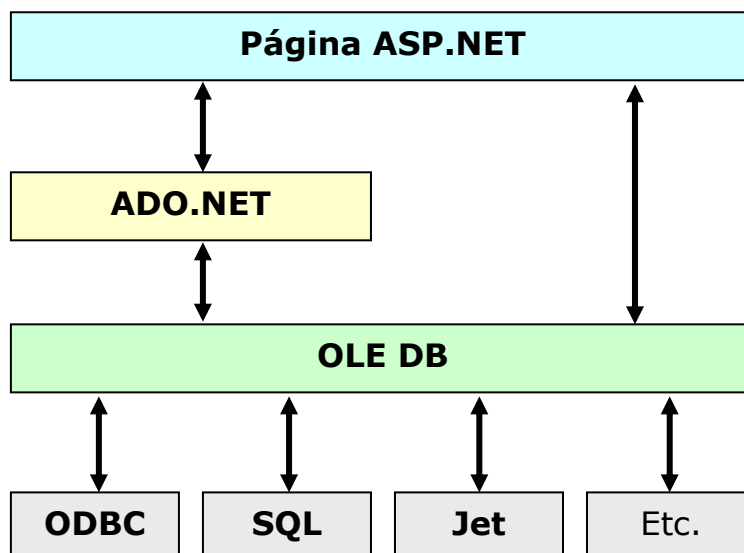
Tabela 7.1 - Objetos da classe System.Data.OleDb.

Objeto	Descrição
OleDbConnection	Define a abertura da conexão.
OleDbCommand	Define a instrução SQL a ser executada.
OleDbDataReader	Define somente para leitura um conjunto de dados.
OleDbDataAdapter	Define a conexão a ser usada para preencher um DataSet, e representa um conjunto de comandos de dados.

Esta classe permite os seguintes provedores:

Provedor	Plataforma
SQLOLEDB	Microsoft OLE DB Provider for SQL Server
MSDAORA	Microsoft OLE DB Provider for Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB Provider for Microsoft Jet

Abaixo temos um modelo de acesso a dados com ADO.NET e ASP.NET.



SQL

Este provedor é para o acesso exclusiva do SQL Server. Pertence a classe `System.Data.SqlClient` e seus principais objetos estão representados na tabela 7.2.

Tabela 7.2 - Objetos da classe `SqlClient`

Objeto	Descrição
<code>SqlConnection</code>	Define a abertura da conexão.
<code>SqlCommand</code>	Define a instrução SQL a ser executada.
<code>SqlDataReader</code>	Define somente para leitura um conjunto de dados.
<code>SqlDataAdapter</code>	Define a conexão a ser usada para preencher um <code>DataSet</code> , e representa um conjunto de comandos de dados.

Se a aplicação desenvolvida utiliza o banco de dados `SQLServer 7.0` ou superior, é altamente recomendável o uso do provedor SQL.

DataSet

O ADO.NET baseia-se no `DataSet`. Esse objeto é um conceito completamente novo para substituir o tradicional `Recordset` do ADO.

O `DataSet` é um armazenamento de dados simples residente na memória, que fornece um modelo de programação consistente de acesso a dados, independentemente do tipo de dados.

Diferentemente do `RecordSet`, o `DataSet` mantém conjuntos completos de dados, incluindo restrições, relacionamentos e até múltiplas tabelas de uma vez, todos estes armazenados na memória do computador.

Podemos afirmar também que, os componentes do `DataSet` foram desenvolvidos para manipular dados com mais rapidez, sendo possível executar comandos de leitura, alteração de dados, stored procedures e enviar ou receber informações parametrizadas.

Já que todas as linhas da seleção são transmitidas de uma vez para a memória, é preciso ter cuidado neste aspecto importante para não comprometer a performance, entupindo a memória e a rede.

Segundo o autor Américo Damasceno Junior o `SELECT` que você faz tem então que ser muito bem estudado para termos um objeto tipo `DataSet` de, no máximo, uns 100 Kbytes (isso daria umas 1000 linhas de 100 dígitos mais ou menos). Mesmo assim, a uma velocidade de 3Kbytes/segundo (que é o máximo que geralmente se consegue numa hora boa de Internet com um modem de 56 Kbits/segundo), seriam 30 segundos para o `DataSet` ir de uma maquina para outra. O ideal é um objeto tipo `DataSet` de uns 5Kbytes (50 linhas de 100 dígitos mais ou menos) que para uma transmissão a 1Kbyte/segundo levaria aceitáveis 5 segundos.

Conectando com um Banco de Dados

Para toda manipulação de dados realizada via ADO.NET, é preciso estabelecer uma conexão para montar um DataSet.

Usando OleDb

Conectando utilizando a linguagem C#.

```
Using System.Data.OleDb;
OleDbConnection conn;
conn = new OleDbConnection ("Provider = SQLOLEDB; server=NomeDoServidor;
                             Database=NomeDoDatabase; user id=NomeUsuario; Pwd=senha" );

conn.Open( );
```

Conectando utilizando a linguagem Visual Basic.Net.

```
Imports System.Data.OleDb
Dim conn As OleDbConnection
conn = New OleDbConnection ("Provider = SQLOLEDB; server=NomeDoServidor;
                             Database=NomeDoDatabase; user id=NomeUsuario; Pwd=senha" )

conn.Open( )
```

Usando SqlClient

Conectando utilizando a linguagem C#.

```
Using System.Data.SqlClient;
SqlConnection conn;
conn = new SqlConnection ("Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind");

SqlCommand cmd = new SqlCommand( INSTRUÇÃO SQL );
cmd.Connection = conn;

conn.Open( );
```

Conectando utilizando a linguagem Visual Basic.Net.

```
Imports System.Data.SqlClient
Dim conn As SqlConnection
conn = New SqlConnection ("Data Source=localhost;Integrated Security=SSPI;Initial Catalog=CBD")

Dim cmd As SqlCommand
cmd = New SqlCommand( INSTRUÇÃO SQL )
cmd.Connection = conn

conn.Open( )
```

Para fechar a conexão, utilize a sintaxe:

Para a linguagem C#.

```
conn.Close( );
```

Para a linguagem Visual Basic.Net.

```
conn.Close( )
```

Lendo um Banco de Dados

Uma vez determinada a conexão como banco de dados, o desenvolvedor já pode ler as informações contidas nas tabelas e a partir daí, gerar o que quiser.

DataReader

Este é o objeto que armazena o resultado de uma consulta ou stored procedure executada. Dependendo da forma de acesso (SQL ou OleDb), o desenvolvedor precisa respeitar os métodos existentes em cada objeto.

Acompanhe a sintaxe para a leitura em Banco de Dados do Access utilizando a linguagem Visual Basic.Net.

```
Dim InstrucaoSql As String = "SELECT * FROM Products"
Dim conn As OleDbConnection
Dim cmd As OleDbCommand
Dim Dr As OleDbDataReader

conn = new OleDbConnection("Provider=SQLOLEDB;server=localhost;" & _
    "database=northwind; user id=sa")

conn.Open( )

cmd = New OleDbCommand(InstrucaoSql, conn)

Dr = cmd.ExecuteReader( )

Do While Dr.Reader( )
    TextBox1.Text += Dr("ProductName") & " - " & Dr("UnitPrice") & vbCrLf
Loop

Dr.Close( )

conn.Close( )
```

OBS: não se esqueça de importar a classe: System.Data.OleDb

Acompanhe a sintaxe para a leitura em Banco de Dados do SQLServer utilizando a linguagem C#.

```
String InstrucaoSql = "SELECT * FROM Products";
SqlConnection conn;
SqlCommand cmd;
SqlDataReader Dr;

conn = new SqlConnection ("server=localhost;database=NomeBanco;" +
                          "uid=SqlService;pwd=senha123");

cmd = New SqlCommand(InstrucaoSql, conn);
cmd.Connection.Open( );

DR = cmd.ExecuteReader(CommandBehavior.CloseConnection);

While (Dr.Reader() )
{
    TextBox1.Text += Dr["Matricula"] + " - " + Dr["Nome"] + "\n";
}

Dr.Close( );

conn.Close( );

OBS: não se esqueça de importar a classe: System.Data.SqlClient
```

Podemos realizar uma consulta com a utilização do objeto SqlDataAdapter. No exemplo abaixo, temos um método chamado Consultar() que retorna um DataSet.

```
public void DataSet consultar(string InstrucaoSql)
{
    SqlConnection objconn = new SqlConnection("server=localhost; "+
                                              "uid=SqlService; pwd=esparta; database=Sql_net01");

    SqlDataAdapter objda = new SqlDataAdapter(InstrucaoSql, objconn);

    DataSet objds = new DataSet();

    objda.Fill(objds, "Listar");
    return objds;
}
```

OBS: não se esqueça de importar a classe: System.Data.SqlClient

Incluir Dados

A inserção de dados é indispensável numa aplicação Web.

Normalmente, aplicativos em que você se cadastra para uma determinada finalidade, têm seus dados inseridos em um banco de dados. Tudo isso é feito por meio de instruções SQL do tipo INSERT.

A sintaxe desta instrução é descrita abaixo:

```
INSERT INTO NomeTabela (Campo1, Campo2, CampoN) VALUES (Valor1, Valor2, ValorN)
```

Todos os dados que forem alfanuméricos precisam estar entre ‘apóstrofes’ inclusive as datas.

Acompanhe a sintaxe para a inserção de dados em um banco de dados.

```
Dim Matricula, Nome As String
Dim InstrucaoSql As String
Dim conn As OleDbConnection
Dim cmd As OleDbCommand
```

```
Matricula = "101245"
Nome = "Juliana Avila"
```

```
InstrucaoSql = "INSERT INTO Tab_Aluno (CampoMatricula, CampoNome)"
InstrucaoSql = InstrucaoSql & "VALUES(' " & Matricula & " ', ' " & Nome & " ' ) "
```

```
conn = New OleDbConnection("Provider=SQLOLEDB; server=localhost;" &
                             "database=BD;user id=sa")
```

```
conn.Open( )
cmd = New OleDbCommand(InstrucaoSql, conn)
cmd.ExecuteNonQuery( )
conn.Close( )
```

A instrução SQL utilizada no exemplo acima na linguagem Visual Basic.Net , possui a mesma sintaxe para a linguagem C#.

Neste exemplo usamos uma string chamada Matricula e outra chamada Nome para facilitar o estudo, mais estas informações poderiam vir de um componente de entrada de dados, como: TextBox, DropDownList, CheckBoxList, dentre outros.

Excluir Dados

Esta operação é tão necessária e utilizada quanto as outras estudadas até este momento.

Porem devemos tratar a exclusão com muita cautela e segurança. Antes da implantação de um aplicativo que se faz uso desta instrução, devemos criar um banco de dados de teste para executar a operação de exclusão.

Vamos acompanhar a sintaxe da instrução de exclusão:

```
DELETE FROM NomeTabela WHERE CONDICAO
```

A clausula CONDICAO contida na instrução acima é opcional.

Para realizar a condição da instrução de exclusão, utilizamos a palavra-chave **WHERE**. Devemos observar que se o campo é alfanumérico, é preciso estar entre apóstrofes.

Vamos acompanhar a exclusão de dados em um banco de dados.

```
Dim Matricula As String
Dim InstrucaoSql As String
Dim conn As OleDbConnection
Dim cmd As OleDbCommand

Matricula = "101245"

InstrucaoSql = "DELETE FROM Tab_Aluno"
InstrucaoSql = InstrucaoSql "WHERE CampoMatricula = ` " & Matricula & " ` "

conn = New OleDbConnection("Provider=SQLOLEDB; server=localhost;" &
    "database=BD;user id=sa")

conn.Open( )
cmd = New OleDbCommand(InstrucaoSql, conn)
cmd.ExecuteNonQuery( )
conn.Close( )
```

A instancia do objeto OleDbCommand executa a instrução SQL para excluir o registro efetivamente.

Como é uma instrução do tipo **DELETE**, então é usado o ExecuteNonQuery para executá-la.

Atualizar Dados

Atualizar dados em um banco de dados é outra prática bastante utilizada nas aplicações.

Os bancos de dados de empresas corporativas passam por constantes mudanças. Imagine o numero de transações de atualizações que ocorrem em um banco de dados de organização de comercio eletrônico.

Então devemos tomar o cuidado de deixar apenas pessoas autorizadas a atualizar os dados nas tabelas.

A atualização de dados deve ser bem estruturada e concisa, pois os critérios devem estar bem claros e definidos, pois a instrução de atualização pode determinar uma mudança em um único produto, ou em vários produtos, como um aumento de preços.

Vejamos a sintaxe para a instrução de exclusão de dados.

```
UPDATE Tabela SET Campo1=Valor1, Campo2=Valor2, CampoN=ValorN WHERE CONDICAO
```

A clausula CONDICAO contida na instrução acima é opcional.

Para realizar a condição da instrução de atualização, utilizamos a palavra-chave **WHERE**. Devemos observar que se o campo é alfanumérico, é preciso estar entre apóstrofos.

Vamos acompanhar a atualização em um banco de dados.

```
Dim Matricula, Nome As String
Dim InstrucaoSql As String
Dim conn As OleDbConnection
Dim cmd As OleDbCommand
```

```
Matricula = "101245"
Nome = "Luciana Silva"
```

```
InstrucaoSql = "UPDATE Tab_Aluno SET CampoNome = ` "& Nome &" ' "`
InstrucaoSql = InstrucaoSql "WHERE CampoMatricula = ` "& Matricula &" ' "`
```

```
conn = New OleDbConnection("Provider=SQLOLEDB; server=localhost;" &
    "database=BD;user id=sa")
```

```
conn.Open( )
cmd = New OleDbCommand(InstrucaoSql, conn)
cmd.ExecuteNonQuery( )
conn.Close( )
```

DataGrid

O DataGrid é uma das melhores maneiras de exibir os dados para o usuário, de uma forma organizada, simples e eficaz.

Este controle utiliza colunas para exibir dados em um layout de grade. Por padrão, o DataGrid gera uma coluna para cada campo no armazenamento de dados. Entretanto é possível especificar os campos para serem exibidos manualmente, bem como a maneira de exibi-los.

O controle DataGrid escolhe automaticamente um tipo de coluna baseada nos dados apresentados, mas é suficientemente fácil alterar o comportamento padrão.

Vamos acompanhar a sintaxe deste controle:

```
<asp:DataGrid id="programmaticID"
  DataSource='<%# DataBindingExpression %>'
  AllowPaging="True|False"
  AllowSorting="True|False"
  AutoGenerateColumns="True|False"
  BackImageUrl="url"
  CellPadding="pixels"
  CellSpacing="pixels"
  DataKeyField="DataSourceKeyField"
  GridLines="None|Horizontal|Vertical|Both"
  HorizontalAlign="Center|Justify|Left|NotSet|Right"
  PagedDataSource
  PageSize="ItemCount"
  ShowFooter="True|False"
  ShowHeader="True|False"
  VirtualItemCount="ItemCount"
  OnCancelCommand="OnCancelCommandMethod"
  OnDeleteCommand="OnDeleteCommandMethod"
  OnEditCommand="OnEditCommandMethod"
  OnItemCommand="OnItemCommandMethod"
  OnItemCreated="OnItemCreatedMethod"
  OnPageIndexChanged="OnPageIndexChangedMethod"
  OnSortCommand="OnSortCommandMethod"
  OnUpdateCommand="OnUpdateCommandMethod"
  runat="server" >

  <AlternatingItemStyle ForeColor="Blue"/>
  <EditItemStyle BackColor="Yellow"/>
  <FooterStyle BorderColor="Gray"/>
  <HeaderStyle BorderColor="Gray"/>
  <ItemStyle Font-Bold="True"/>
  <PagerStyle Font-Name="Ariel"/>
  <SelectedItemStyle BackColor="Blue"/>

</asp:DataGrid>
```

A listagem acima demonstra apenas algumas propriedades deste controle, que possui uma grande quantidade de propriedades que o desenvolvedor pode manipular para conseguir o resultado esperado.

Vamos entender um pouco mais deste controle no próximo exemplo, que demonstra o recurso de paginação.

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>

<html>
  <script runat="server">

    Dim Cart As DataTable
    Dim CartView As DataView

    Function CreateDataSource() As ICollection
      Dim dt As New DataTable()
      Dim dr As DataRow

      dt.Columns.Add(New DataColumn("IntegerValue", GetType(Int32)))
      dt.Columns.Add(New DataColumn("StringValue", GetType(String)))
      dt.Columns.Add(New DataColumn("CurrencyValue", GetType(Double)))

      Dim i As Integer
      For i = 0 To 99
        dr = dt.NewRow()

        dr(0) = i
        dr(1) = "Item " + i.ToString()
        dr(2) = 1.23 *(i + 1)
        dt.Rows.Add(dr)
      Next i

      Dim dv As New DataView(dt)
      Return dv
    End Function

    Sub Page_Load(sender As Object, e As EventArgs)

      If Not IsPostBack Then
        ' É necessário criar os dados apenas uma vez.
        ItemsGrid.DataSource = CreateDataSource()
        ItemsGrid.DataBind()
      End If

      If CheckBox1.Checked Then
        ItemsGrid.PagerStyle.Mode = PagerMode.NumericPages
      Else
        ItemsGrid.PagerStyle.Mode = PagerMode.NextPrev
      End If
    End Sub

    Sub Grid_Change(sender As Object, e As DataGridPageChangedEventArgs)
      'faz a navegação de paginação.
      ItemsGrid.CurrentPageIndex = e.NewPageIndex
      'Atualiza do dados na tabela
      ItemsGrid.DataSource = CreateDataSource()
      ItemsGrid.DataBind()
    End Sub 'Grid_Change

  </script>
```

```
<body>

  <form runat="server">

    <h3>DataGrid Paging Example</h3>

    <asp:DataGrid id="ItemsGrid" runat="server"
      BorderColor="black"
      BorderWidth="1"
      CellPadding="3"
      AllowPaging="true"
      AutoGenerateColumns="false"
      OnPageIndexChanged="Grid_Change">

      <HeaderStyle BackColor="#00aaaa">
      </HeaderStyle>

      <PagerStyle Mode="NextPrev">
      </PagerStyle>

      <Columns>

        <asp:BoundColumn
          HeaderText="Numero"
          DataField="IntegerValue"/>

        <asp:BoundColumn
          HeaderText="Item"
          DataField="StringValue"/>

        <asp:BoundColumn
          HeaderText="Valor"
          DataField="CurrencyValue"
          DataFormatString="{0:c}">

          <ItemStyle HorizontalAlign="right">
          </ItemStyle>

        </asp:BoundColumn>
      </Columns>
    </asp:DataGrid> <br>
    <asp:CheckBox id="CheckBox1"
      Text="Modo de Paginação"
      AutoPostBack="true"
      runat="server"/>

  </form>
</body>
</html>
```

Exemplos e Exercícios

Exemplo 01 – retorna a consulta num componente TextBox.

```
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<head>
  <title>Trabalhando com SQL Server 2000 e asp.net </title>
</head>
<body>
  <form id="form1" runat="server" >
  <p>
    <font face="arial" size="5">
    <strong> Lê um banco de dados com SqlDataReader </strong>
    </font> </p>
    <p>
    <asp:Button id="but1"
      runat="server"
      Text="Ler banco de dados "
      OnClick="Ler_db" >
    </asp:Button> </p>

    <p>
    <asp:TextBox id="saida"
      runat="server"
      TextMode="MultiLine"
      Width="650"
      Height="265" >
    </asp:TextBox> </p>

  </form>
</body>

<script language="C#" runat="server" >
  public void Ler_db(object sender, EventArgs ea)
  {
    string conn = "server=localhost; database=Livro; uid=SQLService; pwd=atenas";
    string sql = "SELECT * FROM Tabela_Cliente";
    SqlConnection myConexao = new SqlConnection(conn);
    SqlCommand myComando = new SqlCommand (sql,myConexao);
    myConexao.Open( );
    SqlDataReader myLeitura = myComando.ExecuteReader( );
    saida.Text=" ";
    try {
      while(myLeitura.Read()) {
        saida.Text+=myLeitura["cod_cli"] + "" + myLeitura["nome_cli"] + "" +
          myLeitura["email_cli"] + "\n";
      }
    }
    finally
    {
      myLeitura.Close( );
      myConexao.Close( );
    }
  }
</script>

</html>
```

Exemplo 02 – Monta um tabela com o resultado de uma consulta.

```

<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<body>
<form id="form1" runat="server">
<font face="tahoma" color=red><b>Consulta uma Tabela no SqlServer 2000</b></font>
</form>
<script language="C#" runat="server">
public void Page_Load(Object sender, EventArgs ea) {
    if(!IsPostBack) {
        string conn = "server=localhost; database=Livro; uid=SQLService; pwd=atenas";
        string sql = "SELECT * FROM Tabela_Cliente";
        SqlConnection myConexao = new SqlConnection(conn);
        myConexao.Open( );
        try {
            SqlCommand myComando = new SqlCommand (sql,myConexao);
            SqlDataReader Leitura = myComando.ExecuteReader( );
            try {
                Response.Write("<b>Lê uma Tabela e exibe com Response.write</b><br><br>");
                Response.Write("<table width='80%' border='1' cellpadding='0' cellspacing='0' bordercolor='#000066' bgcolor='#EEEEEE' >");
                Response.Write("<tr>");
                Response.Write("<th>");
                Response.Write("<b>Código</b>");
                Response.Write("</th>");
                Response.Write("<th>");
                Response.Write("<b>Nome</b>");
                Response.Write("</th>");
                Response.Write("<th>");
                Response.Write("<b>e-mail</b>");
                Response.Write("</th>");
                Response.Write("</tr>");
                while(Leitura.Read()) {
                    Response.Write("<tr>");
                    Response.Write("<th>");
                    Response.Write("<font face='Arial, Helvetica, sans-serif' color='red' size='2' >");
                    Response.Write( Leitura["cod_cli"].ToString( ) );
                    Response.Write(" </font>" );
                    Response.Write("</th>");
                    Response.Write("<td>");
                    Response.Write("<font face='Arial, Helvetica, sans-serif' color='green' size='2'>");
                    Response.Write("<br><br>");
                    Response.Write(Leitura["nome_cli"].ToString( ));
                    Response.Write(" </font>" );
                    Response.Write("</td>");
                    Response.Write("<td>");
                    Response.Write("<font face='Arial, Helvetica, sans-serif' color='#003399' size='2' >");
                    Response.Write("<br><br>");
                    Response.Write(Leitura["email_cli"].ToString( ));
                    Response.Write("</font>");
                    Response.Write("</td>");
                    Response.Write("</tr>");
                }
                Response.Write("</table>");
            }
            finally {
                Leitura.Close( );
            }
        }
        finally {
            myConexao.Close( );
        }
    }
}
</script>
</body>
</html>

```


Exercício 01 – Codificar os exemplos para Visual Basic.Net.

Como você deve ter percebido, todos os exemplos estão codificados na linguagem C#.

Para exercitar os exemplos, codifique-os para a linguagem Visual Basic.net.

Capítulo 9 - Componentes

Introdução a componentes

Componentes podem ser definidos como objetos que podem ser utilizados repetidamente em aplicativos diferentes. Normalmente caracterizam um objeto real.

Um componente é uma forma inteligente de utilizar em diversos aplicativos uma rotina desenvolvida em uma determinada linguagem, como C++, Visual C, Visual Basic ou Delphi. Normalmente são arquivos DLL ou EXE que contêm classes, objetos e propriedades que são manipulados dentro do código.

A plataforma .Net acabou com o incomodo de ter que registrar as DLLs. Agora não há mais a necessidade de registrar os componentes. O papel do Registry do Windows não afetará em nada essas rotinas.

Vantagens

Podemos citar algumas vantagens de utilizar componentes na plataforma .Net, como:

- ☒ Facilidade de invocar os componentes por meio de referências.
- ☒ Garantir que uma rotina seja feita da mesma maneira para todos os aplicativos que utilizam este componente.
- ☒ Facilita a manutenção, uma vez que apenas uma mudança em uma determinada rotina afetará também os outros aplicativos que fazem uso deste componente.
- ☒ Portabilidade para os demais aplicativos que utilizaram a plataforma .Net para o desenvolvimento.
- ☒ Padronizar o uso de rotinas necessárias em todas as aplicações.
- ☒ Adicionar componentes fabricados por outros desenvolvedores.
- ☒ Dispensa o registro no Windows. Eliminando o conflito de registros de diversos componentes com o mesmo nome.
- ☒ Gerenciamento melhor na memória.

Problemas

Podemos citar ainda que poucos, alguns problemas em se utilizar componentes, como:

- ☒ Para determinar o sucesso e a aplicabilidade do componente, é indispensável que o desenvolvedor tenha um conhecimento da linguagem a ser criada, uma vez que este componente tem que abranger todas as possibilidades do uso.
- ☒ O uso de um componente apenas uma única vez, em um único aplicativo é desnecessário, o desenvolvedor vai criar uma complexidade desnecessária.

Namespaces

Para termos uma integração com os objetos tipo Page, ou seja, as páginas **.aspx**, a compilação do componente deve ser feita em um arquivo **.DLL**. Neste caso o disparo é dado pelo **CLR** quando seu environment (no caso o Asp.Net) recebe um **GET** de solicitação de página **.ASPX**, com ou sem o identificador **_viewstate**.

Assim, com o *statement* da definição da página, é criado um objeto de uma classe que está num arquivo **.DLL**, que é instanciado e ligada ao objeto tipo Page.

Para que esta ligação aconteça é preciso que a classe seja organizada dentro de um conjunto chamado **namespace**.

Assim, um programa que define uma classe Aluno pode ser colocado em um **namespace** NamesAluno com a sintaxe descrita abaixo:

```
Namespace NamesAluno {
```

```
    class Aluno {
```

```
        // Aqui a definição da classe C#.
```

```
    }
```

```
}
```

```
Namespace NamesAluno
```

```
    class Aluno
```

```
        // Aqui a definição da classe VB.
```

```
    End class
```

```
End Namespace
```

Não existe uma ligação entre o namespace e um arquivo **.DLL**. Pois estes arquivos podem conter classes de vários namespace, e um namespace pode ter classes em diferentes arquivos **.DLL**. Enfim, o namespace é um agrupamento lógico e não físico.

No início do arquivo que define a criação de uma nova classe, que faz uso de objetos de outras classes, devemos importar estes objetos adicionando nas primeiras linhas a sintaxe mostrada abaixo:

```
Para C#:          using NamespaceAluno;
```

```
Para VB.Net:      Imports NamespaceProfessor
```

No caso da programação em arquivos **.aspx**, se quisermos usar objetos de classes que estão em um determinado **namespace**, temos que usar a cláusula **Import** no início do arquivo.

```
<%@ Import Namespace = "NamesAluno" %>
```

```
<%@ Import Namespace = "CalculoMedia" %>
```

Criando um Componente

Este componente vai acessar um banco de dados e retornar o resultado para um DataSet.

Este componente foi desenvolvido na linguagem C#.

Nome do Arquivo: conexao.cs

```
namespace conexao {  
  
using System;  
using System.Data;  
using System.Data.OleDb;  
  
public class Listar {  
  
public DataSet mostra(String caminho, String InstrucaoSql)  
{  
    OleDbConnection objconn = new  
    OleDbConnection("Provider=Microsoft.JET.OLEDB.4.0; Data Source="+caminho);  
  
    OleDbDataAdapter objcomm = new OleDbDataAdapter( InstrucaoSql, objconn);  
  
    DataSet objds = new DataSet();  
  
    objcomm.Fill(objds, "dsvMostra");  
    return objds;  
}  
  
}  
}
```

Para a compilação deste componentes siga os seguintes passos:

1. crie um diretório chamado **Bin** no diretório corrente da aplicação e salve o componente conexao.cs descrito acima no diretório Bin;
2. abra o prompt do Dos e digite no diretório Bin:
csc /t:Library /out:conexao.dll conexao.cs
3. este comando cria um arquivo **.DLL** que agora poderá ser utilizado em uma página **.ASPX**.

Usando o Componente

Agora ficou fácil utilizar o componente descrito na seção anterior. Basta referenciar o componente na página **.ASPX**, e criar um objeto da classe **Listar** para executar o método **mostra()**, apresentado abaixo:

```
<%@ Import Namespace ="conexao" %>
<%@ Import Namespace ="System.Data" %>

<html>

<form runat="server">
<h1 align="center"> Lista de Telefones </h1>
<p>
<asp:Panel id="panel1"
    BackColor="white"
    HorizontalAlign="center"
    width="100%"
    height="100"
    runat="server">
<asp:DataGrid id="dgLista"
    Width="400"
    BackColor="Gainsboro"
    BorderColor="lightGray"
    ShowFooter="false"
    CellPadding="4"
    CellSpacing="1"
    Font-Name="arial"
    Font-Size="9pt"
    HeadetStyle-BackColor="Gray"
    runat="server"/>
</asp:Panel>
</p>
</form>

<script language="c#" runat="server">

protected void Page_Load (Object sender, EventArgs ea) {
    String myCaminho;
    String mySql;

    myCaminho=Server.MapPath("../db//projeto01.mdb");
    mySql="SELECT codigo_cli, nome, mail FROM TCliente";

    Listar objlt = new Listar();
    DataSet dsvp = objlt.mostra(myCaminho, mySql);

    dgLista.DataSource = dsvp.Tables["dsvMostra"].DefaultView;
    dgLista.DataBind();

}
</script>
</html>
```

Exemplos e Exercícios

Exemplo 01 – Neste exemplo vai ser desenvolvido um componente na linguagem VB.Net chamado: [componente.vb](#), para ser utilizado na chamada da página [componente.aspx](#).

```
Imports System

Namespace Componente

Public class Somar
    Function calcular(Valor1 As Double, Valor2 As Double) As Double
        Return Valor1 + Valor2
    End Function
End Class

Public class Subtrair
    Function calcular(Valor1 As Double, Valor2 As Double) As Double
        Return Valor1 - Valor2
    End Function
End Class

Public class Dividir
    Function calcular(Valor1 As Double, Valor2 As Double) As Double
        If (Valor2 = 0) Then
            Return 0
        Else
            Return Valor1 / Valor2
        End If
    End Function
End Class

Public class Multiplicar
    Function calcular(Valor1 As Double, Valor2 As Double) As Double
        Return Valor1 * Valor2
    End Function
End Class

End Namespace

Salvar como: componente.vb
```

Para compilar o componente siga os seguintes passos:

1. crie um diretório chamado Bin no diretório da aplicação.
2. salve o arquivo: [componente.vb](#), neste diretório.
3. digite o comando para compilar o componente:

```
vbc /t:library /out:componente.dll componente.vb
```

```
<%@ Import Namespace="Componente" %>

<html>
<form runat="server">
<asp:Panel id="panel1"
    BackColor="#E0E0E0"
    HorizontalAlign="Center"
    Width="200"
    runat="server">
<br/> <br/>
<asp:TextBox id="entrada1"
    BackColor="yellow"
    Width="150"
    runat="server"/>
<p>
<asp:TextBox id="entrada2"
    BackColor="yellow"
    Width="150"
    runat="server"/> </p>
<p>
<asp:Button id="bot01"
    Text=" + "
    OnClick="MetodoEnviar"
    ToolTip="Forneça os Valores e Clique"
    runat="server"/>
<asp:Button id="bot02"
    Text=" - "
    OnClick="MetodoEnviar"
    ToolTip="Forneça os Valores e Clique"
    runat="server"/>
<asp:Button id="bot03"
    Text=" * "
    OnClick="MetodoEnviar"
    ToolTip="Forneça os Valores e Clique"
    runat="server"/>
<asp:Button id="bot04"
    Text=" / "
    OnClick="MetodoEnviar"
    ToolTip="Forneça os Valores e Clique"
    runat="server"/> </p>
<p>
<asp:Label id="saida"
    Font-Name="Arial"
    runat="server"/> </p>
<br/> <br/>
</asp:Panel>
</form>
```

```
<script language="C#" runat="server">

    public void MetodoEnviar(Object sender, EventArgs ea){

        string VarSinal = ((Button)sender).Text;
        double VarValor1, VarValor2, VarResultado;

        try {
            VarValor1 = double.Parse(entrada1.Text);
            VarValor2 = Convert.ToDouble(entrada2.Text);

            switch (VarSinal) {
                case " + ": Somar ObjSoma = new Somar( );
                    VarResultado = ObjSoma.calcular(VarValor1, VarValor2);
                    saida.Text = VarResultado.ToString( );
                    break;

                case " - ": Subtrair ObjSub = new Subtrair( );
                    VarResultado = ObjSub.calcular(VarValor1, VarValor2);
                    saida.Text = VarResultado.ToString( );
                    break;

                case " / ": Dividir ObjDiv = new Dividir( );
                    VarResultado = ObjDiv.calcular(VarValor1, VarValor2);
                    saida.Text = VarResultado.ToString( );
                    break;

                case " * ": Multiplicar ObjMult = new Multiplicar( );
                    VarResultado = ObjMult.calcular(VarValor1, VarValor2);
                    saida.Text = VarResultado.ToString( );
                    break;
            }
        }
        catch(Exception e) {
            saida.Text = "Ocorreu algum erro de conversão! ";
        }
    }
}
</script>
</html>
```

Salvar como: componente.aspx

Capítulo 10 - Web Services

Um Web Service é uma maneira uniforme pela qual os objetos em um servidor aceitam solicitações que chegam de clientes que estão usando o mínimo denominador comum do HTTP/XML. Para criar um Web Service você não tem de aprender uma nova maneira de programar. Você simplesmente escreve um objeto .NET como se ele estivesse sendo acessado diretamente por clientes locais, marca-o com um atributo que diz que você quer que ele esteja disponível para os clientes Web, e o ASP.NET faz o resto. O ASP.NET automaticamente monta uma infra-estrutura pré-fabricada que aceita as solicitações que chegam por meio do HTTP e as mapeia para chamadas no seu objeto. Quando você coloca seus objetos em um Web Service, eles podem funcionar com qualquer outro na Web que fale HTTP e XML, que será o caso de todos no universo, independentemente do tipo de sistema operacional e ambiente de processamento em que estiverem funcionando. Você não precisa escrever a infra-estrutura que lida com a comunicação Web; o .NET Framework providencia isso para você.

No lado cliente, o .NET proporciona classes proxies que permitem acesso fácil e baseado em função aos Web Services fornecidos por qualquer servidor que aceite solicitações HTTP. Uma ferramenta de desenvolvedor lê a descrição do Web Service e gera uma classe Proxy contendo funções na linguagem que você usa para desenvolver o cliente. Quando o seu cliente chama uma dessas funções, a classe Proxy cria uma solicitação HTTP e a envia para o servidor. Quando a resposta volta do servidor, a classe Proxy analisa os resultados e os retorna da função. Isso permite que o seu cliente baseado em função possa interagir de forma uniforme com qualquer servidor Web que fale HTTP e XML, que é a linguagem que deverá ser falada por todos.

Exemplo de um Web Service básico:

```
<%@ WebService Language="VB" Class="TimeService"%>
' A linha do cabeçalho acima diz ao ASP.NET que este arquivo contém
' um Web Service escrito na linguagem Visual Basic e que o
' nome da classe que fornece aquele serviço é TimeService
' Importe os namespaces (pense neles como referências)
' necessários para um Web Service.

Imports System
Imports System.Web.Services

' Declare uma nova classe para nosso novo serviço.
' Ela deve herdar da classe-base fornecida pelo sistema, WebService.

Public Class TimeService : Inherits WebService
' Coloque as funções na classe
' Marque-as como WebMethods
Public Function <WebMethod()> GetTime (ShowSeconds as Boolean) As String

' Execute a regra de negócio de nossa função.
' Determine a hora atual, formate conforme solicitado e retorne a string.

Dim dt As DateTime

If (ShowSeconds = True) Then
    GetTime = dt.Now.ToLongTimeString
Else
    GetTime = dt.Now.ToShortTimeString
End If

End Function
End Class
```

SOAP

SOAP que significa Simple Object Access Protocol, é um vocabulário XML que descreve chamadas de função e seus parâmetros.

O pacote SOAP enviado ao servidor contém o nome da função e seus parâmetros, codificados em XML de acordo com um esquema convencionado, como você pode ver na caixa de texto superior. Quando o pacote SOAP alcança o servidor, ASP.NET o reconhece, analisa o nome do método e seus parâmetros no pacote, cria o objeto e faz a chamada. Ele toma o valor de retorno desse método, formata-o em XML e retorna o XML para o cliente.

Criando um WebService

Vamos ver como criar um WebService simples, que faz apenas um cálculo de soma entre dois números inteiros.

Para facilitar o exercício, vamos considerar que o cliente e os servidores são apenas uma única máquina, e que o URL do servidor1 e do servidor2 é **127.0.0.1**.

Acompanhe o exemplo passo a passo:

1. crie uma pasta sob o diretório **Inetpub\Wwwroot** com o nome de **WebTeste** e salve neste diretório o arquivo **Somaserv.asmx**.

```
<%@ WebService Language="C#" Class="Somaserv" %>
```

```
using System;
```

```
using System.Web.Services;
```

```
public class Somaserv {
```

```
[WebMethod] public int soma(int a, int b){  
    return a+b;
```

```
}
```

```
}
```

salve como: Somaserv.asmx

A diretiva superior diz que este é um arquivo de WebService. Precisamos também importar o namespace **System.Web.Services** para qualificar o método soma da classe SomaServ como um método de um WebService.

2. você agora pode testar este WebService abrindo o seu navegador e chamando o arquivo como mostrado abaixo:

<http://127.0.0.1/webteste/somaserv.asmx>

3. agora temos que compilar o WebService. Esta etapa é importantíssima pois vai criar o proxy do Servidor1 para o stub do Servidor2, ou seja, a conexão entre os servidores. Esta compilação vai resultar na criação de um arquivo chamado Somaserv.cs que é o fonte do nosso proxy.

Vamos a compilação:

```
wsdl /out:Somaserv.cs  
/n:WebSoma  
http://127.0.0.1/webteste/Somasrv.asmx?wsdl
```

Temos que observar que: com [/n:WebSoma](#) definimos o nome do namespace da nossa classe.

4. agora temos que compilar o arquivo Somaserv.cs , gerado na compilação do WebService, em uma pasta chamada **Bin** criada no diretório WebTeste. O diretório Bin é um padrão que o .Net FrameWork utiliza para buscar os componentes.

Vamos acompanhar abaixo a sintaxe para a compilação do arquivo Somaserv.cs:

```
csc /t:library /out:bin\Somaserv.dll Somaserv.cs
```

Usando o WebService

Para utilizarmos o WebService, basta seguir os passos abaixo:

1. vamos criar o arquivo: Somaserv.aspx para utilizar o WebService. Note que para chamarmos o arquivo Somaserv.asmx devemos importar a classe Somaserv que esta contida no namespace WebSoma.

Acompanhe o código do arquivo .aspx:

```
<%@ Import Namespace="WebSoma" %>
<html>
<form runat="server">
<asp:Panel id="panel01"
    BackColor="#E0E0E0"
    HorizontalAlign="center"
    Width="250"
    runat="server"> <br/>
    Adicione dois números para a soma          <br/> <br/>
<asp:TextBox id="numero01"
    BackColor="yellow"
    Width="110"
    runat="server"/> <br/>
<asp:TextBox id="numero02"
    BackColor="yellow"
    Width="110"
    runat="server"/> <br/>
<p>
<asp:Button id="bot01"
    Text="Somar Inteiros"
    ToolTip="Adicione dois valores inteiros"
    OnClick="MetodoSomar"
    runat="server"/> </p>
<p>
<asp:Label id="saida"
    Font-Name="Arial"
    Font-Size="16"
    runat="server"/> <p/>
</asp:panel>
</form>
<script language="VB" runat="server">
    Sub MetodoSomar (obj As Object, ea As EventArgs)
        Dim Valor1, Valor2 As Integer
        Try
            Valor1 = Val(numero01.Text)
            Valor2 = CInt(numero02.Text)          'Valor2 = Integer.Parse(numero02.Text)
            Dim ObjSoma As Somaserv
            ObjSoma = new Somaserv()
            Dim Resultado As String
            Resultado = CStr(ObjSoma.soma(Valor1, Valor2))
            saida.Text = Valor1 & " + "& Valor2 & " = "& Resultado
        Catch
            saida.Text = "Ocorreu um erro !"
        End Try
    End Sub
</script>
</html>
```

Referências Bibliográficas

Microsoft. **Documentação do .Net Framework SDK**. USA:Microsoft, 2002.

Platt, David S. **Iniciando Microsoft .Net**. Editora Makron Books, 2002.

Payne, Chris. **Aprenda em 21 dias ASP.Net**. Editora Campus, 2001.

Haddad, Renato. **C# - Aplicações e Soluções**. Editora Érica, 2001.

Haddad, Renato. **VB.Net – Aplicações e Soluções**. Editora Érica, 2002.

Damasceno Jr, Américo. **Aprendendo ASP.Net com C#**. Editora Érica, 2001.